МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных аукционной площадки с использованием средств диагностики»

**Исполнитель**

студент 2 курса 3 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. Г. Логутенко

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

подпись дата инициалы и фамилия

Минск 2022

**Содержание**

[Введение 4](#_Toc133712800)

[1 Анализ требований к программному средству 5](#_Toc133712801)

[1.1 Аналитический обзор аналогов 5](#_Toc133712802)

[1.1.1 Аналог torgi.gov.by 5](#_Toc133712803)

[1.1.2 Аналог ipmtorgi.by 6](#_Toc133712804)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc133712805)

[1.3 Вывод 9](#_Toc133712807)

[2 Разработка архитектуры проекта 10](#_Toc133712808)

[2.1 Обобщенная структура управлением приложения 10](#_Toc133712809)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 10](#_Toc133712810)

[3 Разработка модели базы данных 13](#_Toc133712811)

[3.1 Создание необходимых объектов 13](#_Toc133712812)

[3.1.1 Представления базы данных 14](#_Toc133712813)

[3.1.2 Индексы базы данных 14](#_Toc133712814)

[3.1.3 Триггеры базы данных 15](#_Toc133712815)

[3.2 Описание используемой технологии 16](#_Toc133712816)

[3.3 Вывод 17](#_Toc133712817)

[4 Установка, настройка и использование Oracle 12c 19](#_Toc133712818)

[4.1 Установка Oracle 12c 19](#_Toc133712819)

[4.2 Создание таблиц 19](#_Toc133712820)

[4.3 Создание ролей для разграничения 20](#_Toc133712821)

[4.4 Создание пакетов процедур для базы данных 22](#_Toc133712822)

[4.4.1 Выборка данных из таблиц 23](#_Toc133712823)

[4.4.3 Заполнение таблиц 100 000 строк 25](#_Toc133712825)

[4.4.4 Добавление данных в таблицы 25](#_Toc133712826)

[4.4.5 Удаление данных в таблицы 27](#_Toc133712827)

[4.4.6 Изменение данных в таблицы 28](#_Toc133712828)

[4.4.7 Дополнительные функции 29](#_Toc133712829)

[4.5 Описание процедур экспорта и импорта 30](#_Toc133712830)

[4.6 Вывод 33](#_Toc133712831)

[5 Тестирование 34](#_Toc133712832)

[5.1 Тестирование производительности базы данных 34](#_Toc133712833)

[5.2 Вывод 35](#_Toc133712834)

[Заключение 42](#_Toc133712839)

[Список литературных источников 43](#_Toc133712840)

[Приложение А 44](#_Toc133712841)

# Введение

Современный рынок электронной коммерции, в том числе и онлайн-аукционы, предоставляют бизнесу новые возможности для продвижения товаров и услуг, а также для получения прибыли. Для успешной работы аукционных площадок необходимо иметь надежную и эффективную базу данных, способную обрабатывать большие объемы информации и обеспечивать быстрый доступ к ней. В то же время, любые сбои в работе базы данных могут стать причиной значительных финансовых потерь и потери доверия со стороны клиентов.

Данная курсовая работа посвящена реализации базы данных аукционной площадки с использованием средств диагностики, которые помогут повысить ее надежность и эффективность. В процессе работы будут использоваться средства Oracle, одной из наиболее распространенных систем управления базами данных.

Целью данной курсовой работы является создание базы данных аукционной площадки, которая будет эффективной и надежной. Работа направлена на улучшение процесса продажи и покупки товаров на аукционных площадках в Интернете.

Основными задачами является реализация следующих функций базы данных:

* Управление информацией о текущем ассортименте площадки.
* Занесение данных о покупках.
* Управление данными о сотрудниках компании.
* Организация данных об аукционах, проведённых площадкой.
* Хранение в базе данных ролей сотрудников.

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

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Аукционные площадки являются важной частью электронной коммерции и предоставляют пользователям возможность продажи и покупки товаров и услуг в Интернете. Они являются аналогом традиционных аукционов, но с использованием современных технологий, таких как онлайн-платежи и быстрые коммуникации. Существует множество аукционных площадок, которые предоставляют широкий спектр услуг для пользователей, включая возможность поиска и сравнения товаров, участие в аукционах, управление своими лотами и т.д.

Одним из основных преимуществ аукционных площадок является возможность участия в аукционах из любой точки мира с помощью Интернета, что делает их очень удобными для пользователей. Также аукционные площадки позволяют продавцам быстро продавать свои товары и получать за них высокую цену, а покупателям - находить интересующие их товары по доступной цене.

### 1.1.1 Аналог torgi.gov.by

Torgi.gov.by - это государственный сервис, который позволяет проводить электронные торги в режиме онлайн на территории Республики Беларусь. Этот сервис позволяет продавцам продавать свои товары и услуги, а также покупателям приобретать нужные им товары и услуги, используя интернет-платформу. Пример интерфейса данного сервиса представлен на рисунке 1.1.

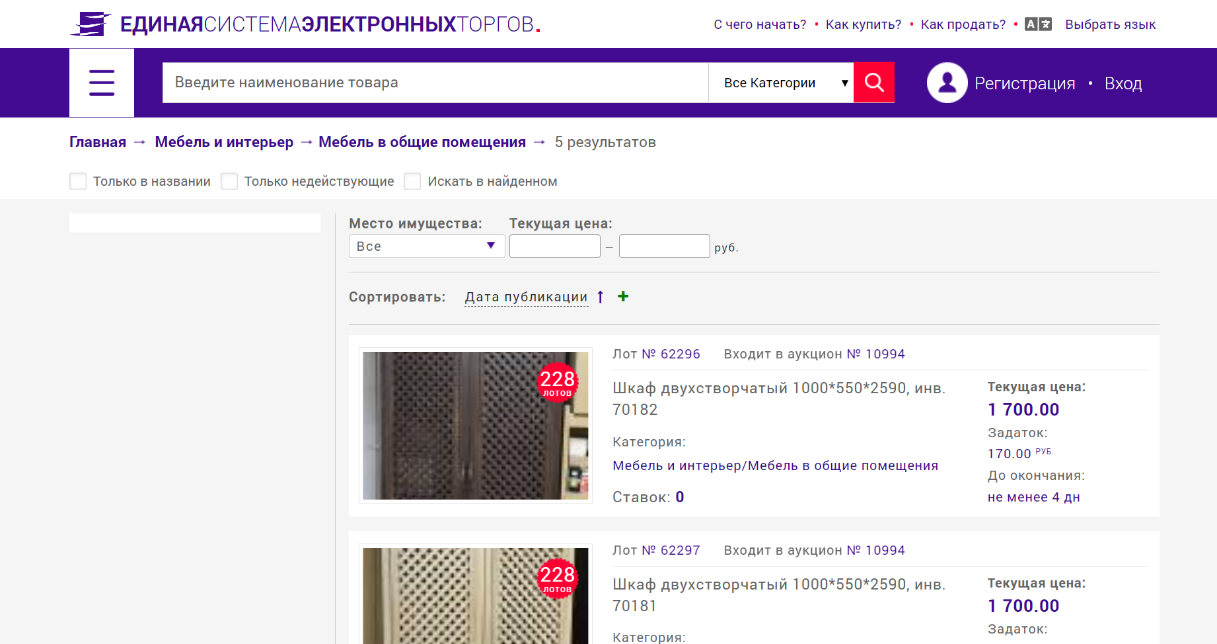


Рисунок 1.1 – Интерфейс сервиса torgi.gov.by

Основная цель сервиса torgi.gov.by - ускорить и упростить процесс проведения торгов, сделать его более прозрачным и безопасным для всех участников. С помощью данного сервиса можно проводить торги на различные категории товаров и услуг, такие как недвижимость, автотранспорт, оборудование, электроника, строительные материалы и многие другие.

Одной из главных преимуществ сервиса является его доступность и удобство использования. Участники торгов могут легко зарегистрироваться на сайте torgi.gov.by, разместить свои лоты и участвовать в торгах онлайн. Сервис также обеспечивает высокую степень безопасности и конфиденциальности для всех участников торгов.

Таким образом, сервис torgi.gov.by представляет собой эффективный и удобный инструмент для проведения электронных торгов, который помогает ускорить и упростить процесс покупки и продажи товаров и услуг.

### 1.1.2 Аналог ipmtorgi.by

Ipmtorgi.by - это официальный интернет-портал ОАО «Инвестпромбанк», который предоставляет информацию о проводимых торгах по продаже имущества банка.

Сервис ipmtorgi.by является эффективным инструментом для поиска и приобретения различных видов имущества банка: зданий, сооружений, транспорта, оборудования, а также объектов недвижимости. На портале размещаются информация о проведении торгов, датах и времени проведения, виды имущества, начальная цена, условия участия в торгах, а также контакты организатора. Пример интерфейса данного сервиса представлен на рисунке 1.2.

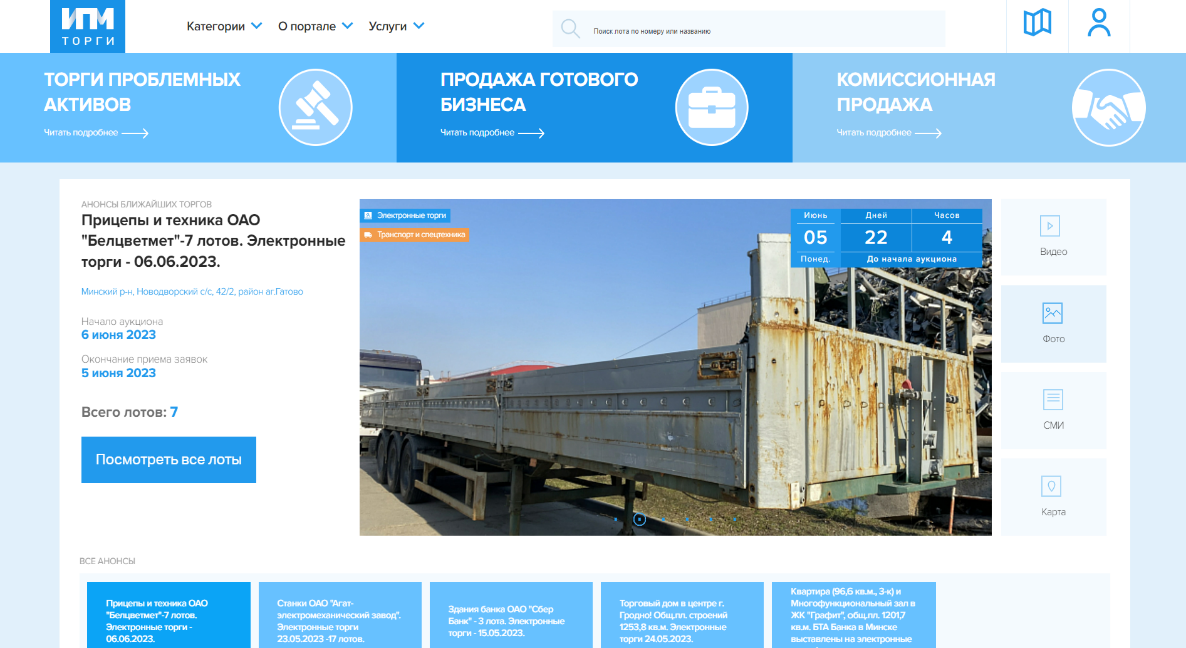


Рисунок 1.2 – Интерфейс сервиса ipmtorgi.by

Для участия в торгах на ipmtorgi.by необходимо зарегистрироваться на портале и подать заявку на участие в торгах. После подачи заявки участнику торгов выдается уникальный номер и он получает доступ к личному кабинету, где он может узнать информацию о статусе своей заявки, получать уведомления об изменениях в расписании торгов и получать полную информацию о проводимых торгах.

Кроме того, сервис ipmtorgi.by предоставляет возможность получить консультацию по вопросам, связанным с участием в торгах, а также предоставляет информацию о возможностях кредитования для приобретения имущества банка на выгодных условиях.

## 1.2 Разработка функциональных требований, определение вариантов использования

Проект предполагает создание базы данных для аукционной площадки с использованием средств диагностики в СУБД Oracle 12c. База данных должна быть реализована с помощью соответствующих процедур для доступа к данным.

Важной частью проекта является обеспечение безопасности базы данных и защита ее от несанкционированного доступа. Для этого необходимо использовать механизмы аутентификации и авторизации, а также применить ролевую модель доступа к данным. Кроме того, в базе данных должна быть реализована система резервного копирования и восстановления данных для предотвращения потери информации в случае сбоя системы или других непредвиденных событий.

Важным требованием к базе данных является импорт данных из XML файлов и экспорт данных в формат XML. Также необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости.

Для подробного описания применяемых системных пакетов, утилит и технологий в базе данных проведен анализ функций пользователей и их ролей в системе.

В проекте присутствуют три вида пользователей: менеджер (manager), продавец (seller) и покупатель (customer). Для каждого пользователя определены их роль и функции в системе, а также созданы соответствующие процедуры и запросы для доступа к данным.

Для более наглядного представления проекта и описания ролей пользователей можно использовать диаграмму UML. Например, диаграмма вариантов использования (use case diagram), представленная на рисунке 1.2, может помочь определить функциональные требования к базе данных для каждого пользователя. На диаграмме определены основные действия, которые пользователи выполняют в системе, и связи между ними.

