МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных интернет-магазина компьютерных комплектующих с применением технологии резервного копирования и восстановления»

Выполнил студент Савко Александр Дмитриевич

(Ф.И.О.)

Руководитель работы

(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 2024

Содержание

[Введение 3](#_Toc153984086)

[1 Постановка задачи 4](#_Toc153984087)

[2 Проектирование базы данных 5](#_Toc153984088)

[2.1 Определение вариантов использования 5](#_Toc153984089)

[2.2 Диаграммы UML, взаимодействие всех компонентов 6](#_Toc153984090)

[2.3 Вывод 6](#_Toc153984091)

[3 Разработка объектов базы данных 7](#_Toc153984092)

[3.1 Разработка таблиц базы данных 7](#_Toc153984093)

[3.2 Разработка представлений базы данных 10](#_Toc153984094)

[3.3 Разработка индексов базы данных 11](#_Toc153984095)

[3.4 Разработка процедур базы данных 11](#_Toc153984097)

[3.5 Разработка функций базы данных 12](#_Toc153984098)

[3.6 Разработка триггеров базы данных 13](#_Toc153984099)

[3.7 Создание пользователей 14](#_Toc153984099)

[3.8 Вывод 14](#_Toc153984100)

[4 Описание процедур импорта и экспорта 15](#_Toc153984101)

[5 Тестирование производительности 16](#_Toc153984102)

[6 Описание технологии и ее применение в базе данных 18](#_Toc153984103)

[7 Руководство пользователя 19](#_Toc153984104)

[Заключение 20](#_Toc153984105)

[Список используемых источников 21](#_Toc153984106)

[Приложение А. Листинг создания таблиц 22](#_Toc153984107)

[Приложение Б. Листинг процедур базы данных 24](#_Toc153984108)

[Приложение В. Листинг функций базы данных 40](#_Toc153984109)

[Приложение Г. Листинг представлений базы данных 42](#_Toc153984110)

[Приложение Д . Листинг кода создания пользователей 43](#_Toc153984111)

[Приложение Е. Листинг процедур для импорта и экспорта в xml 45](#_Toc153984111)

# Введение

Целью работы является разработка базы данных для хранения информации о товарах, их параметрах, заказах и адресах. Создание такой базы данных позволит эффективно организовать торговли, обеспечив удобный доступ к информации о товарах, а также обеспечив точный и надежный учет заказов. Кроме того, целью работы может быть создание интернет-магазина, который будет позволять автоматизировать процесс заказа комплектующих, а также обеспечивать мониторинг и контроль за выполнением заказов. Данная база данных обеспечит эффективное хранение данных, доступ к ним и их фильтрацию, разделение прав доступа к данным.

База данных (БД) – это система, предназначенная для хранения, организации и управления данными в структурированном виде, которая обеспечивает удобный доступ к этим данным и позволяет эффективно обрабатывать и анализировать большие объемы информации. База данных может содержать информацию о различных объектах (например, людях, продуктах, заказах и т.д.), а также связи между ними и правила для доступа и использования этой информации, база данных позволяет повысить скорость доступа к ним, а также избежать дублирования данных. База данных используется в различных областях, включая бизнес, науку, медицину и другие.

База данных должна быть защищена от несанкционированного доступа. Для этого данные изолированы от прямого вмешательства пользователей с помощью функций, процедур и представлений, которые в свою очередь распределены по пользователям в зависимости от необходимого для их роли набора привилегий.

Для обеспечения непрерывной работы базы данных и сохранения целостности данных в ней используется технология резервного копирования базы данных. Это позволяет быстро восстанавливать данные в случае их потери или повреждения, обеспечивая бесперебойную работу базы данных.

Для повышения скорости чтения данных из базы данных были созданы индексы для самых часто используемых наборов столбцов.

В пояснительной записке содержится информация о структуре и реализации проекта, а также инструкции по использованию приложения.

# 1 Постановка задачи

Важным этапом разработки любого нового продукта является анализ существующих на рынке аналогов. Он позволяет определить, какие решения уже представлены потребителям, выявить их сильные и слабые стороны, а также определиться с направлениями для усовершенствований.

Интернет-магазин – это форма торговли, при которой продавец предлагает на продажу товар или услугу, а покупатели оформляют заказ онлайн широко используются в различных отраслях и имеют значительную актуальность. В бизнесе интернет-магазины используются для продажи различных товаров (Программное обеспечение, техника, продукты, и др.).

На текущий момент рынок интернет-магазинов представляет собой множество платформ с различным уровнем функционала и безопасности. Однако, многие из них не могут предложить полноценного комплекса инструментов, сочетающего в себе удобство, гибкость и высокие стандарты безопасности. Это оставляет простор для новых разработок в этой сфере.

Задача проекта: разработать архитектуру приложения, взаимодействие с которой 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.

˗ доступ к данным должен осуществляться только через соответствующие процедуры;

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

˗ 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.

# 2 Проектирование базы данных

## 2.1 Определение вариантов использования

Главное начало функциональных требований базы данных заключается в определении способа обработки данных и обеспечении необходимой функциональности пользовательскому интерфейсу. Важно описать, как данные должны быть хранены и организованы, как осуществляется поиск и выборка данных, как обновляются данные и как обеспечивается безопасность информации. Кроме того, следует учесть взаимодействие базы данных с другими системами и программами. Например, для интернет-магазина функциональные требования могут включать в себя функции для хранения информации аукционах и их лотах.

Кроме функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде Use Case диаграмм, которые описывают, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. На рисунке 2.1 представлена UML диаграмма вариантов использования.

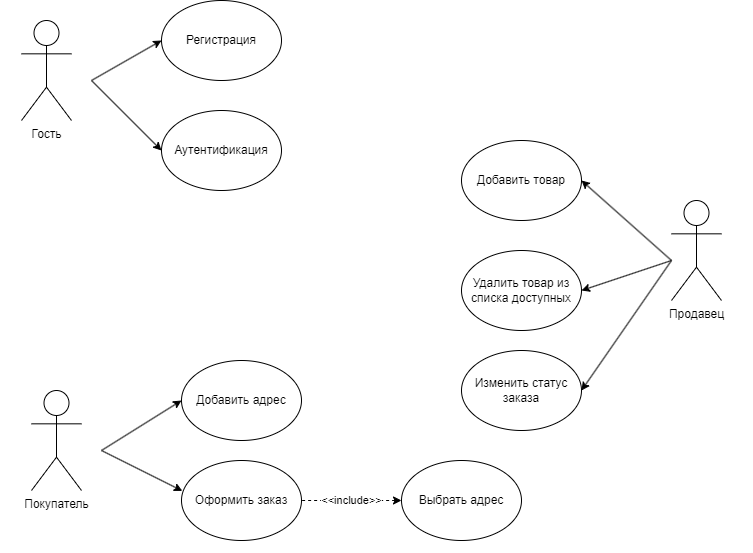


Рисунок 2.1 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. После регистрации гость становится Покупателем (Customer). После регистрации покупатель (Customer) получает возможность оформить заказ.

## 2.2 Диаграммы UML, взаимодействие всех компонентов

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.2.

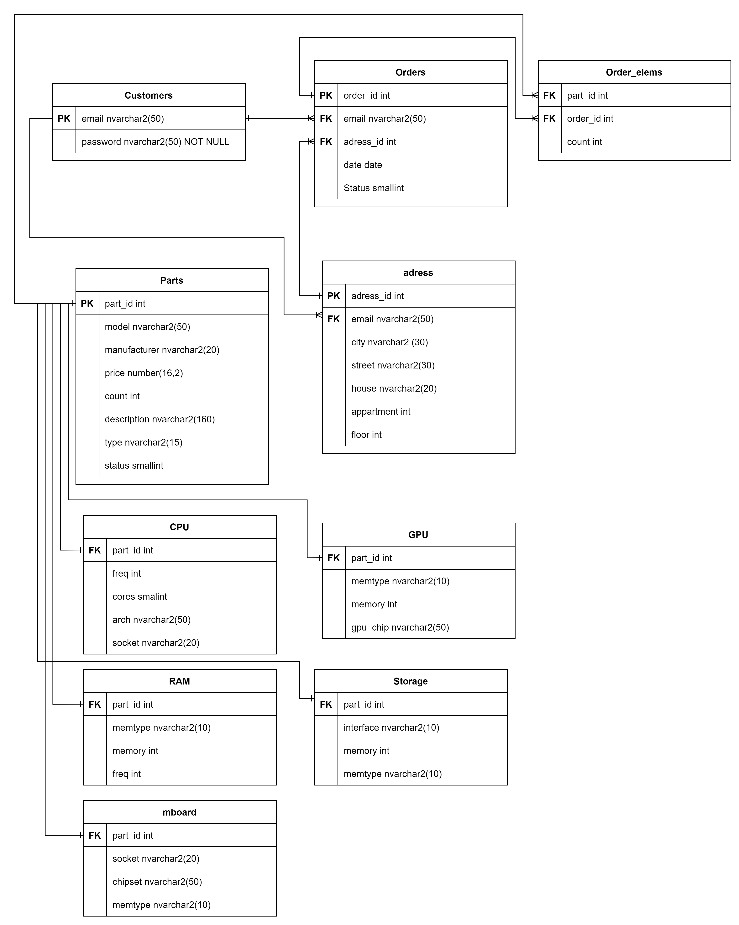


Рисунок 2.2 – Диаграмма базы данных

Видно, что диаграмма показывает связи между таблицами и полями, а также отношения между ними. Одна из основных связей – это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице.

