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 10 Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

База данных «Кинотеатр»

Выполнил студент Реут Владислав Андреевич

(Ф.И.О.)

Руководитель проекта ст.пр. Блинова Е.А.

(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: ст.пр. Блинова Е.А.

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

Нормоконтролер: ст.пр. Блинова Е.А.

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

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит Т страницу пояснительной записки, P иллюстраций, 12 источников литературы, Y приложения.

Основная цель курсового проекта: проектирование базы данных кинотеатра.

В первой главе осуществляется постановка задачи.

Вторая глава посвящена процессу разработки модели базы данных.

В третьей главе приведено описание разработки объектов базы данных.

В четвертой главе описаны процедуры экспорта и импорта данных.

В пятой главе описано тестирование производительности.

Шестая глава посвящена технологиям.

В заключении приведены результаты проделанной работы.

**Abstract**

The explanatory note of the course project contains 41 pages of an explanatory note, 13 illustrations, 12 sources of literature, 3 applications.

The main goal of the course project: designing a database of a train station.

In the first chapter, the problem is formulated.

The second chapter focuses on the process of developing a database model.

The third chapter describes the development of database objects.

The fourth chapter describes the procedures for exporting and importing data.

The fifth chapter describes performance testing.

The sixth chapter is devoted to diagnostic tools and encryption.

In conclusion, the results of the work done are presented.

# **Содержание**

[**Содержание** 4](#_Toc59040167)

[**Введение** 6](#_Toc59040168)

[1. Постановка задачи 7](#_Toc59040169)

[2. Разработка модели базы данных 8](#_Toc59040170)

[3. Разработка необходимых объектов 9](#_Toc59040171)

[3.1. Таблицы 9](#_Toc59040172)

[3.2. Процедуры 10](#_Toc59040173)

[3.3. Пользователи 11](#_Toc59040174)

[4. Описание процедур импорта и экспорта 13](#_Toc59040175)

[5. Тестирование производительности 16](#_Toc59040176)

[6. Описание технологии 17](#_Toc59040177)

[6.1 Технология «Мультимедийные типы данных Oracle» 17](#_Toc59040178)

[6.2 Применение технологии «NoSQl» 19](#_Toc59040179)

[6.3 Технология «Шифрование» 22](#_Toc59040180)

[6.4 Применение технологии «Шифрование» 24](#_Toc59040181)

[Заключение 25](#_Toc59040182)

[**Список используемых источников** 26](#_Toc59040183)

[Приложение A 27](#_Toc59040184)

[Приложение Б 28](#_Toc59040185)

[Приложение В 39](#_Toc59040186)

[40](#_Toc59040187)

# **Введение**

Кинотеатр — общественное учреждение для демонстрации кинофильмов.

Главное помещение кинотеатра — зрительный зал с экраном большого размера и сситемой воспроизведения звука.

Просмотр фильмов стал намного доступнее в связи с развитием информационных технологий и появлением таких сервисов как онлайн кинотеатры, где любой фильм можно купить по приемлемой цене, так и стриминговые платформы, где снимаются фильмы или сериалы не хуже мировых концернов.

Однако это не помешало развиваться сфере услуг по просмотру фильмов в кинотеатрах. Постоянно появляются новые достижения в этой сфере, например такие как: показ культовых фильмов в кинотеатрах, которые уже выходили в прокат (что до этого считалось неправильным), или постоянное улучшение оборудования самих кинотеатров.

Теперь почти каждый торговый центр имеет в своем арсенале небольшой, кинотеатр на верхнем этаже.

Целью данного курсового проекта является создание реляционной базы данных для функционирования кинотеатра. В качестве СУБД была выбрана Oracle Database 12c.

База данных должна содержать данные о пользователях, фильмах, сеансов к ним, кинозалов и билетов.

При разработке базы данных необходимо создать основные объекты: таблицы, процедуры, пользователи. В базе данных должна быть реализована возможность экспорта и импорта данных формата XML в таблицы базы данных. Также после создания объектов и добавления данных в таблицы, необходимо протестировать базу данных на производительность и при необходимости провести оптимизацию. В качестве технологии для курсового проекта была выбрана NoSql

1. Постановка задачи

Основной задачей курсового проекта является разработка базы данных «Кинотеатр». Главная цель разрабатываемой базы данных заключается в возможности полноценного функционирования кинотеатра. Данная база должна упростить работу администраторов, предоставляющих места для работы кинозала, а также оптимизировать процесс заказа билетов для покупателей.

Задачу можно сформулировать следующим образом: создать необходимые сущности, которые будут отражать хранимые данные; охарактеризовать эти сущности с помощью соответствующих атрибутов с корректно подобранными ограничениями целостности; связать таблицы между собой, тем самым получив полноценную реляционную структуру.

Необходимо реализовать средства диагностики и протестировать производительность базы данных на таблицах, содержащих не менее 100 000 строк.

Функционально должны быть выполнены следующие задачи:

* возможность регистрации и авторизации пользователя;
* возможность добавления, удаления и изменения фильмов;
* возможность добавления, удаления и изменения сеансов к фильмам;
* возможность осуществлять заказ билетов.

1. Разработка модели базы данных

Один из главных принципов организации данных – построение взаимосвязей между всеми элементами, что и отличает базу данных от простого набора таблиц. Информация в таблицах реляционной базы данных должна быть соответствующим образом организована. При разработке модели базы данных использовалась нормализация таблиц, которая сводится к устранению недостатков структуры базы данных, приводящих к различным аномалиям и нарушениям целостности данных. Недостатками структуры можно назвать, например, противоречивость данных, а аномалией – возникновение случайных ошибок в процессе эксплуатации базы данных. Таким образом, нормализацией можно считать разбиение таблицы на две или более для исключения повторения информации.

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение.

На рисунке 2.1 изображена модель базы данных, полученная в результате анализа необходимого функционала базы данных. Между сущностями, представленными таблицами, настроены связи, отражающие их взаимосвязь.