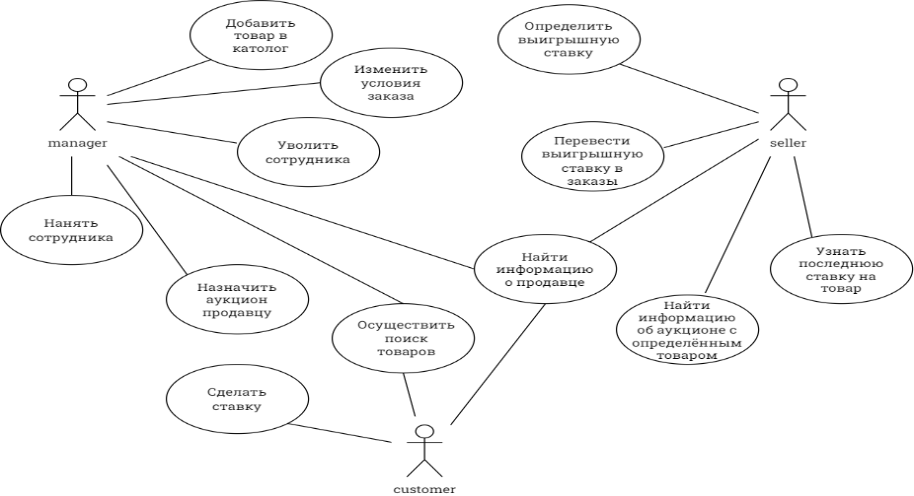


Рисунок 1.2 – Диаграмма вариантов использования базы данных аукционной площадки.

Как видно на диаграмме, у каждого пользователя есть свои функции в системе. Customer может искать товары, сделать ставку на товар, а также найти информацию о продавце. Seller может определить выигрышную ставку, переместить выигрышную ставку в заказы, найти информацию о продавце, найти информацию об аукционе с определенным товаром, а также узнать последнюю ставку на товар. Manager может добавить товар в каталог, изменить условия заказа, уволить или нанять сотрудника, а также осуществлять поиск товаров и найти информацию о продавце.

Таким образом, для успешной реализации проекта проведен анализ функций пользователей и их ролей в системе, а также использована UML диаграммы для наглядного представления проекта и его структуры.

## 1.3 Вывод

Итак, в результате проведенного аналитического обзора аналогов аукционных площадок и сервисов были определены ключевые характеристики и функциональные возможности, которые должны быть реализованы в нашей системе. Были выявлены требования к базе данных, включая импорт и экспорт данных в формате XML, а также проведено тестирование производительности на таблице с более чем 100 000 строками. Также были определены роли пользователей и функциональные требования к системе в зависимости от этих ролей, и для этого была разработана UML-диаграмма вариантов использования системы. В целом, разработка базы данных для аукционной площадки с использованием средств диагностики является важным шагом для обеспечения эффективной работы и удобства использования для всех ее пользователей.

1. Разработка архитектуры проекта

## 2.1 Обобщенная структура управлением приложения

Для обеспечения управления аукционной площадкой с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователям взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также механизмов для извлечения и обработки информации из базы данных.

В функциональность аукционной площадки может входить возможность быстрого и удобного поиска лотов, продавцов и покупателей по различным параметрам, таким как категория, цена, состояние товара и т.д. Также пользователи должны иметь возможность ставить ставки на лоты на сайте без необходимости установки специального программного обеспечения. Для удобства пользователей также должна быть предусмотрена возможность создания собственных списков отслеживания лотов и настройки уведомлений о новых лотах в соответствии с их интересами.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграммы UML - это унифицированный язык моделирования, который позволяет разработчикам и аналитикам описывать структуру, поведение и взаимодействие компонентов системы. В контексте разработки базы данных для аукционной площадки, использование диаграмм UML позволяет детально определить связи между всеми компонентами системы и обеспечить ее эффективное функционирование. В данном разделе будет рассмотрено применение диаграмм UML для описания взаимосвязи всех компонентов базы данных аукционной площадки. В диаграмме, представленной на рисунке 2.1, описаны таблицы, образующие базу данных аукционной площадки. Диаграмма таблиц помогает понять, каким образом компоненты взаимодействуют друг с другом и какие данные они обмениваются.

Для удобства визуализации структуры базы данных была создана диаграмма данных, которая отображает все таблицы и связи между ними. Диаграмма позволяет легко ориентироваться в структуре базы данных и понимать, какие данные хранятся в каждой таблице, а также как они связаны друг с другом. Такая диаграмма является важным инструментом для разработки, тестирования и сопровождения базы данных. Для аукционной площадки важно отобразить все основные таблицы, которые отвечают за хранение данных о продуктах, заказах, клиентах, категориях, сотрудниках, аукционах, аукционных лотах и ставках. Кроме того, важно отразить связи между этими таблицами, чтобы можно было понять, как они взаимодействуют друг с другом. В целом, диаграмма должна показать всю структуру базы данных, отражающую процессы и взаимодействие всех основных сущностей аукционной площадки. Кроме того, диаграмма данных должна также отражать ограничения и свойства таблиц, такие как первичные и внешние ключи, ограничения целостности данных и другие ограничения, которые могут быть установлены для каждой таблицы. Это поможет разработчикам и администраторам баз данных легко понимать, какие ограничения установлены и как они взаимодействуют друг с другом.

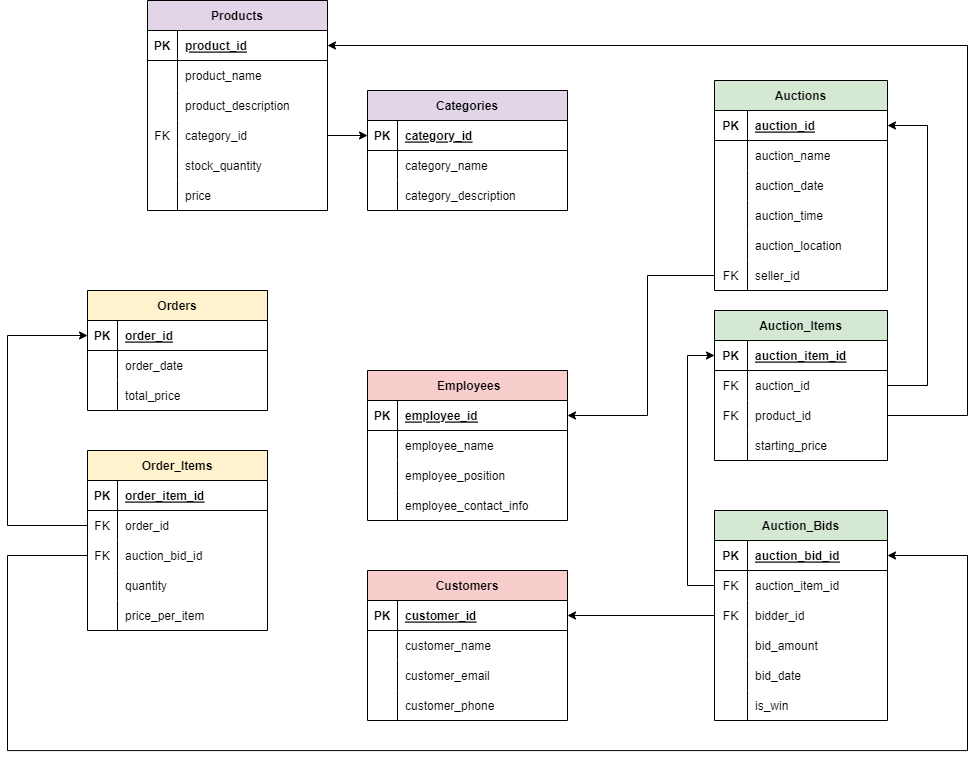


Рисунок 2.1 – Диаграмма базы данных аукционной площадки

Таким образом, диаграмма таблиц для аукционной площадки отражает структуру базы данных, предназначенной для хранения информации о товарах, категориях, покупателях, продавцах, аукционах и заявках на участие в них. Она помогает визуализировать связи между данными, хранящимися в разных таблицах, и понять, как они взаимодействуют друг с другом.

**2.3 Описание информационных объектов**

Данная база данных описывает работу аукционной площадки и состоит из 8 таблиц: Products, Orders, Order\_Items, Customers, Categories, Employees, Auctions, Auction\_Items и Auction\_Bids. Каждая из таблиц содержит информацию, необходимую для эффективного функционирования площадки, включая данные о продуктах, заказах, клиентах, категориях товаров, сотрудниках и аукционах. С помощью этой базы данных администраторы могут отслеживать процесс торговли и управлять заказами, клиентами и товарами на платформе.

Таблица "Products" является центральной таблицей в базе данных, которая хранит информацию о продуктах, которые продает компания. Она содержит данные о наименовании продукта, его цене, количестве на складе и описании. Данная таблица состоит из следующих столбцов:

* product\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор товара
* product\_name (тип: VARCHAR2(50)) - название товара
* product\_description (тип: VARCHAR2(200)) - описание товара
* category\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Categories(category\_id)) - идентификатор категории товара, связь с таблицей Categories
* stock\_quantity (тип: NUMBER) - количество товара на складе
* price (тип: NUMBER) - цена товара

Таблица "Orders" содержит информацию о заказах, размещенных покупателями. Она включает в себя информацию о дате заказа, идентификаторе покупателя, общей стоимости заказа и статусе заказа. Данная таблица состоит из следующих столбцов:

* order\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор заказа
* order\_date (тип: DATE) - дата заказа
* total\_price (тип: NUMBER) - общая цена заказа

Таблица "Order\_Items" содержит информацию о продуктах, включенных в каждый заказ. Она связана с таблицами "Orders" и "Products" и содержит информацию о количестве каждого продукта, цене и общей стоимости для каждого продукта в заказе. Данная таблица состоит из следующих столбцов:

* order\_item\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор позиции в заказе
* order\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Orders(order\_id)) - идентификатор заказа, связь с таблицей Orders
* auction\_bid\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Auction\_Bids(auction\_bid\_id)) - идентификатор заявки на аукцион, связь с таблицей Auction\_Bids
* quantity (тип: NUMBER) - количество товара в заказе
* price\_per\_item (тип: NUMBER) - цена за единицу товара

Таблица "Customers" содержит информацию о покупателях, включая их идентификатор, имя, адрес электронной почты и телефонный номер. Данная таблица состоит из следующих столбцов:

* customer\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор покупателя
* customer\_name (тип: VARCHAR2(50)) - имя покупателя
* customer\_email (тип: VARCHAR2(50)) - email покупателя
* customer\_phone (тип: VARCHAR2(20)) - телефон покупателя

Таблица "Categories" содержит информацию о категориях товаров, включая их идентификатор, название и описание. Данная таблица состоит из следующих столбцов:

* category\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор категории
* category\_name (тип: VARCHAR2(50)) - имя категории
* category\_description (тип: VARCHAR2(200)) - описание категории

Таблица "Employees" содержит информацию о сотрудниках, включая их идентификатор, имя, должность и контактную информацию. Данная таблица состоит из следующих столбцов:

* employee\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор сотрудника
* employee\_name (тип: VARCHAR2(50)) - имя сотрудника
* employee\_position (тип: VARCHAR2(50)) - должность сотрудника. Значения должны быть "seller" или "manager"
* employee\_contact\_info (тип: VARCHAR2(200)) - контактная информация о сотруднике

Таблица "Auctions" содержит информацию об аукционах, включая их идентификатор, название, дату, время, местоположение и идентификатор продавца. Данная таблица состоит из следующих столбцов:

* auction\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор аукциона
* auction\_name (тип: VARCHAR2(50)) - название аукциона
* auction\_date (тип: DATE) - дата проведения аукциона
* auction\_time (тип: VARCHAR2(20)) - время проведения аукциона
* auction\_location (тип: VARCHAR2(50)) - место проведения аукциона
* seller\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Employees(employee\_id)) - идентификатор продавца

Таблица "Auction\_Items" содержит информацию об элементах, продаваемых на аукционах, включая идентификатор элемента, идентификатор аукциона, идентификатор продукта и начальную цену. Данная таблица состоит из следующих столбцов:

* auction\_item\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор товара на аукционе
* auction\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Auctions(auction\_id)) - идентификатор аукциона, на котором продается товар
* product\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Products(product\_id)) - идентификатор продукта, который продается на аукционе
* starting\_price (тип: NUMBER) - начальная цена товара на аукционе

Таблица "Auction\_Bids" содержит информацию о ставках на аукционе, включая идентификатор ставки, идентификатор элемента аукциона, идентификатор покупателя, сумму ставки, дату и время ставки и флаг, указывающий, является ли ставка победившей. Данная таблица состоит из следующих столбцов:

* auction\_bid\_id (тип: NUMBER, ограничение: PRIMARY KEY) - идентификатор заявки на участие в аукционе
* auction\_item\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Auction\_Items(auction\_item\_id)) - идентификатор товара, на который была сделана заявка
* bidder\_id (тип: NUMBER, ограничение: FOREIGN KEY REFERENCES Customers(customer\_id)) - идентификатор покупателя, который сделал заявку
* bid\_amount (тип: NUMBER) - сумма заявки
* bid\_date (тип: DATE) - дата и время, когда была сделана заявка
* is\_win (тип: BOOLEAN) - указывает, является ли заявка выигрышной.

Таким образом, база данных аукционной площадки содержит информацию о продуктах, заказах, клиентах, категориях, сотрудниках, аукционах и ставках на аукционах. Эти таблицы связаны между собой и обеспечивают хранение, обработку и управление данными, связанными с работой аукционной площадки.

**2.4 Вывод**

Разработка архитектуры проекта является важным этапом для создания функционального и эффективного приложения. Описание информационных объектов является неотъемлемой частью этого процесса, так как позволяет определить, какие данные будут использоваться в приложении, и как они будут храниться и обрабатываться. В данной реализации базы данных аукционной площадки были описаны объекты, такие как продукты, заказы, категории, клиенты, сотрудники, аукционы и ставки, что помогло определить структуру и функциональность приложения. Средства диагностики были использованы для обеспечения надежности и безопасности базы данных.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

Для аукционной площадки необходимо создать несколько таблиц, которые будут хранить данные о пользователях, сотрудниках, товарах, заказах и аукционах.

Для музыкальной площадки необходимо создать следующие таблицы:

Product - таблица, содержащая список продуктов на складе и их количество. В этой таблице будут храниться название товаров в наличии на складе, которые могут быть выставлены на аукцион в качестве лота.

Orders - таблица, содержащая данные о заказах. Заказом в контексте данной площадки можно считать выигрышную ставку на товар, которая переходит в статус продаваемой. В этой таблице будут храниться информация о пользователе, такая как имя, фотография, электронная почта, дата рождения и т.д.