Например, таблица parts связана с таблицей order, через внешний ключ part\_id. Также видно, что таблицы customers и adress связаны друг с другом через внешний и первичный ключ email. Все связи представлены на рисунке 2.2, который располагается выше.

Для обеспечения безопасности данных необходимо реализовать обращение к данным через процедуры и реализовать резервное копирование на базе rman.

## 2.3 Вывод

Такая архитектура базы данных предоставляет гибкость в управлении данными, высокую производительность и безопасность. Структурированное разделение на авторизацию и доменную логику позволяет оптимально использовать ресурсы системы и обеспечивать эффективное взаимодействие с данными.

# 3 Разработка объектов базы данных

## 3.1 Разработка таблиц базы данных

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

Для реализации базы данных было разработано 8 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Пользователи, Лоты, Аукционы и Ставки. Ниже будет описание про каждую из них более подробно.

Таблица Customers содержит информацию о пользователях. Состоит из следующих столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы Customers

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Email | Электронная почта, Уникальный идентификатор пользователя, первичный ключ | Nvarchar2(50) |
| password | Пароль пользователя | Nvarchar2(50) |

Таблица parts содержит информацию о предметах, выставленных на продажу. В таблице Address хранятся адреса пользователей. Состоит из следующих столбцов (таблица 3.2):

Таблица 3.2 – Столбцы таблицы Parts

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Part\_id | Уникальный идентификатор товара, первичный ключ | int |
| Model | Имя товара | nvarchar2(50) |
| Manufacturer | Производитель | Nvarchar2(20) |
| Description | Описание товара | nvarchar2(160) |
| Price | Цена товара | Number (16,2) |
| Count | Количество на складе | Int |
| Type | Тип товара | Nvarchar2(15) |
| Status | Статус товара | Smailint |

Таблица Address хранит информацию о адресах для доставки, содержит следующие столбцы (таблица 3.3):

Таблица 3.3 – Столбцы таблицы Adress

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Address\_id | Уникальный идентификатор адреса, первичный ключ | Int |
| Email | Email пользователя | nVARCHAR2(50) |
| City | Город | nVARCHAR2(30) |
| Street | Улица | nVARCHAR2(30) |
| House | Дом | Nvarchar2(20) |
| appartment | Квартира | Int |
| Floor | Этаж | Int |

Таблица orders содержит данные о заказах. Содержит в себе такие столбцы, такие как (таблица 3.4):

Таблица 3.4 – Столбцы таблицы orders

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Order\_id | Уникальный идентификатор элемента заказа, первичный ключ | Int |
| Email | Идентификатор пользователя, который сделал заказ | Nvarchar2(50) |
| Adress\_id | Идентификатор адреса | Int |
| Part\_id | Идентификатор товара | Int |
| Count int | Количество товара | Int |
| Date | Дата для получения заказа | Date |
| Status | Статус заказа | Smallint |

Ниже указаны таблицы с уникальными характеристиками для определенных типов комплектующих (процессоры, видеокарты, накопители и озу).

Таблица 3.5 – Столбцы таблицы CPU

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Part\_id | Уникальный идентификатор товара, первичный ключ | Int |
| Freq | Тактовая частота процессора в мгц. | Int |
| Cores | Кол-во ядер процессора | Int |
| Arch | Архитектура ядер процессора | Nvarchar2(50) |

В таблице cpu хранятся специфичные для процессоров данные.

Таблица 3.6 – Столбцы таблицы Ram

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Part\_id | Уникальный идентификатор товара, первичный ключ | Int |
| Freq | Частота памяти в мгц. | Int |
| Memtype | Тип памяти. | Nvarchar2(10) |
| memory | Объем памяти в гб. | Int |

В таблице Ram хранятся специфичные для оперативной памяти.

Таблица 3.7 – Столбцы таблицы GPU

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Part\_id | Уникальный идентификатор товара, первичный ключ | SERIAL |
| memtype | Тип видеопамяти. | SERIAL |
| Memory | Объем видеопамяти в гб. | int |
| Gpu\_chip | Графический процессор видеокарты. | Nvarchar2(50) |

В таблице Gpu хранятся специфичные для видеокарт.

Таблица 3.8 – Столбцы таблицы Storage

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Part\_id | Уникальный идентификатор товара, первичный ключ. | Int |
| memory | Объем памяти в гб. | Int |
| interface | Интерфейс подключения. | Nvarchar2(10) |
| memtype | Тип накопителя. | Nvarchar2(10) |

В таблице mboard хранятся специфичные для материнских плат.

Таблица 3.9 – Столбцы таблицы mboard

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Part\_id | Уникальный идентификатор товара, первичный ключ. | Int |
| Memtype | Тип памяти. | Nvarchar2(10) |
| Socket | Сокет процессора | Nvarchar2(10) |
| chipset | Чипсет платы. | Nvarchar2(10) |

В таблице order\_elems хранятся данные каждой позиции в заказе.

Таблица 3.10 – Столбцы таблицы order\_elems

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Part\_id | Уникальный идентификатор товара, первичный ключ. | Int |
| Order\_id | Тип памяти. | Int |
| Count | Сокет процессора | Int |

Перед созданием таблиц, для их структуризации и изоляции было принято решение использовать табличные пространства.

Табличное пространство – это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств представлен в листинге 3.1.

|  |
| --- |
| Create tablespace shopts  DATAFILE 'shopts.dbf'  SIZE 10m  AUTOEXTEND ON NEXT 10m MAXSIZE UNLIMITED;  Create temporary tablespace shopts\_tmp  TempFILE 'shopts\_temp.dbf'  SIZE 10m  AUTOEXTEND ON NEXT 10m MAXSIZE UNLIMITED; |

Листинг 3.1 – Cкрипт для создания табличных пространств

Листинг SQL-кода для создания таблиц находится в [приложении А](#_Приложение_А).

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Для таблицы parts был создан индекс на основании столбцов type, manufacturer и model, так как они редко изменяются и часто используются для поиска.

Для более изоляции пользователя от таблиц и облегчения некоторого функционала создано несколько процедур и представлений.

## 3.2 Разработка представлений базы данных

Представление в базе данных – это подмножество базы данных, основанное на запросе, который запущен для одной или нескольких таблиц базы данных. Представления позволяют объединять данные из нескольких таблиц в одну, не изменяя исходных таблиц и их структуры. Представления используются для обеспечения удобства доступа и управления данными в базе данных, а также для изоляции пользователя от оригинальных таблиц. В данном проекте были созданы представления:

* Parts\_v показывает весь список комплектующих, вне зависимости от их типа;
* adress\_v содержит все адреса;
* cpu\_v отображает join таблицы parts(содержит основную информацию о товаре) и cpu(содержит информацию о процессорах).
* Gpu\_v работает по аналогии представления cpu\_v, но относится к видеокартам.

Реализация нескольких из представлений представлено в листинге 3.2. Остальные представления будут аналогичны, работая с другими данными. Их код находится в приложении в

|  |
| --- |
| CREATE OR REPLACE VIEW Parts\_V AS  SELECT \*  FROM parts;  CREATE OR REPLACE VIEW CPU\_V AS  SELECT P.part\_id, P.p\_model, P.manufacturer, P.price, P.p\_count, P.p\_description,  C.freq, C.cores, C.arch, C.socket  FROM parts P  JOIN CPU C ON P.part\_id = C.part\_id;  CREATE OR REPLACE VIEW usermails AS  SELECT email from customers;  CREATE OR REPLACE VIEW GPU\_V AS  SELECT P.part\_id, P.p\_model, P.manufacturer, P.price, P.p\_count, P.p\_description,G.memtype, G.memory, G.gpu\_chip  FROM parts P JOIN GPU G ON P.part\_id = G.part\_id; |

Листинг 3.2 – Представления parts\_v, gpu\_v, и cpu\_v

Представление orders\_v содержит базовые данные о заказах.

Представление orders\_in\_elems\_V содержит данные о заказах и содержимое заказа, это join таблиц orders и order\_elems.

## 3.3 Разработка индексов базы данных

Индекс в базе данных представляет собой объект, который используется для ускорения поиска данных. Если таблица содержит большое количество строк, то последовательный поиск данных может занимать много времени. Индекс создается на основе значений одного или нескольких столбцов таблицы и указывает на соответствующие строки таблицы. Использование индексов помогает улучшить производительность базы данных, поскольку они имеют оптимизированную структуру для поиска, например, сбалансированное дерево. Один из наиболее распространенных алгоритмов индексации – это B-дерево (B-tree). B-дерево – это сбалансированное дерево поиска, которое обеспечивает эффективный поиск элементов, используя ключи для сортировки данных в индексе.

