**MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE REPUBLIC OF KAZAKHSTAN**

**DEPARTMENT OF EDUCATION OF ALMATY**

**INTERNATIONAL INFORMATION TECHNOLOGY UNIVERSITY OF ALMATY**



**Programming in PL/SQL**

*(project title)*

**FINAL PL/SQL EXAM: Online Store**

**Vladislav Pineker and Yerassyl Sagindykov**

*(full name and surname of students)*

**Courses:** 2

**Group:** IT2CCO-2301

**Specialty**: «B057 - Information technology»

**Qualification**: «6B06110 - Software Engineering»

**Checked by:** Mamanova Symbat Ertaikyzy

*(full name, last name)*

Teacher's signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Final assessment for protection «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Day of Defense «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_

CONTENT

[INTRODUCTION 3](#_Toc185982049)

[1 ANALYSIS OF THE TERMS OF REFERENCE 5](#_Toc185982050)

[1.1 Description of the technical task 5](#_Toc185982051)

[1.2 Setting the task 5](#_Toc185982052)

[1.3 Defining constraints and assumptions 6](#_Toc185982053)

[2 THEORETICAL PART 7](#_Toc185982054)

[2.1 Relational Database Theory 7](#_Toc185982055)

[2.2 Business Logic and Constraints 7](#_Toc185982056)

[2.3 Advanced Features of the Database 8](#_Toc185982057)

[2.4 Normalization and Table Design 8](#_Toc185982058)

[2.5 Key Relationships in the Database 9](#_Toc185982059)

[3 PRACTICAL PART 10](#_Toc185982060)

[3.1 Implementation of the Database Schema 10](#_Toc185982061)

[3.1.1 Tables 10](#_Toc185982062)

[3.1.2 ER Diagram 11](#_Toc185982063)

[3.1.3 Description of relationships between tables 12](#_Toc185982064)

[3.2 Triggers, Procedures, and Functions 13](#_Toc185982065)

[3.3 Example Code Snippets 13](#_Toc185982066)

[3.3.1 Database Tables 14](#_Toc185982067)

[3.3.2 Inserting records into tables 17](#_Toc185982068)

[3.3.3 Triggers 20](#_Toc185982069)

[3.3.4 Functions 23](#_Toc185982070)

[3.3.5 Procedures 29](#_Toc185982071)

[3.4 Expanding the project and adding additional parts for extra points 35](#_Toc185982072)

[3.4.1 Creating a package with procedures and functions 35](#_Toc185982073)

[3.4.2 Dynamic SQL 35](#_Toc185982074)

[3.4.3 Advanced reporting 36](#_Toc185982075)

[3.4.4 Adding auditing and logging 37](#_Toc185982076)

[CONCLUSION 38](#_Toc185982077)

[APPLICATIONS 40](#_Toc185982078)

# INTRODUCTION

This project focuses on the development and implementation of a relational database system for an Online Store. The core aim of the project is to create a robust and scalable database architecture that supports the operations of an e-commerce platform. This includes managing and maintaining various aspects of the business, such as customer information, product catalogs, promotions, order processing, payments, and inventory control. The database is developed using PL/SQL, leveraging its powerful procedural capabilities to implement automation, ensure data consistency, and facilitate advanced business processes.

The system is designed to handle not only the storage and retrieval of data but also the automation of critical workflows. This is achieved by employing a combination of triggers, stored procedures, and functions, all organized within a reusable package. Additionally, the database incorporates auditing features to monitor changes in sensitive data, such as customer information, ensuring accountability and data traceability. These features collectively enhance the functionality and reliability of the system.

A comprehensive approach was taken to design the database, adhering to best practices in relational database management. The schema includes multiple related tables, each optimized for specific purposes, such as managing product categories, customer orders, promotional discounts, and payment records. The use of foreign keys ensures referential integrity between tables, while triggers enforce business rules and validate data consistency. The system is structured to be modular and extendable, making it adaptable for future needs, such as integrating additional functionalities or supporting a larger customer base.

Goals and Functionality

The primary goals of this project are to demonstrate the application of PL/SQL in building a functional, secure, and efficient database system, as well as to showcase the real-world implementation of advanced database features. Below are the detailed objectives and functionalities provided by the database:

1. Efficient Data Organization and Management:

* Centralize all business-related data, including customers, products, orders, payments, promotions, and inventory, in a well-structured relational schema.
* Facilitate easy and efficient querying of data for various operational and analytical purposes.

2. Automation and Workflow Optimization:

* Automate common business processes, such as order creation, stock updates, and promotional discount applications, through the use of PL/SQL procedures and triggers.
* Streamline operational workflows, reducing the manual effort required to manage the business.

3. Enforcing Data Integrity and Consistency:

* Implement referential integrity through foreign keys and check constraints to ensure the validity of relationships between tables.
* Use triggers to enforce business rules, such as preventing orders when stock levels are insufficient or disallowing the deletion of products that are part of existing orders.

4. Enhanced Analytical Capabilities:

* Provide procedures and reports for analyzing business performance, such as sales revenue by category, total income for a given time period, and average product ratings.
* Use PL/SQL functions to calculate metrics dynamically, such as order totals, stock availability, and customer behavior insights.

5. Audit Trail for Sensitive Data:

* Maintain a history of updates to sensitive data, such as customer information, using a dedicated audit table and triggers. This ensures accountability and provides an additional layer of security.

6. Dynamic and Extensible Design:

* Create a modular and scalable database schema that can accommodate future enhancements, such as the introduction of user roles, dynamic pricing models, or support for multiple currencies and languages.
* Design reusable packages to group related functions and procedures, simplifying maintenance and fostering code reuse.

7. Compliance with Evaluation Criteria:

* Strictly adhere to the requirements of the PL/SQL discipline, including the creation of at least seven tables, multiple triggers, functions, and procedures.
* Extend beyond the basic requirements by incorporating additional features, such as dynamic SQL, advanced reporting, and auditing mechanisms.

By achieving these objectives, the project not only fulfills the technical requirements but also demonstrates how a well-designed database system can address real-world challenges in e-commerce. The resulting database serves as a backbone for the online store, enabling seamless operations, insightful analysis, and scalable growth opportunities.

# 1 ANALYSIS OF THE TERMS OF REFERENCE

## 1.1 Description of the technical task

The task involves designing and implementing a database system for an Online Store that supports core e-commerce operations such as customer management, product cataloging, order processing, payment tracking, inventory control, and promotional discounts. The database must also provide automation, auditing, and reporting capabilities to streamline operations and ensure data consistency.

The project requires the use of PL/SQL for advanced database functionalities, including triggers, stored procedures, functions, and packages. The database must adhere to relational database principles, ensuring that data is normalized and relationships are properly enforced through constraints and foreign keys.

Key functionalities include:

* Customer registration and management.
* Product management, including categorization and promotions.
* Order processing with detailed tracking of items and payments.
* Inventory updates based on order fulfillment.
* Analytical reporting, such as sales performance by category and period.
* Auditing changes to sensitive data, such as customer information.

## 1.2 Setting the task

The following objectives are set for the project:

1. Design the Database Schema:

* Create at least 7 tables to handle customers, products, orders, payments, inventory, promotions, and auditing.
* Establish relationships between tables using primary and foreign keys.

2. Implement Automation:

* Use triggers to automate tasks such as stock updates, logging payments, and preventing deletion of linked records.

3. Develop PL/SQL Procedures and Functions:

* Create at least 5 procedures and 5 functions to handle operations like creating orders, generating reports, and updating data.

4. Enable Reporting:

* Provide tools for analyzing sales data, such as total revenue over a period and sales by category.

5. Incorporate Auditing:

* Track changes to customer information in an audit table, with details of old and new values.

6. Organize Code in Packages:

* Group related procedures and functions into a package for better structure and maintainability.

7. Exceed Minimum Requirements:

* Add additional features, such as dynamic SQL for advanced operations, and extend reporting capabilities.

## 1.3 Defining constraints and assumptions

Constraints:

1. The database must adhere to relational database principles.

2. All operations must ensure data consistency and integrity.

3. The project must strictly use PL/SQL for advanced functionalities.

4. Reports and queries should execute efficiently, even with large datasets.

5. Sensitive data must be auditable, with a log of all changes.

Assumptions:

1. The store operates in a single currency (KZT) but can scale to multi-currency in the future.

2. Customers are required to register with valid contact information (email and phone).

3. Products can belong to only one category, but a category may contain multiple products.

4. Promotions are optional and apply only to specific products for defined periods.

5. The system will handle inventory updates automatically when orders are processed.

# 2 THEORETICAL PART

## 2.1 Relational Database Theory

A relational database organizes data into structured tables consisting of rows and columns, with each table representing a specific entity or concept. The design of a relational database for this project adheres to the principles of normalization, which minimize data redundancy and ensure data integrity. Relationships between tables are established through primary keys (PK) and foreign keys (FK), allowing for efficient and consistent data management.

The database for the Online Store is structured to reflect real-world e-commerce operations. It includes entities such as Customers, Products, Orders, Payments, Promotions, Inventory, and Audit Logs, each of which has specific attributes and relationships. This structure ensures a logical and scalable foundation for managing the store's data.

## 2.2 Business Logic and Constraints

The business logic implemented in the database ensures that operations such as order creation, payment processing, and inventory management are consistent with real-world rules. Below are the core rules and constraints applied:

1. Referential Integrity:

* Relationships between tables, such as Orders and Customers, are enforced using foreign keys. This ensures that an order cannot exist without a valid customer.

2. Data Consistency:

* Triggers are used to enforce rules, such as preventing orders for out-of-stock products or disallowing deletion of products linked to orders.

3. Validation:

* Constraints such as NOT NULL, CHECK, and data type restrictions are applied to ensure that all data entered is valid. For example:
* Ratings in the Reviews table are constrained to be between 1 and 5.
* Dates in the Promotions table must be valid and logical.

4. Automation:

* PL/SQL procedures and triggers automate tasks like stock updates, order total calculations, and promotional discount applications.