Order\_Items – таблица, содержащая информацию о товарах в заказе: идентификатор товара в заказе, идентификатор заказа, идентификатор заявки на участие в аукционе, количество товаров и цену за единицу товара.

Categories – таблица, содержащая информацию о категориях товаров: идентификатор категории, название и описание.

Employees – таблица, содержащая информацию о сотрудниках: идентификатор сотрудника, имя, должность (продавец или менеджер) и контактную информацию.

Customers - таблица, содержащая информацию об оценках пользователей музыкальных треков. В этой таблице будут храниться данные об оценках, которые пользователи выставляют трекам.

Auctions – таблица, содержащая информацию об аукционах: идентификатор аукциона, название, дату, время, местоположение и идентификатор продавца.

Auction\_Items – таблица, содержащая информацию о товарах, проданных на аукционах: идентификатор товара на аукционе, идентификатор аукциона, идентификатор товара, начальную цену.

Auction\_Bids – таблица, содержащая информацию о заявках на участие в аукционах: идентификатор заявки на участие в аукционе, идентификатор товара на аукционе, идентификатор покупателя, сумму ставки, дату ставки и флаг, указывающий, является ли заявка выигранной.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Например, создать индексы на столбцах customer\_email в таблице Customers, чтобы быстро находить информацию о клиенте по его электронной почте.

### 3.1.1 Представления базы данных

Представление (view) в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно, при этом не изменяя структуру этих таблиц.

В данном проекте были созданы три представления:

* Auction\_Bids\_Info, которое содержит информацию о ставках;
* Products\_Inventory, которое содержит информацию о продуктах и их колличестве;
* Auction\_Sellers, которое содержит информацию о продавцах и товарах, которые они выставили на аукцион.

Данное представление с названием Auction\_Bids\_Info объединяет данные из нескольких таблиц в одной таблице, чтобы получить информацию о заявках на участие в аукционе.Оно будет на листинге 3.1. Остальные представления будут аналогичны, только будут работать с другими таблицами.

|  |
| --- |
| CREATE VIEW Auction\_Bids\_Info AS  SELECT ab.auction\_bid\_id, ai.auction\_id, p.product\_name, c.customer\_name, ab.bid\_amount, ab.bid\_date  FROM Auction\_Bids ab  JOIN Auction\_Items ai ON ab.auction\_item\_id = ai.auction\_item\_id  JOIN Products p ON ai.product\_id = p.product\_id  JOIN Customers c ON ab.bidder\_id = c.customer\_id; |

Листинг 3.1 – Представление all\_info\_track

Таким образом, представление Auction\_Bids\_Info упрощает доступ к данным о заявках на участие в аукционе, объединяя данные из нескольких таблиц в одной таблице.

### 3.1.2 Индексы базы данных

Индекс — объект базы данных, который используется для ускорения поиска данных. В случае большого количества строк в таблице, последовательный поиск данных может занимать много времени. Индекс формируется на основе значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет быстро искать строки, удовлетворяющие заданному критерию поиска. Использование индексов ускоряет работу с базой данных, потому что они имеют оптимизированную структуру для поиска, например, сбалансированное дерево.

Для оптимизации запросов к таблицам базы данных были созданы группы индексов практически для каждого столбца таблицы. Они позволяют ускорять поиск по данным, таким ка email клиента, изначальная цена товара, id объектов базы данных и др. Пример индексов для таблицы product представлен на листинге 3.2.

|  |
| --- |
| --Products  CREATE INDEX idx\_products\_category\_id ON Products(category\_id);  CREATE INDEX idx\_products\_price ON Products(price);  CREATE INDEX idx\_products\_product\_name ON Products(product\_name); |

Листинг 3.2 – Индексы базы данных для таблицы Products.

Кроме индексов idx\_products\_price, idx\_products\_product\_name, idx\_products\_category\_id в базе данных также присутствуют индексы на полях других таблиц, таких как Employees, Customers, Order\_Items и других.

Индекс idx\_order\_id на таблице ORDERS был создан для ускорения поиска пользователя по его ID.

Индекс idx\_products\_price на таблице Products может использоваться для быстрого поиска товаров по их ID.

Индекс idx\_products\_product\_name на таблице Products был создан для ускорения поиска плейлистов пользователя по его ID.

В целом, использование индексов позволяет существенно ускорить операции поиска, сортировки и фильтрации данных в базе данных, особенно в случае большого объема данных. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах, поэтому необходимо сбалансировать количество и тип индексов для оптимальной производительности базы данных.

### 3.1.3 Триггеры базы данных

Триггер базы данных — это объект базы данных, который выполняет некоторое действие автоматически при определенных событиях в таблице или представлении базы данных. Триггер может быть запрограммирован на срабатывание при вставке, обновлении или удалении строк в таблице.

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

В данном случае были созданы триггеры для всех таблиц базы данных, которые срабатывают на операции insert, update и delete и записывают данные с типом XMLType в специальные таблицы (например, Products\_XML). Эти таблицы затем используются процедурами, которые используют пакет UTL\_FILE для записи данных в XML-файл.

Каждый триггер содержит код, который выполняется перед или после операции insert, update или delete на определенной таблице базы данных. Код триггера получает доступ к измененным данным через предопределенные ссылки на старые и новые значения (old и new), которые могут использоваться для формирования XML-данных. Затем, при помощи функции insert, данные вставляются в соответствующую таблицу XML. Пример триггера реагирующего на операцию update для таблицы Products будет представлен на листинге 3.3.

|  |
| --- |
| CREATE OR REPLACE TRIGGER trg\_products\_update  AFTER UPDATE ON Products  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<product>' ||  '<product\_id>' || :new.product\_id || '</product\_id>' ||  '<product\_name>' || :new.product\_name || '</product\_name>' ||  '<product\_description>' || :new.product\_description || '</product\_description>' ||  '<category\_id>' || :new.category\_id || '</category\_id>' ||  '<stock\_quantity>' || :new.stock\_quantity || '</stock\_quantity>' ||  '<price>' || :new.price || '</price>' ||  '</product>'  )  INTO v\_xml  FROM dual;  UPDATE Products\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/product/product\_id/text()') = :new.product\_id;  END;  / |

Листинг 3.3 – Скрипт триггера trg\_products\_update

Второй триггер проверяет, что следующая ставка на некоторый лот аукциона не будет меньше предидущей. Если же ставка меньше, то триггер генерирует исключение с сообщением об ошибке " Bid amount must exceed last bid amount". Данный триггер создан для таблицы Auction\_Bids и выполняется перед выполнением операции вставки (INSERT) для каждой строки (FOR EACH ROW). При вставке новой записи, триггер проверяет, что в поле bid\_amount новой записи указана сумма, превышающая сумму последней ставки для данного аукционного товара (auction\_item\_id). Затем происходит сравнение значений поля bid\_amount новой записи и переменной last\_bid\_amount. Данный триггер обеспечивает контроль за суммой ставок для каждого аукционного товара, не позволяя вставлять новые записи с суммой ставки, меньшей или равной последней ставке для данного товара. Скрипт триггера будет представлен на листинге 3.4.

|  |
| --- |
| CREATE OR REPLACE TRIGGER check\_bid\_amount  BEFORE INSERT ON Auction\_Bids  FOR EACH ROW  DECLARE  last\_bid\_amount NUMBER;  BEGIN  SELECT bid\_amount INTO last\_bid\_amount  FROM Auction\_Bids  WHERE auction\_item\_id = :NEW.auction\_item\_id  ORDER BY bid\_date DESC  FETCH FIRST 1 ROW ONLY;    IF last\_bid\_amount IS NOT NULL AND :NEW.bid\_amount <= last\_bid\_amount THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Bid amount must exceed last bid amount');  END IF;  END; |

Листинг 3.4 – Скрип триггера check\_password\_length

Таким образом, триггер выполняют валидацию данных перед их вставкой или обновлением в таблицу Auction\_Bids, обеспечивая целостность данных.

## 3.2 Описание используемой технологии

Диагностика и мониторинг базы данных являются важными инструментами для обеспечения оптимальной производительности и стабильности базы данных. Они позволяют операторам баз данных контролировать состояние базы данных, определять проблемы, которые могут возникнуть, и принимать меры для их решения. В данном случае была использована технология встроенная в SQLDeveloper, такая как Real-Time SQL Monitor. На рисунке 2.2 показано контекстное меню Tools с разделом Real Time SQL Monitor.

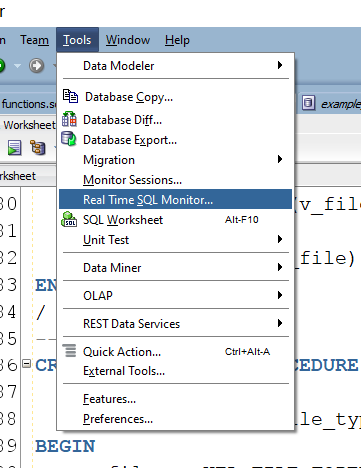


Рисунок 2.2 – Контекстное меню Real-Time SQL Monitor

Real-Time SQL Monitor - это инструмент мониторинга, доступный в Oracle SQL Developer, который позволяет анализировать выполнение SQL запросов в реальном времени. С помощью Real-Time SQL Monitor можно легко и быстро определить, какие запросы работают медленно, а также проследить их выполнение и выявить узкие места.

Real-Time SQL Monitor позволяет просматривать подробную информацию о запросах, включая статистику времени выполнения, планы выполнения запросов, а также информацию о использовании ресурсов базы данных, включая ожидания ввода-вывода, блокировки и другие проблемы, которые могут привести к замедлению выполнения запросов. На рисунке 2.3 изображён интерфейс Real-Time SQL Monitor.

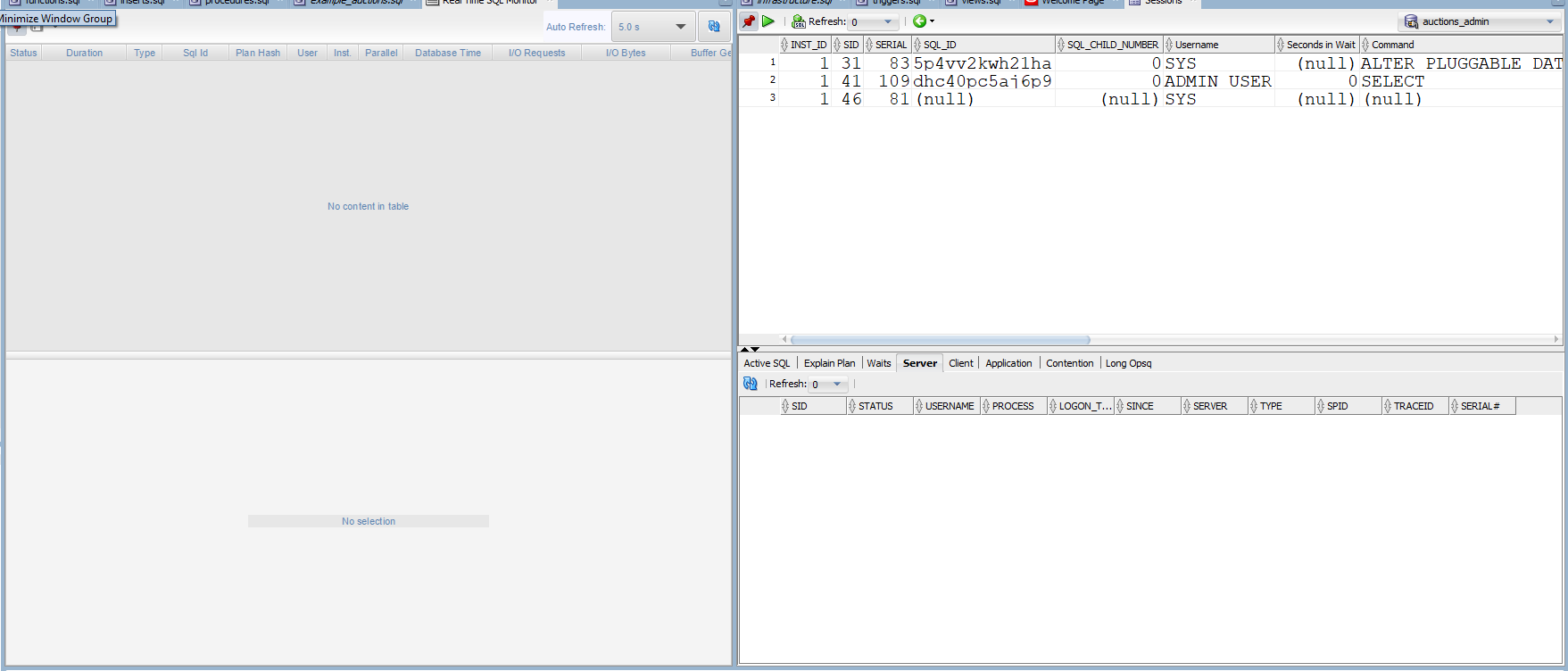


Рисунок 2.3 – интерфейс Real-Time SQL Monitor.

AWR (Automatic Workload Repository) - это инструмент мониторинга и диагностики базы данных в Oracle, который позволяют анализировать производительность базы данных и выявлять проблемы. На рисунке 2.4 представлен пример отчета AWS Rows Details.

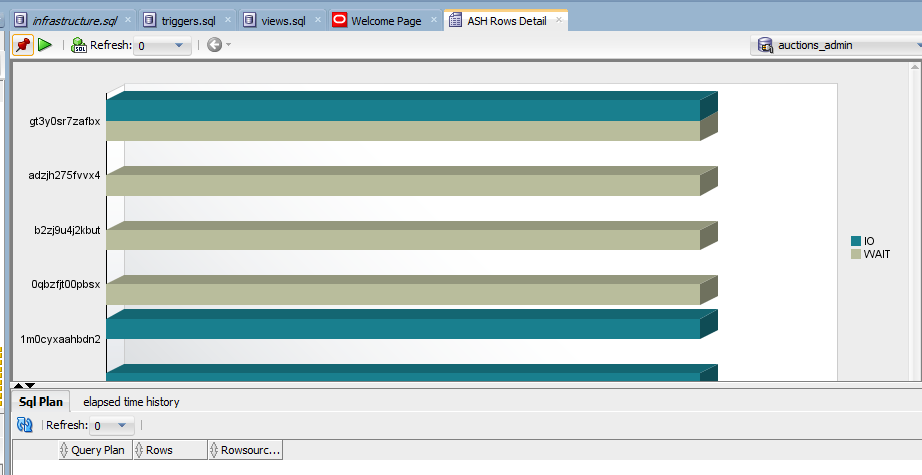


Рисунок 2.4 – отчет AWS Rows Details.