Например, для того чтобы быстро находить комплектующие по их имени, производителю и типу был создан индекс: part\_index по полям TYPE, MANUFACTURER, P\_MODEL, part\_index2 по полям TYPE, P\_MODEL. Создание индексов представлено в листинге 3.3.

|  |
| --- |
| create index part\_index on parts ( TYPE,MANUFACTURER,P\_MODEL);  create index part\_index2 on parts ( TYPE,P\_MODEL);  create index part\_index3 on parts ( MANUFACTURER);  create index part\_index4 on parts ( MANUFACTURER,P\_MODEL); |

Листинг 3.3 – Индексы таблицы parts

Применение индексов в базе данных значительно повышает скорость выполнения операций поиска, сортировки и фильтрации данных, особенно в случае больших объемов информации. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах. Поэтому необходимо балансировать количество и типы индексов, чтобы обеспечить оптимальную производительность базы данных. А также использовать индексы на значениях, которые изменяются редко или вообще не меняются.

## 3.4 Разработка процедур базы данных

Подпрограммы PL / SQL называются блоками PL / SQL, которые могут быть вызваны с помощью набора параметров. PL / SQL предоставляет два вида подпрограмм: функции и процедуры. Процедуры не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Всего было разработано мною 24 процедуры. На листинге 3.4 представлен пример реализации одной из процедур.

|  |
| --- |
| CREATE OR REPLACE procedure create\_customer (mail in nvarchar2,pass in nvarchar2)  is  begin  insert into customers(email,password) values(mail,pass);  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка' || sqlcode);  END; |

Листинг 3.4 – Процедура создания пользователя

Данная процедура создает регистрирует покупателя, в качестве параметров принимает, пароль и email. Процедуры FINISH\_ORDER и UNFINISH\_ORDER нужны для завершения заказа или отмены этого действия, процедуры add\_cpu, edit\_cpu позволяют добавить процессор и изменить комплектующую на процессор с новыми параметрами соответственно. Аналогично работают и другие процедуры с похожими именами. Процедура add\_adress позволяет добавить адрес для приема заказов. Процедуры delete\_part и return\_part позволяют вывести из ассортимента или вернуть туда комплектующую. Процедура make\_order позволяет оформить заказ, а процедуры fill\_order и cut\_order, соответственно, позволяют заполнить заказ товарами или убрать лишнее. Скрипты других процедур приведен в [приложении Б](#_Приложение_В).

## 3.5 Разработка функций базы данных

Функции в Oracle - это именованные блоки программного кода, которые принимают входные значения, обрабатывают их и возвращают результат. Функции используются для выполнения различных операций, например, для вычислений, обработки данных. В отличие от процедур функции используются для получения значений или их получение в обработанном виде. В листинге 3.5 указан код функции

|  |
| --- |
| CREATE OR REPLACE FUNCTION show\_orders\_cust (p\_email IN customers.password%TYPE, p\_pass IN customers.password%TYPE) RETURN sys\_refcursor  IS  ord\_cursor sys\_refcursor;  rcount INT;  BEGIN  SELECT COUNT(\*)  INTO rcount  FROM customers  WHERE customers.email = p\_email  AND customers.password = p\_pass;    IF rcount = 1 THEN  OPEN ord\_cursor FOR  SELECT ord.order\_id, email, adress\_id, elem.part\_id, ord\_date, ord\_status, count  FROM orders ord  JOIN order\_elems elem ON ord.order\_id = elem.order\_id  WHERE email = p\_email;    RETURN ord\_cursor;  ELSE  DBMS\_OUTPUT.PUT\_LINE('проверьте данные.');  RETURN NULL;  END IF;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка '||SQLCODE);  RETURN NULL;  END; |

Листинг 3.5 – Скрипт функции show\_orders\_cust

show\_orders\_cust. Данная функция выводит данные о заказах пользователя в виде курсора, на вход она принимает почту и пароль пользователя. Функция count\_money выводит общую стоимость всех заказов. show\_adress\_cust отображает в виде курсора список всех адресов клиента аналогично функции show\_orders\_cust.

## 3.6 Разработка триггеров базы данных

Триггер в базе данных – это объект, который автоматически выполняет определенные действия при возникновении определенных событий в таблице или представлении базы данных. Триггер может быть настроен на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для поддержки целостности данных, контроля доступа к данным и автоматической обработки данных при выполнении определенных операций в таблице.

в листинге 3.6 приведен код триггера, уведомляющего о добавлении пользователя.

|  |
| --- |
| CREATE OR REPLACE TRIGGER taxWriter  BEFORE INSERT ON parts  FOR EACH ROW  BEGIN  :NEW.price := :NEW.price \* 1.2;  END;  CREATE OR REPLACE TRIGGER manWriter  BEFORE INSERT ON parts  FOR EACH ROW  BEGIN  :NEW.manufacturer := :NEW.manufacturer || 'manufatured by: ';  END;  CREATE OR REPLACE TRIGGER modWriter  BEFORE INSERT ON parts  FOR EACH ROW  BEGIN  :NEW.p\_model := :NEW.p\_model || 'model ';  END; |

Листинг 3.6 – Скрипт триггеров

Триггер taxWriter создается parts и увеличивает цену на 20 процентов от введённого значения. Триггеры modWriter и manWriter приписывают строки “manufactured by ” и “model ” к соответствующим столбцам.

## 3.7 Создание пользователей

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 3 типа пользователей для подключения к базе данных: покупатель, продавец и разработчик.

Создание пользователей представлено в листинге 3.7.

|  |
| --- |
| create role dev\_role not identified;  create role customer\_role not identified;  create role salesman\_role not identified;  Create user dev identified by ads  DEFault tablespace shopts  temporary tablespace shopts\_tmp  profile pf\_dev  account unlock;  password expire;  Create user customer identified by 111  DEFault tablespace shopts  temporary tablespace shopts\_tmp  profile pf\_customer  account unlock;  password expire;  Create user salesman identified by 111  DEFault tablespace shopts  temporary tablespace shopts\_tmp  profile pf\_salesman  account unlock; |

Листинг 3.7 – Создание пользователей

Далее для созданного пользователя базы данных были выданы необходимые привилегии необходимые для функционала клиента. Код выдачи привилегий показан в приложении Д.

## 3.8 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для интернет-магазина компьютерных комплектующих. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе.

# 4 Описание процедур импорта и экспорта

Для таблицы adress в базе данных доступна функция экспорта и импорта данных в формате xml. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в xml, была разработана функция, результатом которой стало создание xml файла, с записанными строками из определенной таблицы в xml формат. Скрипт процедуры представлен в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE GenerateAddressXML IS  xml\_data CLOB;  BEGIN  SELECT XMLELEMENT("Adress",  XMLFOREST(adress\_id AS "AddressID",  email AS "Email",  city AS "City",  street AS "Street",  house AS "House",  a\_floor AS "Floor",  appartment AS "Apartment")  ).getClobVal()  INTO xml\_data  FROM adress;  DBMS\_OUTPUT.PUT\_LINE(xml\_data);  END; |

Листинг 4.1 – процедура GenerateAddressXML

Для обратного экспорта данные в xml формат из файла, реализован запросом, который обращается к xml, и выбирает данные в обычные строки, это можно реализовать с помощью функции, либо по мере надобности записать данные во временную таблицу, либо таблицу в другой базе данных. Скрипт запроса представлен в [приложении Е](#_Приложение_Г).

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах xml, что делает управление базой данных более удобным и эффективным.

# 5 Тестирование производительности

Одной из ключевых задач в процессе разработки является тестирование производительности базы данных. Такое тестирование позволяет оценить способность базы данных обрабатывать запросы и возвращать результаты с высокой скоростью. Оценка производительности важна для определения эффективности базы данных и выявления возможных узких мест, которые могут вызывать задержки или проблемы в работе приложения.

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

Для данной задачи мы создали анонимный блок, и так как работаем именно с действующей базой данных, то можем использовать готовую процедуру для вставки данных в таблицу. Разработанный анонимный блок позволяет добавить большое количество строк за одно выполнение (листинг 5.1).

|  |
| --- |
| DECLARE  -- Объявляем переменные для случайных значений  j NUMBER;  v\_model parts.p\_model%TYPE;  v\_manufacturer parts.manufacturer%TYPE;  v\_price parts.price%TYPE;  v\_count parts.p\_count%TYPE;  v\_description parts.p\_description%TYPE;  BEGIN  -- Генерируем случайные значения для параметров  FOR j IN 1..200000 LOOP  v\_model := 'Model\_' || DBMS\_RANDOM.STRING('A', 5); -- Генерация случайной строки для модели  v\_manufacturer := DBMS\_RANDOM.STRING('A', 10); -- Генерация случайной строки для производителя  v\_price := ROUND(DBMS\_RANDOM.VALUE(10, 1000), 2); -- Генерация случайной цены (от 10 до 1000)  v\_count := ROUND(DBMS\_RANDOM.VALUE(1, 100)); -- Генерация случайного количества (от 1 до 100)  v\_description := DBMS\_RANDOM.STRING('A', 20); -- Генерация случайной строки для описания  -- Вызываем процедуру Add\_Other\_Part с сгенерированными значениями  Add\_Other\_Part(v\_model, v\_manufacturer, v\_price, v\_count, v\_description);  END LOOP;  end; |

Листинг 5.1 – Заполнение таблицы 200000 строк

Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| select \* from parts where MANUFACTURER like 'e%' and p\_model like 'Model\_ad%'; |

Листинг 5.2 – Запрос к таблице parts

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время планирования – 28 мс. Результаты запроса будут представлены на рисунке 5.1.