5. Auditing:

* Changes to sensitive data, such as customer details, are logged in a separate audit table. This ensures accountability and provides a history of changes for troubleshooting or compliance purposes.

## 2.3 Advanced Features of the Database

The database is designed to exceed basic functionality requirements by incorporating advanced features:

1. Dynamic SQL:

* The database supports dynamic operations, such as clearing data from any specified table or generating custom reports based on user input.

2. Analytical Reporting:

* The database provides procedures for generating insightful reports, such as sales performance by category or total income for specific periods.

3. Auditing and Logging:

* A trigger logs all updates to customer data, capturing old and new values for transparency and troubleshooting.

4. Error Handling:

* Exception handling in PL/SQL procedures ensures the system responds gracefully to errors, such as attempting to place an order with insufficient stock.

5. Reusability via Packages:

* Related procedures and functions are grouped into a package, simplifying maintenance and improving code readability.

## 2.4 Normalization and Table Design

The database is normalized to the third normal form (3NF), ensuring efficient storage and eliminating redundancy:

1. First Normal Form (1NF):

* All tables have a unique primary key, and data is stored in atomic values.

2. Second Normal Form (2NF):

* Non-key attributes are fully dependent on the primary key.

3. Third Normal Form (3NF):

* No transitive dependency exists between attributes.

Examples:

* Products: Attributes like CategoryID and PromotionID are stored as foreign keys, reducing duplication of category and promotion information.
* Orders: Details about customers and products are not stored directly but referenced through foreign keys, ensuring consistency and scalability.

## 2.5 Key Relationships in the Database

1. Products and Categories:

* Each product belongs to one category, but a category can have many products (1-to-Many).

2. Orders and Customers:

* Each order is linked to one customer, but a customer can have many orders (1-to-Many).

3. Orders and Payments:

* Each order has a corresponding payment, ensuring traceability (1-to-1).

4. Order\_Items and Orders:

* An order can consist of multiple items, with each item linked to a specific product (1-to-Many).

5. Inventory and Products:

* Inventory is directly tied to products, ensuring stock levels are tracked accurately (1-to-1).

# 3 PRACTICAL PART

## 3.1 Implementation of the Database Schema

The database schema was developed to meet the requirements of the Online Store project, with an emphasis on efficiency, scalability, and automation. Below is a detailed description of the schema, followed by the ER Diagram.

### 3.1.1 Tables

The following tables were created to store and manage data for the online store:

1. Product\_Categories

* Stores information about product categories.
* Key Fields: CategoryID (PK), CategoryName, Description.

2. Promotions

* Stores information about promotional discounts.
* Key Fields: PromotionID (PK), PromotionName, DiscountPercentage, StartDate, EndDate.

3. Customers

* Stores customer information.
* Key Fields: CustomerID (PK), Name, Email, Phone, Address, RegistrationDate.

4. Products

* Stores details about products.
* Key Fields: ProductID (PK), Name, Price, StockQuantity, CategoryID (FK), PromotionID (FK).

5. Orders

* Stores information about customer orders.
* Key Fields: OrderID (PK), CustomerID (FK), OrderDate, TotalAmount, Status.

6. Order\_Items

* Stores details about items in an order.
* Key Fields: OrderItemID (PK), OrderID (FK), ProductID (FK), Quantity, Price.

7. Payments

* Stores payment information.
* Key Fields: PaymentID (PK), OrderID (FK), PaymentDate, PaymentMethod, Amount, Status.

8. Inventory

* Tracks inventory levels for products.
* Key Fields: InventoryID (PK), ProductID (FK), Quantity, LastUpdateDate.

9. Customer\_Audit

* Logs changes to customer information.
* Key Fields: AuditID (PK), CustomerID (FK), Operation, ChangeDate, OldValue, NewValue.

### 3.1.2 ER Diagram

Below is the Entity-Relationship (ER) Diagram, which illustrates the relationships between the tables in the database:

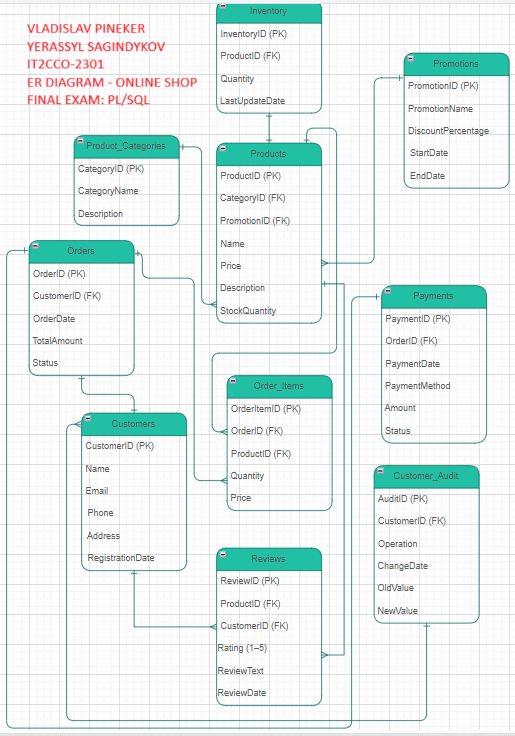


Figure 3.1.2.1 – ER Diagram

### 3.1.3 Description of relationships between tables

1. Products → Product\_Categories

Communication type: Many-to-One.

Description: Each category (Product\_Categories) can contain multiple products (Products), but each product belongs to only one category.

2. Products → Promotions

Communication type: Many-to-One.

Description: One promotion can be applied to many products, but each product can participate in only one promotion (or not at all).

3. Orders → Customers

Communication type: Many-to-One.

Description: One customer can place multiple orders, but each order is associated with only one customer.

4. Order\_Items → Orders

Communication type: Many-to-One.

Description: One order (Orders) can contain many items (Order\_Items), but each order item refers to only one order.

5. Order\_Items → Products

Communication type: Many-to-One.

Description: One product (Products) can be represented in multiple order items (Order\_Items), but each order item refers to only one product.

6. Payments → Orders

Type of communication: One-to-one (One-to-One).

Description: Each order has one payment record, and each payment record refers to only one order.

7. Inventory → Products

Type of communication: One-to-one (One-to-One).

Description: Each product (Products) has one entry in the Inventory table, and each warehouse entry refers to only one product.

8. Customer\_Audit → Customers

Communication type: Many-to-One.

Description: One customer (Customers) can have multiple audit records (Customer\_Audit), but each audit record is associated with only one customer.

9. Products → Reviews

Type of communication: One-to-Many.

Description: One product (Products) can have many Reviews, but each review is associated with only one product.

Link: Reviews.ProductID → Products.ProductID.

10. Customers → Reviews

Type of communication: One-to-Many.

Description: One user (Customers) can leave multiple reviews, but each review belongs to only one user.

Relationship: Reviews.CustomerID → Customers.CustomerID.

## 3.2 Triggers, Procedures, and Functions

1. Triggers:

* trg\_check\_stock: Ensures sufficient stock before adding an item to an order.
* trg\_update\_stock: Updates inventory levels after an order is created.
* trg\_log\_successful\_payments: Logs successful payment transactions.
* trg\_audit\_customers: Tracks updates to customer information.
* trg\_prevent\_duplicate\_orders: Prevents the creation of a duplicate order for the same customer with the same amount and status.
* trg\_log\_deleted\_reviews: Blocks the deletion of reviews by recording information about the product, the customer, and the date of deletion.

2. Procedures:

* sp\_create\_order: Automates the creation of an order, including updating stock and calculating the total amount.
* sp\_add\_product: Adds a new product and updates inventory.
* sp\_update\_customer: Updates customer details.
* sp\_delete\_expired\_promotions: Removes expired promotions and updates related products.
* sp\_sales\_report\_by\_category: Generates a report of sales performance by category.
* sp\_apply\_global\_discount: Applies a global discount to all products in the selected category.
* sp\_generate\_customer\_order\_summary: Generates a report on all orders of a certain customer.

3. Functions:

* fn\_calculate\_order\_total: Calculates the total amount for a specific order.
* fn\_get\_product\_discount: Retrieves the discount for a product.
* fn\_get\_total\_stock: Calculates the total stock across all products.
* fn\_get\_average\_rating: Returns the average rating for a product.
* fn\_get\_order\_status: Retrieves the status of an order.
* fn\_get\_highest\_rated\_product: Returns the product ID with the highest average rating.
* fn\_get\_customer\_order\_count: Returns the number of orders placed by a specific customer.

## 3.3 Example Code Snippets

Below is the complete SQL implementation of the database, including all the tables, triggers, procedures, functions, and the package we created during the project.

