МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Разработка базы данных для магазина по продаже шоколада с применением технологии мультимедийных типов данных в БД»

Выполнил студент Подобед Владислав Георгиевич

(Ф.И.О.)

Руководитель работы асс. Нистюк О.А.

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

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

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

Курсовая работа защищена с оценкой

Минск 2023

**Содержание**

[Введение 3](#_Toc154002562)

[1 Постановка задача 4](#_Toc154002563)

[1.1 Программное средство «Shokobox» 4](#_Toc154002564)

[1.2 Программное средство «Shokoladki» 5](#_Toc154002565)

[1.3 Определение цели и задачи 5](#_Toc154002566)

[2 Проектирование базы данных 7](#_Toc154002567)

[2.1 Вывод по проектированию модели базы данных 8](#_Toc154002568)

[3 Разработка объектов базы данных 9](#_Toc154002569)

[3.1 Разработка таблиц базы данных 9](#_Toc154002570)

[3.2 Последовательность 10](#_Toc154002571)

[3.3 Процедуры 11](#_Toc154002572)

[3.4 Пользователи 11](#_Toc154002573)

[3.5 Вывод по разработке объектов базы данных 12](#_Toc154002574)

[4 Описание процедур импорта и экспорта 13](#_Toc154002575)

[4.1 Описание процедуры экспорта данных 13](#_Toc154002576)

[4.2 Описание процедуры импорта данных 14](#_Toc154002577)

[5 Тестирование производительности 15](#_Toc154002578)

[6 Описание технологии и ее применения в базе данных 17](#_Toc154002579)

[7 Краткое описание приложения для демонстрации 18](#_Toc154002580)

[8 Руководство пользователя 19](#_Toc154002581)

[Заключение 20](#_Toc154002582)

[Список используемых источников 21](#_Toc154002583)

[ПРИЛОЖЕНИЕ А 22](#_Toc154002584)

[ПРИЛОЖЕНИЕ Б 23](#_Toc154002585)

[ПРИЛОЖЕНИЕ В 37](#_Toc154002586)

# **Введение**

В современном мире, где электронная коммерция продолжает расти и развиваться, разработка и управление эффективной базой данных становятся все более важными задачами для предприятий. В рамках данного курсового проекта предлагается рассмотреть тему разработки базы данных для магазина по продаже шоколада с применением технологии мультимедийных типов данных в БД.

Магазин по продаже шоколада — это ресурс, предлагающий широкий ассортимент шоколадных изделий различных брендов и вкусовых направлений. Важной особенностью такого магазина является наличие фотографий, видео и других мультимедийных материалов, которые помогают клиентам визуально оценить товары перед покупкой.

Целью данного курсового проекта является разработка базы данных, которая обеспечивает эффективное хранение, управление и просмотр мультимедийных материалов, связанных с шоколадными продуктами. Использование технологии мультимедийных типов данных позволяет эффективно работать с изображениями, видео и звуком в контексте базы данных.

Приложение будет обладать следующими функционалом:

* добавление товара в корзину;
* оформление заказа
* сортировка продукции
* поиск продукции;
* определение ролей (администратор, пользователь);
* управление товаром администратором (добавление, удаление);
* управление корзиной пользователем (добавление, удаление);

Разработка базы данных для магазина по продаже шоколада с применением технологии мультимедийных типов данных в БД предоставит возможность создать эффективную систему управления данными.

# **Постановка задача**

В наши дни множество аналогов программных средств можно найти на различных веб-ресурсах, связанных с данной тематикой. Перед тем как приступить к разработке требовалось провести анализ существующих аналогов.

1. **Программное средство «Shokobox»**

На рисунке 1.1 предствлен скриншот веб-приложения «Shokobox».



Рисунок 1.1 – Скриншот аналога веб-приложения «Shokobox»

Shokobox – это веб-приложение, которое предоставляет удобную и простую платформу для заказа и доставки конфет и шоколадных изделий. Это онлайн-сервис, который позволяет пользователям выбирать и приобретать разнообразные вкусности прямо из удобства своего дома или офиса.

Плюсы Shokobox:

- Огромный выбор конфет. Shokobox предлагает широкий ассортимент конфет и сладостей различных брендов и вкусов.

- Удобство покупки. Приобрести конфеты на Shokobox.ru очень просто.

Минусы Shokoladki:

- Нет изменения статуса заказа.

1. **Программное средство «Shokoladki»**

На рисунке 1.2 предствлен скриншот веб-приложения «Shokoladki».

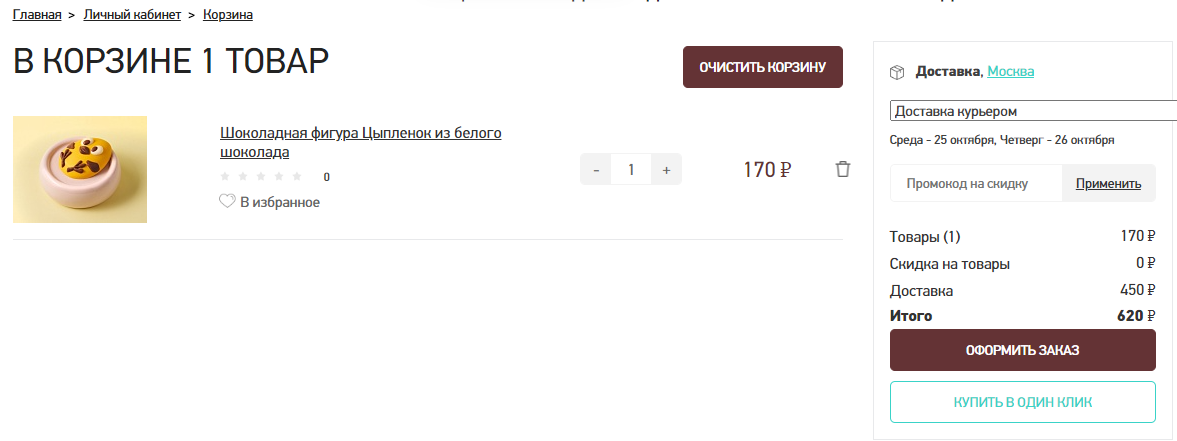


Рисунок 1.2 – Cкриншот аналога веб-приложения «Shokoladki»

Shokoladki – это веб-приложение, которое предоставляет удобную и простую платформу для заказа и доставки шоколадных конфет и сладостей. Это онлайн-сервис, который позволяет пользователям выбирать из широкого ассортимента шоколадных изделий и оформлять заказы прямо через интернет.