AWS Rows Details содержит информацию о таких аспектах как время выполнения операций, количество обращений к строкам таблиц, блокировки и другие метрики, которые могут быть полезны для диагностики и оптимизации производительности базы данных.

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для электронной аукционной площадки. Tакже была описана технология диагностики и мониторинга базы данных. Технология диагностики и мониторинга базы данных является неотъемлемой частью работы с базами данных, так как позволяет контролировать состояние системы, выявлять проблемы и решать их до того, как они приведут к серьезным последствиям.

4 Установка, настройка и использование Oracle 12c

## 4.1 Установка Oracle 12c

Oracle 12c - это реляционная база данных, выпущенная компанией Oracle Corporation. Она предлагает множество функций, таких как масштабируемость, высокую производительность, защиту данных, поддержку многопоточности и транзакционность. Одной из особенностей Oracle 12c является использование контейнерной архитектуры, где база данных разделена на отдельные контейнеры (CDB) и содержащиеся в них базы данных подчинены этим контейнерам (PDB).

После установки Oracle 12c на сервер, была произведена конфигурация сервера для оптимальной работы с базой данных. Затем была создана база данных с названием auctions\_pdb, которая будет использоваться в дальнейшем для хранения данных музыкального приложения.

## 4.2 Создание таблиц

В данном разделе мы создадим таблицы для нашей базы данных. Но перед тем, как приступить к созданию таблиц, нам нужно создать табличное пространство.

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 будет представлен на листинге 4.1.

|  |
| --- |
| CREATE TABLESPACE AUCTIONS  DATAFILE 'AUCTIONS.dat'  SIZE 2G  AUTOEXTEND ON  NEXT 500M  MAXSIZE 10G;  CREATE TEMPORARY TABLESPACE AUCTIONS\_TEMP  TEMPFILE 'AUCTIONS\_TEMP.dat'  SIZE 1G  AUTOEXTEND ON  NEXT 300M  MAXSIZE 5G; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создано 2 табличных пространство: Auctions и Auctions\_Temp. Auctions\_Temp в свою очередь является временным табличным пространством, в котором будут храниться временные объекты.

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. 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, у каждого заказа может быть множество лотов, которые он добавил в свою библиотеку. Для этого мы добавим в таблицу Orders внешний ключ (FOREIGN KEY) на таблицу Order\_Items, который будет указывать на идентификатор заказанного товара. На листинге 4.2 представлен код создания таблицы Orders.

|  |
| --- |
| CREATE TABLE Order\_Items (  order\_item\_id NUMBER PRIMARY KEY,  order\_id NUMBER REFERENCES Orders(order\_id),  auction\_bid\_id NUMBER REFERENCES Auction\_Bids(auction\_bid\_id),  quantity NUMBER,  price\_per\_item NUMBER  ) TABLESPACE AUCTIONS; |

Листинг 4.2 – Cкрипт создание таблицы Order\_Items

Таким образом, было описано создание табличного пространства для базы данных, а также таблиц, которые будут храниться в этих пространствах. Были созданы четыре табличных пространства: TS\_USER, TS\_TRACK, TS\_PLAYLIST и .

## Создание ролей для разграничения

В этом разделе создаются роли для ограничения доступа к базе данных. Создание ролей позволяет установить границы доступа к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

Будут созданы три роли для разграничения доступа к базе данных: manager\_role, seller\_role и customer\_role. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Роли seller\_role и customer\_role являются основными, в то время как роль manager\_role будет иметь больший набор привелегий, по сравнению с seller\_role.

Роль admin\_role имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выданные привилегии роли admin\_role можно увидеть на листинге 4.3.

|  |
| --- |
| GRANT dba TO admin\_role;  GRANT CREATE ANY DIRECTORY TO admin\_role;  GRANT WRITE ON DIRECTORY xml\_dir TO admin\_role;  GRANT SELECT ANY DICTIONARY TO admin\_role;  GRANT SELECT\_CATALOG\_ROLE TO admin\_role;  GRANT SELECT ANY TRANSACTION TO admin\_role;  GRANT SELECT ANY TABLE TO admin\_role;  GRANT ALTER SYSTEM TO admin\_role; |

Листинг 4.3 – Привилегии, выданные роли admin\_role

Роль manager\_role имеет права на выполнение различных функций и процедур, связанных с управлением пользователями, товарами и аукционами в базе данных. Эта роль может выполнять функции добавления, редактирования и удаления товаров в ассортименте, а также обновления информации о пользователях. Выданные привилегии роли manager\_role можно увидеть на листинге 4.4.

|  |
| --- |
| CREATE ROLE manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Orders TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Order\_Items TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Customers TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Products TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Auctions TO manager\_role; |

Листинг 4.4 – Привилегии, выданные роли manager\_role

Роль customer может делать ставки, просматривать каталог товаров и видеть некоторую информацию о продавце. Кроме того, customer имеет возможность редактировать некоторую информацию о себе. Выданные привилегии роли customer\_role можно увидеть на листинге 4.5.

|  |
| --- |
| CREATE ROLE customer\_role;  GRANT SELECT, INSERT, UPDATE ON Auction\_Bids TO customer\_role;  GRANT SELECT ON Orders TO customer\_role;  GRANT SELECT ON Products TO customer\_role;  GRANT SELECT ON seller\_employees TO customer\_role; |

Листинг 4.5 – Привилегии, выданные роли customer\_role

Таким образом, были созданы три роли для ограничения доступа к базе данных: admin\_role, customer\_role, manager\_role и seller\_role. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль admin\_role имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль customer\_role имеет ограниченный набор привилегий, который позволяет пользователю только просматривать каталог товаров и делать ставки. Роль manager\_role имеет средний уровень привилегий и может выполнять функции, связанные с управлением пользователями и лотами в базе данных.

## 4.4 Создание пакетов процедур для базы данных

Для управления данными через приложение пользователи и администраторы используют хранимые процедуры и функции. Хранимая процедура представляет собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Функция также представляет собой набор SQL-инструкций, но возвращает значение, которое может быть использовано внутри другой инструкции SQL.

Написанные в ходе разработки курсового проекта процедуры и функции можно разбить на несколько категорий:

1. Выборка данных из таблиц;
2. Выборка данных по поисковому запросу;
3. Заполнение таблиц 100 000 строк;
4. Добавление данных в таблицы;
5. Удаление данных из таблиц;
6. Изменение данных в таблицах;
7. Дополнительные функции.

Отличие функций от процедур состоит в том, что функции возвращают значение, которое может быть использовано в других SQL-запросах, а процедуры не возвращают значение. Кроме того, функции могут быть использованы в выражениях SQL, например, для вычисления значения поля в запросе SELECT.

В зависимости от того, какую задачу необходимо выполнить, следует использовать хранимую процедуру или функцию. Хранимые процедуры могут использоваться для выполнения сложных операций над данными, таких как массовые изменения в таблицах, а также для оптимизации производительности приложения. Функции же наиболее полезны в случаях, когда требуется выполнить вычисление на основе данных в базе данных, например, для подсчета статистики или фильтрации данных.

### 4.4.1 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: GetUsers, GetUserById, GetTracks, get\_users\_from\_track, GetTracksUser, GetTrackById, GetPlaylistTracksByID, GetAllPlayListByUserId, GetAllPlaylists, GetTrackFromLibraryByUserID, GetRatingUsers, get\_recent\_tracks. Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

GetUsers, GetTracks, GetAllPlaylists - функции для выборки списка пользователей, треков и всех плейлистов.

GetUserById, GetTrackById, GetPlaylistTracksByID - функции для получения информации об определенном пользователе, треке или плейлисте.

get\_users\_from\_track, GetTracksUser - функции для получения связанных записей между пользователями и треками.

GetAllPlayListByUserId, GetTrackFromLibraryByUserID - функции для выборки плейлистов и треков, которые относятся к определенному пользователю.

GetRatingUsers, get\_recent\_tracks - функции для выборки наиболее популярных пользователей и недавно проигранных треков.

На листинге 4.6 будет функция GetUsers, которая предназначена для выборки всех пользователей из таблицы Users.

|  |
| --- |
| CREATE OR REPLACE FUNCTION GETUSERS()  RETURNS TABLE (  USER\_ID INTEGER,  USER\_NAME VARCHAR(255),  USER\_IMG BYTEA,  USER\_EMAIL VARCHAR(255),  USER\_PASSWORD VARCHAR(255),  USER\_DATE\_OF\_BIRTH DATE,  ROLE\_NAME VARCHAR(255)  )  AS $$  BEGIN  RETURN QUERY SELECT \*  FROM ALL\_INFO\_USER  ORDER BY ALL\_INFO\_USER.USER\_ID;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.6 – Функция GetUsers

Все остальные функции и процедуры будут аналогичны, также предназначены для выборки данных из различных таблиц базы данных.

### 4.4.3 Заполнение таблиц 100 000 строк

Для заполнения таблицы Products была разработана функция insert\_products, которая вставляет 100000 строк в таблицу. Функция представлена на листинге 4.8.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE insert\_products AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Products (  product\_id,  product\_name,  product\_description,  category\_id,  stock\_quantity,  price  ) VALUES (  i,  'Product ' || i,  'Product ' || i || ' description',  TRUNC(DBMS\_RANDOM.VALUE(1, 10)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100)),  TRUNC(DBMS\_RANDOM.VALUE(1, 1000))  );  END LOOP;  COMMIT;  END; |

Листинг 4.8 – Функция заполнения таблицы Products

Функция insert\_products была создана для заполнения таблицы Products 100000 строками. В теле функции используется цикл FOR, который проходит по значениям от 1 до 100000 и для каждого значения выполняет вставку новой строки в таблицу Products с именем 'Products' и порядковым номером из цикла. Процедура не имеет выходного параметра и не принимает аргументов.

### 4.4.4 Добавление данных в таблицы

Для добавления значений в таблицы были разработаны процедуры, являющиеся оболочками для вставки значений в определённые таблицы базы данных. На листинге 4.9 показана процедура add\_product, добавляющая новые строки в таблицу Products.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  INSERT INTO Products (product\_id, product\_name, product\_description, category\_id, stock\_quantity, price)  VALUES (p\_id, p\_name, p\_desc, cat\_id, stock\_qty, p\_price);  END;$$; |

Листинг 4.9 – Процедура добавления товара

Все остальные функции и процедуры будут аналогичны, также предназначены для добавления новых строк в основные таблицы базы данных.

### 4.4.6 Изменение данных в таблицы

Для изменения значений в таблицы по id были разработаны процедуры, являющиеся оболочками для обновления строк в определённые таблицы базы данных. На листинге 4.10 показана процедура update\_product, добавляющая новые строки в таблицу Products.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  UPDATE Products  SET product\_name = p\_name,  product\_description = p\_desc,  category\_id = cat\_id,  stock\_quantity = stock\_qty,  price = p\_price  WHERE product\_id = p\_id;  END; |

Листинг 4.10 – Процедура обновления товара

Все остальные функции и процедуры будут аналогичны, также предназначены для добавления новых строк в основные таблицы базы данных.

### 4.4.6 Удаление данных из таблицы

Для удаления строк из таблицы по id были разработаны процедуры, являющиеся оболочками для обновления строк в определённые таблицы базы данных. На листинге 4.11 показана процедура delete\_product, добавляющая новые строки в таблицу Products.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  UPDATE Products  SET product\_name = p\_name,  product\_description = p\_desc,  category\_id = cat\_id,  stock\_quantity = stock\_qty,  price = p\_price  WHERE product\_id = p\_id;  END; |

Листинг 4.10 – Процедура удаления товара

Все остальные функции и процедуры будут аналогичны, также предназначены для удаления строк из основных таблиц базы данных.

### 4.4.7 Дополнительные функции

Дополнительные функции в базе данных могут быть полезны для решения различных задач, которые не решаются стандартными запросами.

Процедура get\_customer\_winning\_bids позволяет получить список выигранных ставок клиента по его id, что также может быть полезно при использовании базы данных. Функция представлена на листинге 4.12.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE get\_customer\_winning\_bids(  p\_customer\_id IN NUMBER,  p\_bids OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_bids FOR  SELECT \*  FROM Auction\_Bids  WHERE bidder\_id = p\_customer\_id  AND is\_win = 1;  END get\_customer\_winning\_bids; |

Листинг 4.12 – Процедура mark\_bid\_as\_winner

Процедура mark\_bid\_as\_winner позволяет определить выигранную ставку и перевести её в категорию заказов для дальнейшей продажи. Процедура представлена на листинге 4.13.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE mark\_bid\_as\_winner(  p\_auction\_bid\_id IN NUMBER,  p\_quantity IN NUMBER  ) AS  v\_auction\_item\_id NUMBER;  v\_product\_id NUMBER;  v\_seller\_id NUMBER;  v\_bid\_amount NUMBER;  BEGIN  -- Получаем информацию о ставке  SELECT ai.auction\_item\_id, ai.product\_id, a.seller\_id, ab.bid\_amount  INTO v\_auction\_item\_id, v\_product\_id, v\_seller\_id, v\_bid\_amount  FROM Auction\_Bids ab  JOIN Auction\_Items ai ON ab.auction\_item\_id = ai.auction\_item\_id  JOIN Auctions a ON ai.auction\_id = a.auction\_id  WHERE ab.auction\_bid\_id = p\_auction\_bid\_id;    -- Помечаем ставку как выигрышную  UPDATE Auction\_Bids  SET is\_win = 1  WHERE auction\_bid\_id = p\_auction\_bid\_id;    -- Создаем новый заказ  INSERT INTO Orders(order\_id, order\_date, total\_price)  VALUES (order\_id\_seq.NEXTVAL, SYSDATE, v\_bid\_amount \* p\_quantity);    -- Добавляем запись в таблицу Order\_Items  INSERT INTO Order\_Items(order\_item\_id, order\_id, auction\_bid\_id, quantity, price\_per\_item)  VALUES (order\_item\_id\_seq.NEXTVAL, order\_id\_seq.CURRVAL, p\_auction\_bid\_id, p\_quantity, v\_bid\_amount);    -- Обновляем количество товара на складе  UPDATE Products  SET stock\_quantity = stock\_quantity - p\_quantity  WHERE product\_id = v\_product\_id;    -- Удаляем товар из таблицы Products, если его количество на складе стало равным 0  IF (SELECT stock\_quantity FROM Products WHERE product\_id = v\_product\_id) = 0 THEN  DELETE FROM Products WHERE product\_id = v\_product\_id;  END IF;    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Bid with ID ' || p\_auction\_bid\_id || ' has been marked as winner and a new order has been created.');  END mark\_bid\_as\_winner; |

Листинг 4.13 – Процедура mark\_bid\_as\_winner.