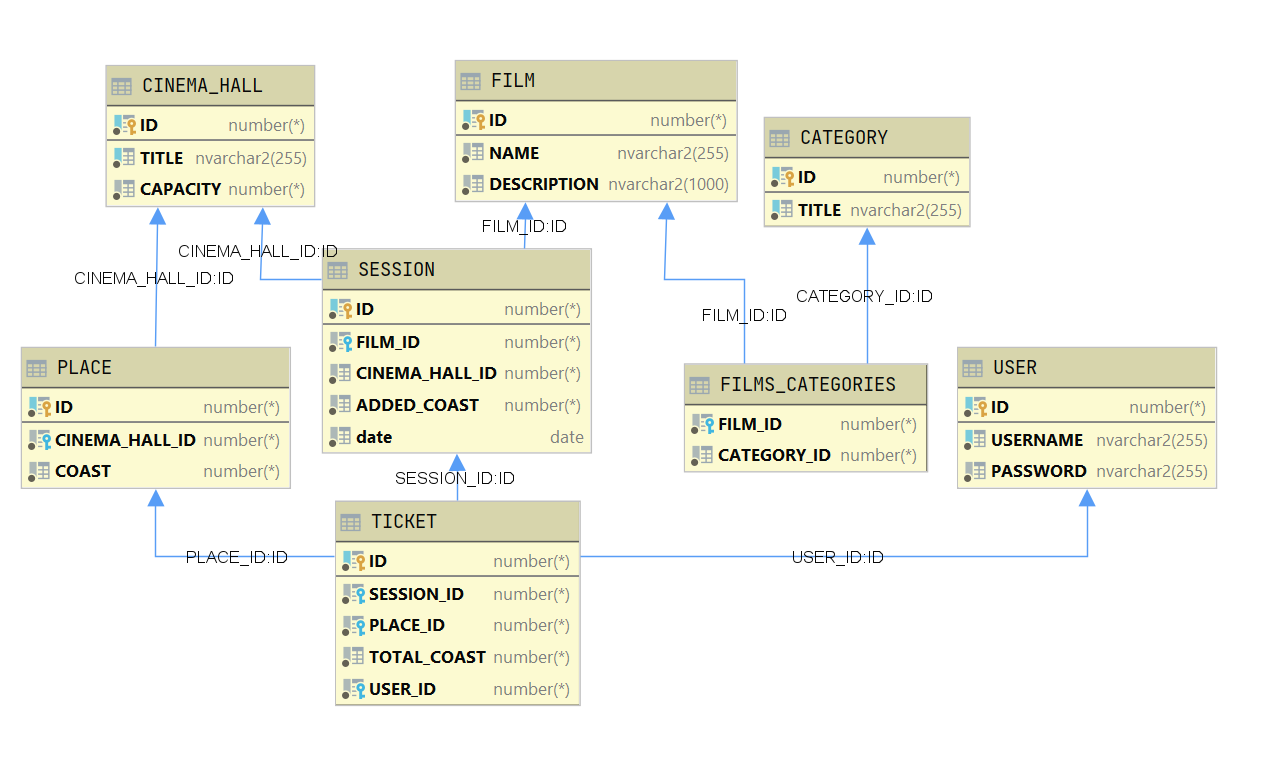


Рисунок 2.1 – Структурная схема базы данных

На рисунке 2.1 проиллюстрированы связи, которые использовались в разработке реляционной базы данных. В построении применялись связи между таблицами один-ко-многим и многие ко многим.

1. Разработка необходимых объектов

В ходе выполнения курсового проекта была разработана подключаемая база данных cinemapdb со следующими объектами: таблицы, роль, пользователи, хранимые процедуры.

* 1. Таблицы

Центральным объектом любой базы данных является связанная совокупность таблиц. Таблицы ограничивают и упорядочивают хранимую информацию, а также обеспечивают связанность за счет внешних ключей.

База данных на представленной схеме на рис. 2.1 в полной мере описывает необходимые таблицы для решения поставленной задачи данного курсового проекта. Для реализации базы данных «Кинотеатр» были разработаны следующие 8 таблиц.

Таблица User содержит информацию о пользователях и включает 3 столбца:

* столбец id является первичным ключом, тип данных number, инкрементируется при добавлении пользователя;
* столбец username хранит логин пользователя, тип данных varchar2(30), не может быть null и может хранить только уникальные значения;
* столбец password хранит пароль пользователя, тип данных varchar2(100) и не может быть null.

Таблица Film содержит информацию о поездах и включает 3 столбца:

* столбец id является первичным ключом, тип данных number, инкрементируется при добавлении фильма;
* столбец name хранит название фильма, тип данных nvarchar2(255) и не может быть null.
* столбец description хранит описание к фильму, тип данных nvarchar2(1000).

Таблица Category содержит информацию о возможных категориях к фильму включает 2 столбца:

* столбец Id является первичным ключом, тип данных number, инкрементируется при добавлении категории;
* столбец title хранит название категории тип данных nvarchar2(255)

Таблица мост Films\_Categories содержит информацию о том какой фильм к каким жанрам относится:

* столбец film\_id является внешним ключом к таблице film;
* столбец category\_id является внешним ключом к таблице category

Таблица Cinema\_hall содержит информацию о кинозале:

* столбец Id является первичным ключом, тип данных number, инкрементируется при добавлении
* столбец title отражает понятное название зала, тип данных nvarchar2(255)
* столбец capacity является отражением максимального количества человек, которое может вместить в себя зал, исходя из площади помещения и качества вентиляции воздуха

Таблица Session содержит информацию о сеансе

* столбец Id является первичным ключом, тип данных number, инкрементируется при добавлении
* столбец film\_id хранит внешний ключ на id фильма к сеансу, тип данных integer и не может быть null;
* столбец cinema\_hall\_id внешний ключ, хранит идентификатор зала в котором проводится фильм, не может быть null;
* столбец added\_coast, хранит добавочную стоимость к сеансу тип данных integer.
* столбец date, хранит дату проведения сеанса к фильму.

Таблица place содержит информацию о каждом месте в кинозале, содержит 3 столбца:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении места;
* столбец cinema\_hall\_id является внешним ключом к таблице кинозала к которому прикреплено место, тип данных int.
* столбец coast отображает стоимость места исходя из его удобства, тип данных integer.

Таблица Ticket содержит информацию о железнодорожных билетах и включает 5 столбцов:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении билета;
* столбец session\_id внешний ключ на таблицу к сеансу фильма, тип данных integer и не может быть null;
* столбец place\_id внешний ключ на таблицу места в кинозале, тип данных integer и не может быть null;
* столбец user\_id внешний ключ на таблицу пользователя, купившего билет, тип данных integer и не может быть null;

Листинги создания таблиц представлены в приложении А.

* 1. Процедуры

Процедура представляет собой модуль, выполняющий одно или несколько действий. Поскольку вызов процедуры в T-SQL является отдельным исполняемым оператором, блок кода T-SQL может состоять только из вызова процедуры. Процедуры относятся к числу ключевых компонентов модульного кода, обеспечивающих оптимизацию и повторное использование программной логики [3].

Для реализации базы данных «Кинотеатр» были разработаны 22 процедуры, реализующие функции CRUD [1].

Процедуры, связанные c манипулированием данными в таблице Users находятся в файле UserProcedures.sql:

* CheckUser – проверка наличия пользователя в базе данных;
* EncryptPassword – шифрование пароля пользователя;
* RegistrationUser – регистрация пользователя;
* AuthorizationUser – авторизация пользователя;
* DeleteUser – удаление пользователя;
* UpdateLogin – изменение логина пользователя.

Процедуры, связанные c манипулированием данными в таблице Category находятся в файле CategoryService.sql:

* createCategory – добавление категории
* DeleteCategoryById – удаление категории по id;
* DeleteCategoryByTitle – удаление категории по названию;
* UpdateCategory – изменение названия категории по id;

Процедуры, связанные c манипулированием данными в таблице Film находятся в файле FilmService.sql:

* CreateFilm– создание фильма;
* DeleteFilmById – удаление фильма по id;
* DeleteFilmByName – удаление фильма по названию;
* UpdateFilm – изменить название в фильме и описание к фильму по id;

Процедуры, связанные c манипулированием данными в таблице Session находятся в файле SessionService.sql:

* CreateSession– создание сеанса;
* DeleteSessionById – удаление сеанса по id;
* UpdateSession – изменение сеанса;

Процедуры, связанные c манипулированием данными в таблице FilmsCategories находятся в файле filmsCategoriesService.sql:

* addCategoryToFilm – добавление категории к фильму;
* deleteCategoryFromFilm – удаление категории с фильма;

Процедуры, связанные c манипулированием данными в таблице Ticket находятся в файле TicketService.sql:

* BuyTicket– купить билет к фильму;

Процедуры, связанные c манипулированием данными в таблице Place находятся в файле PlaceService.sql:

* CreatePlace–добавление места;
* DeletePlaceById – удаление места по id;
* UpdatePlace – изменение места по id;

Процедуры, осуществляющие с импортом и экспортом данных, находятся в файлах Import.sql и Export.sql соответственно:

* Import – импорт данных в таблицу;
* Export – экспорт данных из таблицы.