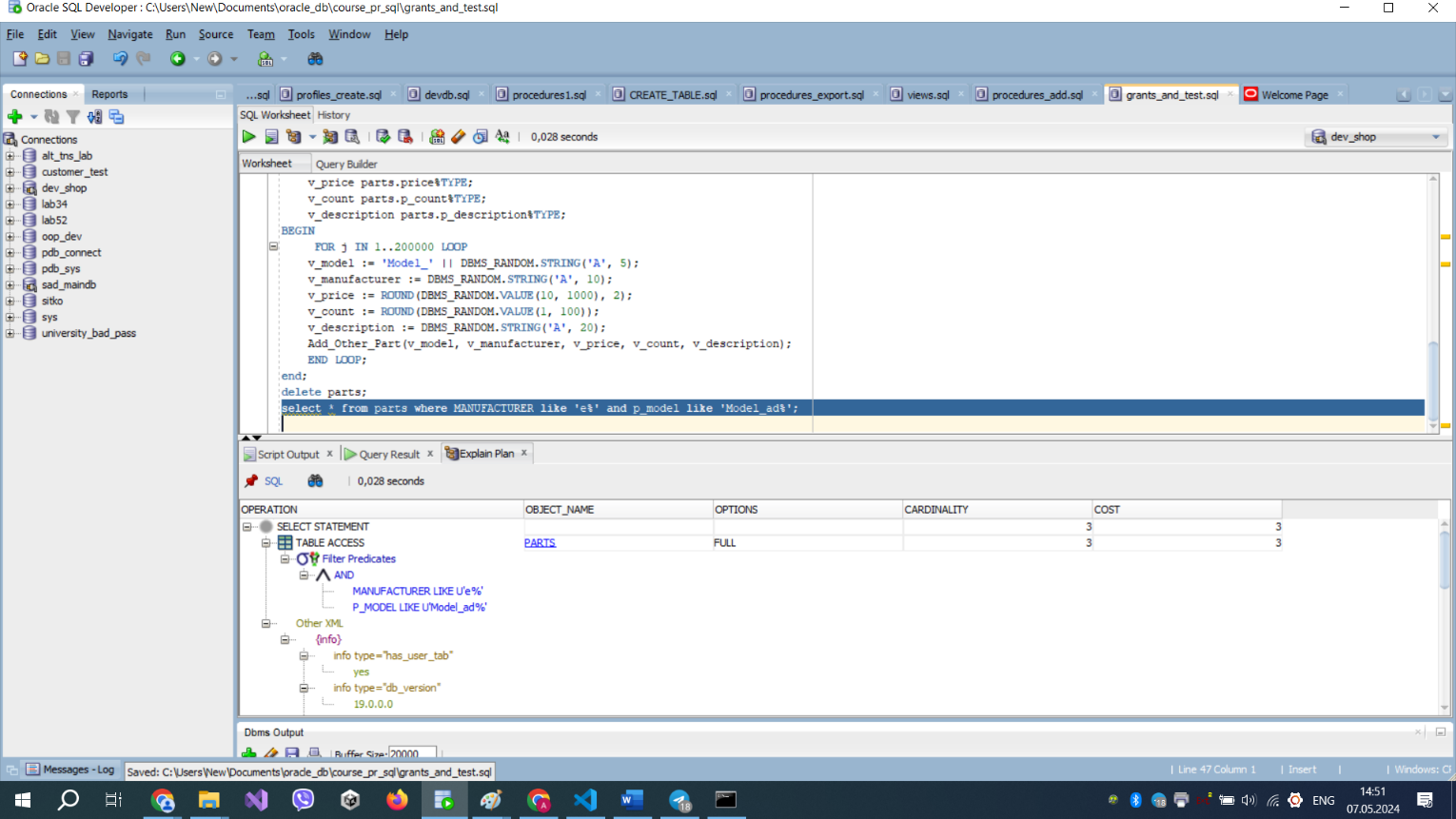


Рисунок 5.1 – Результат выполнения запроса

Для ускорения данного процесса создадим индекс на поле Наименование так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.2.

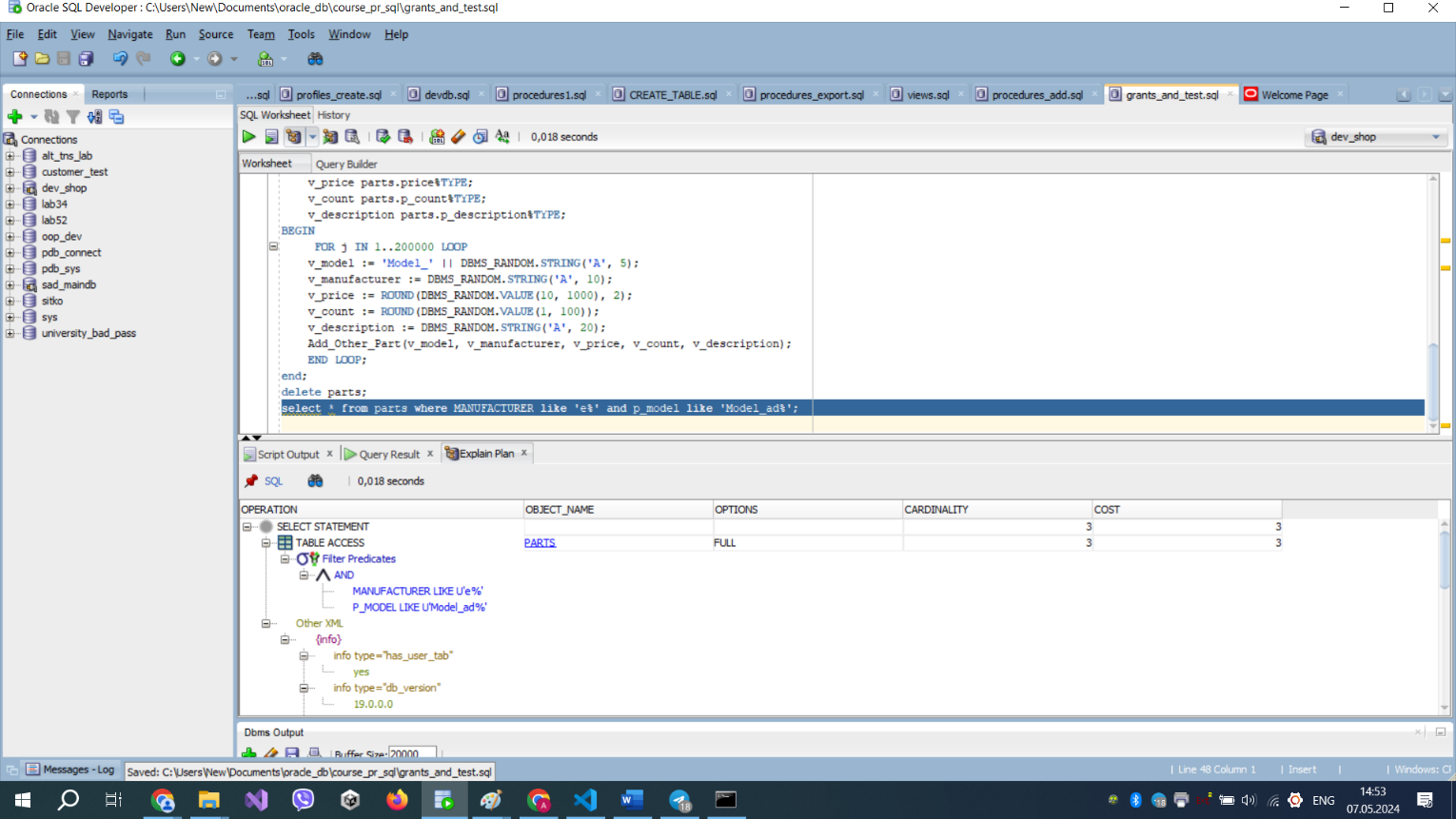


Рисунок 5.2 – Результат выполнения запроса

После создания индексов видно, что лоты стали генерировать намного быстрее. Время планирования заняло 0.18 мс.

Результаты тестирования говорят о том, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

# 6 Описание технологии и ее применение в базе данных

Резервное копирование и восстановление в базах данных представляет собой систему методов и инструментов, предназначенных для сохранения и восстановления данных в случае их потери или повреждения. В данной базе данных используется технология резервного копирования для обеспечения надежности и безопасности хранения информации.

На рисунке 6.1 иллюстрируется процесс резервного копирования данных в базе.