В целом, эти функции могут быть полезны для повышения безопасности хранения паролей пользователей, а также для получения дополнительной информации в базе данных.

## 4.5 Описание процедур экспорта в XML

Для базы данных была предусмотрена функция экспорта данных таблиц базы данных в xml файлы, хранящиеся на сервере.

В начале были созданы соответствующие таблицы для каждой из таблиц базы данных, состоящие из одно столбца типа XMLType. В Листинге 4.14. приведён пример создание данной таблицы.

|  |
| --- |
| CREATE TABLE Products\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML; |

Листинг 4.14 – Функция EXPORT\_USERS\_TO\_JSON\_FILE

Данные таблицы, содержащие xml данные, используются триггерами таблиц базы данных при удалении, изменении, добавлении строк в оригинальные таблицы. В листинге 4.16. представлен триггер, который записывает данные параллельно в таблицу Products\_XML при их добавлении в таблицу Products.

|  |
| --- |
| CREATE OR REPLACE TRIGGER trg\_products\_insert  AFTER INSERT ON Products  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<product>' ||  '<product\_id>' || :new.product\_id || '</product\_id>' ||  '<product\_name>' || :new.product\_name || '</product\_name>' ||  '<product\_description>' || :new.product\_description || '</product\_description>' ||  '<category\_id>' || :new.category\_id || '</category\_id>' ||  '<stock\_quantity>' || :new.stock\_quantity || '</stock\_quantity>' ||  '<price>' || :new.price || '</price>' ||  '</product>'  )  INTO v\_xml  FROM dual;  INSERT INTO Products\_XML (xml\_data) VALUES (v\_xml);  END;  / |

Листинг 4.15 – Триггер trg\_products\_insert

Для экспорта и сохранения данных в xml формате на сервере базы данных была разработана процедура SAVE\_PRODUCTS\_XML, которая для записи в файл использует пакет функций UTL\_FILE. Процедура представлена на листинге 4.16.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE SAVE\_PRODUCTS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'products.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<products>');    FOR rec IN (SELECT xml\_data FROM Products\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</products>');  UTL\_FILE.FCLOSE(v\_file);  END;  / |

Листинг 4.16 – Процедура SAVE\_PRODUCTS\_XML

Таким образом, пользователи базы данных могут легко экспортировать данные и сохранять данные в формате XML, что делает управление базой данных более удобным и эффективным.

## Вывод

В данном разделе были рассмотрены основные этапы установки, настройки и использования Oracle 12c. Были описаны процедуры создания таблиц, ролей для разграничения доступа к базе данных и пакетов процедур для выполнения различных операций с данными.

Были также представлены процедуры экспорта данных в формате Xml, а также были проведены тесты производительности базы данных. В результате тестирования было установлено, что база данных Oracle 12c обладает высокой производительностью и способна быстро обрабатывать запросы на получение данных.

Итак, можно сделать вывод, что Oracle 12c является мощной и надежной системой управления базами данных, которая может быть использована для хранения и обработки больших объемов данных. Правильная установка и настройка Oracle 12c а также оптимизация запросов, позволят обеспечить высокую производительность и эффективность работы с базой данных.

5 Тестирование

## 5.1 Тестирование производительности базы данных

Тестирование производительности является важным этапом разработки, поскольку позволяет определить, насколько хорошо база данных может обрабатывать запросы и как быстро она может возвращать результаты.

Для тестирования производительности базы данных была выбрана таблица Products, содержащая 100000 строк. Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| SELECT \* FROM Products WHERE product\_name = 'Product 33'; |

Листинг 5.1 – Запрос к таблице genre

Запрос к таблице Products показывает, что его стоимость выполнения с Total Cost равным 238 говорит о том, что выполнение запроса требует значительных затрат времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составляет 0,031 с. Результаты запроса будут представлены на рисунке 5.2.

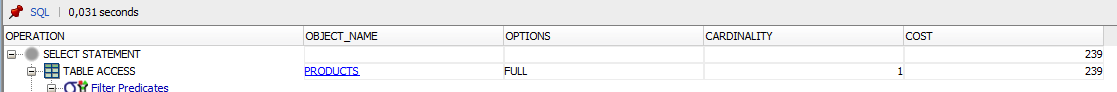


Рисунок 5.2 – Результат выполнения запроса без индекса idx\_product\_name

Для ускорения данного процесса можно создать индекс на поле product\_name, 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 5.3

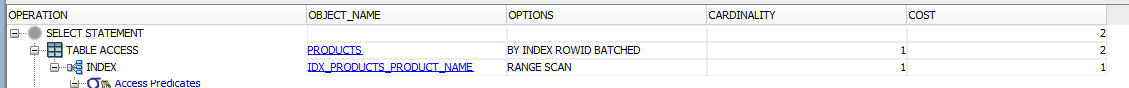


Рисунок 5.3 – Результат выполнения запроса с индексом idx\_product\_name

После создания индекса на поле product\_name запрос к таблице Products имеет стоимость выполнения с Total Cost равным 2, что является значительным улучшением по сравнению с предыдущим результатом, составляющим 238. Время выполнения запроса сократилось до 0,002 с. Результаты запроса также будут представлены на рисунке 5.3.

Тестирование показало, что создание индекса на поле product\_name значительно улучшило производительность запроса к таблице Products. Стоимость выполнения запроса уменьшилась с 238 до 2. При работе с большими объемами данных каждая миллисекунда может иметь значение. Поэтому создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

## 5.2 Вывод

В данном разделе было рассмотрено важное понятие тестирования производительности базы данных. Для проведения тестирования была выбрана таблица Products с большим количеством данных. Проведенный тест показал, что создание индекса на поле product\_name значительно улучшило производительность запроса к таблице Products, сократив время выполнения запроса и уменьшив стоимость выполнения запроса. Выводом является то, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных, что особенно важно при работе с большими объемами данных.

# Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление информацией. В данной работе была поставлена задача разработки базы данных для электронной аукционной площадки с использованием технологии диагностики и мониторинга.

На начальном этапе проектирования был проведён предметный анализ области аукционных площадок, были выделены некоторые основные объекты необходимые для корректной работы проекта.

Далее была реализована инфраструктура базы данных, в частности табличные пространства, профили безопасности, роли, пользователи, привелегии. В контексте этой инфраструктуры были спроектированы и созданы на основе ранее проведённого анализа. Также были созданы вспомогательные элементы для функционирования базы данных, такие как индексы, триггеры, представления, функции и процедуры.

Тестирование базы данных было проведено при использовании большого объема данных, и результаты были положительными. Были реализованы процедуры для импорта и экспорта данных в формат XML, что обеспечило удобство использования и управления данными, а также их сохранность.

В контексте базы данных были полностью реализованы поставленные задачи, такие как:

* Управление информацией о текущем ассортименте площадки.
* Занесение данных о покупках.
* Управление данными о сотрудниках компании.
* Организация данных об аукционах, проведённых площадкой.
* Хранение в базе данных ролей сотрудников.

В целом, результаты работы говорят о том, что база данных полностью соответствует поставленным требованиям и может успешно использоваться в рамках аукционной площадки.

# Список литературных источников

1. Чарльз Ким и Сэм М. Аллапати. Oracle Database 11g [Электронный ресурс] / – Дата доступа: 18.04.2023.

2. Стивен Фейерштейн. Oracle PL/SQL для профессионалов [Электронный ресурс] / – Дата доступа: 18.04.2023.

3 Use SQL Developer to Monitor Database Perfomance [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/en/cloud/paas/exadata-express-cloud/csdbp/use-sql-developer-monitor-database-performance-ee> – Дата доступа: 18.04.2023.

# Приложение А

|  |
| --- |
| -------------------------------------  ---------- TABLESPACES ------------  -------------------------------------  CREATE TABLESPACE AUCTIONS  DATAFILE 'AUCTIONS.dat'  SIZE 2G  AUTOEXTEND ON  NEXT 500M  MAXSIZE 10G;  CREATE TEMPORARY TABLESPACE AUCTIONS\_TEMP  TEMPFILE 'AUCTIONS\_TEMP.dat'  SIZE 1G  AUTOEXTEND ON  NEXT 300M  MAXSIZE 5G;  -------------------------------------  ------------- TABLES --------------  -------------------------------------  -- создание таблицы "Products"  CREATE TABLE Products (  product\_id NUMBER PRIMARY KEY,  product\_name VARCHAR2(50),  product\_description VARCHAR2(200),  category\_id NUMBER REFERENCES Categories(category\_id),  stock\_quantity NUMBER,  price NUMBER  ) TABLESPACE AUCTIONS;  -- создание таблицы "Orders"  CREATE TABLE Orders (  order\_id NUMBER PRIMARY KEY,  order\_date DATE,  total\_price NUMBER  ) TABLESPACE AUCTIONS;  -- создание таблицы "Order\_Items"  -- таблица "Order\_Items" содержит те товары, которые приобрёл определённый покупатель во время аукциона.  CREATE TABLE Order\_Items (  order\_item\_id NUMBER PRIMARY KEY,  order\_id NUMBER REFERENCES Orders(order\_id),  auction\_bid\_id NUMBER REFERENCES Auction\_Bids(auction\_bid\_id),  quantity NUMBER,  price\_per\_item NUMBER  ) TABLESPACE AUCTIONS;  -- создание таблицы "Customers"  CREATE TABLE Customers (  customer\_id NUMBER PRIMARY KEY,  customer\_name VARCHAR2(50),  customer\_email VARCHAR2(50),  customer\_phone VARCHAR2(20)  ) TABLESPACE AUCTIONS;  -- создание таблицы "Categories"  CREATE TABLE Categories (  category\_id NUMBER PRIMARY KEY,  category\_name VARCHAR2(50),  category\_description VARCHAR2(200)  ) TABLESPACE AUCTIONS;  -- создание таблицы "Employees"  CREATE TABLE Employees (  employee\_id NUMBER PRIMARY KEY,  employee\_name VARCHAR2(50),  employee\_position VARCHAR2(50) CHECK (employee\_position IN ('seller', 'manager')),  employee\_contact\_info VARCHAR2(200)  ) TABLESPACE AUCTIONS;  -- создание таблицы "Auctions"  CREATE TABLE Auctions (  auction\_id NUMBER PRIMARY KEY,  auction\_name VARCHAR2(50),  auction\_date DATE,  auction\_time VARCHAR2(20),  auction\_location VARCHAR2(50),  seller\_id NUMBER REFERENCES Employees(employee\_id)  ) TABLESPACE AUCTIONS;  -- создание таблицы "Auction\_Items"  -- таблица "Auction\_Items" содержит те товары, которые были проданы во время определенного аукциона.  CREATE TABLE Auction\_Items (  auction\_item\_id NUMBER PRIMARY KEY,  auction\_id NUMBER REFERENCES Auctions(auction\_id),  product\_id NUMBER REFERENCES Products(product\_id),  starting\_price NUMBER  ) TABLESPACE AUCTIONS;  -- создание таблицы "Auction\_Bids"  -- таблица "Auction\_Bids" содержит информацию о заявках на участие в аукционе.  CREATE TABLE Auction\_Bids (  auction\_bid\_id NUMBER PRIMARY KEY,  auction\_item\_id NUMBER REFERENCES Auction\_Items(auction\_item\_id),  bidder\_id NUMBER REFERENCES Customers(customer\_id),  bid\_amount NUMBER,  bid\_date DATE,  is\_win BOOLEAN  ) TABLESPACE AUCTIONS;  -------------------------------------  ------------- INDEXES -------------  -------------------------------------  --Products  CREATE INDEX idx\_products\_category\_id ON Products(category\_id);  CREATE INDEX idx\_products\_price ON Products(price);  CREATE INDEX idx\_products\_product\_name ON Products(product\_name);  --Orders  CREATE INDEX idx\_orders\_order\_date ON Orders(order\_date);  CREATE INDEX idx\_orders\_total\_price ON Orders(total\_price);  --Order\_Items  CREATE INDEX idx\_order\_items\_order\_id ON Order\_Items(order\_id);  CREATE INDEX idx\_order\_items\_auction\_bid\_id ON Order\_Items(auction\_bid\_id);  CREATE INDEX idx\_order\_items\_quantity ON Order\_Items(quantity);  CREATE INDEX idx\_order\_items\_price\_per\_item ON Order\_Items(price\_per\_item);  --Customers  CREATE INDEX idx\_customers\_customer\_email ON Customers(customer\_email);  CREATE INDEX idx\_customers\_customer\_name ON Customers(customer\_name);  --Categories  CREATE INDEX idx\_categories\_category\_name ON Categories(category\_name);  --Employees  CREATE INDEX idx\_employees\_employee\_position ON Employees(employee\_position);  CREATE INDEX idx\_employees\_employee\_name ON Employees(employee\_name);  --Auctions  CREATE INDEX idx\_auctions\_seller\_id ON Auctions(seller\_id);  CREATE INDEX idx\_auctions\_auction\_date ON Auctions(auction\_date);  --Auction\_Items  CREATE INDEX idx\_auction\_items\_auction\_id ON Auction\_Items(auction\_id);  CREATE INDEX idx\_auction\_items\_product\_id ON Auction\_Items(product\_id);  CREATE INDEX idx\_auction\_items\_starting\_price ON Auction\_Items(starting\_price);  --Auction\_Bids  CREATE INDEX idx\_auction\_bids\_auction\_item\_id ON Auction\_Bids(auction\_item\_id);  CREATE INDEX idx\_auction\_bids\_bidder\_id ON Auction\_Bids(bidder\_id);  CREATE INDEX idx\_auction\_bids\_bid\_amount ON Auction\_Bids(bid\_amount);  CREATE INDEX idx\_auction\_bids\_bid\_date ON Auction\_Bids(bid\_date);  CREATE INDEX idx\_auction\_bids\_is\_win ON Auction\_Bids(is\_win);  DROP INDEX idx\_products\_product\_name;  -------------------------------------  ------------- ROLES ---------------  -------------------------------------  --manager\_role  CREATE ROLE manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Orders TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Order\_Items TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Customers TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Products TO manager\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Auctions TO manager\_role;  --seller\_role  CREATE ROLE seller\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Orders TO seller\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Order\_Items TO seller\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Customers TO seller\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Auction\_Items TO seller\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON Auction\_Bids TO seller\_role;  GRANT SELECT, UPDATE, DELETE ON Products TO manager\_role;  GRANT SELECT ON Auctions TO seller\_role;  GRANT SELECT ON Products TO seller\_role;  --customer\_role  CREATE ROLE customer\_role;  GRANT SELECT, INSERT, UPDATE ON Auction\_Bids TO customer\_role;  GRANT SELECT ON Orders TO customer\_role;  GRANT SELECT ON Products TO customer\_role;  GRANT SELECT ON seller\_employees TO customer\_role;  ----------  CREATE VIEW seller\_employees AS  SELECT \*  FROM Employees  WHERE employee\_position = 'seller';  ----------  --admin\_role  CREATE ROLE admin\_role;  GRANT dba TO admin\_role;  GRANT CREATE ANY DIRECTORY TO admin\_role;  GRANT WRITE ON DIRECTORY xml\_dir TO admin\_role;  GRANT SELECT ANY DICTIONARY TO admin\_role;  GRANT SELECT\_CATALOG\_ROLE TO admin\_role;  GRANT SELECT ANY TRANSACTION TO admin\_role;  GRANT SELECT ANY TABLE TO admin\_role;  GRANT ALTER SYSTEM TO admin\_role;  -------------------------------------  ------------ PROFILE --------------  -------------------------------------  CREATE PROFILE PF\_USER LIMIT  SESSIONS\_PER\_USER 3  CPU\_PER\_SESSION 30000  CPU\_PER\_CALL 1000  CONNECT\_TIME 180  IDLE\_TIME 30  LOGICAL\_READS\_PER\_SESSION 10000  LOGICAL\_READS\_PER\_CALL 1000  PRIVATE\_SGA 20M  COMPOSITE\_LIMIT 200M  PASSWORD\_LIFE\_TIME 180  PASSWORD\_REUSE\_TIME 365  PASSWORD\_REUSE\_MAX UNLIMITED  PASSWORD\_LOCK\_TIME 1  PASSWORD\_GRACE\_TIME 7;  CREATE USER admin\_user  IDENTIFIED BY password  DEFAULT TABLESPACE AUCTIONS  TEMPORARY TABLESPACE AUCTIONS\_TEMP  PROFILE PF\_USER  ACCOUNT UNLOCK  PASSWORD EXPIRE;  -------------------------------------  ------------- USERS ---------------  -------------------------------------  --manager  CREATE USER manager\_user  IDENTIFIED BY password  DEFAULT TABLESPACE AUCTIONS  TEMPORARY TABLESPACE AUCTIONS\_TEMP  PROFILE PF\_USER  ACCOUNT UNLOCK  PASSWORD EXPIRE;  GRANT manager\_role TO manager\_user;  --seller  CREATE USER seller\_user  IDENTIFIED BY password  DEFAULT TABLESPACE AUCTIONS  TEMPORARY TABLESPACE AUCTIONS\_TEMP  PROFILE PF\_USER  ACCOUNT UNLOCK  PASSWORD EXPIRE;  GRANT seller\_role TO seller\_user;  --customer  CREATE USER customer\_user  IDENTIFIED BY password  DEFAULT TABLESPACE AUCTIONS  TEMPORARY TABLESPACE AUCTIONS\_TEMP  PROFILE PF\_USER  ACCOUNT UNLOCK  PASSWORD EXPIRE;  GRANT customer\_role TO customer\_user;  --admin  CREATE USER admin\_user  IDENTIFIED BY password  DEFAULT TABLESPACE AUCTIONS  TEMPORARY TABLESPACE AUCTIONS\_TEMP  PROFILE PF\_ADMIN  ACCOUNT UNLOCK  PASSWORD EXPIRE;  GRANT admin\_role TO admin\_user; |