Плюсы Shokoladki:

- Удобство заказа: Shokoladki предоставляет удобную платформу для заказа шоколадных конфет и сладостей прямо через интернет, что позволяет сэкономить время и усилия.

- Широкий ассортимент: платформа предлагает разнообразные вкусы и виды шоколадных конфет, позволяя выбрать из множества вариантов и насладиться разнообразием вкусов.

Минусы Shokoladki:

- Возможность недостаточного выбора: возможно, некоторые пользователи могут не найти на платформе Shokoladki конкретные виды или марки шоколадных конфет, которые они ищут.

1. **Определение цели и задачи**

На текущий момент рынок инструментов для управления магазина по продаже шоколада представляет собой множество решений с различным уровнем функционала. Однако, многие из них не могут предложить полноценного комплекса инструментов, сочетающего в себе удобство и гибкость, что оставляет простор для новых разработок в этой сфере. Задача проекта: разработать архитектуру приложения, создать интерфейс, взаимодействие с которым будет понятно любому пользователю. Построить базу данных и выполнить тестирование готового продукта.

Должны быть выполнены следующие требования:

˗ база данных должна быть реализована в СУБД Oracle 19c.

˗ доступ к данным должен осуществляться только через соответствующие процедуры;

˗ должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON;

˗ необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице;

˗ применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.

# **Проектирование базы данных**

В самом начале разработки нужно определить из каких таблиц будет состоять база данных, какие связи будут между ними и какие типы данных будут представлять поля для заполнения таблиц.

Диаграмма таблиц и их отношения представлены на рисунке 2.1.

|  |
| --- |
|  |

Рисунок 2.1 – Схема базы данных

Далее необходимо создать пользователей с различными правами, чтобы распределить процедуры для взаимодействия с базой данных.

В моем проекте роли пользователей включают Admin и User. Администратор будет иметь полный доступ ко всем процедурам, в то время как клиент будет иметь ограниченный доступ только к определенным процедурам и данным, соответствующим его роли.

Use-case диаграмма взаимодействие пользователей с базой данных на рисунке 2.2.

|  |
| --- |
|  |

Рисунок 2.2 – Диаграмма доменной логики

По итогу, будет создано две группы пользователей – Администратор (пользователь, который выполняет управление системой), а также обычные пользователи. Все действия данных пользователей будут выполняться через хранимые процедуры.

## **Вывод по проектированию модели базы данных**

Такая архитектура базы данных предоставляет гибкость в управлении данными и высокую производительность.

# **Разработка объектов базы данных**

1. **Разработка таблиц базы данных**

При разработке приложения для курсового проекта была использована база данных Oracle.

В базе данных представлены 7 таблиц: USERS, PRODUCT, PURCHASE, MARK, PRODUCER, ORDERS, TASTE\_CHOCOLATE.

Таблица USERS хранит информацию о пользователях. Состоит из следующих столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы USERS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_User | Уникальный идентификатор пользователя, первичный ключ | INT |
| LOGIN | Логин пользователя | VARCHAR2(255) |
| PASSWORD | Пароль пользователя | VARCHAR2(255) |

Таблица PRODUCT хранит информацию о товаре. Состоит из следующих столбцов (таблица 3.2):

Таблица 3.2 – Столбцы таблицы PRODUCT

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_PRODUCT | Код товара | INT |
| TYPE\_OF\_CHOCOLATE | Вид шоколада | INT |
| ID\_TYPE\_OF\_CHOCOLATE | Код вкуса шоколада | INT |
| BRAND\_CODE | Код марки | INT |
| MANUFACTURERS\_CODE | Код производителя | INT |
| WEIGHT | Вес | VARCHAR(255) |
| PRICE | Цена | INT |
| FOTO | Фото | VARCHAR(255) |
| STRUCTURE | Состав | VARCHAR(255) |

Таблица PURCHASE хранит информацию о покупки товаров. Состоит из следующих столбцов (таблица 3.3):

Таблица 3.3 – Столбцы таблицы PURCHASE

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_BUY | Код покупки | INT |
| ID\_PRODUCT | Код товара | INT |
| ID\_USER | Код пользователя | INT |
| QUANITY | Количество товара | INT |
| SUM | Сумма | INT |
| DATE\_BUY | Дата покупки | INT |

Таблица MARK хранит информацию о марки товара. Состоит из следующих столбцов (таблица 3.4):

Таблица 3.4 – Столбцы таблицы MARK

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| BRAND\_CODE | Код марки | INT |
| BRAND\_NAME | Название марки | VARCHAR(255) |

Таблица PRODUCER хранит информацию о производителя товара. Состоит из следующих столбцов (таблица 3.5):

Таблица 3.5 – Столбцы таблицы PRODUCER

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| MANUFACTURERS\_CODE | Код производителя | INT |
| MANUFACTURERS\_NAME | Наименование производителя | VARCHAR(255) |

Таблица ORDERS хранит информацию о оформлении заказа. Состоит из следующих столбцов (таблица 3.6):

Таблица 3.6 – Столбцы таблицы ORDERS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_BUY | Код покупки | INT |
| ID\_PRODUCT | Код товара | INT |
| QUANITY | Количество товара. | INT |
| SUM | Сумма | INT |
| DATE\_BUY | Дата покупки | INT |

Таблица TASTE\_CHOCOLATE предназначена для реализации связи many-to-many для таблиц MOVIES и GENRES. Состоит из следующих столбцов (таблица 3.7):

Таблица 3.7 – Столбцы таблицы TASTE\_CHOCOLATE

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_TYPE\_OF\_CHOCOLATE | Код вкуса шоколада | INT |
| THE\_NAME\_THE\_TASTE\_OF\_CHOCOLATE | Название вкуса шоколада | VARCHAR(255) |

## **Последовательность**

Последовательность – это объект базы данных, подобный таблице или представлению, который представляет собой последовательность целых чисел, которая может использоваться любой таблицей или представлением в глобальном пространстве имен базы данных.

Последовательность используется для генерации уникальных идентификаторов имен базы данных.

При каждом использовании PRODUCT\_SEQ будут генерироваться последовательные значения, начиная с 1 и увеличиваясь на 1. Последовательность представлена в листинге 3.1.

|  |
| --- |
| CREATE SEQUENCE PRODUCT\_SEQ START WITH 1 INCREMENT BY 1; |

Листинг 3.1 – Последовательность PRODUCT\_SEQ

Для создания последовательности, которая может генерировать уникальные числа.

## **Процедуры**

Процедуры, представленные ниже, обеспечивают базовые операции управления данными в базе данных, такие как вставка, обновление и удаление. Эти процедуры охватывают ключевые аспекты системы, включая управление пользователями, шоколадам и связанным с ними содержимым, а именно вкус, марка.