Листинги создания процедур представлены в приложениях Б соответственно.

* 1. Пользователи

Пользователь БД – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

В данном курсовом проекте созданы 2 пользователя: администратор базы данных и пользователь. Администратор базы данных имеет все привилегии.

Для создания пользователя и назначения привилегий была создана роль role\_cinema\_user. В качестве привилегий ему были назначены привилегии на создание сессии, получение данных из некоторых таблиц и выполнение определенных процедур.

Скрипт создания пользователя представлен на рисунке 3.1.

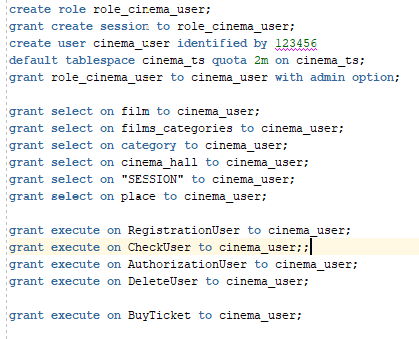


Рисунок 3.1 – Роли пользователя

**3.4 Представления**

В ходе курсового проекта были разработаны 3 различных представления.  
Films – представляющее отображение фильмов в читаемом формате.  
Записи отсортированы по id, в порядке убывания, чтобы во время работы или тестирования приложения видеть какие записи добавляются.  
 

Рисунок 3.2 – представление films

Sessions – таблица session содержит внешние ключи, указывающие на cinema\_hall и film таблицы, так как внешний ключ в виде id не несет много информации обычному пользователю, поэтому была разработано данное представление. С помощью таблицы session и соединением с ней таблиц cinema\_hall и film, вывели удобочитаемые поля.

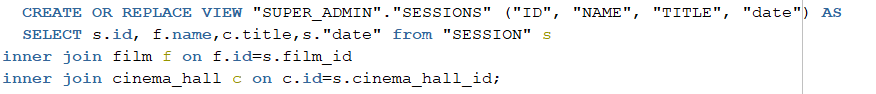


Рисунок 3.3 – представление sessions



Рисунок 3.3 – результат выполнения sessions

Tickets – представление разработано для отображения всех купленных билетов.  
содержит соединение с таблицей user, для того чтобы отобразить какой билет какой пользователь купил

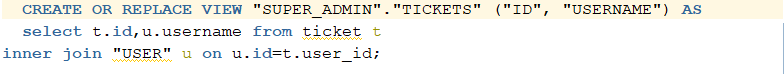


Рисунок 3.4 – представление Tickets

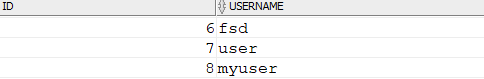


Рисунок 3.4 – результат выполнения Tickets

1. Описание процедур импорта и экспорта

Возможность импорта и экспорта данных таблиц – важный элемент приложения, так как такой формат передачи данных может быть использован для пересылки данных другим приложениям.

Для импорта данных в таблицу Film в ходе выполнения курсового проекта была разработана процедура Import Films.

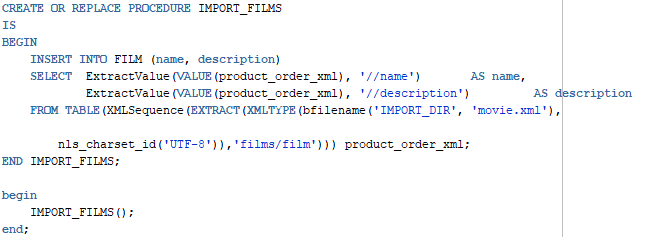


Рисунок 4.1 процедура ImportFilms

Здесь функция EXTRACTVALUE принимает в себя аргументы XPath выражения, и собственно, место, где нужно эти значения найти.

Для экспорта данных из таблиц Category, Film, cinemaHall в ходе выполнения курсового проекта была разработаны соответствующие процедуры.

Процедура экспорта данных будет рассмотрена на процедуре ExportFIlm, изображенной на рисунке 4.2.



Рисунок 4.2 – процедура Export

Первым действием требуется создать файл, куда будут экспортированы данные таблицы. Для этого используется функция fopen пакета UTL\_FILE. Чтобы созданный xml-документ был валидным, туда записывается декларация и открывается корневой тег.

Далее, двигаясь по открытому курсору, создается xml-запись о каждом фильме: тег film содержит два дочерних тега title и description, содержащих название и описание

После обработки всех записей в курсоре в файл нужно записать закрывающий корневой тег и закрыть файл.

1. Тестирование производительности

В данном разделе сравним скорость выполнения некоторых запросов для двух баз данных, Oracle и MongoDB.

Время выполнения запроса в Oracle будет фиксироваться с помощью SQL developer,  
в MongoDB будем добавлять функцию explain("executionStats"); для отображения длительности запроса.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название операции | запрос Mongo | Среднее время выполнения Mongo | запрос Oracle | Среднее Время выполнения Oracle |
| Вывод всех значений | db.film.find().explain("executionStats"); | 65мс | select \* from film | 130мс |
| Найти запись по названию | db.film.find({title: "Toy Story"}).explain("executionStats"); | 70мс | select \* from film where film.name= ‘Story’ | 380мс |
| Регулярное выражение | db.film.find({name:/Tre\w+/i}).explain("executionStats"); | 150мс | select \* from film where film.name like 'T%'; | 180мс |

1. Описание технологий

6.1 Технология «Мультимедийные типы данных Oracle»

Oracle Multimedia (в прошлом Oracle interMedia)  - это возможность Oracle Database 11g для управления мультимедийным контентом (изображения, звук, видео).  Использование этой функции позволяет управлять мультимедийными данными интегрированно с реляционной информацией. Oracle Multimedia умеет работать с большинством популярных мультимедийных форматов. Хранить такие мультимедийные данные можно разными способами (внутри и вовне базы). Также предоставляется целый спектр интерфейсов для использования этих сервисов (объектный, реляционный). Для разработки приложений Oracle поставляет набор Java интерфейсов и компонентов.

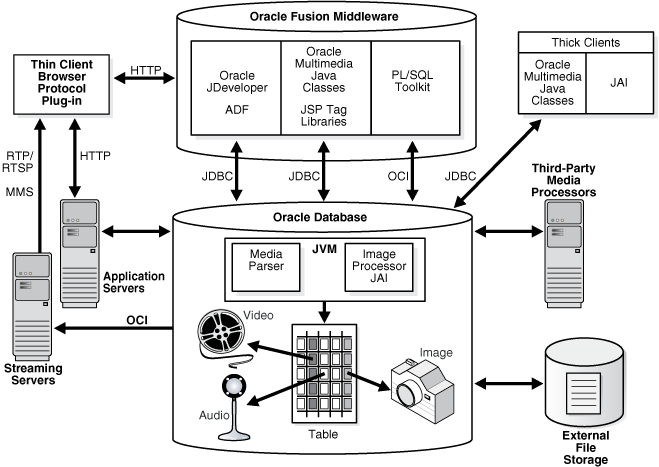


Рисунок 6.1 устройство Oracle Multimedia

Контент для мультимедийных типов данных может состоять из 4 различных типов. Коротко о каждом из них:

* BFILE — двоичный файл. Переменная этого типа содержит локатор файла, указывающий на файл операционной системы вне базы данных. Oracle интерпретирует содержимое файла как двоичные данные.
* BLOB — большой двоичный объект. Переменная этого типа содержит локатор LOB, указывающий на большой двоичный объект, хранящийся в базе данных.
* CLOB — большой символьный объект. Переменная этого типа содержит локатор LOB, указывающий на хранящийся в базе данных большой блок текстовых данных в наборе символов базы данных.
* NCLOB — большой символьный объект с поддержкой символов национальных языков (NLS). Переменная этого типа содержит локатор LOB, указывающий на хранящийся в базе данных большой блок текстовых данных с национальным набором символов.