Листинг 1 – infrastructure.sql

|  |
| --- |
| -------------------------------------  ------------- INSERTS -------------  -------------------------------------  DROP TRIGGER update\_products\_stats;  DROP TRIGGER update\_orders\_stats;  DROP TRIGGER update\_order\_items\_stats;  DROP TRIGGER update\_employees\_stats;  DROP TRIGGER update\_customers\_stats;  DROP TRIGGER update\_categories\_stats;  DROP TRIGGER update\_auctions\_stats;  DROP TRIGGER update\_auction\_items\_stats;  DROP TRIGGER update\_auction\_bids\_stats;  ---- (обновление статистики таблицы ...  -- ... products)  CREATE OR REPLACE TRIGGER update\_products\_stats  AFTER INSERT OR UPDATE OR DELETE ON Products  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Products'  );  END;  -- ... orders)  CREATE OR REPLACE TRIGGER update\_orders\_stats  AFTER INSERT OR UPDATE OR DELETE ON Orders  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Orders'  );  END;  -- ... customers)  CREATE OR REPLACE TRIGGER update\_customers\_stats  AFTER INSERT OR UPDATE OR DELETE ON Customers  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Customers'  );  END;  -- ... auctions)  CREATE OR REPLACE TRIGGER update\_auctions\_stats  AFTER INSERT OR UPDATE OR DELETE ON Auctions  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Auctions'  );  END;  -- ... auctions\_items)  CREATE OR REPLACE TRIGGER update\_auction\_items\_stats  AFTER INSERT OR UPDATE OR DELETE ON Auction\_Items  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Auction\_Items'  );  END;  -- ... auction\_bids)  CREATE OR REPLACE TRIGGER update\_auction\_bids\_stats  AFTER INSERT OR UPDATE OR DELETE ON Auction\_Bids  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Auction\_Bids'  );  END;  -- ... categories)  CREATE OR REPLACE TRIGGER update\_categories\_stats  AFTER INSERT OR UPDATE OR DELETE ON Categories  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Categories'  );  END;  -- ... employees)  DROP TRIGGER check\_bid\_amount;  CREATE OR REPLACE TRIGGER update\_employees\_stats  AFTER INSERT OR UPDATE OR DELETE ON Employees  BEGIN  DBMS\_STATS.GATHER\_TABLE\_STATS(  ownname => USER,  tabname => 'Employees'  );  END;  -- (проверяет, чтобы размер новой ставки был больше последней)  CREATE OR REPLACE TRIGGER check\_bid\_amount  BEFORE INSERT ON Auction\_Bids  FOR EACH ROW  DECLARE  last\_bid\_amount NUMBER;  BEGIN  SELECT bid\_amount INTO last\_bid\_amount  FROM Auction\_Bids  WHERE auction\_item\_id = :NEW.auction\_item\_id  ORDER BY bid\_date DESC  FETCH FIRST 1 ROW ONLY;    IF last\_bid\_amount IS NOT NULL AND :NEW.bid\_amount <= last\_bid\_amount THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Bid amount must exceed last bid amount');  END IF;  END;  --------------------  ------- XML --------  --------------------  CREATE TABLE Products\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Orders\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Order\_Items\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Customers\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Categories\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Employees\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Auctions\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Auction\_Items\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  CREATE TABLE Auction\_Bids\_XML (xml\_data XMLType) XMLTYPE COLUMN xml\_data STORE AS BINARY XML;  -- insert  CREATE OR REPLACE TRIGGER trg\_products\_insert  AFTER INSERT ON Products  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<product>' ||  '<product\_id>' || :new.product\_id || '</product\_id>' ||  '<product\_name>' || :new.product\_name || '</product\_name>' ||  '<product\_description>' || :new.product\_description || '</product\_description>' ||  '<category\_id>' || :new.category\_id || '</category\_id>' ||  '<stock\_quantity>' || :new.stock\_quantity || '</stock\_quantity>' ||  '<price>' || :new.price || '</price>' ||  '</product>'  )  INTO v\_xml  FROM dual;  INSERT INTO Products\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_orders\_insert  AFTER INSERT ON Orders  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<order>' ||  '<order\_id>' || :new.order\_id || '</order\_id>' ||  '<order\_date>' || :new.order\_date || '</order\_date>' ||  '<total\_price>' || :new.total\_price || '</total\_price>' ||  '</order>'  )  INTO v\_xml  FROM dual;  INSERT INTO Orders\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_order\_items\_insert  AFTER INSERT ON Order\_Items  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<order\_item>' ||  '<order\_item\_id>' || :new.order\_item\_id || '</order\_item\_id>' ||  '<order\_id>' || :new.order\_id || '</order\_id>' ||  '<auction\_bid\_id>' || :new.auction\_bid\_id || '</auction\_bid\_id>' ||  '<quantity>' || :new.quantity || '</quantity>' ||  '<price\_per\_item>' || :new.price\_per\_item || '</price\_per\_item>' ||  '</order\_item>'  )  INTO v\_xml  FROM dual;  INSERT INTO Order\_Items\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_customers\_insert  AFTER INSERT ON Customers  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<customer>' ||  '<customer\_id>' || :new.customer\_id || '</customer\_id>' ||  '<customer\_name>' || :new.customer\_name || '</customer\_name>' ||  '<customer\_email>' || :new.customer\_email || '</customer\_email>' ||  '<customer\_phone>' || :new.customer\_phone || '</customer\_phone>' ||  '</customer>'  )  INTO v\_xml  FROM dual;  INSERT INTO Products\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_categories\_insert  AFTER INSERT ON Categories  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<category>' ||  '<category\_id>' || :new.category\_id || '</category\_id>' ||  '<category\_name>' || :new.category\_name || '</category\_name>' ||  '<category\_description>' || :new.category\_description || '</category\_description>' ||  '</category>'  )  INTO v\_xml  FROM dual;  INSERT INTO Categories\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_employees\_update  AFTER UPDATE ON Employees  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<employee>' ||  '<employee\_id>' || :new.employee\_id || '</employee\_id>' ||  '<employee\_name>' || :new.employee\_name || '</employee\_name>' ||  '<employee\_position>' || :new.employee\_position || '</employee\_position>' ||  '<employee\_contact\_info>' || :new.employee\_contact\_info || '</employee\_contact\_info>' ||  '</employee>'  )  INTO v\_xml  FROM dual;  INSERT INTO Employees\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_auctions\_update  AFTER UPDATE ON Auctions  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<auction>' ||  '<auction\_id>' || :new.auction\_id || '</auction\_id>' ||  '<auction\_name>' || :new.auction\_name || '</auction\_name>' ||  '<auction\_date>' || :new.auction\_date || '</auction\_date>' ||  '<auction\_time>' || :new.auction\_time || '</auction\_time>' ||  '<auction\_location>' || :new.auction\_location || '</auction\_location>' ||  '<seller\_id>' || :new.seller\_id || '</seller\_id>' ||  '</auction>'  )  INTO v\_xml  FROM dual;  INSERT INTO Auctions\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_auction\_items\_insert  AFTER INSERT ON Auction\_Items  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<auction\_item>' ||  '<auction\_item\_id>' || :new.auction\_item\_id || '</auction\_item\_id>' ||  '<auction\_id>' || :new.auction\_id || '</auction\_id>' ||  '<product\_id>' || :new.product\_id || '</product\_id>' ||  '<starting\_price>' || :new.starting\_price || '</starting\_price>' ||  '</auction\_item>'  )  INTO v\_xml  FROM dual;  INSERT INTO Auction\_Items\_XML (xml\_data) VALUES (v\_xml);  END;  /  CREATE OR REPLACE TRIGGER trg\_auction\_bids\_insert  AFTER INSERT ON Auction\_Bids  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<auction\_bid>' ||  '<auction\_bid\_id>' || :new.auction\_bid\_id || '</auction\_bid\_id>' ||  '<auction\_item\_id>' || :new.auction\_item\_id || '</auction\_item\_id>' ||  '<bidder\_id>' || :new.bidder\_id || '</bidder\_id>' ||  '<bid\_amount>' || :new.bid\_amount || '</bid\_amount>' ||  '<bid\_date>' || :new.bid\_date || '</bid\_date>' ||  '<is\_win>' || :new.is\_win || '</is\_win>' ||  '</auction\_bid>'  )  INTO v\_xml  FROM dual;  INSERT INTO Auction\_Bids\_XML (xml\_data) VALUES (v\_xml);  END;  /  ----  ---- update  CREATE OR REPLACE TRIGGER trg\_products\_update  AFTER UPDATE ON Products  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<product>' ||  '<product\_id>' || :new.product\_id || '</product\_id>' ||  '<product\_name>' || :new.product\_name || '</product\_name>' ||  '<product\_description>' || :new.product\_description || '</product\_description>' ||  '<category\_id>' || :new.category\_id || '</category\_id>' ||  '<stock\_quantity>' || :new.stock\_quantity || '</stock\_quantity>' ||  '<price>' || :new.price || '</price>' ||  '</product>'  )  INTO v\_xml  FROM dual;  UPDATE Products\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/product/product\_id/text()') = :new.product\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_orders\_update  AFTER UPDATE ON Orders  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<order>' ||  '<order\_id>' || :new.order\_id || '</order\_id>' ||  '<order\_date>' || :new.order\_date || '</order\_date>' ||  '<total\_price>' || :new.total\_price || '</total\_price>' ||  '</order>'  )  INTO v\_xml  FROM dual;  UPDATE Orders\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/order/order\_id/text()') = :new.order\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_customers\_update  AFTER UPDATE ON Customers  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<customer>' ||  '<customer\_id>' || :new.customer\_id || '</customer\_id>' ||  '<customer\_name>' || :new.customer\_name || '</customer\_name>' ||  '<customer\_email>' || :new.customer\_email || '</customer\_email>' ||  '<customer\_phone>' || :new.customer\_phone || '</customer\_phone>' ||  '</customer>'  )  INTO v\_xml  FROM dual;  UPDATE Customers\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/customer/customer\_id/text()') = :new.customer\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_order\_items\_update  AFTER UPDATE ON Order\_Items  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<order\_item>' ||  '<order\_item\_id>' || :new.order\_item\_id || '</order\_item\_id>' ||  '<order\_id>' || :new.order\_id || '</order\_id>' ||  '<auction\_bid\_id>' || :new.auction\_bid\_id || '</auction\_bid\_id>' ||  '<quantity>' || :new.quantity || '</quantity>' ||  '<price\_per\_item>' || :new.price\_per\_item || '</price\_per\_item>' ||  '</order\_item>'  )  INTO v\_xml  FROM dual;  UPDATE Order\_Items\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/order\_item/order\_item\_id/text()') = :new.order\_item\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_categories\_update  AFTER UPDATE ON Categories  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<category>' ||  '<category\_id>' || :new.category\_id || '</category\_id>' ||  '<category\_name>' || :new.category\_name || '</category\_name>' ||  '<category\_description>' || :new.category\_description || '</category\_description>' ||  '</category>'  )  INTO v\_xml  FROM dual;  UPDATE Categories\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/category/category\_id/text()') = :new.category\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_employees\_update  AFTER UPDATE ON Employees  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<employee>' ||  '<employee\_id>' || :new.employee\_id || '</employee\_id>' ||  '<employee\_name>' || :new.employee\_name || '</employee\_name>' ||  '<employee\_position>' || :new.employee\_position || '</employee\_position>' ||  '<employee\_contact\_info>' || :new.employee\_contact\_info || '</employee\_contact\_info>' ||  '</employee>'  )  INTO v\_xml  FROM dual;  UPDATE Employees\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/employee/employee\_id/text()') = :new.employee\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_auctions\_update  AFTER UPDATE ON Auctions  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<auction>' ||  '<auction\_id>' || :new.auction\_id || '</auction\_id>' ||  '<auction\_name>' || :new.auction\_name || '</auction\_name>' ||  '<auction\_date>' || :new.auction\_date || '</auction\_date>' ||  '<auction\_time>' || :new.auction\_time || '</auction\_time>' ||  '<auction\_location>' || :new.auction\_location || '</auction\_location>' ||  '<seller\_id>' || :new.seller\_id || '</seller\_id>' ||  '</auction>'  )  INTO v\_xml  FROM dual;  UPDATE Auctions\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/auction/auction\_id/text()') = :new.auction\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_auction\_items\_update  AFTER UPDATE ON Auction\_Items  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<auction\_item>' ||  '<auction\_item\_id>' || :new.auction\_item\_id || '</auction\_item\_id>' ||  '<auction\_id>' || :new.auction\_id || '</auction\_id>' ||  '<product\_id>' || :new.product\_id || '</product\_id>' ||  '<starting\_price>' || :new.starting\_price || '</starting\_price>' ||  '</auction\_item>'  )  INTO v\_xml  FROM dual;  UPDATE Auction\_Items\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/auction\_item/auction\_item\_id/text()') = :new.auction\_item\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_auction\_bids\_update  AFTER UPDATE ON Auction\_Bids  FOR EACH ROW  DECLARE  v\_xml XMLType;  BEGIN  SELECT XMLType(  '<auction\_bid>' ||  '<auction\_bid\_id>' || :new.auction\_bid\_id || '</auction\_bid\_id>' ||  '<auction\_item\_id>' || :new.auction\_item\_id || '</auction\_item\_id>' ||  '<bidder\_id>' || :new.bidder\_id || '</bidder\_id>' ||  '<bid\_amount>' || :new.bid\_amount || '</bid\_amount>' ||  '<bid\_date>' || :new.bid\_date || '</bid\_date>' ||  '<is\_win>' || :new.is\_win || '</is\_win>' ||  '</auction\_bid>'  )  INTO v\_xml  FROM dual;  UPDATE Auction\_Bids\_XML SET xml\_data = v\_xml WHERE EXTRACTVALUE(xml\_data, '/auction\_bid/auction\_bid\_id/text()') = :new.auction\_bid\_id;  END;  /  ----  ---- delete  CREATE OR REPLACE TRIGGER trg\_customers\_delete  AFTER DELETE ON Customers  FOR EACH ROW  BEGIN  DELETE FROM Customers\_XML WHERE EXTRACTVALUE(xml\_data, '/customer/customer\_id/text()') = :old.customer\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_categories\_delete  AFTER DELETE ON Categories  FOR EACH ROW  BEGIN  DELETE FROM Categories\_XML WHERE EXTRACTVALUE(xml\_data, '/category/category\_id/text()') = :old.category\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_order\_items\_delete  AFTER DELETE ON Order\_Items  FOR EACH ROW  BEGIN  DELETE FROM Order\_Items\_XML WHERE EXTRACTVALUE(xml\_data, '/order\_item/order\_item\_id/text()') = :old.order\_item\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_orders\_delete  AFTER DELETE ON Orders  FOR EACH ROW  BEGIN  DELETE FROM Orders\_XML WHERE EXTRACTVALUE(xml\_data, '/order/order\_id/text()') = :old.order\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_products\_delete  AFTER DELETE ON Products  FOR EACH ROW  BEGIN  DELETE FROM Products\_XML WHERE EXTRACTVALUE(xml\_data, '/product/product\_id/text()') = :old.product\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_employees\_delete  AFTER DELETE ON Employees  FOR EACH ROW  BEGIN  DELETE FROM Employees\_XML WHERE EXTRACTVALUE(xml\_data, '/employee/employee\_id/text()') = :old.employee\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_auctions\_delete  AFTER DELETE ON Auctions  FOR EACH ROW  BEGIN  DELETE FROM Auctions\_XML WHERE EXTRACTVALUE(xml\_data, '/auction/auction\_id/text()') = :old.auction\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_auction\_items\_delete  AFTER DELETE ON Auction\_Items  FOR EACH ROW  BEGIN  DELETE FROM Auction\_Items\_XML WHERE EXTRACTVALUE(xml\_data, '/auction\_item/auction\_item\_id/text()') = :old.auction\_item\_id;  END;  /  CREATE OR REPLACE TRIGGER trg\_auction\_bids\_delete  AFTER DELETE ON Auction\_Bids  FOR EACH ROW  BEGIN  DELETE FROM Auction\_Bids\_XML WHERE EXTRACTVALUE(xml\_data, '/auction\_bid/auction\_bid\_id/text()') = :old.auction\_bid\_id;  END;  /  ---- |