Листинг PL/SQL-кода для создания процедур находится в приложении Б.

Процедуры, разработанные в рамках курсового проекта:

* AddProduct – добавление нового шоколада;
* UpdateProductPriceAndFoto – позволяет обновить данные существующего шоколада;
* DeleteProduct – позволяет удалить из базы данных существующий шоколад;
* AddMark – позволяет добавлять новые марки;
* UpdateMarkName – позволяет обновлять существующие марки;
* DeleteMark – позволяет удалять существующие марки;
* AddProducer – позволяет добавлять нового производителя;
* DeleteProducer – позволяет удалять существующего производителя;
* UpdateProducer – позволяет обновлять существующего производителя;
* AddUser – позволяет добавить пользователя;
* DeleteUser – позволяет удалить пользователя;
* SortAndDisplayProducts – позволяет сортировать товар по PRICE и по WEIGHT.
* SearchProduct – позволяет осуществить поиск продукта.
* MovePurchaseToOrders – оформление заказа.
* UpdatePassword – позволяет обновить пользователя;
* AddChocolateTaste – позволяет добавлять новый вкус шоколада;
* DeleteChocolateTaste – позволяет удалять существующего вкус шоколада;
* UpdateChocolateTaste – позволяет обновлять существующего вкус шоколада;

## **Пользователи**

Пользователь базы данных — это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

При проектировании базы данных было создано 2 типа пользователей: клиент, осуществляющий доступные ему операции и администратор имеющий полный доступ ко всем объектам базы данных.

Создание пользователей представлено в листинге 3.3.

|  |
| --- |
| create user User1 IDENTIFIED by 1234;  create user Admin1 IDENTIFIED by 1234; |

Листинг 3.2 – Создание пользователей «Пользователь» и «Администратор» для базы данных

Далее для созданного пользователя базы данных были выданы необходимые привилегии необходимые для функционала клиента.

## **Вывод по разработке объектов базы данных**

Важным моментом создания хорошо спроектированных объектов базы данных является определение связей между различными таблицами или данными. Связи позволяют связывать информацию из разных таблиц, обеспечивая связность и целостность данных.

Один из способов установления связей - использование внешних ключей. Они позволяют ссылаться на данные в другой таблице, обеспечивая целостность ссылочных цепочек между объектами.

Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря хорошо спроектированным объектам базы данных система способна обеспечивать надежность, своевременное быстродействие и целостность данных.

Не следует забывать о документации базы данных. Подробное описание объектов, их связей и правил использования помогает разработчикам, администраторам и другим заинтересованным лицам более эффективно работать с базой данных, обращаться к ней и понимать ее структуру и логику.

Хорошо спроектированные объекты базы данных не ограничиваются только этими аспектами, но важно учитывать их при разработке и настройке базы данных, чтобы обеспечить ее эффективность, надежность и легкость использования.

# **Описание процедур импорта и экспорта**

В данной курсовой работе реализованы процедуры экспорта и импорта данных из JSON файла в базу данных таблицы Users и наоборот. При данных операциях работает с файловой системой приложение, а разбором и генерацией JSON занимается SQL Developer.

1. **Описание процедуры экспорта данных**

В представленном блоке кода описаны две основные процедуры: EXPORT\_JSON и IMPORT\_JSON. Эти процедуры позволяют экспортировать и импортировать данные из таблицы MARK в формат JSON.

Код процедура экспорта данных в формат JSON представлена в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE EXPORT\_JSON IS  v\_file UTL\_FILE.FILE\_TYPE;  v\_cursor SYS\_REFCURSOR;  v\_row MARK%ROWTYPE;  v\_json CLOB;  BEGIN  v\_file := UTL\_FILE.FOPEN('JSONDIR', 'JSON\_DATA.json', 'W');  OPEN v\_cursor FOR SELECT \* FROM MARK;  v\_json := '[';  LOOP  FETCH v\_cursor INTO v\_row;  EXIT WHEN v\_cursor%NOTFOUND;  IF v\_json != '[' THEN  v\_json := v\_json || ',';  END IF;  v\_json := v\_json || '{';  v\_json := v\_json || '"BRAND\_CODE":' || NVL(TO\_CHAR(v\_row.BRAND\_CODE),'NULL') || ',';  v\_json := v\_json || '"BRAND\_NAME":' || NVL(TO\_CHAR(v\_row.BRAND\_NAME), 'NULL') || '}';  END LOOP;  CLOSE v\_cursor;  v\_json := v\_json || ']';  UTL\_FILE.PUT\_LINE(v\_file, v\_json);  UTL\_FILE.FCLOSE(v\_file);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);  RAISE;  END; |

Листинг 4.1 – Процедура экспорта данных из таблицы MARK в формат JSON

Процедура EXPORT\_JSON предназначена для экспорта данных из таблицы Ratings в формат JSON и сохранения их в файл 'JSON\_DATA.json'. Процедура принимает один параметр p, который представляет собой имя файла, в который будут экспортированы данные.

Процедура начинается с попытки открыть файл для записи. Если файл существует, выбрасывается ошибка. Затем данные из таблицы USERS выбираются и преобразуются в формат JSON. Каждая строка данных добавляется в файл с учетом правильного форматирования JSON. После завершения экспорта файла, он закрывается. Процедура завершается вызовом export\_to\_json, что инициирует процесс экспорта.

## **Описание процедуры импорта данных**

Процедура IMPORT\_JSON не принимает явных параметров. Её цель - вставить данные из JSON-файла ('JSON\_DATA.json') в таблицу Ratings. При попытке открыть файл для записи, если файл уже существует, процедура генерирует ошибку. В случае возникновения ошибки, она выводит сообщение об ошибке в консоль и повторно возбуждает исключение. Код процедуры импорта представлена в листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE IMPORT\_JSON  IS  BEGIN  INSERT INTO MARK (BRAND\_CODE, BRAND\_NAME)  SELECT BRAND\_CODE, BRAND\_NAME  FROM JSON\_TABLE(BFILENAME('JSONDIR', 'JSON\_DATA.json'), '$[\*]' COLUMNS (  BRAND\_CODE NUMBER PATH '$.BRAND\_CODE',  BRAND\_NAME NUMBER PATH '$.BRAND\_NAME'  )  );  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);  RAISE;  END; |

Листинг 4.2 – Процедура импорта из JSON-файла в таблицу MARK

Обе процедуры содержат механизмы обработки исключений для обработки ошибок, связанных с работой с файлами и другими возможными ошибками.