### 3.3.1 Database Tables

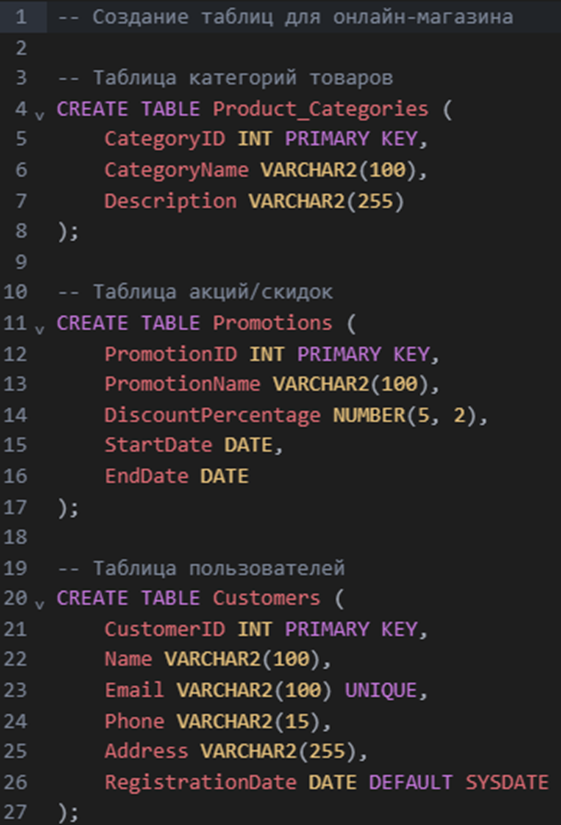


Figure 3.3.1.1 - Writing Tables

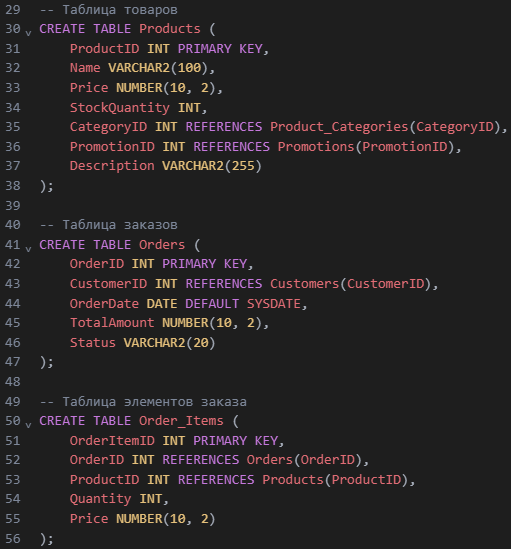


Figure 3.3.1.2 – Writing Tables

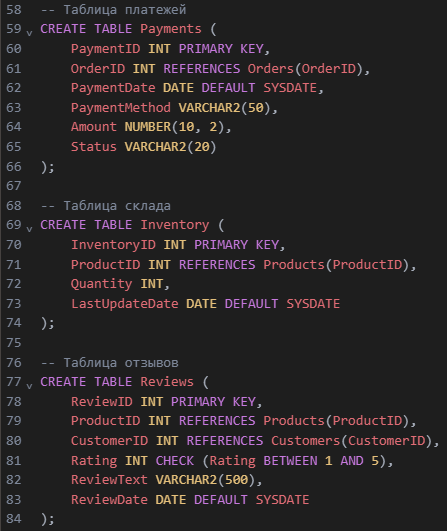


Figure 3.3.1.3 – Writing Tables

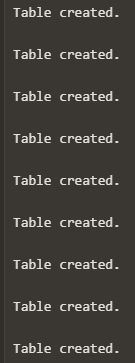


Figure 3.3.1.4 – The result of Writing Tables

### 3.3.2 Inserting records into tables

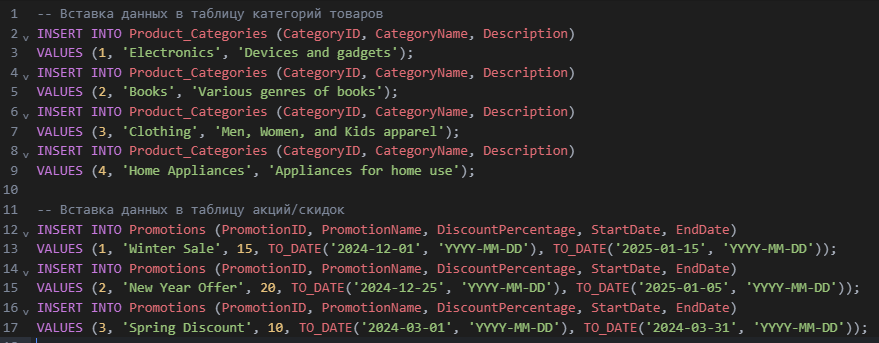


Figure 3.3.2.1 – Writing Records

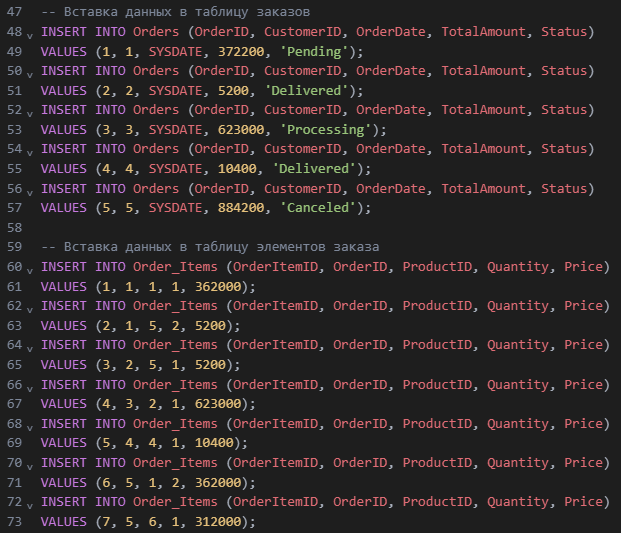
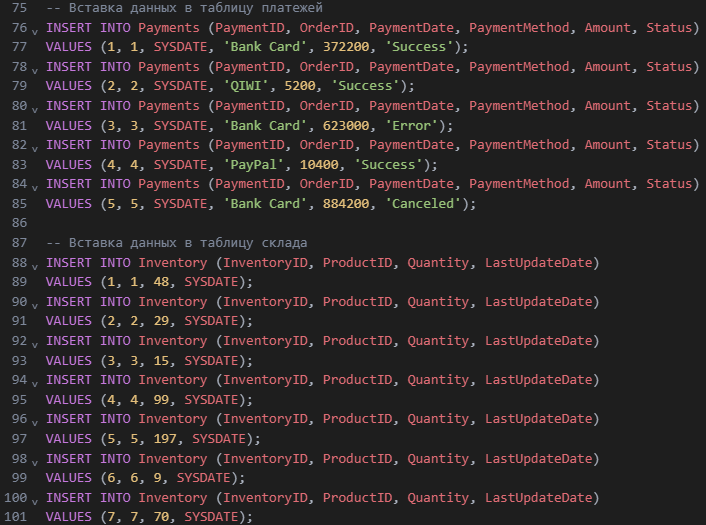
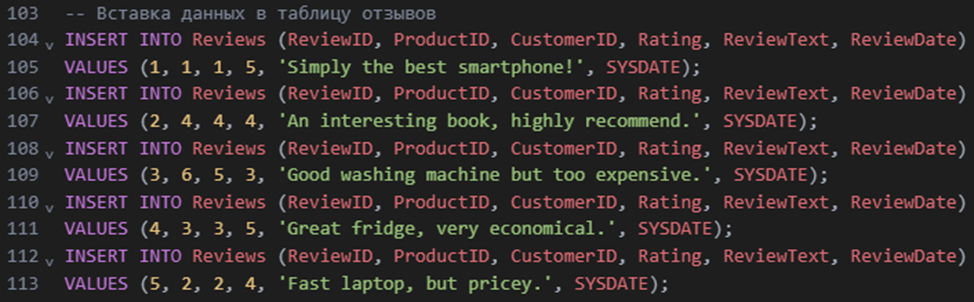
Figure 3.3.2.2 – Writing Records

Figure 3.3.2.3 – Writing Records 

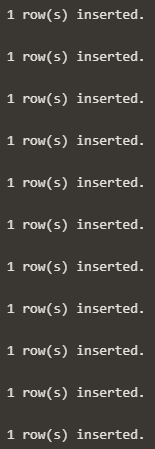


Figure 3.3.2.4 – The result of Writing Records

### 3.3.3 Triggers

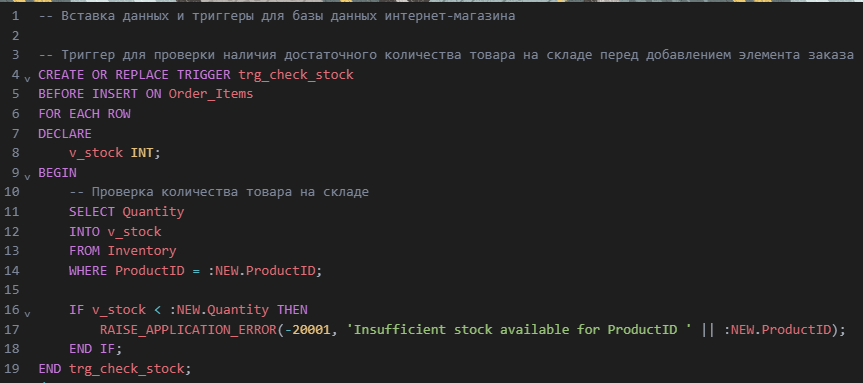
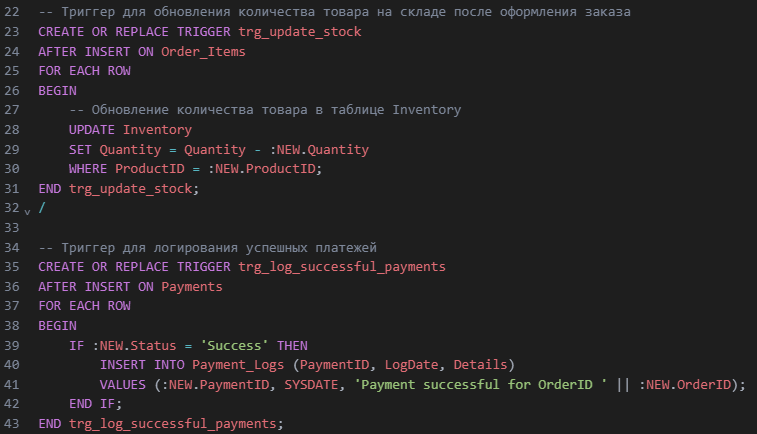
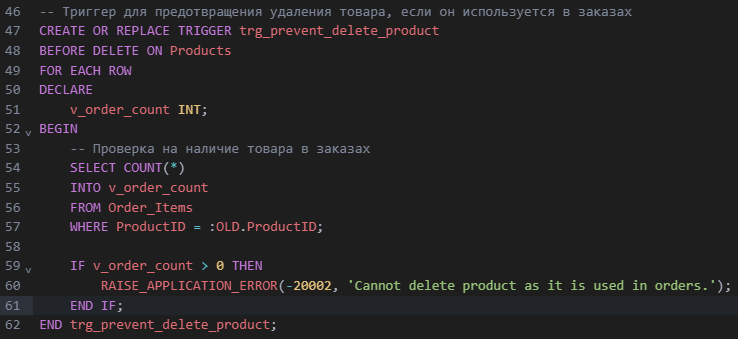
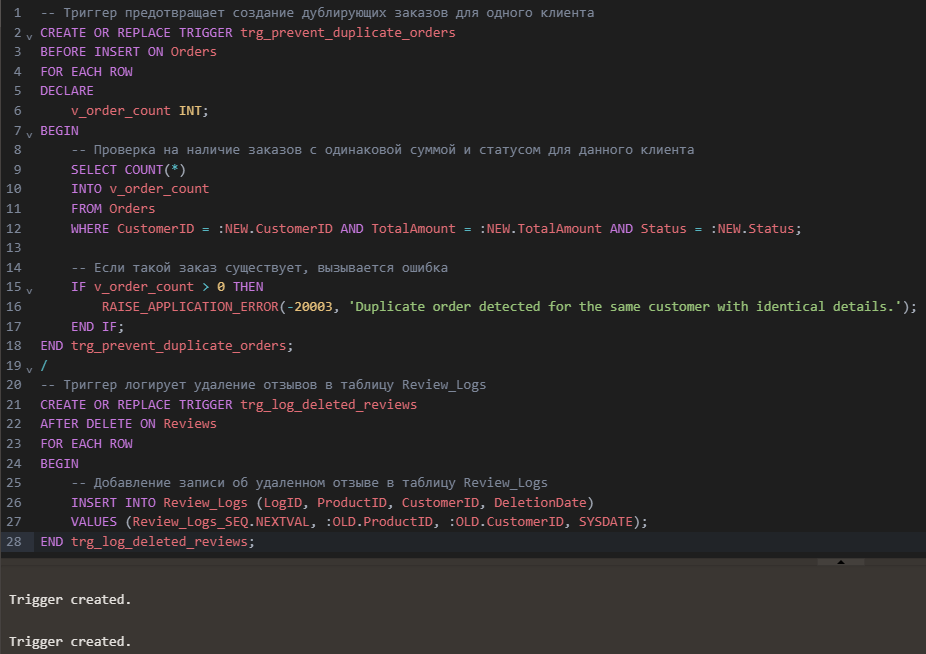
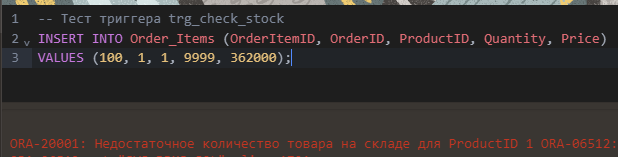