Листинг 2 – triggers.sql

|  |
| --- |
| -------------------------------------  ----------- PROCEDURES ------------  -------------------------------------  ----insert  CREATE OR REPLACE PROCEDURE add\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  INSERT INTO Products (product\_id, product\_name, product\_description, category\_id, stock\_quantity, price)  VALUES (p\_id, p\_name, p\_desc, cat\_id, stock\_qty, p\_price);  END;  CREATE OR REPLACE PROCEDURE add\_order (  o\_id IN NUMBER,  o\_date IN DATE,  o\_total\_price IN NUMBER  ) AS  BEGIN  INSERT INTO Orders (order\_id, order\_date, total\_price)  VALUES (o\_id, o\_date, o\_total\_price);  END;  CREATE OR REPLACE PROCEDURE add\_order\_item (  oi\_id IN NUMBER,  o\_id IN NUMBER,  ab\_id IN NUMBER,  qty IN NUMBER,  price\_per\_item IN NUMBER  ) AS  BEGIN  INSERT INTO Order\_Items (order\_item\_id, order\_id, auction\_bid\_id, quantity, price\_per\_item)  VALUES (oi\_id, o\_id, ab\_id, qty, price\_per\_item);  END;  ----update  CREATE OR REPLACE PROCEDURE update\_product (  p\_id IN NUMBER,  p\_name IN VARCHAR2,  p\_desc IN VARCHAR2,  cat\_id IN NUMBER,  stock\_qty IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  UPDATE Products  SET product\_name = p\_name,  product\_description = p\_desc,  category\_id = cat\_id,  stock\_quantity = stock\_qty,  price = p\_price  WHERE product\_id = p\_id;  END;  CREATE OR REPLACE PROCEDURE update\_order (  o\_id IN NUMBER,  o\_date IN DATE,  o\_total\_price IN NUMBER  ) AS  BEGIN  UPDATE Orders  SET order\_date = o\_date,  total\_price = o\_total\_price  WHERE order\_id = o\_id;  END;  CREATE OR REPLACE PROCEDURE update\_order\_item (  oi\_id IN NUMBER,  o\_id IN NUMBER,  ab\_id IN NUMBER,  qty IN NUMBER,  price\_per\_item IN NUMBER  ) AS  BEGIN  UPDATE Order\_Items  SET order\_id = o\_id,  auction\_bid\_id = ab\_id,  quantity = qty,  price\_per\_item = price\_per\_item  WHERE order\_item\_id = oi\_id;  END;  ----delete  CREATE OR REPLACE PROCEDURE delete\_product (p\_id IN NUMBER) AS  BEGIN  DELETE FROM Products WHERE product\_id = p\_id;  END;  CREATE OR REPLACE PROCEDURE delete\_order (o\_id IN NUMBER) AS  BEGIN  DELETE FROM Orders WHERE order\_id = o\_id;  END;  CREATE OR REPLACE PROCEDURE delete\_order\_item (oi\_id IN NUMBER) AS  BEGIN  DELETE FROM Order\_Items WHERE order\_item\_id = oi\_id;  END;  ------------------  ---- customer ----  ------------------  -- (размещение ставки клиентом)  CREATE OR REPLACE PROCEDURE place\_bid(  p\_customer\_id IN NUMBER,  p\_auction\_item\_id IN NUMBER,  p\_bid\_amount IN NUMBER  ) AS  v\_current\_bid\_amount NUMBER;  BEGIN  -- Получаем текущую максимальную ставку на данный товар  SELECT MAX(bid\_amount)  INTO v\_current\_bid\_amount  FROM Auction\_Bids  WHERE auction\_item\_id = p\_auction\_item\_id;    -- Проверяем, что новая ставка больше текущей максимальной ставки  IF p\_bid\_amount > v\_current\_bid\_amount THEN  -- Добавляем новую ставку в таблицу Auction\_Bids  INSERT INTO Auction\_Bids(auction\_bid\_id, auction\_item\_id, bidder\_id, bid\_amount, bid\_date, is\_win)  VALUES (auction\_bid\_id\_seq.NEXTVAL, p\_auction\_item\_id, p\_customer\_id, p\_bid\_amount, SYSDATE, 0);    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Your bid has been placed successfully.');  ELSE  DBMS\_OUTPUT.PUT\_LINE('Your bid amount must be greater than the current highest bid.');  END IF;  END place\_bid;  -- (получение списка всех выигранных ставок клиента)  CREATE OR REPLACE PROCEDURE get\_customer\_winning\_bids(  p\_customer\_id IN NUMBER,  p\_bids OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_bids FOR  SELECT \*  FROM Auction\_Bids  WHERE bidder\_id = p\_customer\_id  AND is\_win = 1;  END get\_customer\_winning\_bids;  -- (получение списка всех ставок клиента)  CREATE OR REPLACE PROCEDURE get\_customer\_bids(  p\_customer\_id IN NUMBER,  p\_bids OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_bids FOR  SELECT \*  FROM Auction\_Bids  WHERE bidder\_id = p\_customer\_id;  END get\_customer\_bids;  ----------------  ---- seller ----  ----------------  -- (добавление товара на аукцион)  CREATE OR REPLACE PROCEDURE add\_auction\_item(  p\_auction\_id IN NUMBER,  p\_product\_id IN NUMBER,  p\_starting\_price IN NUMBER  ) AS  BEGIN  -- Проверяем, что аукцион существует  IF NOT EXISTS(SELECT 1 FROM Auctions WHERE auction\_id = p\_auction\_id) THEN  DBMS\_OUTPUT.PUT\_LINE('Auction with ID ' || p\_auction\_id || ' does not exist.');  RETURN;  END IF;    -- Проверяем, что товар существует  IF NOT EXISTS(SELECT 1 FROM Products WHERE product\_id = p\_product\_id) THEN  DBMS\_OUTPUT.PUT\_LINE('Product with ID ' || p\_product\_id || ' does not exist.');  RETURN;  END IF;    -- Добавляем новый товар на аукцион  INSERT INTO Auction\_Items(auction\_item\_id, auction\_id, product\_id, starting\_price)  VALUES (auction\_item\_id\_seq.NEXTVAL, p\_auction\_id, p\_product\_id, p\_starting\_price);    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('New auction item has been added successfully.');  END add\_auction\_item;  -- (определение выигрышной ставки и добавление её в заказы)  CREATE OR REPLACE PROCEDURE mark\_bid\_as\_winner(  p\_auction\_bid\_id IN NUMBER,  p\_quantity IN NUMBER  ) AS  v\_auction\_item\_id NUMBER;  v\_product\_id NUMBER;  v\_seller\_id NUMBER;  v\_bid\_amount NUMBER;  BEGIN  -- Получаем информацию о ставке  SELECT ai.auction\_item\_id, ai.product\_id, a.seller\_id, ab.bid\_amount  INTO v\_auction\_item\_id, v\_product\_id, v\_seller\_id, v\_bid\_amount  FROM Auction\_Bids ab  JOIN Auction\_Items ai ON ab.auction\_item\_id = ai.auction\_item\_id  JOIN Auctions a ON ai.auction\_id = a.auction\_id  WHERE ab.auction\_bid\_id = p\_auction\_bid\_id;    -- Помечаем ставку как выигрышную  UPDATE Auction\_Bids  SET is\_win = 1  WHERE auction\_bid\_id = p\_auction\_bid\_id;    -- Создаем новый заказ  INSERT INTO Orders(order\_id, order\_date, total\_price)  VALUES (order\_id\_seq.NEXTVAL, SYSDATE, v\_bid\_amount \* p\_quantity);    -- Добавляем запись в таблицу Order\_Items  INSERT INTO Order\_Items(order\_item\_id, order\_id, auction\_bid\_id, quantity, price\_per\_item)  VALUES (order\_item\_id\_seq.NEXTVAL, order\_id\_seq.CURRVAL, p\_auction\_bid\_id, p\_quantity, v\_bid\_amount);    -- Обновляем количество товара на складе  UPDATE Products  SET stock\_quantity = stock\_quantity - p\_quantity  WHERE product\_id = v\_product\_id;    -- Удаляем товар из таблицы Products, если его количество на складе стало равным 0  IF (SELECT stock\_quantity FROM Products WHERE product\_id = v\_product\_id) = 0 THEN  DELETE FROM Products WHERE product\_id = v\_product\_id;  END IF;    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Bid with ID ' || p\_auction\_bid\_id || ' has been marked as winner and a new order has been created.');  END mark\_bid\_as\_winner;  -- (получение списка всех товаров продавца)  CREATE OR REPLACE PROCEDURE get\_seller\_orders(  p\_seller\_id IN NUMBER,  p\_orders OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_orders FOR  SELECT \*  FROM Orders o  JOIN Order\_Items oi ON o.order\_id = oi.order\_id  JOIN Auction\_Bids ab ON oi.auction\_bid\_id = ab.auction\_bid\_id  JOIN Auction\_Items ai ON ab.auction\_item\_id = ai.auction\_item\_id  JOIN Auctions a ON ai.auction\_id = a.auction\_id  WHERE a.seller\_id = p\_seller\_id;  END get\_seller\_orders;  -----------------  ---- manager ----  -----------------  -- (обновление информации о товаре)  CREATE OR REPLACE PROCEDURE update\_product(  p\_product\_id IN NUMBER,  p\_product\_name IN VARCHAR2,  p\_product\_description IN VARCHAR2,  p\_category\_id IN NUMBER,  p\_stock\_quantity IN NUMBER,  p\_price IN NUMBER  ) AS  BEGIN  -- Обновляем информацию о товаре  UPDATE Products  SET product\_name = p\_product\_name,  product\_description = p\_product\_description,  category\_id = p\_category\_id,  stock\_quantity = p\_stock\_quantity,  price = p\_price  WHERE product\_id = p\_product\_id;    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Product with ID ' || p\_product\_id || ' has been updated.');  END update\_product;  -- (удаление заказа)  CREATE OR REPLACE PROCEDURE delete\_order(  p\_order\_id IN NUMBER  ) AS  BEGIN  -- Удаляем запись из таблицы Order\_Items  DELETE FROM Order\_Items WHERE order\_id = p\_order\_id;    -- Удаляем запись из таблицы Orders  DELETE FROM Orders WHERE order\_id = p\_order\_id;    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Order with ID ' || p\_order\_id || ' has been deleted.');  END delete\_order;  --------------------  ------- XML --------  --------------------  -- products  CREATE OR REPLACE PROCEDURE SAVE\_PRODUCTS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'products.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<products>');    FOR rec IN (SELECT xml\_data FROM Products\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</products>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --orders  CREATE OR REPLACE PROCEDURE SAVE\_ORDERS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'orders.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<orders>');    FOR rec IN (SELECT xml\_data FROM Orders\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</orders>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --order\_items  CREATE OR REPLACE PROCEDURE SAVE\_ORDER\_ITEMS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'order\_items.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<order\_items>');    FOR rec IN (SELECT xml\_data FROM Order\_Items\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</order\_items>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --employees  CREATE OR REPLACE PROCEDURE SAVE\_EMPLOYEES\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'employees.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<employees>');    FOR rec IN (SELECT xml\_data FROM Employees\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</employees>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --customers  CREATE OR REPLACE PROCEDURE SAVE\_CUSTOMERS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'customers.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<customers>');    FOR rec IN (SELECT xml\_data FROM Customers\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</customers>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --categories  CREATE OR REPLACE PROCEDURE SAVE\_CATEGORIES\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'categories.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<categories>');    FOR rec IN (SELECT xml\_data FROM Categories\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</categories>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --auctions  CREATE OR REPLACE PROCEDURE SAVE\_AUCTIONS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'auctions.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<auctions>');    FOR rec IN (SELECT xml\_data FROM Auctions\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</auctions>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --auction\_items  CREATE OR REPLACE PROCEDURE SAVE\_AUCTION\_ITEMS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'auction\_items.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<auction\_items>');    FOR rec IN (SELECT xml\_data FROM Auction\_Items\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</auction\_items>');  UTL\_FILE.FCLOSE(v\_file);  END;  /  --auction\_bids  CREATE OR REPLACE PROCEDURE SAVE\_AUCTION\_BIDS\_XML AS  v\_xml XMLType;  v\_file UTL\_FILE.file\_type;  BEGIN  v\_file := UTL\_FILE.FOPEN('XML\_DIR', 'auction\_bids.xml', 'W');  UTL\_FILE.PUT\_LINE(v\_file, '<?xml version="1.0" encoding="utf-8"?>');  UTL\_FILE.PUT\_LINE(v\_file, '<auction\_bids>');    FOR rec IN (SELECT xml\_data FROM Auction\_Bids\_XML)  LOOP  v\_xml := rec.xml\_data;  UTL\_FILE.put\_line(v\_file, v\_xml.getClobVal());  END LOOP;    UTL\_FILE.PUT\_LINE(v\_file, '</auction\_bids>');  UTL\_FILE.FCLOSE(v\_file);  END;  / |