# **Тестирование производительности**

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для данной задачи была разработана процедура, которая за одно выполнение позволяет вставить в таблицу 100 000 строк. (рисунок 5.1).

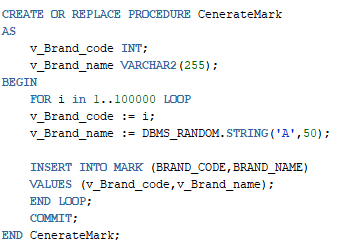


Рисунок 5.1 –Процедура для заполнения таблицы большим количеством данных;

Ниже на рисунке 5.2 представлен результат статистического анализа и компиляции SQL Developer до изменения таблицы



Рисунок 5.2 – Статический анализ перед изменением таблицы

Как можно увидеть, затраченное время достаточно велико. Для оптимизации поиска была изменена таблицы путем добавления двух индексов, код которых представлен на рисунке 5.3.



Рисунок 5.3 – Создание двух индексов к таблице Comments

Ниже на рисунке 5.4 представлен результат статистического анализа и компиляции SQL Developer после создания двух индексов.



Рисунок 5.4 – Статический анализ после изменение таблицы

На основе использованных индексов можно сделать вывод о сокращении времени поиска данных.

Проблемы выявления узких мест производительности баз данных напрямую связаны с метриками, методами измерений производительности и технологией.

Для того чтобы правильно организовать процесс тестирования БД, тестировщики должны обладать хорошими знаниями SQL и DML и иметь ясное представление о внутренней структуре БД. Это самый лучший и надежный способ тестирования БД особенно для приложений с низким и средним уровнем сложности. Данный метод не только дает уверенность, что тестирование выполнено качественно, но также повышает мастерство написания SQL-запросов.

В итоге, использование данных индексов позволяет значительно сократить время поиска данных, улучшить производительность запросов и эффективность использования ресурсов базы данных. Это приводит к более отзывчивой системе и повышению общей производительности.

# **Описание технологии и ее применения в базе данных**

Технология Oracle Multimedia в Oracle Database обеспечивает эффективную обработку и анализ мультимедийных типов данных, повышая функциональность системы. Процессорные ядра выполняют многозадачное сканирование мультимедийных столбцов с использованием векторных SIMD-инструкций, обеспечивая высокую скорость обработки. Эффективные операции объединения данных из различных мультимедийных таблиц улучшают производительность запросов. Технология также поддерживает мультимедийные агрегационные операции, оптимизированные для анализа медийных данных, ускоряя формирование отчетов и аналитики. В итоге, Oracle Multimedia предоставляет средства для эффективной обработки и взаимодействия с мультимедийными данными в базе данных Oracle. Чтобы не загромождать оперативную память, используется сжатие столбцов в памяти.

Применение технологии мультимедийных типов данных можно увидеть на рисунке 6.1.

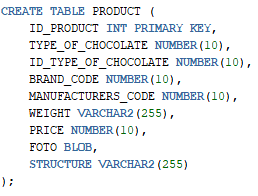


Рисунок 6.1 – Применение технологии мультимедийных типов данных

Из рисунка 6.1 видно, что в таблице есть атрибут FOTO, который имеет тип данных BLOB. BLOB (Binary Large Object) в Oracle представляет собой тип данных для хранения бинарных объектов, таких как изображения. Этот двоичный тип данных используется для работы с крупными объемами информации, и он отличается от обычных строковых данных. Важно отметить, что BLOB предназначен для хранения неструктурированных двоичных данных и не обладает семантикой сортировки или поиска. Размер объекта BLOB ограничен доступным объемом памяти в системе, и его использование включает манипуляции через пакет DBMS\_LOB. BLOB-колонки могут быть включены в определение таблицы для хранения бинарных данных, и операции вставки и извлечения данных выполняются с использованием специальных функций. Тип данных BLOB находит применение в программном обеспечении для хранения и обработки мультимедийных файлов, предоставляя гибкость при работе с большими объемами бинарных данных в контексте мультимедийных контентов в базе данных Oracle.

# **Краткое описание приложения для демонстрации**

Демонстрация работоспособности базы данных будет выполняться с помощью интегрированной среды разработки SQL Developer. В ходе работы активно использовался SQL Developer для написания и отладки SQL-запросов, а также для мониторинга и оптимизации производительности базы данных.

SQL Developer предоставил удобное окружение для взаимодействия с Oracle Database, что существенно упростило процесс разработки и обеспечило эффективную работу с данными.

В SQL Developer имеется мощный редактор, позволяющий создавать и редактировать SQL-запросы. Редактор предоставляет подсветку синтаксиса, автодополнение и другие функции для упрощения процесса написания SQL-кода.

SQL Developer включает инструменты для анализа и оптимизации SQL-запросов. Он позволяет просматривать планы выполнения запросов, оценивать их производительность и предлагать улучшения для оптимизации.

# **Руководство пользователя**

В приложении присутствует два типа пользователей: Администратор и Пользователь. В зависимости от роли пользователя ему будут предоставлен доступ к соответствующим процедурам. Администраторы имеет доступ абсолютно ко всем процедурам, в то время как обычный пользователь имеет ограниченный набор процедур. Список процедур доступных обычному пользователю представлен на рисунке 8.1

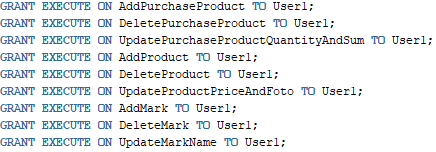


Рисунок 8.1 – Список процедурных к которым имеет доступ обычный пользователь

Из рисунка 8.1 видно, что пользователь может получить полный список шоколада, добавить товар в корзину, оформить заказ.

Заключение

В результате выполнения данного курсового проекта были реализованы ключевые элементы базы данных, включая таблицы для хранения информации о пользователях, фильмах, трейлерах, рейтингах, комментариях и жанрах. Индексы были применены для оптимизации производительности запросов, а процедуры обеспечивают управление данными, соблюдая принципы безопасности.

Также были созданы пользователи базы данных с различными уровнями доступа – клиент и администратор, что обеспечивает отделение прав доступа и повышает безопасность системы.

Разработанные процедуры обеспечивают базовые операции управления данными, а также функционал поиска, сортировки и фильтрации, что сделает взаимодействие с системой интуитивно понятным и удобным для пользователей.

Благодаря внедрению индексов, процедур, и оптимизированным структурам таблиц, система способна эффективно обрабатывать запросы, в том числе на больших объемах данных, что существенно повышает производительность системы.