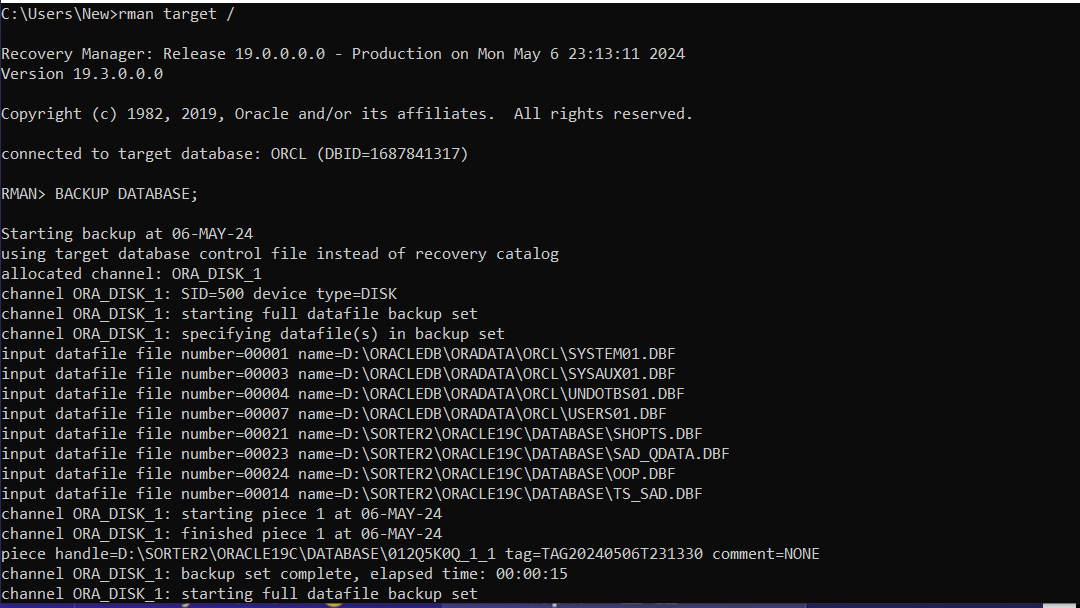


Рисунок 6.1 – Процесс резервного копирования данных

Применение резервного копирования и восстановления обладает несколькими ключевыми преимуществами:

* 1. Сохранение целостности данных. Регулярные резервные копии помогают предотвратить потерю информации в случае аварийных ситуаций, таких как сбои оборудования или программного обеспечения.
  2. Обеспечение безопасности. Резервные копии данных могут служить важным инструментом в защите от угроз безопасности, таких как вирусы или злонамеренные атаки, позволяя быстро восстановить информацию после инцидента.
  3. Восстановление в случае необходимости. Благодаря наличию резервных копий базы данных возможно оперативное восстановление данных до момента их утраты или повреждения.
  4. Эффективное управление данными. Возможность регулярного создания резервных копий и их последующего использования дает возможность контролировать и управлять данными с большей уверенностью.

Технология резервного копирования позволяет обеспечить сохранность и доступность данных, играя ключевую роль в обеспечении непрерывности работы базы данных и предоставлении средств для восстановления информации в случае необходимости.

# 7 Руководство пользователя

При входе в магазин, мы будем иметь статус гостя, это значит – ограниченный функционал. Мы сможем лишь просмотреть, какие только просмотреть товары. Главная функция у гостя – регистрация (процедура create\_customer) и авторизация осуществляется непосредственно при выполнении действий, требующих быть зарегистрированным пользователем (добавление адреса, оформление заказа), для регистрации нужно запустить процедуру create\_customer указав почту, в качестве логина и пароль, после чего регистрация завершена.

После регистрации, пользователь уже имеет одноименный статус – Пользователь. У него открываются новые возможности, он может добавить адрес и оформить заказ, просмотреть собственные заказы и адрес, для выполнения данных операций пользователь должен указать в параметрах соответствующих процедур почту и пароль. Перед оформлением заказа необходимо создать адрес, указав почту, пароль, город, улицу, дом, этаж, квартиру. Для оформления заказа пользователь должен создать заказ, указав дату, почту, пароль, id адреса, после чего он должен вызвать fill\_order для каждого торгового наименования, указав почту, пароль, количество, id заказа и id комплектующей.

Если вход был произведен с аккаунта продавца, то мы сможем редактировать статус заказа, добавлять комплектующие в ассортимент, выводить их оттуда и возвращать их обратно, кроме того, продавец может импортировать и экспортировать адреса. Для того чтобы просмотреть существующие заказы нужно обратиться к представлению orders\_V или orders\_in\_elems\_V, последнее представление показывает join заказов и их содержимого, после того как заказ будет доставлен, продавец должен завершить заказ выполнив процедуру finish\_order. В случае ошибки продавец может откатить эту операцию процедурой unfinish\_order, для этих процедур нужно указать id заказа. Для добавления комплектующих необходимо вызвать соответствующую процедуру, там необходимо указать общие параметры(цена, количество на складе) и специфичные параметры товара, также можно изменить товары, в случае ошибки создания, для товаров, продажа которых приостановлена нужно указать статус с помощью процедуры delete\_part, если товар вернулся в продажу необходимо запустить процедуру restore\_part, данные процедуры принимают id товара.

Вход с учетной записью разработчика дает нам полное управление базой данных интернет-магазина.

Реализация каждой из вышеупомянутой роли приведена в [приложении Д](#_Приложение_Д).

# Заключение

В данном проекте была поставлена задача разработки базы данных для интернет-магазина компьютерных комплектующих с использованием технологии резервного копирования и восстановления в СУБД oracle 19c.

В ходе выполнения проекта были использованы различные объекты, такие как таблицы, представления, процедуры, индексы триггеры, специальные функции, чтобы обеспечить структурированное хранение данных, изолированность данных таблиц от прямого вмешательства и удобный доступ к ним. В итоге, поставленная цель была успешно достигнута, и на выходе получена готовая база данных.

В рамках тестирования базы данных был использован поток данных, и результаты тестирования оказались положительными. Также были разработаны процедуры импорта и экспорта данных в формате xml, что значительно облегчило управление данными и повысило удобство использования базы данных.

Разработанная база данных для интернет-магазина компьютерных комплектующих имеет ряд важных особенностей, которые позволяют эффективно управлять ей. Одной из главных особенностей является технология резервного копирования и восстановления, которая обеспечивает сохранность данных в случае любых сбоев или ошибок. Это означает, что все данные интернет-магазина, могут быть быстро восстановлены, минимизируя потерю информации и обеспечивая непрерывность работы площадки.

Кроме того, база данных обладает гибкой системой привилегий, которая позволяет разграничивать доступ к различным функциям и данным в зависимости от роли пользователя. Это обеспечивает безопасность данных и предотвращает несанкционированный доступ.

Делая вывод, можно сказать, что поставленные цели были выполнены. интернет-магазина компьютерных комплектующих выполняет необходимую функцию хранения и управления данными.

# Список используемых источников

1. Использование диаграммы вариантов использования UML при проектировании программного обеспечения [Электронный ресурс] – https://habr.com/ru/articles/566218/ – Дата доступа 20.04.2024
2. IBM [Электронный ресурс] – https://www.ibm.com/docs/ru/mfci/7.6.2?topic=structure-views – Дата доступа 21.04.2024
3. oracleplsql [Электронный ресурс] – https://oracleplsql.ru/procedures.html – Дата доступа 21.04.2024
4. Администрирование баз данных Oracle [Электронный ресурс] – <https://oracle-dba.ru/database/backup-and-restore/rman/about-oracle-rman/> – Дата доступа 23.04.2024

# Приложение А. Листинг создания таблиц

|  |
| --- |
| create table customers(  email nvarchar2(50),password nvarchar2(50) Encrypt not null,  CONSTRAINT user\_pk PRIMARY KEY (email)  );  --дом текстовый ввиду наличия корпусов, индексов букв и т.д  create table adress(  adress\_id int GENERATED BY DEFAULT ON NULL AS IDENTITY START WITH 1, email nvarchar2(50), city nvarchar2(30) Encrypt not null,street nvarchar2(30) Encrypt not null,  house nvarchar2(30) Encrypt not null,a\_floor int Encrypt not null,appartment int Encrypt not null,  CONSTRAINT adress\_pk PRIMARY KEY (adress\_id),  CONSTRAINT adr\_mail\_fk foreign key(email)references customers(email)  );  create table orders(order\_id int GENERATED BY DEFAULT ON NULL AS IDENTITY START WITH 1,  email nvarchar2(50),adress\_id int,  ord\_date date Encrypt,  CONSTRAINT order\_pk PRIMARY KEY (order\_id),  CONSTRAINT ord\_mail\_fk foreign key(email)references customers(email),  CONSTRAINT adress\_fk foreign key(adress\_id)references adress(adress\_id),  ord\_status smallint  );  CREATE TABLE Order\_elems (  order\_id INT,  part\_id INT,  count INT,  FOREIGN KEY (order\_id) REFERENCES Orders(order\_id) ON DELETE CASCADE,  FOREIGN KEY (part\_id) REFERENCES Parts(part\_id)  );  create table parts(  part\_id int GENERATED BY DEFAULT ON NULL AS IDENTITY START WITH 1,  p\_model nvarchar2(50),  manufacturer nvarchar2(50),  price number(16,2)  ,p\_count int,p\_description nvarchar2(160),  type nvarchar2(15),p\_status smallint default 0,  CONSTRAINT part\_pk PRIMARY KEY (part\_id)  );  CREATE TABLE CPU (  part\_id INT, FOREIGN KEY (part\_id) REFERENCES Parts(part\_id),  freq INT,  cores SMALLINT,  arch NVARCHAR2(50),  socket NVARCHAR2(20)  );  CREATE TABLE GPU (  part\_id INT, FOREIGN KEY (part\_id) REFERENCES Parts(part\_id),  memtype NVARCHAR2(10),  memory INT,  gpu\_chip NVARCHAR2(50)  );  CREATE TABLE RAM (  part\_id INT, FOREIGN KEY (part\_id) REFERENCES Parts(part\_id),  memtype NVARCHAR2(10),  memory INT,  freq INT  );  CREATE TABLE Storage (  part\_id INT, FOREIGN KEY (part\_id) REFERENCES Parts(part\_id),  interface NVARCHAR2(10),  memory INT,  memtype NVARCHAR2(10)  );  CREATE TABLE Mboard (  part\_id INT, FOREIGN KEY (part\_id) REFERENCES Parts(part\_id),  socket NVARCHAR2(20),  chipset NVARCHAR2(50),  memtype NVARCHAR2(10)  ); |