Пример использования типа Clob приводилось ранее, в функции импорта.

Далее стояло требование: сделать возможность добавления фотографии пользователя, чтобы данные этой фотографии не хранились в базе данных целиком, что позволит сократить количество запросов.

Для этой цели идеально подошел объектный тип bfile, который в себе хранит ссылку на файл операционной системы.

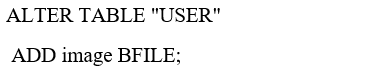


Рисунок 6.2 добавление нового поля в таблицу user

Для начала внесем изменения в таблицу User, добавив новый столбец – image,

типа BFILE. Будем подразумевать, что backend-server, будет сохранять все фотографии пользователей в одну папку, и называть файл номером id пользователя,

например, фотография пользователя с id=3 будет называться 3.png.

Напишем процедуру displayBFILE\_proc, которая отобразит байты картинки конкретного пользователя.

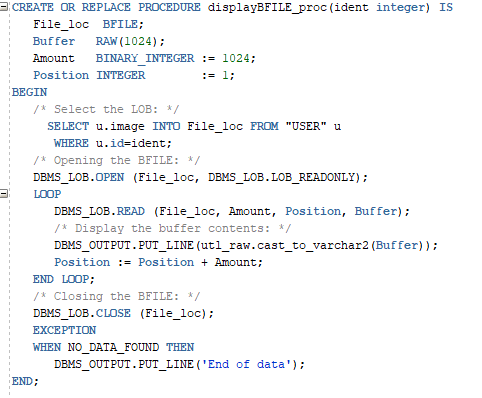


Рисунок 6.3 процедура displayBFILE\_proc

В ходе написания процедуры использовался поток вывода DBMS.OUPUT для наглядности.

CREATE OR REPLACE DIRECTORY IMAGE\_DIR AS 'D:\curse project Oracle\multi\data';

Рисунок 6.4 создание директории для изображений

Создадим директорию для расположения картинок.

Добавим в таблицу примитивного пользователя с картинкой, и скопируем в эту папку какой-нибудь png файл с названием 3.png.

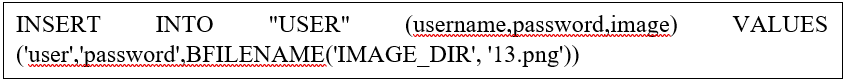


Рисунок 6.5 добавление пользователя с фотографией

Вызовем процедуру:

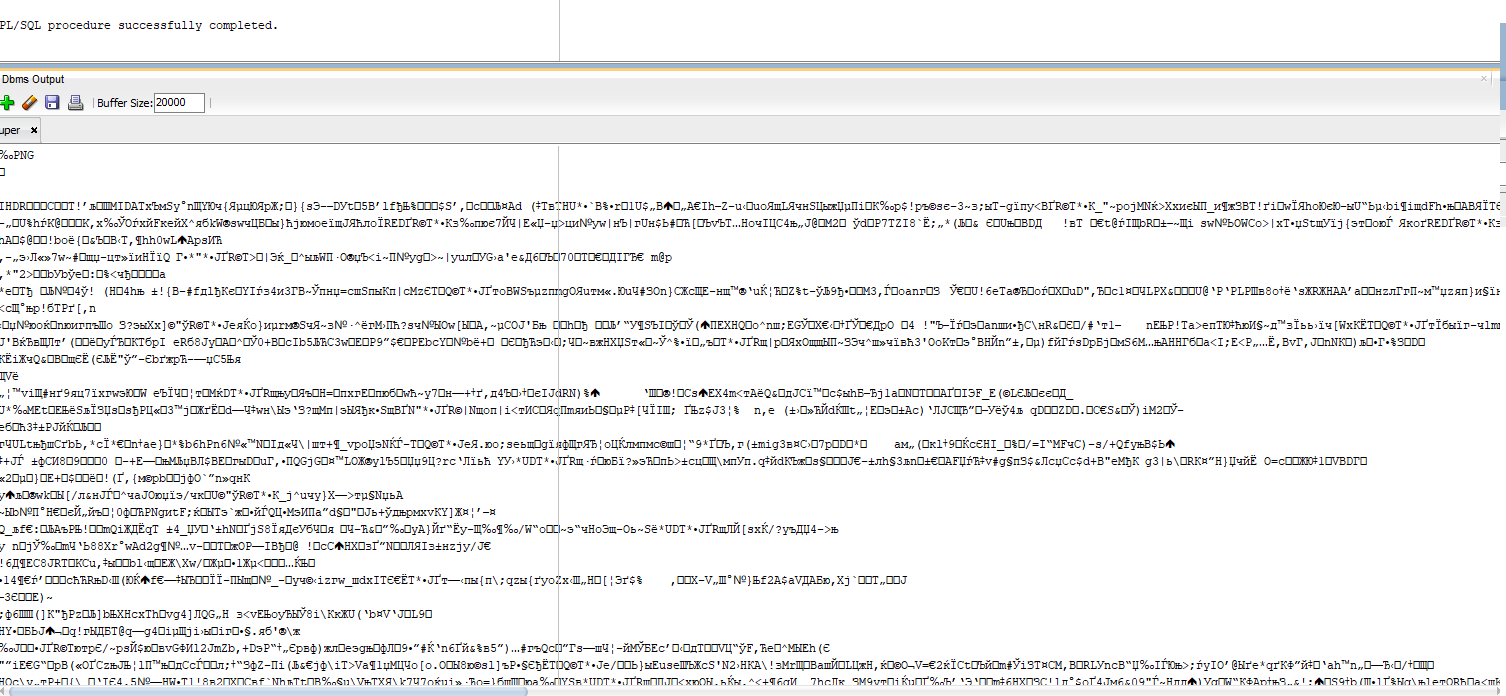


Рисунок 6.6 файл в бинарном виде

6.2 Применение технологии «NoSQl»

NoSQL базы данных – обширная и быстро развивающаяся область. Они позволяют хранить и обрабатывать неслыханные до сих пор объемы данных, но за это приходится платить. В этих базах данных нет многих привычных по реляционным базам данных возможностей.

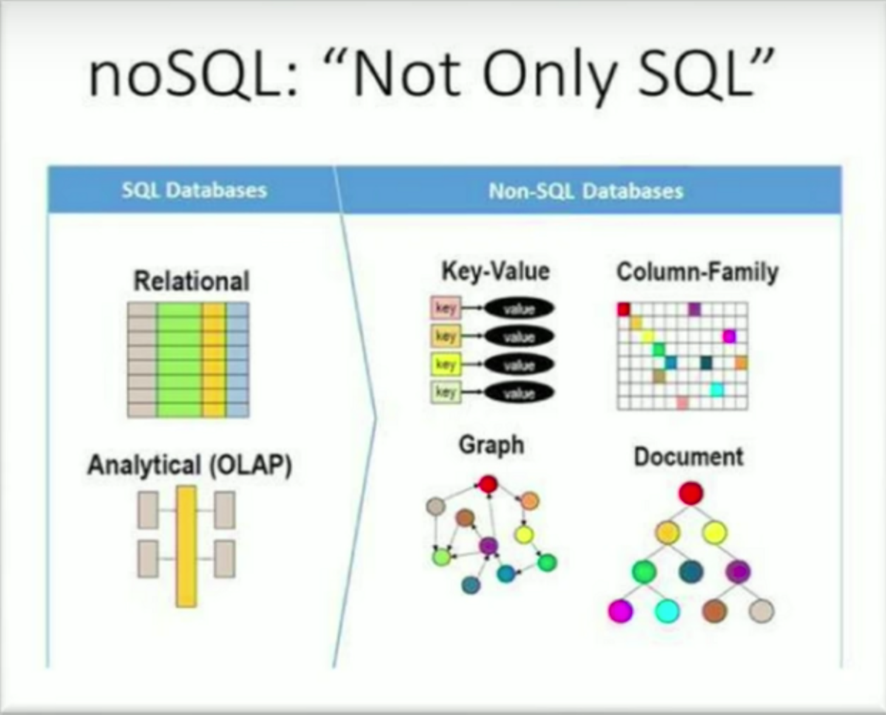


Рисунок 6.7 виды не реляционных баз данных

В качестве NoSQL базы данных была выбрана Mongo DB, за простоту изучения и быстрого освоения в этой технологии.