В целом, разработанная база данных представляет собой мощный инструмент для управления онлайн-кинотеатром, сочетающий в себе гибкость, производительность и безопасность. Разработанная база данных также включает использование мультимедийных типов данных для хранения и управления информацией о трейлерах и изображениях фильмов. Это обеспечивает возможность хранения медиаконтента в формате BLOB (Binary Large Object), что является важным аспектом для кинотеатров, поскольку они часто работают с изображениями постеров фильмов и видео трейлерами.

.

Список используемых источников

1. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа ­– URL: https://learn.microsoft.com/ru-ru/ – Дата доступа: 14.10.2023

URL: http://msdn.microsoft.com/library/rus/ – Дата доступа: 20.10.2023

2. Примеры массового импорта и экспорта XML-документов (SQL Developer) [Электронный ресурс] / Режим доступа – URL: https://stackoverflow.com/questions/9672780/importing-xml-data-from-xml-file-to-sql-database ­ – Дата доступа: 30.10.2023

3. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа – URL: https://metanit.com – Дата доступа: 04.11.2023

4. Нистюк О. А. Курс лекций по базам данных, [Электронный ресурс]. – Режим доступа: https://diskstation.belstu.by:5001/. – Дата доступа: 01.09.2023.

ПРИЛОЖЕНИЕ А

Создание базы данных

CREATE TABLE PURCHASE (

ID\_BUY INT PRIMARY KEY,

ID\_PRODUCT INT,

QUANTITY INT,

SUM INT,

DATE\_BUY DATE,

FOREIGN KEY (ID\_PRODUCT) REFERENCES PRODUCT(ID\_PRODUCT) ON DELETE CASCADE

);

CREATE TABLE PRODUCT (

ID\_PRODUCT INT PRIMARY KEY,

TYPE\_OF\_CHOCOLATE NUMBER(10),

ID\_TYPE\_OF\_CHOCOLATE NUMBER(10),

BRAND\_CODE NUMBER(10),

MANUFACTURERS\_CODE NUMBER(10),

WEIGHT VARCHAR2(255),

PRICE NUMBER(10),

FOTO BLOB,

STRUCTURE VARCHAR2(255)

);

CREATE TABLE Users (

ID\_User INT PRIMARY KEY,

LOGIN VARCHAR2(255),

PASSWORD VARCHAR2(255)

);

CREATE TABLE MARK (

BRAND\_CODE INT,

BRAND\_NAME VARCHAR2(255),

PRIMARY KEY (BRAND\_CODE)

);

CREATE TABLE PRODUCER (

MANUFACTURERS\_CODE INT,

MANUFACTURERS\_NAME VARCHAR2(255),

PRIMARY KEY (MANUFACTURERS\_CODE)

);

CREATE TABLE TASTE\_CHOCOLATE (

ID\_TYPE\_OF\_CHOCOLATE INT,

THE\_NAME\_THE\_TASTE\_OF\_CHOCOLATE VARCHAR2(255),

PRIMARY KEY (ID\_TYPE\_OF\_CHOCOLATE)

);

ПРИЛОЖЕНИЕ Б

Создание процедур

CREATE OR REPLACE PROCEDURE AddUser (

p\_id\_user IN INT,

p\_login IN VARCHAR2,

p\_password IN VARCHAR2

)

IS

BEGIN

INSERT INTO Users (ID\_User, LOGIN, PASSWORD)

VALUES (p\_id\_user, p\_login, p\_password);

COMMIT;

END AddUser;

EXEC AddUser(1, 'User2', 123 );

--------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE DeleteUser (

p\_id\_user IN INT

)

IS

BEGIN

DELETE FROM Users WHERE ID\_User = p\_id\_user;

COMMIT;

END DeleteUser;

-------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE UpdatePassword (

p\_id\_user IN INT,

p\_new\_password IN VARCHAR2

)

IS

BEGIN

UPDATE Users SET PASSWORD = p\_new\_password WHERE ID\_User = p\_id\_user;

COMMIT;

END UpdatePassword;

CREATE OR REPLACE PROCEDURE AddProduct (

p\_id\_product IN INT,

p\_type\_chocolate IN NUMBER,

p\_id\_type\_chocolate IN NUMBER,

p\_brand\_code IN NUMBER,

p\_manufacturers\_code IN NUMBER,

p\_weight IN VARCHAR2,

p\_price IN NUMBER,

p\_foto IN VARCHAR2,

p\_structure IN VARCHAR2,

p\_quantity IN INT,

p\_sum IN INT,

p\_date\_buy IN DATE

)

IS

v\_Blob BLOB;

v\_File BFILE;

v\_id\_buy INT;

v\_id\_product INT;

BEGIN

-- Открываем файл с изображением

v\_File := BFILENAME('DIR', p\_foto);

DBMS\_LOB.fileopen(v\_File, DBMS\_LOB.file\_readonly);

-- Создаем BLOB и копируем данные из BFILE

DBMS\_LOB.createtemporary(v\_Blob, TRUE);

DBMS\_LOB.loadfromfile(v\_Blob, v\_File, DBMS\_LOB.getLength(v\_File));

-- Закрываем BFILE

DBMS\_LOB.fileclose(v\_File);

-- Вставляем данные в таблицу PRODUCT

INSERT INTO PRODUCT (

ID\_PRODUCT,

TYPE\_OF\_CHOCOLATE,

ID\_TYPE\_OF\_CHOCOLATE,

BRAND\_CODE,

MANUFACTURERS\_CODE,

WEIGHT,

PRICE,

FOTO,

STRUCTURE

) VALUES (

p\_id\_product,

p\_type\_chocolate,

p\_id\_type\_chocolate,

p\_brand\_code,

p\_manufacturers\_code,

p\_weight,

p\_price,

v\_Blob,

p\_structure

);

-- Получаем ID\_PRODUCT

SELECT ID\_PRODUCT INTO v\_id\_product FROM PRODUCT WHERE ID\_PRODUCT = p\_id\_product;

-- Вставляем данные в таблицу PURCHASE

INSERT INTO PURCHASE (

ID\_BUY,

ID\_PRODUCT,

QUANTITY,

SUM,

DATE\_BUY

) VALUES (

p\_id\_product,

p\_id\_product,

p\_quantity,

p\_sum,

p\_date\_buy

);

-- Вставляем данные в таблицу PURCHASE\_PRODUCT

INSERT INTO PURCHASE\_PRODUCT (

ID\_BUY,

ID\_PRODUCT

) VALUES (

p\_id\_product,

p\_id\_product

);

-- Фиксируем изменения

COMMIT;

-- Освобождаем временные ресурсы BLOB

DBMS\_LOB.freetemporary(v\_Blob);