# Приложение Б. Листинг процедур базы данных

|  |
| --- |
| CREATE OR REPLACE procedure create\_customer (mail in nvarchar2,pass in nvarchar2)  is  begin  insert into customers(email,password) values(mail,pass);  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка' || sqlcode);  END;  CREATE OR REPLACE procedure create\_adress (  p\_email in adress.email%TYPE,p\_pass in customers.password%TYPE, p\_city in adress.city%TYPE, p\_street in adress.street%TYPE,  p\_house in adress.house%TYPE,p\_a\_floor in adress.a\_floor%TYPE,p\_appartment in adress.appartment%TYPE  )  is  rcount int;  begin  select count(\*) into rcount from customers where email=p\_email and password=p\_pass;  IF rcount = 1 THEN  insert into adress values(null, p\_email,p\_city,p\_street,p\_house,p\_a\_floor,p\_appartment);  dbms\_output.put\_line('ок');  ELSE  dbms\_output.put\_line('проверьте пароль или почту');  END IF;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  end;  CREATE OR REPLACE PROCEDURE make\_order (part\_id\_p IN INT, qty IN INT)  IS  before\_qty INT;  table\_stat SMALLINT;  BEGIN  SELECT p\_count, p\_status  INTO before\_qty, table\_stat  FROM parts  WHERE part\_id = part\_id\_p;    IF before\_qty >= qty THEN  INSERT INTO orders VALUES (11);  END IF;    IF table\_stat = 1 AND before\_qty < 1 THEN  DELETE FROM parts WHERE part\_id = part\_id\_p;  END IF;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END;  drop procedure make\_order;  select \* from user\_procedures;  --select \* from parts;  begin  create\_customer('ads','testpass');  end;  -- Процедура для добавления значений в таблицу parts  CREATE OR REPLACE PROCEDURE Add\_Part(  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  type IN parts.type%TYPE,  p\_status IN parts.p\_status%TYPE DEFAULT 0  )  IS  BEGIN  if p\_count < 0 then  null;  INSERT INTO parts (p\_model, manufacturer, price, p\_count, p\_description, type, p\_status)  VALUES (p\_model, manufacturer, price, p\_count, p\_description, type, p\_status);  COMMIT;  else  DBMS\_OUTPUT.PUT\_LINE('количество меньше нуля, откат.');  end if;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Add\_Part;  -- Процедура для добавления значений в таблицу, соответствующую типу товара CPU  CREATE OR REPLACE PROCEDURE Add\_CPU(  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  freq IN CPU.freq%TYPE,  cores IN CPU.cores%TYPE,  arch IN CPU.arch%TYPE,  socket IN CPU.socket%TYPE  )  IS  part\_id NUMBER;  BEGIN  Add\_Part(p\_model, manufacturer, price, p\_count, p\_description, 'CPU');  SELECT part\_id INTO part\_id FROM parts WHERE p\_model = p\_model AND type = 'CPU';  INSERT INTO CPU (part\_id, freq, cores, arch, socket)  VALUES (part\_id, freq, cores, arch, socket);  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Add\_CPU;  -- Процедура для добавления значений в таблицу, соответствующую типу товара GPU  CREATE OR REPLACE PROCEDURE Add\_GPU(  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  memtype IN GPU.memtype%TYPE,  memory IN GPU.memory%TYPE,  gpu\_chip IN GPU.gpu\_chip%TYPE  )  IS  part\_id NUMBER;  BEGIN  Add\_Part(p\_model, manufacturer, price, p\_count, p\_description, 'GPU');  SELECT part\_id INTO part\_id FROM parts WHERE p\_model = p\_model AND type = 'GPU';  INSERT INTO GPU (part\_id, memtype, memory, gpu\_chip)  VALUES (part\_id, memtype, memory, gpu\_chip);  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Add\_GPU;  -- Процедура для добавления значений в таблицу, соответствующую типу товара RAM  CREATE OR REPLACE PROCEDURE Add\_RAM(  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  memtype IN RAM.memtype%TYPE,  memory IN RAM.memory%TYPE,  freq IN RAM.freq%TYPE  )  IS  part\_id NUMBER;  BEGIN  Add\_Part(p\_model, manufacturer, price, p\_count, p\_description, 'RAM');  SELECT part\_id INTO part\_id FROM parts WHERE p\_model = p\_model AND type = 'RAM';  INSERT INTO RAM (part\_id, memtype, memory, freq)  VALUES (part\_id, memtype, memory, freq);  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Add\_RAM;  /  -- Процедура для добавления значений в таблицу, соответствующую типу товара Storage  CREATE OR REPLACE PROCEDURE Add\_Storage(  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  interface IN Storage.interface%TYPE,  memory IN Storage.memory%TYPE,  memtype IN Storage.memtype%TYPE  )  IS  part\_id NUMBER;  BEGIN  Add\_Part(p\_model, manufacturer, price, p\_count, p\_description, 'Storage');  SELECT part\_id INTO part\_id FROM parts WHERE p\_model = p\_model AND type = 'Storage';  INSERT INTO Storage (part\_id, interface, memory, memtype)  VALUES (part\_id, interface, memory, memtype);  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Add\_Storage;  -- Процедура для добавления значений в таблицу, соответствующую типу товара Mboard  CREATE OR REPLACE PROCEDURE Add\_Mboard(  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  socket IN Mboard.socket%TYPE,  chipset IN Mboard.chipset%TYPE,  memtype IN Mboard.memtype%TYPE  )  IS  part\_id NUMBER;  BEGIN  Add\_Part(p\_model, manufacturer, price, p\_count, p\_description, 'Mboard');  SELECT part\_id INTO part\_id FROM parts WHERE p\_model = p\_model AND type = 'Mboard';  INSERT INTO Mboard (part\_id, socket, chipset, memtype)  VALUES (part\_id, socket, chipset, memtype);  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Add\_Mboard;  -- Процедура для добавления значений в таблицу parts для типа товара "other"  CREATE OR REPLACE PROCEDURE Add\_Other\_Part(  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE  )  IS  BEGIN  Add\_Part(p\_model, manufacturer, price, p\_count, p\_description, 'other');  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Add\_Other\_Part;  select \* from orders;  CREATE OR REPLACE procedure make\_order (  p\_email in adress.email%TYPE,p\_pass in customers.password%TYPE, p\_adress in adress.adress\_id%TYPE,  ord\_date in date  )  is  rcount int;  begin  select count(\*) into rcount from customers where email=p\_email and password=p\_pass;  IF rcount = 1 THEN  insert into orders values(null, p\_email,p\_adress,ord\_date,0);  dbms\_output.put\_line('ок');  ELSE  dbms\_output.put\_line('проверьте пароль или почту');  END IF;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  end;  CREATE OR REPLACE procedure fill\_order (  p\_order\_id in order\_elems.order\_id%TYPE,p\_part\_id in order\_elems.part\_id%TYPE , p\_count in order\_elems.order\_id%TYPE)  is  begin  if p\_count >0 then  insert into order\_elems values(p\_order\_id,p\_part\_id,p\_count);  dbms\_output.put\_line('ok');  else  dbms\_output.put\_line('проверьте введеные данные');end if;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  end;  CREATE OR REPLACE procedure cut\_order (  p\_order\_id in order\_elems.order\_id%TYPE)  is  begin  delete order\_elems where order\_id = p\_order\_id;  EXCEPTION  WHEN OTHERS THEN  dbms\_output.put\_line('ошибка '||sqlcode);  end;  select \* from orders;  CREATE OR REPLACE procedure finish\_order (  p\_order\_id in orders.order\_id%TYPE  )  is  begin  update orders set ord\_status = 1 where p\_order\_id = order\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка '||sqlcode);  end;  CREATE OR REPLACE procedure unfinish\_order (  p\_order\_id in orders.order\_id%TYPE  )  is  begin  update orders set ord\_status = 0 where p\_order\_id = order\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка '||sqlcode);  end;  CREATE OR REPLACE PROCEDURE Update\_Mboard(  part\_id IN parts.part\_id%TYPE,  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  p\_status IN parts.p\_status%TYPE,  socket IN Mboard.socket%TYPE,  chipset IN Mboard.chipset%TYPE,  memtype IN Mboard.memtype%TYPE  )  IS  type\_var parts.type%TYPE;  BEGIN  -- Изменение типа товара и