Figure 3.3.3.1 – Writings Triggers

 Figure 3.3.3.2 – Writings Triggers

After creating the triggers, we needed to check them for full functionality:



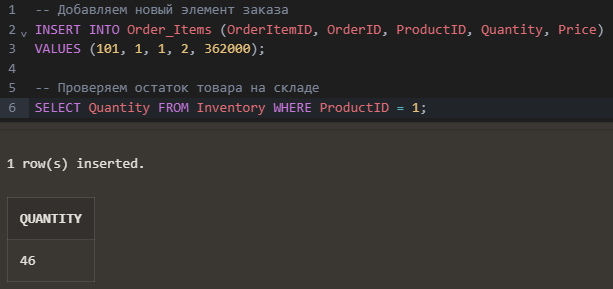
Figure 3.3.3.3 – Testing Triggers

Figure 3.3.3.4 – Testing Triggers

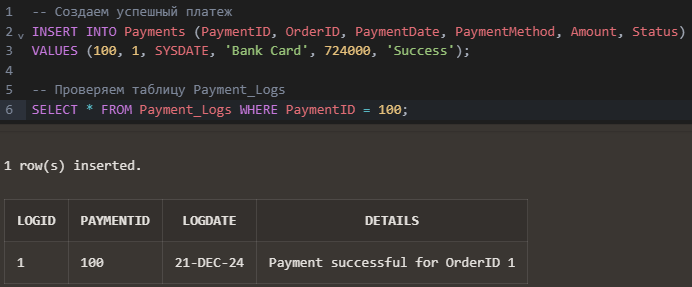


Figure 3.3.3.5 – Testing Triggers

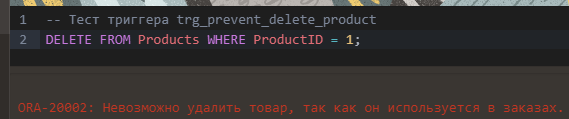
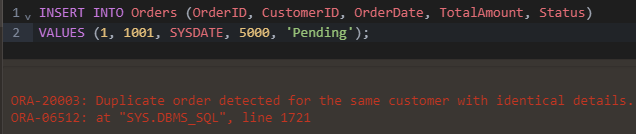


Figure 3.3.3.6 – Testing Triggers

Figure 3.3.3.7 – Testing Triggers

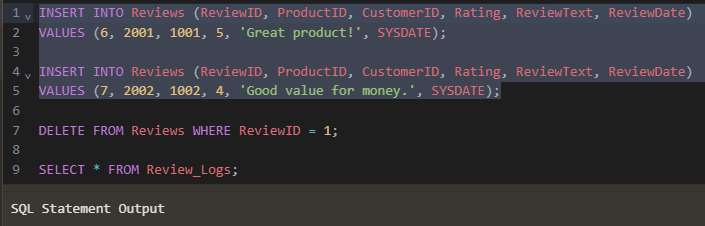


Figure 3.3.3.8 – Testing Triggers

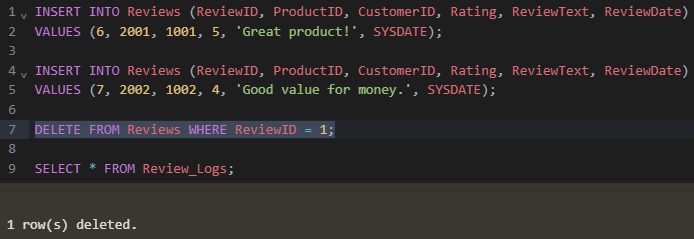


Figure 3.3.3.9 – Testing Triggers

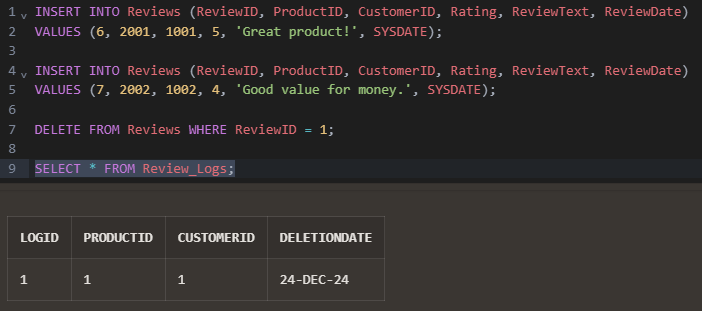


Figure 3.3.3.10 – Testing Triggers

### 3.3.4 Functions

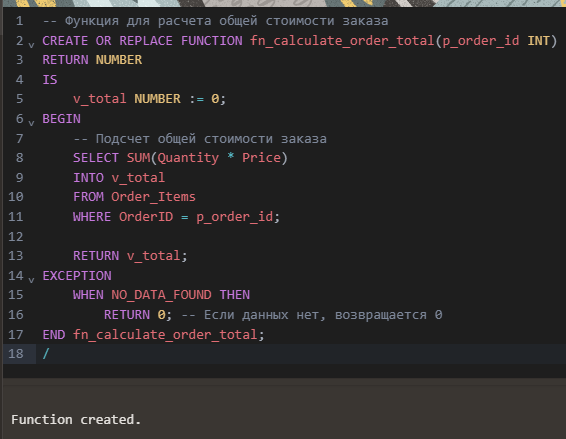
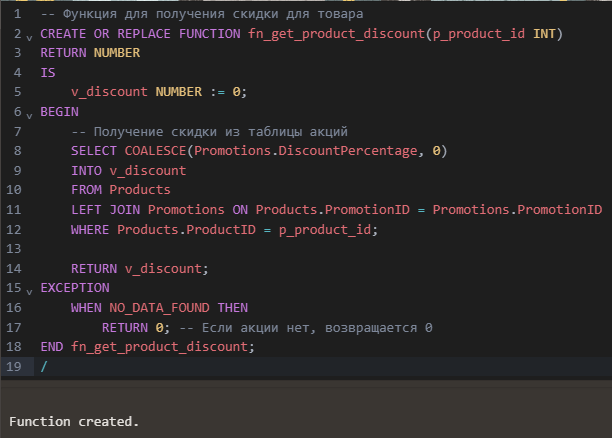