END AddProduct;

-- Процедура для удаления данных из таблицы PRODUCT

CREATE OR REPLACE PROCEDURE DeleteProduct (

p\_id\_product IN VARCHAR2

)

IS

n\_id\_product INT;

BEGIN

-- Пробуем конвертировать строку в число

BEGIN

n\_id\_product := TO\_NUMBER(p\_id\_product);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20004, 'ID продукта должен быть числом');

RETURN;

END;

IF n\_id\_product <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'ID продукта должен быть положительным числом');

ELSIF MOD(n\_id\_product, 1) <> 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'ID продукта должен быть целым числом');

ELSE

-- Удаляем продукт, все связанные записи в PURCHASE\_PRODUCT и PURCHASE будут удалены каскадно

DELETE FROM PRODUCT WHERE ID\_PRODUCT = n\_id\_product;

COMMIT;

END IF;

EXCEPTION

-- Обрабатываем ситуацию, когда не найден продукт с указанным ID

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Продукт с указанным ID не найден');

WHEN OTHERS THEN

-- Обрабатываем другие ошибки

DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);

END DeleteProduct;

-- Процедура для изменения данных в таблице PRODUCT

CREATE OR REPLACE PROCEDURE UpdateProductPriceAndFoto (

p\_id\_product IN INT,

p\_price IN NUMBER,

p\_foto IN VARCHAR2

)

IS

v\_Blob BLOB;

v\_File BFILE;

BEGIN

-- Проверка на положительные числа

IF p\_id\_product <= 0 OR p\_price <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'ERROR: Product ID and price must be positive integers');

END IF;

-- Проверка на существование файла

BEGIN

v\_File := BFILENAME('DIR', p\_foto);

DBMS\_LOB.fileopen(v\_File, DBMS\_LOB.file\_readonly);

DBMS\_LOB.fileclose(v\_File);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20002, 'ERROR: File not found');

END;

-- Открываем BFILE для чтения

v\_File := BFILENAME('DIR', p\_foto);

DBMS\_LOB.fileopen(v\_File, DBMS\_LOB.file\_readonly);

-- Создаем BLOB и копируем данные из BFILE

DBMS\_LOB.createtemporary(v\_Blob, TRUE);

DBMS\_LOB.loadfromfile(v\_Blob, v\_File, DBMS\_LOB.getLength(v\_File));

-- Закрываем BFILE

DBMS\_LOB.fileclose(v\_File);

-- Обновляем запись в таблице PRODUCT

UPDATE PRODUCT

SET PRICE = p\_price, FOTO = v\_Blob

WHERE ID\_PRODUCT = p\_id\_product;

-- Обновляем записи в таблице PURCHASE

UPDATE PURCHASE

SET PRICE = p\_price

WHERE ID\_PRODUCT = p\_id\_product;

-- Обновляем записи в таблице PURCHASE\_PRODUCT

UPDATE PURCHASE\_PRODUCT

SET PRICE = p\_price

WHERE ID\_PRODUCT = p\_id\_product;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Обработка ошибок

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

END UpdateProductPriceAndFoto;

CREATE OR REPLACE PROCEDURE AddMark (

p\_brand\_code IN INT,

p\_brand\_name IN VARCHAR2

)

IS

brand\_code\_exists INT;

BEGIN

-- Проверяем, существует ли код бренда

SELECT COUNT(\*) INTO brand\_code\_exists FROM MARK WHERE BRAND\_CODE = p\_brand\_code;

IF p\_brand\_code <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Код бренда должен быть положительным числом');

ELSIF brand\_code\_exists > 0 THEN

RAISE\_APPLICATION\_ERROR(-20007, 'Бренд с указанным кодом уже существует');

END IF;

FOR i IN 1..LENGTH(p\_brand\_name) LOOP

IF ASCII(SUBSTR(p\_brand\_name, i, 1)) BETWEEN 48 AND 57 THEN

RAISE\_APPLICATION\_ERROR(-20006, 'Название бренда не может содержать числа');

END IF;

END LOOP;

INSERT INTO MARK (BRAND\_CODE, BRAND\_NAME)

VALUES (p\_brand\_code, p\_brand\_name);

-- Добавляем запись в таблицу PRODUCT

INSERT INTO PRODUCT (BRAND\_CODE)

VALUES (p\_brand\_code);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Обрабатываем другие ошибки

DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);

END AddMark;

-- Процедура для удаления данных из таблицы MARK

CREATE OR REPLACE PROCEDURE DeleteMark (

p\_brand\_code IN NUMBER

)

IS

n\_id\_product INT;

BEGIN

BEGIN

n\_id\_product := TO\_NUMBER(p\_brand\_code);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20004, 'ID продукта должен быть числом');

RETURN;

END;

IF n\_id\_product <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'ID продукта должен быть положительным числом');

ELSIF MOD(n\_id\_product, 1) <> 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'ID продукта должен быть целым числом');

-- Удаляем запись из таблицы PRODUCT

DELETE FROM PRODUCT WHERE BRAND\_CODE = p\_brand\_code;

-- Удаляем запись из таблицы MARK

DELETE FROM MARK WHERE BRAND\_CODE = p\_brand\_code;

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

-- Обрабатываем другие ошибки

DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);

END DeleteMark;

-- Процедура для изменения данных в таблице MARK

CREATE OR REPLACE PROCEDURE UpdateMarkName (

p\_brand\_code IN INT,

p\_brand\_name IN VARCHAR2

)

IS

v\_brand\_name\_check NUMBER; -- Добавим переменную для проверки числового значения

BEGIN

-- Проверка на положительное значение первого параметра

IF p\_brand\_code <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'ERROR: Brand code must be a positive integer');

END IF;

-- Проверка на то, что второй параметр не является числом

BEGIN

-- Пробуем преобразовать строку в число

SELECT TO\_NUMBER(p\_brand\_name) INTO v\_brand\_name\_check FROM DUAL;

RAISE\_APPLICATION\_ERROR(-20002, 'ERROR: Brand name must be a non-numeric string');

EXCEPTION

WHEN VALUE\_ERROR THEN

NULL; -- Продолжаем выполнение, если значение не является числом

END;

-- Обновляем запись в таблице MARK

UPDATE MARK

SET BRAND\_NAME = p\_brand\_name

WHERE BRAND\_CODE = p\_brand\_code;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Обработка ошибок

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

END UpdateMarkName;

CREATE SEQUENCE PRODUCT\_SEQ START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE PROCEDURE AddProducer (

p\_manufacturers\_code IN INT,

p\_manufacturers\_name IN VARCHAR2

)