обновление данных  UPDATE parts  SET p\_model = p\_model,  manufacturer = manufacturer,  price = price,  p\_count = p\_count,  p\_description = p\_description,  p\_status = p\_status  WHERE part\_id = part\_id;        SELECT type INTO type\_var FROM parts WHERE part\_id = part\_id;  IF type\_var = 'Mboard' THEN  UPDATE Mboard  SET socket = socket,  chipset = chipset,  memtype = memtype  WHERE part\_id = part\_id;  Else  Delete\_Added\_data(part\_id);  INSERT INTO Mboard values(part\_id, socket, chipset, memtype);    UPDATE parts  set parts.type='Mboard';  END IF;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Update\_Mboard;  CREATE OR REPLACE PROCEDURE Delete\_Added\_data(  part\_id IN parts.part\_id%TYPE  )  IS  type\_var parts.type%TYPE;  BEGIN  -- Получение типа товара по part\_id  SELECT type INTO type\_var FROM parts WHERE part\_id = part\_id;    -- Удаление данных в зависимости от типа товара  CASE type\_var  WHEN 'CPU' THEN  DELETE FROM CPU WHERE part\_id = part\_id;  WHEN 'GPU' THEN  DELETE FROM GPU WHERE part\_id = part\_id;  WHEN 'RAM' THEN  DELETE FROM RAM WHERE part\_id = part\_id;  WHEN 'Storage' THEN  DELETE FROM Storage WHERE part\_id = part\_id;  WHEN 'Mboard' THEN  DELETE FROM Mboard WHERE part\_id = part\_id;  END CASE;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  COMMIT;  END Delete\_Added\_data;  create or replace procedure delete\_part(  part\_id IN parts.part\_id%TYPE)  is  begin  --Delete\_Added\_data (part\_id);  update parts set p\_status=1 where parts.part\_id = part\_id;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  end delete\_part;  create or replace procedure return\_part(  part\_id IN parts.part\_id%TYPE)  is  begin  update parts set p\_status=0 where parts.part\_id = part\_id;  end return\_part;  select \* from user\_procedures;  select \* from user\_tables;  -- Процедура для обновления данных товара типа CPU  CREATE OR REPLACE PROCEDURE Update\_CPU(  part\_id IN parts.part\_id%TYPE,  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  p\_status IN parts.p\_status%TYPE,  freq IN CPU.freq%TYPE,  cores IN CPU.cores%TYPE,  arch IN CPU.arch%TYPE,  socket IN CPU.socket%TYPE  )  IS  type\_var parts.type%TYPE;  BEGIN  -- Получение типа товара по part\_id  SELECT type INTO type\_var FROM parts WHERE part\_id = part\_id;    -- Обновление данных в соответствующей таблице и типе товара  IF type\_var = 'CPU' THEN  UPDATE CPU  SET freq = freq,  cores = cores,  arch = arch,  socket = socket  WHERE part\_id = part\_id;  ELSE  -- Вставка данных в таблицу CPU  INSERT INTO CPU (part\_id, freq, cores, arch, socket)  VALUES (part\_id, freq, cores, arch, socket);    -- Обновление типа товара в таблице parts  UPDATE parts  SET type = 'CPU'  WHERE part\_id = part\_id;  END IF;    -- Обновление общих данных товара в таблице parts  UPDATE parts  SET p\_model = p\_model,  manufacturer = manufacturer,  price = price,  p\_count = p\_count,  p\_description = p\_description,  p\_status = p\_status  WHERE part\_id = part\_id;    COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Update\_CPU;  -- Процедура для обновления данных товара типа GPU  CREATE OR REPLACE PROCEDURE Update\_GPU(  part\_id IN parts.part\_id%TYPE,  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  p\_status IN parts.p\_status%TYPE,  memtype IN GPU.memtype%TYPE,  memory IN GPU.memory%TYPE,  gpu\_chip IN GPU.gpu\_chip%TYPE  )  IS  type\_var parts.type%TYPE;  BEGIN  -- Получение типа товара по part\_id  SELECT type INTO type\_var FROM parts WHERE part\_id = part\_id;    -- Обновление данных в соответствующей таблице и типе товара  IF type\_var = 'GPU' THEN  UPDATE GPU  SET memtype = memtype,  memory = memory,  gpu\_chip = gpu\_chip  WHERE part\_id = part\_id;  ELSE  -- Вставка данных в таблицу GPU  INSERT INTO GPU (part\_id, memtype, memory, gpu\_chip)  VALUES (part\_id, memtype, memory, gpu\_chip);    -- Обновление типа товара в таблице parts  UPDATE parts  SET type = 'GPU'  WHERE part\_id = part\_id;  END IF;    -- Обновление общих данных товара в таблице parts  UPDATE parts  SET p\_model = p\_model,  manufacturer = manufacturer,  price = price,  p\_count = p\_count,  p\_description = p\_description,  p\_status = p\_status  WHERE part\_id = part\_id;    COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Update\_GPU;  -- Процедура для обновления данных товара типа RAM  CREATE OR REPLACE PROCEDURE Update\_RAM(  part\_id IN parts.part\_id%TYPE,  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  p\_status IN parts.p\_status%TYPE,  memtype IN RAM.memtype%TYPE,  memory IN RAM.memory%TYPE,  freq IN RAM.freq%TYPE  )  IS  type\_var parts.type%TYPE;  BEGIN  -- Получение типа товара по part\_id  SELECT type INTO type\_var FROM parts WHERE part\_id = part\_id;    -- Обновление данных в соответствующей таблице и типе товара  IF type\_var = 'RAM' THEN  UPDATE RAM  SET memtype = memtype,  memory = memory,  freq = freq  WHERE part\_id = part\_id;  ELSE  -- Вставка данных в таблицу RAM  INSERT INTO RAM (part\_id, memtype, memory, freq)  VALUES (part\_id, memtype, memory, freq);    -- Обновление типа товара в таблице parts  UPDATE parts  SET type = 'RAM'  WHERE part\_id = part\_id;  END IF;    -- Обновление общих данных товара в таблице parts  UPDATE parts  SET p\_model = p\_model,  manufacturer = manufacturer,  price = price,  p\_count = p\_count,  p\_description = p\_description,  p\_status = p\_status  WHERE part\_id = part\_id;    COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Update\_RAM;  -- Процедура для обновления данных товара типа Storage  CREATE OR REPLACE PROCEDURE Update\_Storage(  part\_id IN parts.part\_id%TYPE,  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  p\_status IN parts.p\_status%TYPE,  interface IN Storage.interface%TYPE,  memory IN Storage.memory%TYPE,  memtype IN Storage.memtype%TYPE  )  IS  type\_var parts.type%TYPE;  BEGIN  -- Получение типа товара по part\_id  SELECT type INTO type\_var FROM parts WHERE part\_id = part\_id;    -- Обновление данных в соответствующей таблице и типе товара  IF type\_var = 'Storage' THEN  UPDATE Storage  SET interface = interface,  memory = memory,  memtype = memtype  WHERE part\_id = part\_id;  ELSE  -- Вставка данных в таблицу Storage  INSERT INTO Storage (part\_id, interface, memory, memtype)  VALUES (part\_id, interface, memory, memtype);    -- Обновление типа товара в таблице parts  UPDATE parts  SET type = 'Storage'  WHERE part\_id = part\_id;  END IF;    -- Обновление общих данных товара в таблице parts  UPDATE parts  SET p\_model = p\_model,  manufacturer = manufacturer,  price = price,  p\_count = p\_count,  p\_description = p\_description,  p\_status = p\_status  WHERE part\_id = part\_id;    COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Update\_Storage;  CREATE OR REPLACE PROCEDURE Update\_Other(  part\_id IN parts.part\_id%TYPE,  p\_model IN parts.p\_model%TYPE,  manufacturer IN parts.manufacturer%TYPE,  price IN parts.price%TYPE,  p\_count IN parts.p\_count%TYPE,  p\_description IN parts.p\_description%TYPE,  p\_status IN parts.p\_status%TYPE,  socket IN Mboard.socket%TYPE,  chipset IN Mboard.chipset%TYPE,  memtype IN Mboard.memtype%TYPE  )  IS  type\_var parts.type%TYPE;  BEGIN  -- Изменение типа товара и обновление данных  SELECT type INTO type\_var FROM parts WHERE part\_id = part\_id;  IF type\_var = 'other' THEN  UPDATE Mboard  SET socket = socket,  chipset = chipset,  memtype = memtype  WHERE part\_id = part\_id;  Else  INSERT INTO Mboard values(part\_id, socket, chipset, memtype);  UPDATE parts  set parts.type='Mboard';  END IF;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  rollback;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('ошибка'||sqlcode);  END Update\_Other; |