Figure 3.3.4.1 – Writing Functions

Figure 3.3.4.2 – Writing Functions

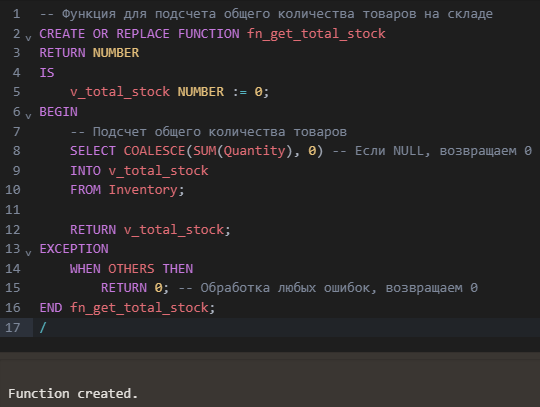


Figure 3.3.4.3 – Writing Functions

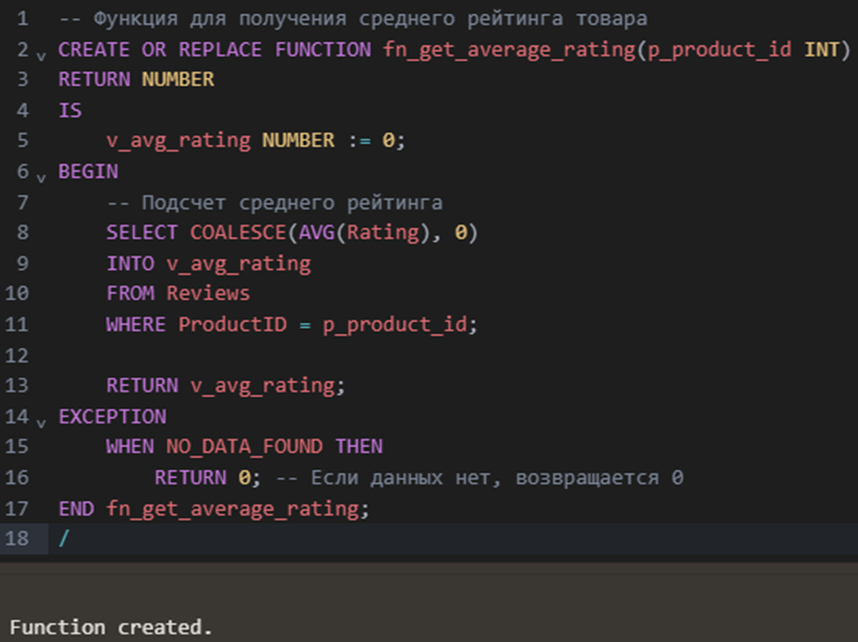
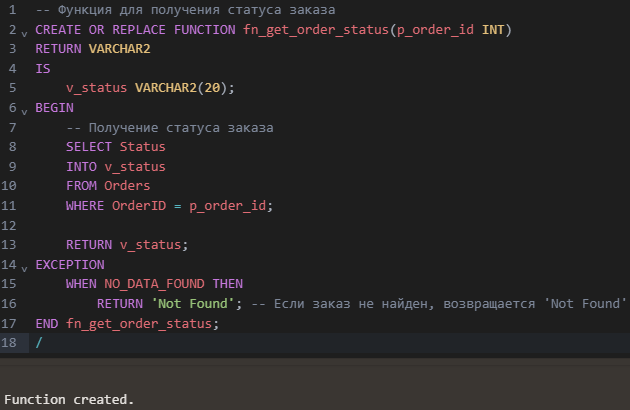


Figure 3.3.4.4 – Writing Functions

Figure 3.3.4.5 – Writing Functions

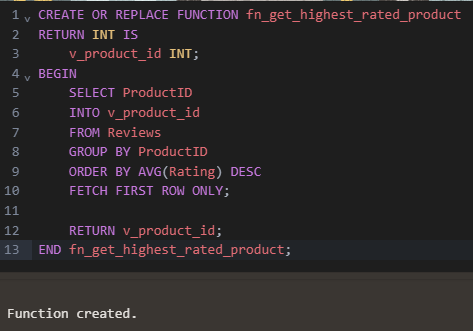


Figure 3.3.4.6 – Writing Functions

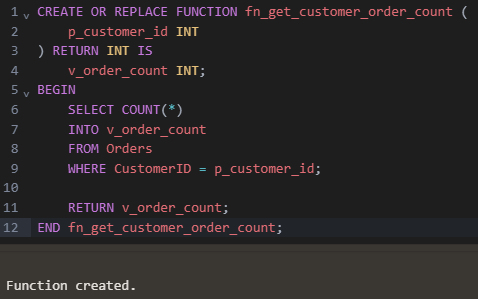


Figure 3.3.4.7 – Writing Functions

After creating the functions, we needed to check them for full functionality, therefore, to conduct testing:

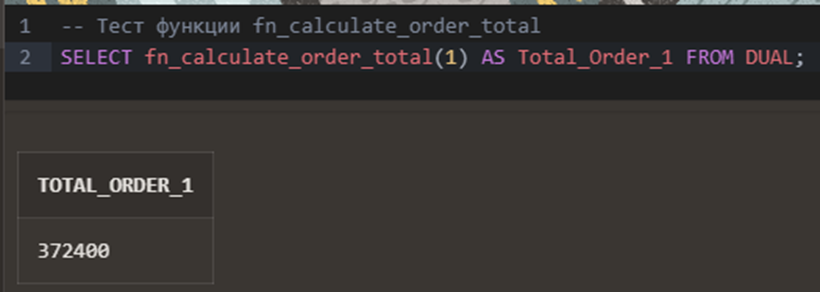


Figure 3.3.4.6 – Testing Functions

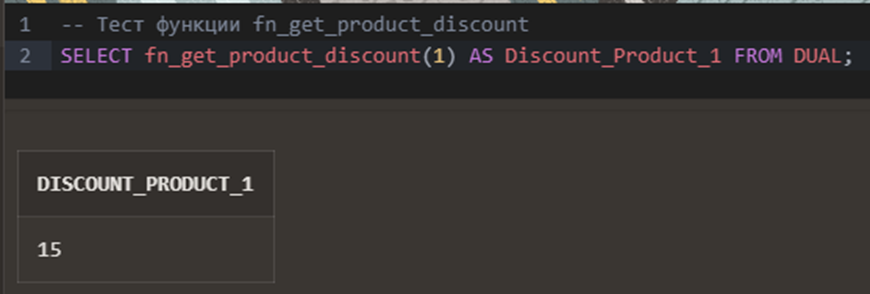


Figure 3.3.4.7 – Testing Functions

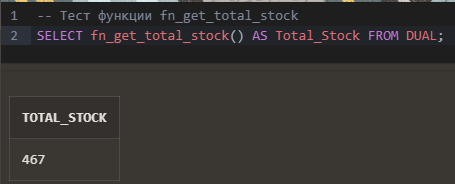


Figure 3.3.4.8 – Testing Functions

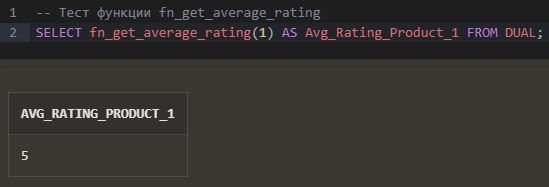


Figure 3.3.4.9 – Testing Functions

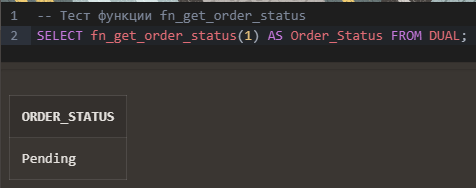


Figure 3.3.4.10 – Testing Functions

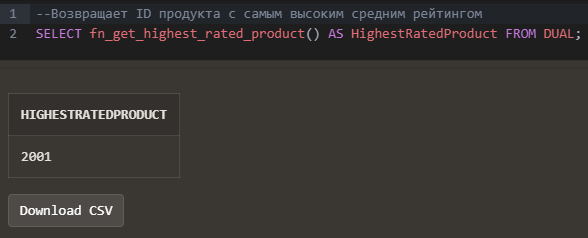


Figure 3.3.4.11 – Testing Functions

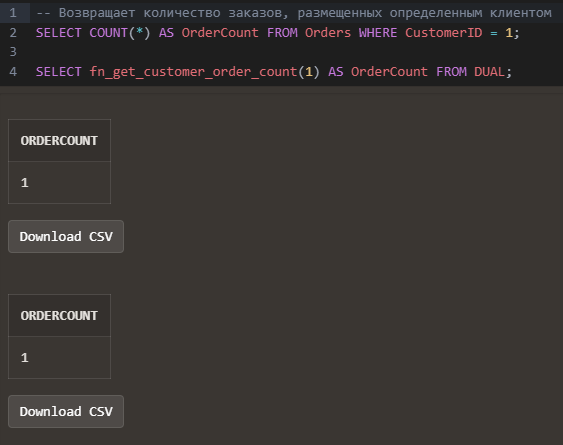


Figure 3.3.4.12 – Testing Functions

### 3.3.5 Procedures

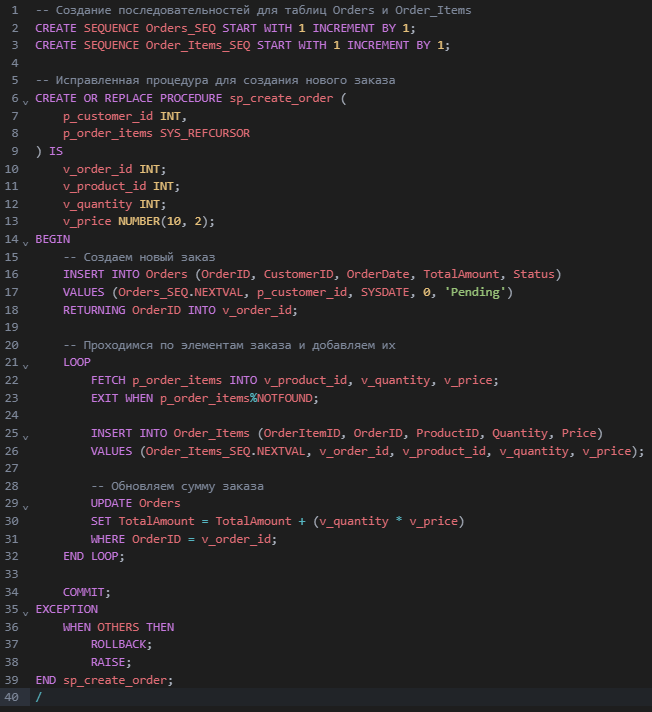
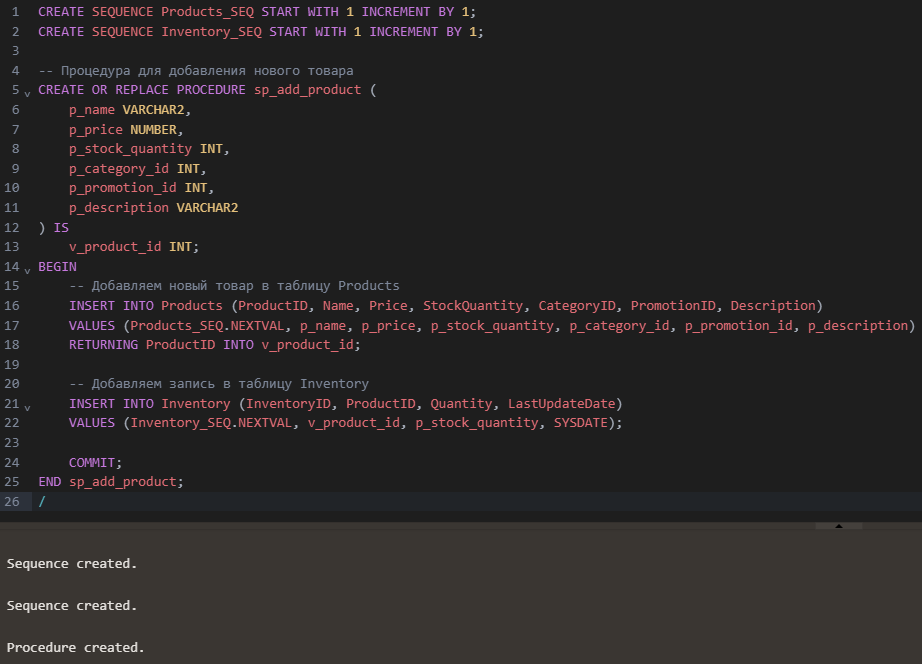