IS

v\_seq\_val NUMBER; -- переменная для хранения значения из последовательности

BEGIN

-- Проверка на положительное значение первого параметра

IF p\_manufacturers\_code <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'ERROR: Manufacturers code must be a positive integer');

END IF;

-- Проверка на то, что второй параметр является нечисловой строкой

BEGIN

-- Пробуем преобразовать строку в число

v\_seq\_val := TO\_NUMBER(p\_manufacturers\_name);

RAISE\_APPLICATION\_ERROR(-20002, 'ERROR: Manufacturers name must be a non-numeric string');

EXCEPTION

WHEN VALUE\_ERROR THEN

NULL; -- Продолжаем выполнение, если значение является нечисловой строкой

END;

-- Добавляем запись в таблицу PRODUCER

BEGIN

INSERT INTO PRODUCER (MANUFACTURERS\_CODE, MANUFACTURERS\_NAME)

VALUES (p\_manufacturers\_code, p\_manufacturers\_name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

RAISE\_APPLICATION\_ERROR(-20003, 'ERROR: Manufacturers code already exists');

END;

-- Получаем значение из последовательности PRODUCT\_SEQ

SELECT PRODUCT\_SEQ.NEXTVAL INTO v\_seq\_val FROM DUAL;

-- Добавляем запись в таблицу PRODUCT

INSERT INTO PRODUCT (ID\_PRODUCT, MANUFACTURERS\_CODE)

VALUES (v\_seq\_val, p\_manufacturers\_code);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Обработка ошибок

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

END AddProducer;

-- Процедура для удаления данных из таблицы PRODUCER

CREATE OR REPLACE PROCEDURE DeleteProducer (

p\_manufacturers\_code IN INT

)

IS

v\_exists NUMBER; -- переменная для проверки наличия записи

BEGIN

-- Проверка на положительное значение параметра

IF p\_manufacturers\_code <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'ERROR: Manufacturers code must be a positive integer');

END IF;

-- Проверка на существование элемента

SELECT COUNT(\*) INTO v\_exists FROM PRODUCER WHERE MANUFACTURERS\_CODE = p\_manufacturers\_code;

IF v\_exists = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'ERROR: Manufacturers code does not exist');

END IF;

-- Удаляем запись из таблицы PRODUCT

DELETE FROM PRODUCT WHERE MANUFACTURERS\_CODE = p\_manufacturers\_code;

-- Удаляем запись из таблицы PRODUCER

DELETE FROM PRODUCER WHERE MANUFACTURERS\_CODE = p\_manufacturers\_code;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Обработка ошибок

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

END DeleteProducer;

-- Процедура для изменения данных в таблице PRODUCER

CREATE OR REPLACE PROCEDURE UpdateProducerName

(

p\_manufacturers\_code IN INT,

p\_manufacturers\_name IN VARCHAR2

)

IS

l\_exists

NUMBER;

BEGIN -- Проверка на положительное значение первого параметра

IF p\_manufacturers\_code <= 0 THEN RAISE\_APPLICATION\_ERROR(-20001, 'ERROR: Manufacturers code must be a positive integer');

END IF;

-- Проверка на то, что второй параметр является нечисловой строкой

BEGIN

-- Пробуем преобразовать строку в число

l\_exists := TO\_NUMBER(p\_manufacturers\_name);

RAISE\_APPLICATION\_ERROR(-20002, 'ERROR: Manufacturers name must be a non-numeric string');

EXCEPTION

WHEN VALUE\_ERROR THEN

NULL; -- Продолжаем выполнение, если значение является нечисловой строкой

END;

-- Проверка на существование элемента

SELECT COUNT(\*) INTO l\_exists

FROM PRODUCER

WHERE MANUFACTURERS\_CODE = p\_manufacturers\_code;

IF l\_exists = 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'ERROR: Manufacturers code does not exist');

END IF;

-- Обновляем запись в таблице PRODUCER

UPDATE PRODUCER

SET MANUFACTURERS\_NAME = p\_manufacturers\_name

WHERE MANUFACTURERS\_CODE = p\_manufacturers\_code;

COMMIT;

EXCEPTION WHEN OTHERS THEN -- Обработка ошибок DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

END UpdateProducerName;

CREATE OR REPLACE PROCEDURE AddChocolateTaste (

p\_id\_product IN INT,

p\_id\_type\_of\_chocolate IN INT,

p\_the\_name\_the\_taste\_of\_chocolate IN VARCHAR2

)

IS

BEGIN

INSERT INTO TASTE\_CHOCOLATE (ID\_TYPE\_OF\_CHOCOLATE, THE\_NAME\_THE\_TASTE\_OF\_CHOCOLATE)

VALUES (p\_id\_type\_of\_chocolate, p\_the\_name\_the\_taste\_of\_chocolate);

-- Добавляем запись в таблицу PRODUCT

INSERT INTO PRODUCT (ID\_PRODUCT, ID\_TYPE\_OF\_CHOCOLATE)

VALUES (p\_id\_product, p\_id\_type\_of\_chocolate);

COMMIT;

END AddChocolateTaste;

-- Процедура для удаления данных из таблицы TASTE\_CHOCOLATE

CREATE OR REPLACE PROCEDURE DeleteChocolateTaste (

p\_id\_type\_of\_chocolate IN INT

)

IS

BEGIN

-- Удаляем запись из таблицы PRODUCT

DELETE FROM PRODUCT WHERE ID\_TYPE\_OF\_CHOCOLATE = p\_id\_type\_of\_chocolate;

-- Удаляем запись из таблицы TASTE\_CHOCOLATE

DELETE FROM TASTE\_CHOCOLATE

WHERE ID\_TYPE\_OF\_CHOCOLATE = p\_id\_type\_of\_chocolate;

COMMIT;

END DeleteChocolateTaste;

-- Процедура для изменения данных в таблице TASTE\_CHOCOLATE

CREATE OR REPLACE PROCEDURE UpdateChocolateTasteName (

p\_id\_type\_of\_chocolate IN INT,

p\_the\_name\_the\_taste\_of\_chocolate IN VARCHAR2

)

IS

BEGIN

-- Обновляем запись в таблице TASTE\_CHOCOLATE

UPDATE TASTE\_CHOCOLATE

SET THE\_NAME\_THE\_TASTE\_OF\_CHOCOLATE = p\_the\_name\_the\_taste\_of\_chocolate

WHERE ID\_TYPE\_OF\_CHOCOLATE = p\_id\_type\_of\_chocolate;

COMMIT;

END UpdateChocolateTasteName;