Если в Oracle Sql есть базы данных, таблицы, строки, то в MongoDb это базы данных, коллекции, и документы.

Требование: посмотреть на скорость выполнения долгих запросов в 2-х разных СУБД: Oracle и MongoDB

Для графического отображения использовался MongoDB Compass, официальное приложения от разработчиков Mongo.

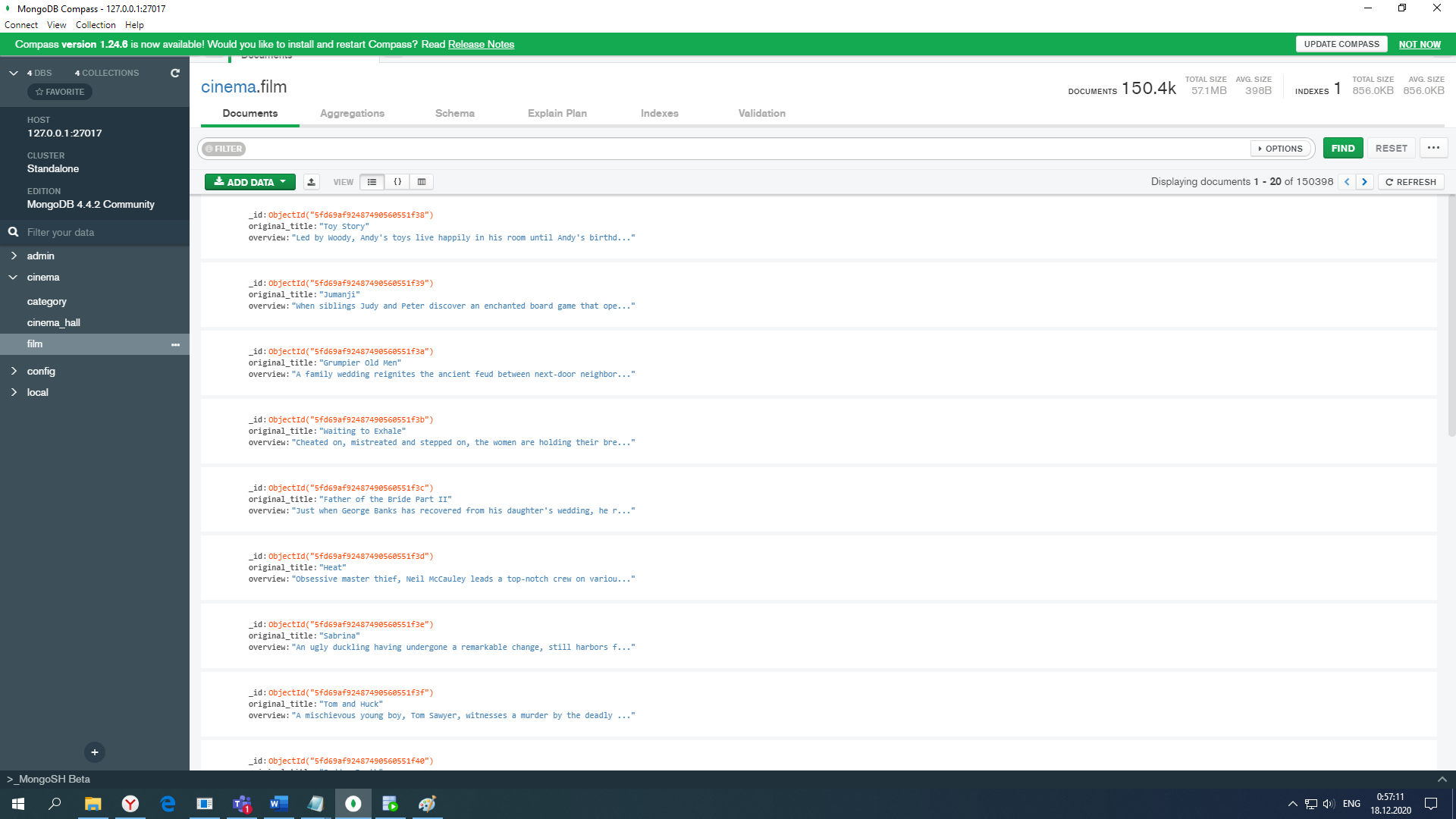


Рисунок 6.8 графический клиент Mongo

Создадим первую базу данных – cinema. Для этого в консольной строке Mongo введем:> use cinema, что означает- выбрать базу данных cinema, или если ее нет, создать и выбрать. База данных в Mongo не может существовать без коллекции.  
Создадим коллекцию коммандой db.film.save({title: “pirats”}), в данном примере мы создали объект в коллекции film с названием пираты, и так как такой коллекции у нас не было, Mongo создаст ее для нас сама. Аналогично создадим остальные нужные коллекции.

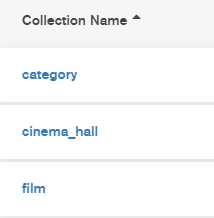


Рисунок 6.9 нужные коллекции

У нас сохранился XML файлы, с экспортом фильмов, залов и категорий.  
Воспользуемся онлайн сервисом: <https://jsonformatter.org/xml-to-json> для конвертации XML в JSON, понятный Mongo формат.

|  |  |
| --- | --- |
| было | стало |
| films>  <film>  <Id>1</Id>  <name>Toy Story</name>  <description>Led by Woody, Andy's toys live happily in his room until Andy's birthday brings Buzz Lightyear description>  </film>  <film>  <Id>2</Id>  <name>Jumanji</name>  <description>When siblings Judy and Peter discover an enchanted board game that opens the door to a creatures.</description>  </film>  <film> | {  "Id": "1",  "name": "Toy Story",  "description": "Led by Woody, Andy's toys live happily in his room until Andy's birthday brings."  },  {  "Id": "2",  "name": "Jumanji",  "description": "When siblings Judy and Peter discover an enchanted board game that opens the door to a magical world, they unwittingly invite Alan."  },  {  "Id": "3",  "name": "Grumpier Old Men",  "description": "A family wedding reignites the ancient feud between next-door neighbors and f."  }, |

В MongoDB Compass есть встроенная функция, которая заполняет коллекцию, исходя из данных JSON. Воспользуемся этой функцией и заполним все наши коллекции данными JSON, конвертированными из XML, и экспортированных из таблиц. Выведем количество получившихся записей



Рисунок 6.10 количество записей в коллекциях

6.3 Технология «Шифрование»

Шифрование представляет собой «маскировку» данных, или их преобразование, при котором данные не могут использоваться посторонними.