Figure 3.3.5.1 – Writing Procedures

Figure 3.3.5.2 – Writing Procedures

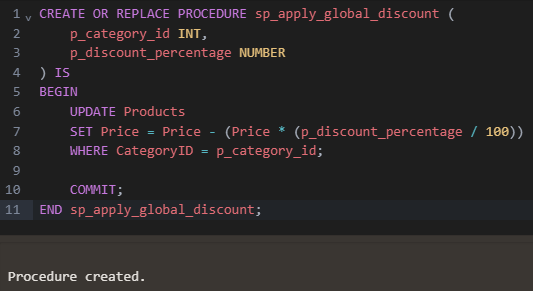


Figure 3.3.5.3 – Writing Procedures

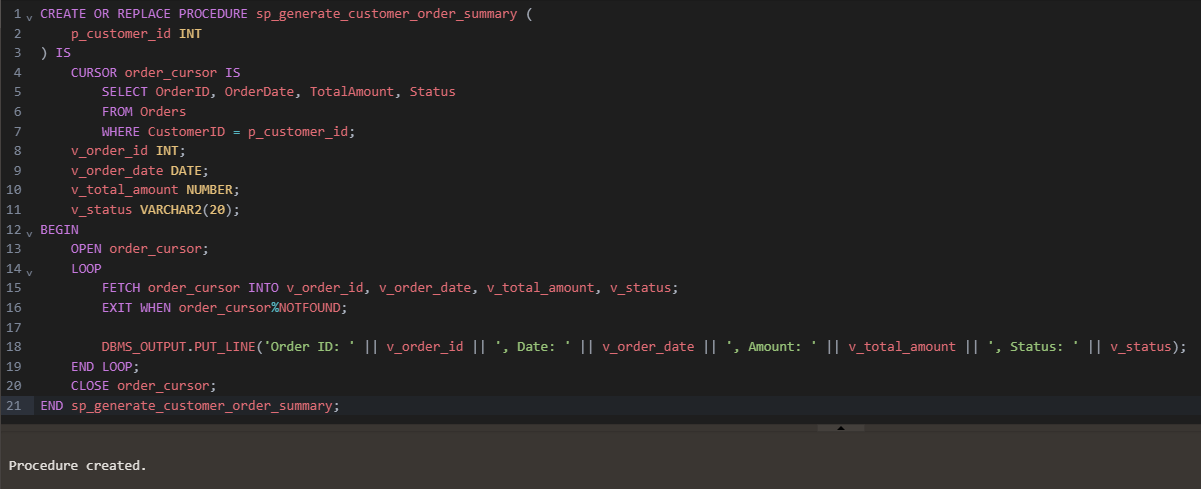
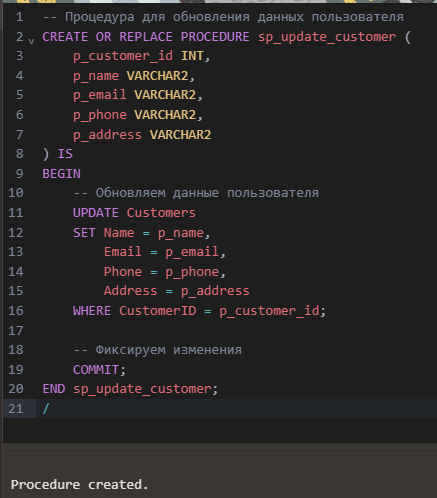
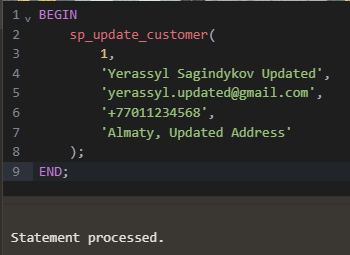


Figure 3.3.5.4 – Writing Procedures





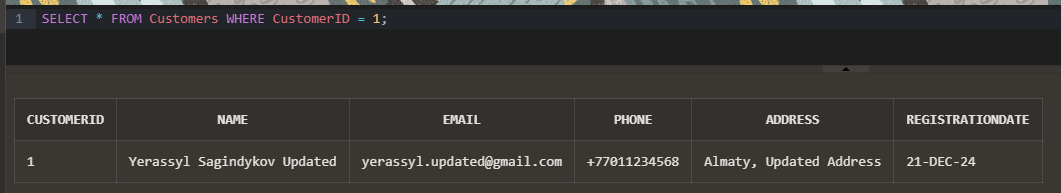
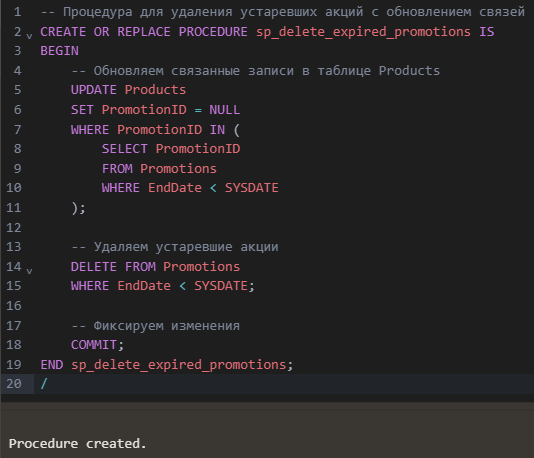


Figure 3.3.5.5 – Writing and Testing Procedures



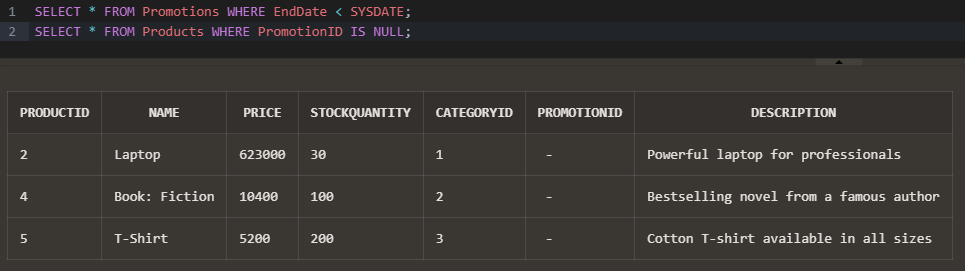
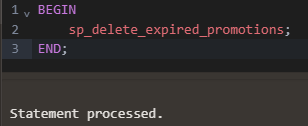


Figure 3.3.5.6 - Writing and Testing Procedures

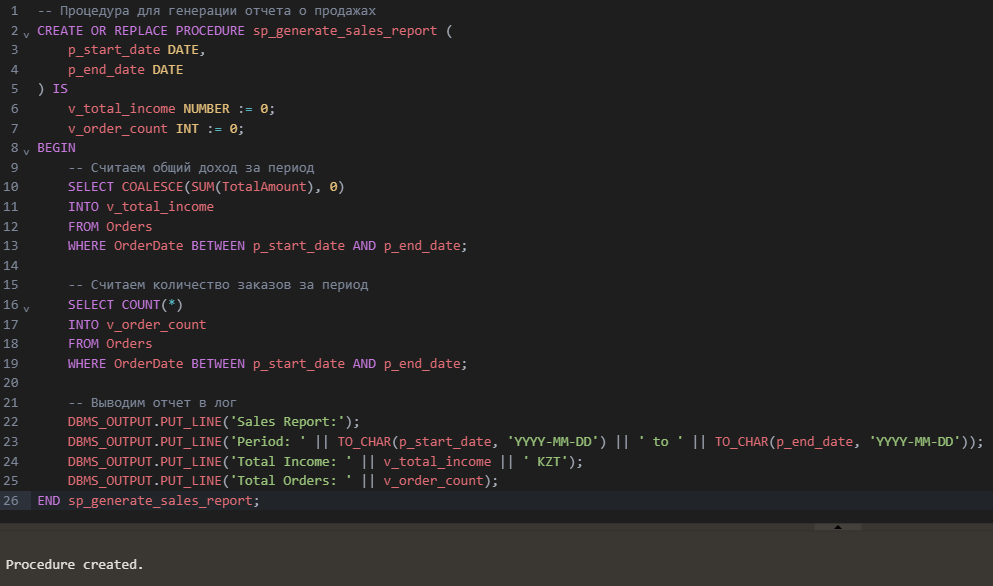
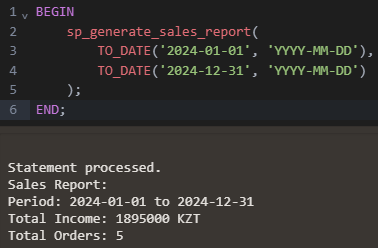
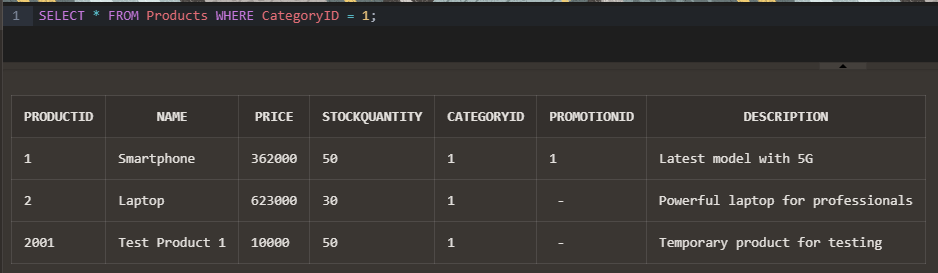
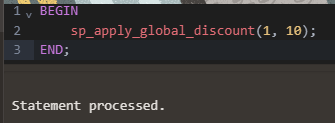