# Приложение В. Листинг функций базы данных

|  |
| --- |
| CREATE OR REPLACE FUNCTION show\_orders\_cust (p\_email IN customers.password%TYPE, p\_pass IN customers.password%TYPE) RETURN sys\_refcursor  IS  ord\_cursor sys\_refcursor;  rcount INT;  BEGIN  SELECT COUNT(\*)  INTO rcount  FROM customers  WHERE customers.email = p\_email  AND customers.password = p\_pass;  IF rcount = 1 THEN  OPEN ord\_cursor FOR  SELECT ord.order\_id, email, adress\_id, elem.part\_id, ord\_date, ord\_status, count  FROM orders ord  JOIN order\_elems elem ON ord.order\_id = elem.order\_id  WHERE email = p\_email;  RETURN ord\_cursor;  ELSE  DBMS\_OUTPUT.PUT\_LINE('проверьте данные.');  RETURN NULL;  END IF;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка '||SQLCODE);  RETURN NULL;  END;  CREATE OR REPLACE FUNCTION show\_adress\_cust (  p\_email in customers.password%TYPE,  p\_pass in customers.password%TYPE) RETURN sys\_refcursor  is  ord\_cursor sys\_refcursor;  rcount int;  begin  select count(\*) into rcount from customers where customers.email = p\_email and customers.password = p\_pass;  if rcount = 1 then  open ord\_cursor for  select \* from adress where email = p\_email;  return ord\_cursor;  else  DBMS\_OUTPUT.PUT\_LINE('проверьте данные.');  return NULL;  end if;  EXCEPTION  WHEN OTHERS THEN  dbms\_output.put\_line('ошибка '||sqlcode);  end;  CREATE or replace FUNCTION count\_money(order\_id VARCHAR) RETURN NUMBER IS  total\_cost NUMBER;  BEGIN  BEGIN  SELECT SUM(count \* (select price from parts where part\_id=oe.part\_id)) INTO total\_cost  FROM order\_elems oe;    IF total\_cost IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Ничего не найдено');  END IF;    EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Ничего не найдено');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Ошибка'||sqlcode);  END;  RETURN total\_cost;  END; |

# Приложение Г. Листинг представлений базы данных

|  |
| --- |
| CREATE OR REPLACE VIEW Parts\_V AS  SELECT \*  FROM parts;  CREATE OR REPLACE VIEW CPU\_V AS  SELECT P.part\_id, P.p\_model, P.manufacturer, P.price, P.p\_count, P.p\_description,  C.freq, C.cores, C.arch, C.socket  FROM parts P  JOIN CPU C ON P.part\_id = C.part\_id;  -- Представление для таблицы GPU  CREATE OR REPLACE VIEW GPU\_V AS  SELECT P.part\_id, P.p\_model, P.manufacturer, P.price, P.p\_count, P.p\_description,  G.memtype, G.memory, G.gpu\_chip  FROM parts P  JOIN GPU G ON P.part\_id = G.part\_id;  -- Представление для таблицы RAM  CREATE OR REPLACE VIEW RAM\_V AS  SELECT P.part\_id, P.p\_model, P.manufacturer, P.price, P.p\_count, P.p\_description,  R.memtype, R.memory, R.freq  FROM parts P  JOIN RAM R ON P.part\_id = R.part\_id;  -- Представление для таблицы Storage  CREATE OR REPLACE VIEW Storage\_V AS  SELECT P.part\_id, P.p\_model, P.manufacturer, P.price, P.p\_count, P.p\_description,  S.interface, S.memory, S.memtype  FROM parts P  JOIN Storage S ON P.part\_id = S.part\_id;  CREATE OR REPLACE VIEW Mboard\_V AS  SELECT P.part\_id, P.p\_model, P.manufacturer, P.price, P.p\_count, P.p\_description,  M.socket, M.chipset, M.memtype  FROM parts P  JOIN Mboard M ON P.part\_id = M.part\_id;  CREATE OR REPLACE VIEW adress\_V AS  SELECT \*  FROM adress;  CREATE OR REPLACE VIEW orders\_V AS  SELECT \*  FROM orders;  CREATE OR REPLACE VIEW orders\_in\_elems\_V AS  SELECT ord.order\_id,email,adress\_id,elem.part\_id,ord\_date,ord\_status,count  FROM orders ord join order\_elems elem  on ord.order\_id = elem.order\_id;  CREATE OR REPLACE VIEW usermails AS  SELECT email from customers; |

# Приложение Д. Листинг кода создания пользователей

|  |
| --- |
| create role customer\_role not identified;  CREATE PROFILE pf\_customer LIMIT  PASSWORD\_LIFE\_TIME 640  SESSIONS\_PER\_USER 5000  FAILED\_LOGIN\_ATTEMPTS 7  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 10  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 180  IDLE\_TIME 30;  Create user customer identified by 111  DEFault tablespace shopts  temporary tablespace shopts\_tmp  profile pf\_customer  account unlock;  password expire;  grant select on Parts\_V to customer\_role;  grant select on gpu\_V to customer\_role;  grant select on cpu\_V to customer\_role;  grant select on storage\_V to customer\_role;  grant select on ram\_V to customer\_role;  grant select on mboard\_V to customer\_role;  GRANt execute on create\_customer to customer\_role;  GRANt execute on make\_order to customer\_role;  GRANt execute on cut\_order\_cust to customer\_role;  GRANt execute on fill\_order\_cust to customer\_role;  GRANt execute on create\_adress to customer\_role;  create role salesman\_role not identified;  CREATE PROFILE pf\_salesman LIMIT  PASSWORD\_LIFE\_TIME 640  SESSIONS\_PER\_USER 200  FAILED\_LOGIN\_ATTEMPTS 7  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 10  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 180  IDLE\_TIME 30;  Create user salesman identified by 111  DEFault tablespace shopts  temporary tablespace shopts\_tmp  profile pf\_salesman  account unlock;  password expire;  Grant create session to salesman\_role;  Grant salesman\_role to salesman;  GRANt execute on UNFINISH\_ORDER to salesman\_role;  GRANt execute on FINISH\_ORDER to salesman\_role;  GRANt execute on update\_cpu to salesman\_role;  GRANt execute on update\_gpu to salesman\_role;  GRANt execute on update\_storage to salesman\_role;  GRANt execute on update\_ram to salesman\_role;  GRANt execute on update\_other to salesman\_role;  GRANt execute on update\_mboard to salesman\_role;  GRANt execute on add\_cpu to salesman\_role;  GRANt execute on add\_gpu to salesman\_role;  GRANt execute on add\_storage to salesman\_role;  GRANt execute on add\_ram to salesman\_role;  GRANt execute on add\_other\_part to salesman\_role;  GRANt execute on add\_mboard to salesman\_role;  GRANt execute on delete\_part to salesman\_role;  GRANt execute on return\_part to salesman\_role;  GRANt execute on cut\_order to salesman\_role;  GRANt execute on fill\_order to salesman\_role;  GRANt execute on count\_money to salesman\_role;  GRANt execute on return\_part to salesman\_role;  GRANt execute on cut\_order to salesman\_role;  GRANt execute on fill\_order to salesman\_role;  grant select on Parts\_V to salesman\_role;  grant select on gpu\_V to salesman\_role;  grant select on cpu\_V to salesman\_role;  grant select on storage\_V to salesman\_role;  grant select on ram\_V to salesman\_role;  grant select on mboard\_V to salesman\_role;  grant select on adress\_V to salesman\_role;  grant select on orders\_in\_elems\_V to salesman\_role;  grant select on adress\_V to salesman\_role;  grant execute on edit\_part\_count to salesman\_role;  grant select on orders\_V to salesman\_role;  create role salesman\_role not identified;  CREATE PROFILE pf\_salesman LIMIT  PASSWORD\_LIFE\_TIME 640  SESSIONS\_PER\_USER 200  FAILED\_LOGIN\_ATTEMPTS 7  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 10  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 180  IDLE\_TIME 30;  Create user salesman identified by 111  DEFault tablespace shopts  temporary tablespace shopts\_tmp  profile pf\_salesman  account unlock;  password expire; |

# Приложение Е. Листинг процедур для импорта и экспорта в xml

|  |
| --- |
| CREATE OR REPLACE PROCEDURE GenerateAddressXML IS  xml\_data CLOB;  BEGIN  SELECT XMLELEMENT("Adress",  XMLFOREST(adress\_id AS "AddressID",email AS "Email",  city AS "City",street AS "Street", house AS "House", a\_floor AS "Floor", appartment AS "Apartment")).getClobVal()  INTO xml\_data  FROM adress;  DBMS\_OUTPUT.PUT\_LINE(xml\_data);  END;  CREATE OR REPLACE Procedure import\_xml  (xml\_text in nvarchar2) is  v\_xml XMLType;  BEGIN  -- Загружаем XML документ  v\_xml := XMLType(xml\_text);  INSERT INTO adress (adress\_id, email, city, street, house, a\_floor, appartment)  VALUES (null,  v\_xml.extract('/Address/Email/text()').getStringVal(),  v\_xml.extract('/Address/City/text()').getStringVal(),  v\_xml.extract('/Address/Street/text()').getStringVal(),  v\_xml.extract('/Address/House/text()').getStringVal(),  v\_xml.extract('/Address/Floor/text()').getNumberVal(),  v\_xml.extract('/Address/Apartment/text()').getNumberVal());  COMMIT;  EXCEPTION  WHEN VALUE\_ERROR THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Ошибка в значениях данных.');  WHEN NO\_DATA\_FOUND THEN  -- Обработка ошибок отсутствия данных  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Не удалось найти необходимые данные в XML.');  WHEN DUP\_VAL\_ON\_INDEX THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Нарушение уникального ограничения. Дублирующиеся значения.');  WHEN OTHERS THEN  -- Обработка других исключений  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка во время выполнения операции.');  END;  END; |