Oracle поддерживает шифрование сетевых данных посредством своей опции Advanced Security. Для обеспечения шифрования данных Oracle предлагает два пакета PL/SQL, один из которых, DBMS\_OBFUSCATION\_TOOLKIT, является более старым. Этот пакет позволяет шифровать данные по алгоритму DES. Для обеспечения наивысшего уровня безопасности этот пакет инструментов поддерживает тройное DES-шифрование. Кроме того, он поддерживает использование безопасного криптографического хеша MD5.

Для шифрования и дешифрации данных можно также применять более новый пакет шифрования PL/SQL по имени DBMS\_CRYPTO. По сравнению с DBMS\_OBFUSCATION\_TOOLKIT пакет DBMS\_CRYPTO предлагает более широкое множество шифровальных и криптографических алгоритмов для обеспечения повышенной безопасности и проще в использовании. Этот пакет предназначен для замены более старого пакета DBMS\_OBFUSCATION\_TOOLKIT. Но какой бы пакет не использовался, придется управлять ключами шифрования данных, что является нетривиальной задачей. Часто нужно создавать представления, которые облегчают дешифрацию и шифрование данных, что увеличивает количество задач управления. Кроме того, согласно рекомендациям Oracle, шифрованные данные нельзя индексировать, что в некоторых случаях снижает ценность обоих этих пакетов шифрования.

Существует также и третий, более простой подход: шифрование с помощью функции прозрачного шифрования данных. Далее будет показано, как зашифровать табличные данные Oracle с использованием Oracle Wallet (Бумажник Oracle) для хранения ключей шифрования. При этом можно также индексировать столбцы шифрованной таблицы, тем самым избавляясь от одного из наибольших недостатков применения пакетов шифрования Oracle.

Существует немало распространенных, пользующихся коммерческой поддержкой алгоритмов шифрования. Все алгоритмы, поддерживаемые Oracle для приложений PL/SQL, относятся к категории алгоритмов с закрытым ключом (иногда называемых симметричными алгоритмами).

В Oracle чаще всего используются следующие алгоритмы:

* DES (Data Encryption Standard). Традиционно алгоритм DES занимал ведущие позиции в области шифрования. Он был разработан более 20 лет назад для Национального бюро стандартов (позднее переименованного в Национальный институт стандартов и технологий), и с тех пор был принят в качестве стандарта ISO. Об алгоритме DES и его истории можно рассказать очень много, но моей задачей является не описание алгоритма, а краткое описание его применения. Алгоритму DES необходим 64-разрядный ключ, но 8 бит ключа не используются. Чтобы подобрать ключ, злоумышленнику придется перебрать до 72 057 594 037 927 936 комбинаций. Возможностей DES было достаточно в течение нескольких десятилетий, но сейчас он постепенно уходит в прошлое. Современные мощные компьютеры способны перебрать даже огромное число комбинаций, необходимое для взлома ключа DES.
* DES3. В этой схеме, базирующейся на исходном алгоритме DES, данные шифруются дважды или трижды (в зависимости от режима вызова). DES3 использует 128- или 192-разрядный ключ; его длина определяется количеством проходов. Надежность алгоритма DES3 тоже была приемлемой в течение некоторого времени, но сейчас и этот алгоритм постепенно устаревает и не обеспечивает защиты от целенаправленных атак.
* AES. В ноябре 2001 года был одобрен новый стандарт AES (Advanced Encryption Standard), вступивший в силу в мае 2002 года. Полный текст стандарта можно найти по этому адресу.

Пакет DBMS\_CRYPTO появился в Oracle10g. В более ранних версиях пакет DBMS\_OBFUSCATION\_TOOLKIT предоставлял похожую (но не идентичную) функциональность. Старый пакет все еще остается доступным, но сейчас он считается устаревшим, и вместо него рекомендуется использовать новый пакет.

Для выполнения шифрования кроме входных данных необходимы еще четыре компонента:

* ключ шифрования;
* алгоритм шифрования;
* метод заполнения;
* метод сцепления.

Ключ шифрования предоставляете вы, а остальные компоненты предоставляет Oracle. Выбор осуществляется при помощи соответствующих констант пакета DBMS\_CRYPTO [4].

6.4 Применение технологии «Шифрование»

При реализации технологии в курсовом проекте было разработано 2 процедуры: encryptPassword и encryptPassport. Рассмотрим применение технологии на примере процедуры encryptPassword.

Для реализации технологии применяется функция DBMS\_CRYPTO.ENCRYPT. Функция получает четыре аргумента:

* src – исходные данные, подлежащие шифрованию должны иметь тип данных raw.
* key – ключ шифрования, тип данных raw. Длина ключа должна соответствовать вы­бранному алгоритму. Например, для алгоритма DES она должна быть не менее 64 бит.
* typ – определение трех компонентов алгоритм, механизм заполнения и метод сце­пления в виде суммы соответствующих констант.
* iv – необязательный вектор инициализации, еще один компонент схемы шифрования, затрудняющий анализ «закономерностей» в зашифрованном тексте.

В курсовой работе будут использоваться:

* алгоритм – AES c 128-разрядным ключом;
* метод сцепления – CBC;
* механизм заполнения – PKCS#5.

На рисунке 6.3 продемонстрирована процедура шифрования паролей от пользовательских аккаунтов.

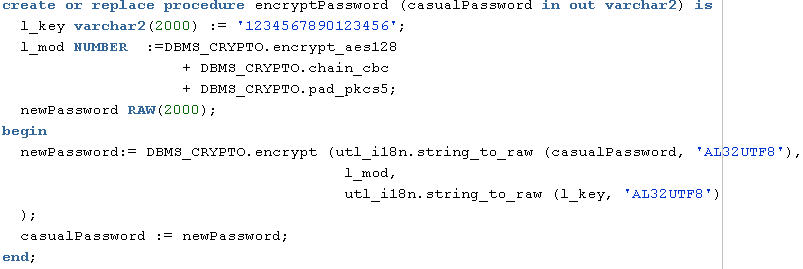


Рисунок 6.3 – Процедура шифрования пароля

Данная процедура принимает пароль в качестве единственного параметра и возвращает зашифрованный (рисунок 6.4).



Рисунок 6.3 – Зашифрованный пароль

Заключение

В ходе выполнения курсового проекта была достигнута поставленная цель по созданию базы данных для программного обеспечения «Кинотеатр». Были созданы необходимые объекты: 8 таблиц, 25 процедур, 3 представления, 2 директории, 1 роль, 2 пользователя.

При разработке проекта выполнены следующие пункты:

* возможность регистрации и авторизации пользователя;
* возможность добавления, удаления и изменения фильмов;
* возможность добавления, удаления и изменения сеансов к фильмам;
* возможность добавления, удаления и покупка билетов к сеансам;
* возможность осуществлять заказ билетов.

Были изучены технологии «Мультимедийные типы данных Oracle», «NoSql» и «Шифрование». В ходе реализации технологии рассмотрены системные пакеты DBMS\_HM и DBMS\_CRYPTO.

В результате выполнения курсового проекта были получены навыки по взаимодействию с документами в формате xml, импорту и экспорту данных формата xml.

Простота разработанной модели базы данных и необходимый набор процедур делают проект интуитивным для разработчиков и пользователей.

По результатам выполнения программы можно сделать вывод о том, что разработанная программа работает должным образом и требования технического задания полностью выполнены.