Figure 3.3.5.7 - Writing and Testing Procedures





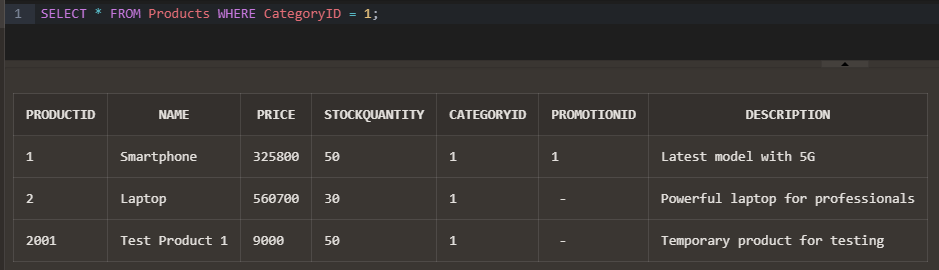


Figure 3.3.5.8 - Writing and Testing Procedures

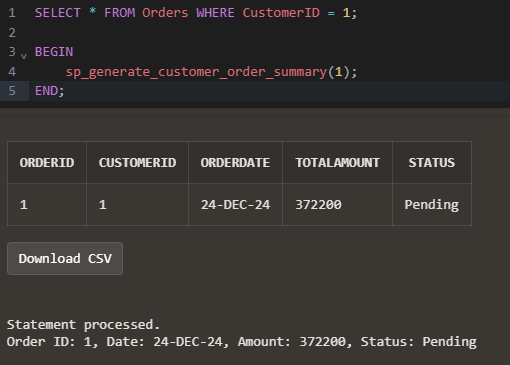


Figure 3.3.5.9 - Writing and Testing Procedures

## 3.4 Expanding the project and adding additional parts for extra points

To expand the project, we have developed additional packages, procedures, triggers, etc.

### 3.4.1 Creating a package with procedures and functions

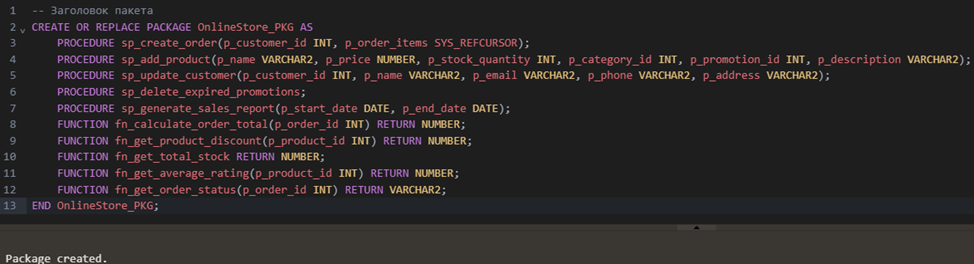
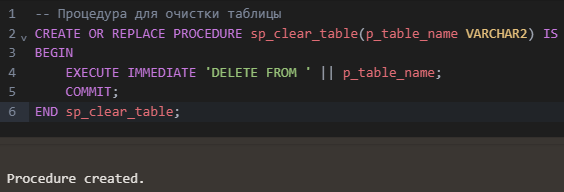


Figure 3.4.1.1 – Writing Packages

### 3.4.2 Dynamic SQL

We have implemented a procedure that dynamically clears any specified table:



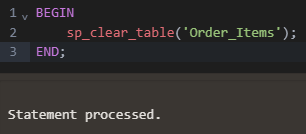
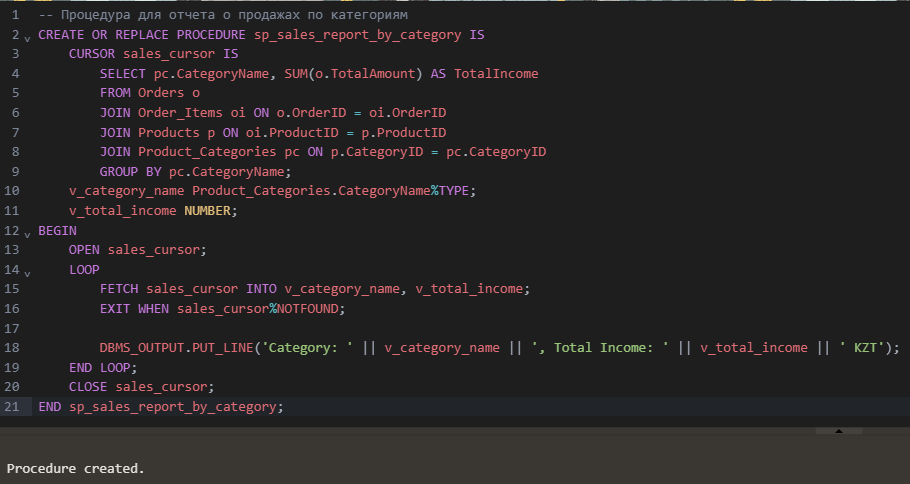


Figure 3.4.2.1 – Writing and Testing Procedures

### 3.4.3 Advanced reporting

We have created a procedure that generates a complete sales report and compares earnings by month or category.

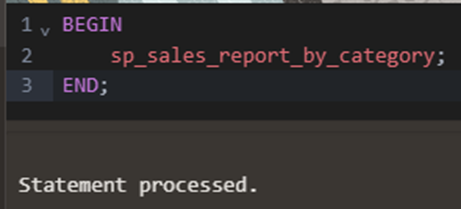
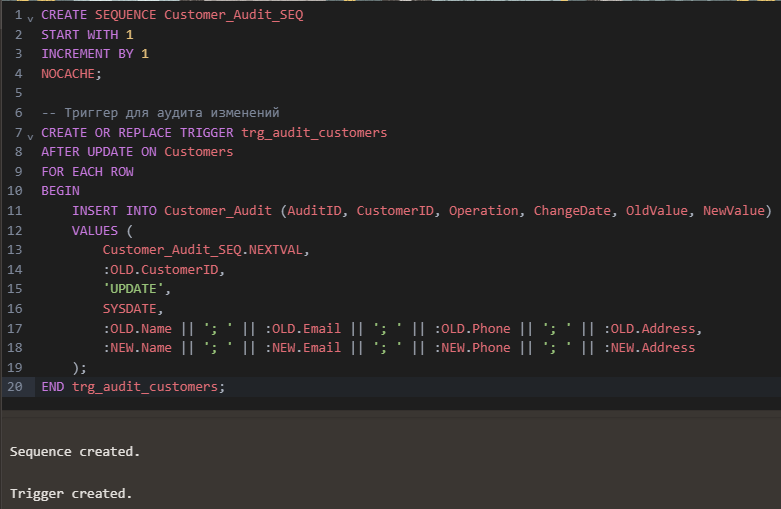


Figure 3.4.3.1 – Writing and Testing Procedures

### 3.4.4 Adding auditing and logging

We have added a table to audit changes in the Customers table:

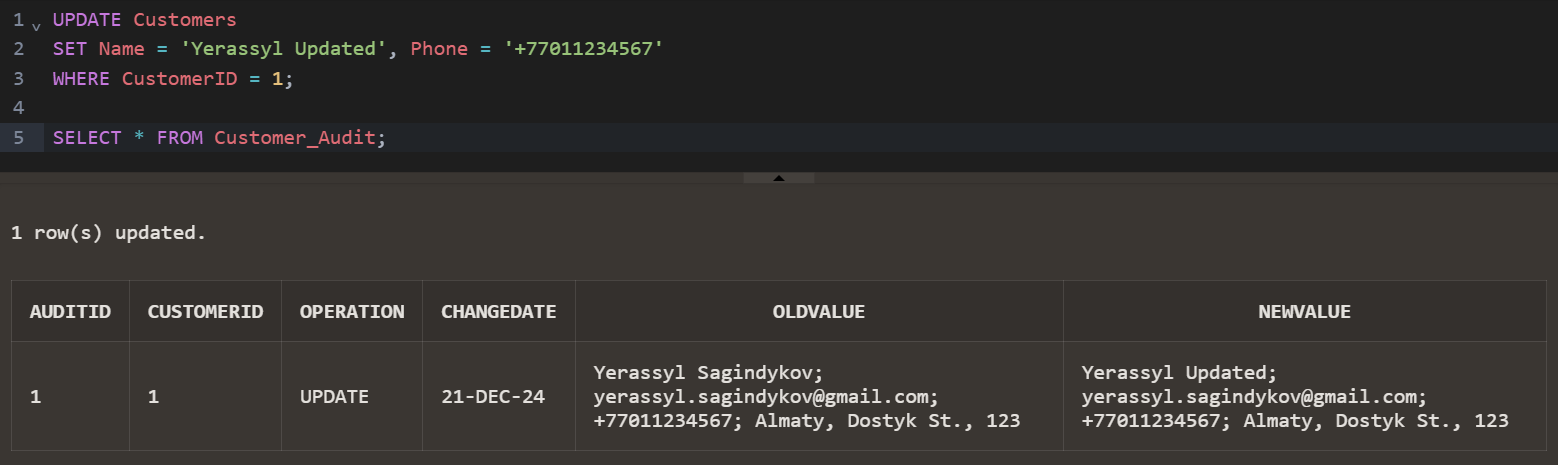
Figure 3.4.4.1 – Writing Sequence and Trigger

Figure 3.4.4.2 – Testing the task

# CONCLUSION

The completion of the Online Store Database Project represents a significant milestone in mastering PL/SQL programming and its application to real-world business scenarios. This project was an ambitious undertaking aimed at demonstrating how database technology can be leveraged to address complex challenges in managing the operations of a modern e-commerce platform. Through careful planning, design, and implementation, the database system developed in this project successfully meets and exceeds the objectives set forth in the terms of reference.