Листинг 3 – procedures.sql

|  |
| --- |
| -------------------------------------  ------------ FUNCTIONS ------------  -------------------------------------  -- (получение информации о товаре по ID)  CREATE OR REPLACE FUNCTION get\_product\_info\_by\_id(  p\_product\_id IN NUMBER  ) RETURN Products%ROWTYPE AS  v\_product\_info Products%ROWTYPE;  BEGIN  -- Получаем информацию о товаре по его ID  SELECT \*  INTO v\_product\_info  FROM Products  WHERE product\_id = p\_product\_id;    RETURN v\_product\_info;  END;  -- (получение информации о товаре по ID)  CREATE OR REPLACE FUNCTION get\_order\_info\_by\_id(  p\_order\_id IN NUMBER  ) RETURN Orders%ROWTYPE AS  v\_order\_info Orders%ROWTYPE;  BEGIN  -- Получаем информацию о заказе по его ID  SELECT \*  INTO v\_order\_info  FROM Orders  WHERE order\_id = p\_order\_id;    RETURN v\_order\_info;  END;  -- (получение информации о клиенте по ID)  CREATE OR REPLACE FUNCTION get\_customer\_info\_by\_id(  p\_customer\_id IN NUMBER  ) RETURN Customers%ROWTYPE AS  v\_customer\_info Customers%ROWTYPE;  BEGIN  -- Получаем информацию о покупателе по ID  SELECT \*  INTO v\_customer\_info  FROM Customers  WHERE customer\_id = p\_customer\_id;    RETURN v\_customer\_info;  END;  -- (получение информации о сотруднике по ID)  CREATE OR REPLACE FUNCTION get\_employee\_info\_id(  p\_employee\_id IN NUMBER  ) RETURN Employees%ROWTYPE AS  v\_employee\_info Employees%ROWTYPE;  BEGIN  -- Получаем информацию о продавце по его ID  SELECT \*  INTO v\_employee\_info  FROM Employees  WHERE employee\_id = p\_employee\_id;    RETURN v\_employee\_info;  END;  -- (получение информации о аукционе по ID)  CREATE OR REPLACE FUNCTION get\_auction\_bid\_info\_by\_id(  p\_auction\_bid\_id IN NUMBER  ) RETURN Auction\_Bids%ROWTYPE AS  v\_auction\_bid\_info Auction\_Bids%ROWTYPE;  BEGIN  -- Получаем информацию о заявке на участие в аукционе по ее ID  SELECT \*  INTO v\_auction\_bid\_info  FROM Auction\_Bids  WHERE auction\_bid\_id = p\_auction\_bid\_id;    RETURN v\_auction\_bid\_info;  END; |

Листинг 4 – functions.sql

|  |
| --- |
| -------------------------------------  ------------- INSERTS -------------  -------------------------------------  CREATE OR REPLACE PROCEDURE insert\_products AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Products (  product\_id,  product\_name,  product\_description,  category\_id,  stock\_quantity,  price  ) VALUES (  i,  'Product ' || i,  'Product ' || i || ' description',  TRUNC(DBMS\_RANDOM.VALUE(1, 10)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100)),  TRUNC(DBMS\_RANDOM.VALUE(1, 1000))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_orders AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Orders (  order\_id,  order\_date,  total\_price  ) VALUES (  i,  SYSDATE,  TRUNC(DBMS\_RANDOM.VALUE(1, 10000))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_order\_items AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Order\_Items (  order\_item\_id,  order\_id,  auction\_bid\_id,  quantity,  price\_per\_item  ) VALUES (  i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 10)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_customers AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Customers (  customer\_id,  customer\_name,  customer\_email,  customer\_phone  ) VALUES (  i,  'Customer ' || i,  'customer' || i || '@example.com',  '555-555-' || LPAD(i, 4, '0')  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_categories AS  BEGIN  FOR i IN 1..10 LOOP  INSERT INTO Categories (  category\_id,  category\_name,  category\_description  ) VALUES (  i,  'Category ' || i,  'Category ' || i || ' description'  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_auctions AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Auctions (  auction\_id,  auction\_name,  auction\_date,  auction\_time,  auction\_location,  seller\_id  ) VALUES (  i,  'Auction ' || i,  TRUNC(SYSDATE + DBMS\_RANDOM.VALUE(1, 30)),  LPAD(TRUNC(DBMS\_RANDOM.VALUE(0, 23.99)), 2, '0') || ':' || LPAD(TRUNC(DBMS\_RANDOM.VALUE(0, 59.99)), 2, '0'),  'Location ' || i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_employees AS  BEGIN  FOR i IN 1..10000 LOOP  INSERT INTO Employees (  employee\_id,  employee\_name,  employee\_position,  employee\_contact\_info  ) VALUES (  i,  'Employee ' || i,  CASE WHEN MOD(i, 2) = 0 THEN 'seller' ELSE 'manager' END,  'employee' || i || '@example.com'  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_auction\_items AS  BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Auction\_Items (  auction\_item\_id,  auction\_id,  product\_id,  starting\_price  ) VALUES (  i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100))  );  END LOOP;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE insert\_auction\_bids AS  BEGIN  FOR i IN 1..200000 LOOP  INSERT INTO Auction\_Bids (  auction\_bid\_id,  auction\_item\_id,  bidder\_id,  bid\_amount,  bid\_date,  is\_win  ) VALUES (  i,  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 100000)),  TRUNC(DBMS\_RANDOM.VALUE(1, 1000)),  SYSDATE,  CASE WHEN MOD(i, 10) = 0 THEN 1 ELSE 0 END  );  END LOOP;  COMMIT;  END;  EXEC insert\_products;  EXEC insert\_orders;  EXEC insert\_order\_items;  EXEC insert\_employees;  EXEC insert\_categories;  EXEC insert\_customers;  EXEC insert\_auctions;  EXEC insert\_auction\_items;  EXEC insert\_auction\_bids; |

Листинг 5 – inserts.sql

|  |
| --- |
| -------------------------------------  -------------- VIEWS --------------  -------------------------------------  -- (информация о продуктах и их количестве)  CREATE VIEW Products\_Inventory AS  SELECT product\_id, product\_name, stock\_quantity  FROM Products;  -- (информация о заявках на участие в аукционе)  CREATE VIEW Auction\_Bids\_Info AS  SELECT ab.auction\_bid\_id, ai.auction\_id, p.product\_name, c.customer\_name, ab.bid\_amount, ab.bid\_date  FROM Auction\_Bids ab  JOIN Auction\_Items ai ON ab.auction\_item\_id = ai.auction\_item\_id  JOIN Products p ON ai.product\_id = p.product\_id  JOIN Customers c ON ab.bidder\_id = c.customer\_id;  -- (информация о продавцах и товарах, которые они выставили на аукцион)  CREATE VIEW Auction\_Sellers AS  SELECT e.employee\_name, p.product\_name, ai.starting\_price  FROM Auctions a  JOIN Employees e ON a.seller\_id = e.employee\_id  JOIN Auction\_Items ai ON a.auction\_id = ai.auction\_id  JOIN Products p ON ai.product\_id = p.product\_id;  -- (информация о заказах и соответствующим им товарах)  CREATE VIEW Order\_Details AS  SELECT o.order\_id, o.order\_date, c.customer\_name, p.product\_name, oi.quantity, oi.price\_per\_item, o.total\_price  FROM Orders o  JOIN Customers c ON o.customer\_id = c.customer\_id  JOIN Order\_Items oi ON o.order\_id = oi.order\_id  JOIN Products p ON oi.product\_id = p.product\_id;  -- (сколько товаров было продано на аукционе)  CREATE VIEW Auction\_Sales AS  SELECT a.auction\_id, a.auction\_name, COUNT(DISTINCT ab.auction\_bid\_id) AS num\_sales  FROM Auctions a  JOIN Auction\_Items ai ON a.auction\_id = ai.auction\_id  JOIN Auction\_Bids ab ON ai.auction\_item\_id = ab.auction\_item\_id  WHERE ab.is\_win = TRUE  GROUP BY a.auction\_id, a.auction\_name; |

Листинг 6 – views.sql