# **Список используемых источников**

1. Базы Данных. Учебное пособие [Электронный ресурс] /Базы данных – Режим доступа – URL: <https://siblec.ru/informatika-i-vychislitelnaya-tekhnika/bazy-dannykh> – Дата доступа: 25.11.2020
2. Блинова Е.А. Курс лекций по базам данных/ Е.А. Блинова
3. Patches IT Community [Электронный ресурс]: Дата доступа: 25.11.2020. Режим доступа – https://oracle-patches.com db/sql/3800-shifrovanie-i-deshifrovanie-v-pl-sql-dlya-bd-oracle-na-primere
4. Patches IT Community [Электронный ресурс]: Дата доступа: 25.11.2020. Режим доступа – https://oracle-patches.com/ db/sql/3611-процедуры-pl-sql-программирование-на-примерах
5. Patches IT Community [Электронный ресурс]: Дата доступа: 25.11.2020. Режим доступа – https://oracle-patches.com/ oracle/prof/3043-пользователи-oracle-управление-и-безопасность-базы-данных
6. Студопедия [Электронный ресурс]. Дата доступа: 25.11.2020. Режим доступа – https://studopedia.su/16\_42650\_polzovateli-baz-dannih-administrator-bazi-dannih-ego-funktsii.html
7. Patches IT Community [Электронный ресурс]: Дата доступа: 25.11.2020. Режим доступа – https://oracle-patches.com/db/sql/3645-что-такое-триггеры-pl-sql-код,-выполняемый-по-событию
8. Oracle PL/SQL учебник [Электронный ресурс]: Дата доступа: 25.11.2020. Режим доступа – https://oracleplsql.ru/contents-oracle-plsql.html
9. Patches IT Community [Электронный ресурс]: Дата доступа: 25.11.2020. Режим доступа – https://oracle-patches.com/ oracle/begin/3016-представления-oracle-database
10. Patches IT Community [Электронный ресурс]: Дата доступа: 25.11.2020. Режим доступа – https://oracle-patches.com/ oracle/prof/3126-oracle-xml-db

Приложение A

CREATE TABLE CINEMA\_HALL(

id INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

title NVARCHAR2(255) UNIQUE NOT NULL,

capacity INT NOT NULL,

constraint CINEMA\_HALL\_PK PRIMARY KEY (id));

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CREATE TABLE CATEGORY (

id INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

title NVARCHAR2(255) UNIQUE NOT NULL,

constraint CATEGORY\_PK PRIMARY KEY (id));

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/

CREATE TABLE FILM (

id INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

name NVARCHAR2(255) NOT NULL,

description NVARCHAR2(1000) NOT NULL,

constraint FILM\_PK PRIMARY KEY (id));

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/

CREATE TABLE "SESSION" (

id INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

film\_id INT NOT NULL,

cinema\_hall\_id INT NOT NULL,

added\_coast INT DEFAULT 0 NOT NULL ,

"date" DATE NOT NULL,

constraint SESSION\_PK PRIMARY KEY (id));

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/

CREATE TABLE FILMS\_CATEGORIES(

film\_id INT NOT NULL,

category\_id INT NOT NULL);

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/

CREATE TABLE PLACE(

id INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

cinema\_hall\_id INT NOT NULL,

coast INT NOT NULL,

constraint PLACE\_PK PRIMARY KEY (id));

/

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CREATE TABLE TICKET(

id INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

session\_id INTEGER NOT NULL,

place\_id INTEGER NOT NULL,

total\_coast INTEGER NOT NULL,

user\_id INTEGER NOT NULL,

constraint TICKET\_PK PRIMARY KEY (id));

/

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

create table "USER"(

id INTEGER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),

username NVARCHAR2(255) not null unique,

password NVARCHAR2(255) not null,

constraint user\_pk primary key(id));

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALTER TABLE "SESSION" ADD CONSTRAINT "session\_fk0" FOREIGN KEY (film\_id) REFERENCES FILM(id)on delete cascade;

ALTER TABLE "SESSION" ADD CONSTRAINT "session\_fk1" FOREIGN KEY (cinema\_hall\_id) REFERENCES CINEMA\_HALL(id)on delete cascade;

ALTER TABLE FILMS\_CATEGORIES ADD CONSTRAINT "films\_categories\_fk0" FOREIGN KEY (film\_id) REFERENCES FILM(id)on delete cascade;

ALTER TABLE FILMS\_CATEGORIES ADD CONSTRAINT "films\_categories\_fk1" FOREIGN KEY (category\_id) REFERENCES CATEGORY(id)on delete cascade;

ALTER TABLE PLACE ADD CONSTRAINT "place\_fk0" FOREIGN KEY (cinema\_hall\_id) REFERENCES CINEMA\_HALL(id)on delete cascade;

ALTER TABLE TICKET ADD CONSTRAINT "ticket\_fk0" FOREIGN KEY (session\_id) REFERENCES "SESSION"(id)on delete cascade;

ALTER TABLE TICKET ADD CONSTRAINT "ticket\_fk1" FOREIGN KEY (place\_id) REFERENCES PLACE(id)on delete cascade;

ALTER TABLE TICKET ADD CONSTRAINT "ticket\_fk2" FOREIGN KEY (user\_id) REFERENCES "USER"(id)on delete cascade;

Приложение Б

--Check if user exists

create or replace procedure CheckUser (inLogin in varchar2, inPassword in varchar2)

is

cuser number;

cursor c1 is select u.id from Users u where u.login = inLogin and u.password = inPassword;

begin

open c1;

fetch c1 into cuser;

if c1%notfound then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

close c1;

end;

--Registration

create or replace procedure RegistrationUser (inLogin in nvarchar2, inPassword in varchar2)

is

cuser number;

cursor c1 is select u.id from Users u where u.Login = inLogin and u.password = inPassword;

newPassword varchar2(2000);

begin

newPassword := inPassword;

encryptPassword(newPassword);

open c1;

fetch c1 into cuser;

if c1%found then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

insert into Users (login, password) values (inLogin, newPassword);

commit;

close c1;

end;

--Encrypt password

create or replace procedure encryptPassword (casualPassword in out varchar2) is

l\_key varchar2(2000) := '1234567890123456';

l\_mod NUMBER :=DBMS\_CRYPTO.encrypt\_aes128

+ DBMS\_CRYPTO.chain\_cbc

+ DBMS\_CRYPTO.pad\_pkcs5;

newPassword RAW(2000);

begin

newPassword:= DBMS\_CRYPTO.encrypt (utl\_i18n.string\_to\_raw (casualPassword, 'AL32UTF8'),

l\_mod,

utl\_i18n.string\_to\_raw (l\_key, 'AL32UTF8')

);

casualPassword := newPassword;

end;

--Authorization

create or replace procedure AuthorizationUser (inLogin in varchar2, inPassword in varchar2)

is

dpassword varchar2 (2000);

begin

dpassword := inPassword;

encryptPassword(dpassword);

CheckUser(inLogin, dpassword);

end;

--Delete

create or replace procedure DeleteUser (inLogin in varchar2, inPassword in varchar2)

is

dpassword varchar2 (2000);

begin

dpassword := inPassword;

encryptPassword(dpassword);

CheckUser(inLogin, dPassword);

delete from Users u where u.login = inLogin and u.password = dpassword;

commit;

end;

--Update login

create or replace procedure UpdateLogin(inLogin in varchar2, inPassword in varchar2)

is

dpassword varchar2 (2000);

begin

dpassword := inPassword;

encryptPassword(dpassword);

update Users u set u.login = inLogin where u.Password = dPassword;

commit;

end;

--check trip

create or replace procedure CheckTrip (inDateDeparture in date, inTrainId in int)

is

ctrip number;

cursor c1 is select t.id from Trips t where t.DateDeparture = inDateDeparture and t.TrainId = inTrainId;

begin

open c1;

fetch c1 into ctrip;

if c1%notfound then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

close c1;

end;