CREATE OR REPLACE PROCEDURE SortAndDisplayProducts (

p\_sort\_column VARCHAR2

)

IS

v\_column\_count NUMBER;

v\_sql\_statement VARCHAR2(4000);

v\_cursor SYS\_REFCURSOR;

v\_rec PRODUCT%ROWTYPE;

BEGIN

-- Проверка наличия колонки сортировки

SELECT COUNT(\*)

INTO v\_column\_count

FROM USER\_TAB\_COLUMNS

WHERE TABLE\_NAME = 'PRODUCT' AND COLUMN\_NAME = UPPER(p\_sort\_column);

IF v\_column\_count = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Указанной колонки для сортировки не существует.');

RETURN;

END IF;

-- Динамический SQL для сортировки

v\_sql\_statement := 'SELECT \* FROM PRODUCT ORDER BY ' || p\_sort\_column;

-- Открываем курсор для динамического SQL

OPEN v\_cursor FOR v\_sql\_statement;

-- Обрабатываем результаты запроса

LOOP

FETCH v\_cursor INTO v\_rec;

EXIT WHEN v\_cursor%NOTFOUND;

-- Выводим результаты на экран

DBMS\_OUTPUT.PUT\_LINE(

v\_rec.ID\_PRODUCT || ', ' || v\_rec.TYPE\_OF\_CHOCOLATE || ', ' || v\_rec.ID\_TYPE\_OF\_CHOCOLATE || ', ' ||

v\_rec.BRAND\_CODE || ', ' || v\_rec.MANUFACTURERS\_CODE || ', ' || v\_rec.WEIGHT || ', ' ||

v\_rec.PRICE || ', ' || TO\_CHAR(v\_rec.FOTO) || ', ' || v\_rec.STRUCTURE

);

END LOOP;

-- Закрываем курсор

CLOSE v\_cursor;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);

END SortAndDisplayProducts;

BEGIN

SortAndDisplayProducts('PRICE');

END;

-- ИЛИ

BEGIN

SortAndDisplayProducts('WEIGHT');

END;

--------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE SearchProduct (

p\_search\_term VARCHAR2

)

IS

BEGIN

FOR rec IN (SELECT \* FROM PRODUCT WHERE UPPER(STRUCTURE) LIKE '%' || UPPER(p\_search\_term) || '%')

LOOP

DBMS\_OUTPUT.PUT\_LINE(

rec.ID\_PRODUCT || ', ' || rec.TYPE\_OF\_CHOCOLATE || ', ' || rec.ID\_TYPE\_OF\_CHOCOLATE || ', ' ||

rec.BRAND\_CODE || ', ' || rec.MANUFACTURERS\_CODE || ', ' || rec.WEIGHT || ', ' ||

rec.PRICE || ', ' || TO\_CHAR(rec.FOTO) || ', ' || rec.STRUCTURE

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);

END SearchProduct;

BEGIN

SearchProduct('Milk');

END;

CREATE OR REPLACE PROCEDURE EXPORT\_JSON IS

v\_file UTL\_FILE.FILE\_TYPE;

v\_cursor SYS\_REFCURSOR;

v\_row MARK%ROWTYPE;

v\_json CLOB;

BEGIN

v\_file := UTL\_FILE.FOPEN('JSONDIR', 'JSON\_DATA.json', 'W');

OPEN v\_cursor FOR SELECT \* FROM MARK;

v\_json := '[';

LOOP

FETCH v\_cursor INTO v\_row;

EXIT WHEN v\_cursor%NOTFOUND;

IF v\_json != '[' THEN

v\_json := v\_json || ',';

END IF;

v\_json := v\_json || '{';

v\_json := v\_json || '"BRAND\_CODE":' || NVL(TO\_CHAR(v\_row.BRAND\_CODE),'NULL') || ',';

v\_json := v\_json || '"BRAND\_NAME":' || NVL(TO\_CHAR(v\_row.BRAND\_NAME), 'NULL') || '}';

END LOOP;

CLOSE v\_cursor;

v\_json := v\_json || ']';

UTL\_FILE.PUT\_LINE(v\_file, v\_json);

UTL\_FILE.FCLOSE(v\_file);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

RAISE;

END;

exec InsertRating(5, 1, 1);

exec InsertRating(3, 1, 1);

exec EXPORT\_JSON;

exec IMPORT\_JSON;

truncate table Ratings;

CREATE OR REPLACE PROCEDURE IMPORT\_JSON

IS

BEGIN

INSERT INTO MARK (BRAND\_CODE, BRAND\_NAME)

SELECT BRAND\_CODE, BRAND\_NAME

FROM JSON\_TABLE(BFILENAME('JSONDIR', 'JSON\_DATA.json'), '$[\*]' COLUMNS (

BRAND\_CODE NUMBER PATH '$.BRAND\_CODE',

BRAND\_NAME NUMBER PATH '$.BRAND\_NAME'

)

);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

RAISE;

END;

CREATE OR REPLACE PROCEDURE MovePurchaseToOrders (

p\_user\_id IN INT

)

IS

BEGIN

-- Вставляем данные из PURCHASE в ORDERS

INSERT INTO ORDERS (ID\_PRODUCT, ID\_User, QUANTITY, SUM, DATE\_BUY)

SELECT ID\_PRODUCT, ID\_User, QUANTITY, SUM, DATE\_BUY

FROM PURCHASE

WHERE ID\_User = p\_user\_id;

-- Удаляем строки с указанным ID\_User из PURCHASE

DELETE FROM PURCHASE WHERE ID\_User = p\_user\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Обработка ошибок

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

END MovePurchaseToOrders;

CREATE OR REPLACE PROCEDURE CascadeDeleteUser (

p\_user\_id IN INT

)

IS

BEGIN

-- Удаляем записи из таблицы PURCHASE связанные с пользователем

DELETE FROM PURCHASE WHERE ID\_User = p\_user\_id;

-- Удаляем пользователя из таблицы Users

DELETE FROM Users WHERE ID\_User = p\_user\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Обработка ошибок

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

END CascadeDeleteUser;

ПРИЛОЖЕНИЕ В

Добавление 100 000 строк

CREATE OR REPLACE PROCEDURE CenerateMark

AS

v\_Brand\_code INT;

v\_Brand\_name VARCHAR2(255);

BEGIN

FOR i in 1..10 LOOP

v\_Brand\_code := i;

v\_Brand\_name := DBMS\_RANDOM.STRING('A',50);

INSERT INTO MARK (BRAND\_CODE,BRAND\_NAME)

VALUES (v\_Brand\_code,v\_Brand\_name);

END LOOP;

COMMIT;

END CenerateMark;