--Add Trip

create or replace procedure AddTrip (inDateDeparture in DATE,

inDateArrival in DATE,

inPlaceDeparture in NVARCHAR2,

inPlaceArrival in NVARCHAR2,

inCost in FLOAT,

inNumberSeats in INT,

inTrainId in INT)

is

ctrip number;

cursor c1 is select t.id from Trips t where t.DateDeparture = inDateDeparture and t.DateArrival = inDateArrival

and t.PlaceDeparture = inPlaceDeparture and t.PlaceArrival = inPlaceArrival and t.Cost = inCost and t.NumberSeats = inNumberSeats and t.TrainId = inTrainId;

begin

open c1;

fetch c1 into ctrip;

if c1%found then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

insert into Trips (DateDeparture, DateArrival, PlaceDeparture, PlaceArrival, Cost, NumberSeats, TrainId) values (inDateDeparture,

inDateArrival,

inPlaceDeparture,

inPlaceArrival,

inCost,

inNumberSeats,

inTrainId);

commit;

close c1;

end;

--delete trip

create or replace procedure DeleteTrip (inPlaceDeparture in nvarchar2, inId in number)

is

begin

delete from Trips t where t.PlaceDeparture = inPlaceDeparture and t.Id = inId;

commit;

end;

--update DateDeparture

create or replace procedure UpdateDateDeparture(inDateDeparture in date, inId in number)

is

begin

update Trips t set t.DateDeparture = inDateDeparture where t.id = inId;

commit;

end;

--update DateArrival

create or replace procedure UpdateDateArrival(inDateArrival in date, inId in number)

is

begin

update Trips t set t.DateArrival = inDateArrival where t.id = inId;

commit;

end;

--update PlaceDeparture

create or replace procedure UpdatePlaceDeparture(inPlaceDeparture in nvarchar2, inId in number)

is

begin

update Trips t set t.PlaceDeparture = inPlaceDeparture where t.id = inId;

commit;

end;

--update PlaceArrival

create or replace procedure UpdatePlaceArrival(inPlaceArrival in nvarchar2, inId in number)

is

begin

update Trips t set t.PlaceArrival = inPlaceArrival where t.id = inId;

commit;

end;

--update Cost

create or replace procedure UpdateCost (inCost in float, inId in number)

is

begin

update Trips t set t.Cost = inCost where t.id = inId;

commit;

end;

--update NumberSeats

create or replace procedure UpdateNumberSeats(inNumberSeats in int, inId in number)

is

begin

update Trips t set t.NumberSeats = inNumberSeats where t.id = inId;

commit;

end;

--Remove NumberSeats

create or replace procedure RemoveNumberSeats (inTripId in int)

is

begin

update Trips t set t.NumberSeats = t.NumberSeats - 1 where t.Id = inTripId and t.NumberSeats > 0;

commit;

end;

--Check train

create or replace procedure CheckTrain (inType in nvarchar2, inNumberTrain in nvarchar2)

is

ctrain number;

cursor c1 is select t.id from Trains t where t.Type = inType and t.NumberTrain = inNumberTrain;

begin

open c1;

fetch c1 into ctrain;

if c1%notfound then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

close c1;

end;

--Add train

create or replace procedure AddTrain (inType in nvarchar2, inNumberTrain in nvarchar2)

is

ctrain number;

cursor c1 is select t.id from Trains t where t.Type = inType and t.NumberTrain = inNumberTrain;

begin

open c1;

fetch c1 into ctrain;

if c1%found then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

insert into Trains (type, numbertrain) values (inType, innumbertrain);

commit;

close c1;

end;

--delete train

create or replace procedure DeleteTrain (inType in nvarchar2, inNumberTrain in nvarchar2)

is

begin

CheckTrain(inType, inNumberTrain);

delete from Trains t where t.Type = inType and t.NumberTrain = inNumberTrain;

commit;

end;

--update train type

create or replace procedure UpdateType(inType in nvarchar2, inId in number)

is

begin

update Trains t set t.type = inType where t.id = inId;

commit;

end;

--update number of train

create or replace procedure UpdateNumber (inNumberTrain in nvarchar2, inId in number)

is

begin

update Trains t set t.numbertrain = inNumberTrain where t.id = inId;

commit;

end;

--Check ticket

create or replace procedure CheckTicket (inPassport in nvarchar2, inTripId in int)

is

cticket number;

cursor c1 is select t.id from Tickets t where t.Passport = inPassport and t.TripId = inTripId;

begin

open c1;

fetch c1 into cticket;

if c1%notfound then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

close c1;

end;

--Encrypt Passport

create or replace procedure encryptPassport (casualPassport in out varchar2) is

l\_key varchar2(2000) := '1234567890123456';

l\_mod NUMBER :=DBMS\_CRYPTO.encrypt\_aes128

+ DBMS\_CRYPTO.chain\_cbc

+ DBMS\_CRYPTO.pad\_pkcs5;

newPassport RAW(2000);

begin

newPassport:= DBMS\_CRYPTO.encrypt (utl\_i18n.string\_to\_raw (casualPassport, 'AL32UTF8'),

l\_mod,

utl\_i18n.string\_to\_raw (l\_key, 'AL32UTF8')

);

casualPassport := newPassport;

end;

--Add ticket

create or replace procedure AddTicket ( inName in NVARCHAR2, inPassport in VARCHAR2, inTripId in INT)

is

cticket number;

cursor c1 is select t.id from Tickets t where t.Name = inName and t.Passport = inPassport and t.Tripid = inTripId;

newPassport varchar2(2000);

begin

newPassport := inPassport;

encryptPassport(newPassport);

open c1;

fetch c1 into cticket;

if c1%found then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

insert into Tickets (name, passport, tripId) values (inName, newPassport, inTripId);

RemoveNumberSeats(inTripId);

commit;

close c1;

end;

--delete ticket

create or replace procedure DeleteTicket (inName in nvarchar2, inTripId in int)

is

begin

delete from Tickets t where t.Name = inName and t.TripId = inTripId;

commit;

end;

--update name

create or replace procedure UpdateName(inName in nvarchar2, inId in number)

is

begin

update Tickets t set t.name = inName where t.id = inId;

commit;

end;

--Check order

create or replace procedure CheckOrder (inDates in date, inTicketId in int)

is

corder number;

cursor c1 is select r.id from Orders r where r.Dates = inDates and r.TicketId = inTicketId;

begin

open c1;

fetch c1 into corder;

if c1%notfound then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

close c1;

end;

--Add order

create or replace procedure AddOrder ( inUserId in int, inTicketId in int)

is

corder number;

cursor c1 is select r.id from Orders r where r.UserId = inUserId and r.TicketId = inTicketId;

begin

open c1;

fetch c1 into corder;

if c1%found then

raise\_application\_error(-20010, 'Error detected - '||SQLCODE||' ERROR '||SQLERRM);

end if;

insert into Orders (Dates, States, UserId, TicketId) values (sysdate, 'formed', inUserId, inTicketId);

commit;

close c1;

end;

--Delete order

create or replace procedure DeleteOrder (inUserId in int, inTicketId in int)

is

begin

delete from Orders r where r.UserId = inUserId and r.TicketId = inTicketId;

commit;

end;

--Cancel Order

create or replace procedure CancelOrder (inId in number)

is

begin

update Orders r set r.States = 'canceled' where r.id = inId;

commit;

end;

--Confirm order

create or replace procedure ConfirmOrder (inId in number)

is

begin

update Orders r set r.States = 'confirmed' where r.id = inId and r.States = 'formed';

commit;

end;

Приложение В