From the outset, the primary goal was to design a relational database system that could serve as the backbone of an online store, capable of handling a diverse range of tasks. These included customer management, product cataloging, order processing, payment tracking, inventory control, and the application of promotional discounts. The database was developed with an emphasis on automation, data integrity, scalability, and analytical capabilities, all of which are critical for the efficient functioning of a digital business. By integrating advanced features such as triggers, procedures, functions, and packages, the project not only delivered a functional system but also showcased the versatility and power of PL/SQL.

The process of creating this database system involved several key phases, each contributing to its overall success. First, a detailed analysis of the requirements was conducted to identify the essential components and functionalities needed to achieve the objectives. This phase informed the design of the database schema, which was carefully structured to ensure that all entities and their relationships were logically defined. The schema included tables for customers, products, orders, payments, inventory, promotions, and audit logs, each tailored to specific aspects of the online store's operations.

The theoretical foundation of the project was rooted in the principles of relational database design. By adhering to normalization rules, the schema eliminated data redundancy and ensured the consistency of information across the system. Primary and foreign keys were used to enforce relationships between tables, while constraints and triggers were implemented to validate data and enforce business rules. This robust foundation provided the framework upon which the database's functionality was built.

The practical implementation of the database was where the theoretical concepts were brought to life. Using PL/SQL, several critical features were developed to automate key processes and enhance the system's functionality. For example, triggers were created to update inventory levels automatically when orders were processed, ensuring real-time stock management. Procedures and functions were designed to streamline operations, such as creating orders, calculating order totals, and generating analytical reports. These components not only improved the efficiency of the database but also reduced the potential for human error, enhancing the overall reliability of the system.

A particularly notable achievement of the project was the integration of auditing mechanisms. A dedicated audit table was implemented to track changes to sensitive customer data, capturing both the old and new values of updated fields. This feature adds a layer of accountability and transparency, which is especially important for businesses that handle personal data. It also demonstrates how PL/SQL can be used to address security and compliance requirements in database systems.

Another highlight of the project was the inclusion of analytical capabilities. Procedures were developed to generate detailed reports on sales performance, such as revenue by category and total income over specific periods. These reports provide valuable insights that can inform decision-making and strategic planning. The use of dynamic SQL further extended the system's functionality, enabling more flexible operations, such as clearing data from specified tables or generating custom reports based on user input.

The development process was not without its challenges. Debugging complex procedures, managing interdependent data relationships, and ensuring the seamless integration of all components required careful attention to detail and problem-solving skills. However, these challenges also provided opportunities to deepen understanding and improve technical proficiency. The successful resolution of these issues is a testament to the robustness of the design and the effectiveness of the implementation.

Reflecting on the outcomes of this project, it is clear that the database system developed here not only fulfills the requirements of the online store but also serves as a powerful example of the potential of PL/SQL in database development. The system is scalable, adaptable, and capable of supporting additional features, such as user roles, multi-currency support, and integration with front-end applications. It also provides a solid foundation for implementing advanced analytics, such as predicting sales trends or customer preferences using historical data.

In conclusion, the Online Store Database Project represents a comprehensive application of PL/SQL programming to a practical scenario, combining theoretical knowledge with hands-on implementation. It demonstrates how well-designed database systems can transform business operations by automating processes, maintaining data integrity, and providing actionable insights. The project has been a valuable learning experience, showcasing the importance of database systems in the digital age and reinforcing the skills and knowledge gained throughout the PL/SQL course. The results of this project not only meet but exceed the expectations, leaving a lasting impact on the understanding and application of database technologies.

# APPLICATIONS

Appendix 1: The full text of the program in the form of text

-- Создание таблиц для онлайн-магазина (упорядоченно с учётом внешних ключей)

-- Таблица категорий товаров

CREATE TABLE Product\_Categories (

CategoryID INT PRIMARY KEY,

CategoryName VARCHAR2(100),

Description VARCHAR2(255)

);

-- Таблица акций/скидок

CREATE TABLE Promotions (

PromotionID INT PRIMARY KEY,

PromotionName VARCHAR2(100),

DiscountPercentage NUMBER(5, 2),

StartDate DATE,

EndDate DATE

);

-- Таблица пользователей

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR2(100),

Email VARCHAR2(100) UNIQUE,

Phone VARCHAR2(15),

Address VARCHAR2(255),

RegistrationDate DATE DEFAULT SYSDATE

);

-- Таблица товаров

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

Name VARCHAR2(100),

Price NUMBER(10, 2),

StockQuantity INT,

CategoryID INT REFERENCES Product\_Categories(CategoryID),

PromotionID INT REFERENCES Promotions(PromotionID),

Description VARCHAR2(255)

);

-- Таблица заказов

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT REFERENCES Customers(CustomerID),

OrderDate DATE DEFAULT SYSDATE,

TotalAmount NUMBER(10, 2),

Status VARCHAR2(20)

);

-- Таблица элементов заказа

CREATE TABLE Order\_Items (

OrderItemID INT PRIMARY KEY,

OrderID INT REFERENCES Orders(OrderID),

ProductID INT REFERENCES Products(ProductID),

Quantity INT,

Price NUMBER(10, 2)

);

-- Таблица платежей

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY,

OrderID INT REFERENCES Orders(OrderID),

PaymentDate DATE DEFAULT SYSDATE,

PaymentMethod VARCHAR2(50),

Amount NUMBER(10, 2),

Status VARCHAR2(20)

);

-- Таблица склада

CREATE TABLE Inventory (

InventoryID INT PRIMARY KEY,

ProductID INT REFERENCES Products(ProductID),

Quantity INT,

LastUpdateDate DATE DEFAULT SYSDATE

);

-- Таблица отзывов

CREATE TABLE Reviews (

ReviewID INT PRIMARY KEY,

ProductID INT REFERENCES Products(ProductID),

CustomerID INT REFERENCES Customers(CustomerID),

Rating INT CHECK (Rating BETWEEN 1 AND 5),

ReviewText VARCHAR2(500),

ReviewDate DATE DEFAULT SYSDATE

);

-- Вставка данных в таблицу категорий товаров

INSERT INTO Product\_Categories (CategoryID, CategoryName, Description)

VALUES (1, 'Electronics', 'Devices and gadgets');

INSERT INTO Product\_Categories (CategoryID, CategoryName, Description)

VALUES (2, 'Books', 'Various genres of books');

INSERT INTO Product\_Categories (CategoryID, CategoryName, Description)

VALUES (3, 'Clothing', 'Men, Women, and Kids apparel');

INSERT INTO Product\_Categories (CategoryID, CategoryName, Description)

VALUES (4, 'Home Appliances', 'Appliances for home use');

-- Вставка данных в таблицу акций/скидок

INSERT INTO Promotions (PromotionID, PromotionName, DiscountPercentage, StartDate, EndDate)

VALUES (1, 'Winter Sale', 15, TO\_DATE('2024-12-01', 'YYYY-MM-DD'), TO\_DATE('2025-01-15', 'YYYY-MM-DD'));

INSERT INTO Promotions (PromotionID, PromotionName, DiscountPercentage, StartDate, EndDate)

VALUES (2, 'New Year Offer', 20, TO\_DATE('2024-12-25', 'YYYY-MM-DD'), TO\_DATE('2025-01-05', 'YYYY-MM-DD'));

INSERT INTO Promotions (PromotionID, PromotionName, DiscountPercentage, StartDate, EndDate)

VALUES (3, 'Spring Discount', 10, TO\_DATE('2024-03-01', 'YYYY-MM-DD'), TO\_DATE('2024-03-31', 'YYYY-MM-DD'));

-- Вставка данных в таблицу пользователей

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address, RegistrationDate)

VALUES (1, 'Yerassyl Sagindykov', 'yerassyl.sagindykov@gmail.com', '+77011234567', 'Almaty, Dostyk St., 123', SYSDATE);

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address, RegistrationDate)

VALUES (2, 'Vladislav Pineker', 'vladislav.pineker@icloud.com', '+77019876543', 'Astana, Kabanbai Batyr Ave., 45', SYSDATE);

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address, RegistrationDate)

VALUES (3, 'Aliya Amanova', 'aliya.amanova@example.kz', '+77021122334', 'Shymkent, Abai St., 78', SYSDATE);

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address, RegistrationDate)

VALUES (4, 'Maksim Smirnov', 'maksim.smirnov@example.kz', '+77037778899', 'Karaganda, Mira St., 99', SYSDATE);

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address, RegistrationDate)

VALUES (5, 'Diana Yesimova', 'diana.esimova@example.kz', '+77041234567', 'Aktobe, Bauyrzhan Momyshuly St., 32', SYSDATE);

-- Вставка данных в таблицу товаров

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (1, 'Smartphone', 362000, 50, 1, 1, 'Latest model with 5G');

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (2, 'Laptop', 623000, 30, 1, NULL, 'Powerful laptop for professionals');

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (3, 'Refrigerator', 520000, 15, 4, 2, 'Energy-saving refrigerator');

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (4, 'Book: Fiction', 10400, 100, 2, 3, 'Bestselling novel from a famous author');

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (5, 'T-Shirt', 5200, 200, 3, NULL, 'Cotton T-shirt available in all sizes');

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (6, 'Washing Machine', 312000, 10, 4, 1, 'Automatic washing machine');

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (7, 'E-Reader', 57200, 70, 2, 2, 'Device for reading e-books');

-- Вставка данных в таблицу заказов

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (1, 1, SYSDATE, 372200, 'Pending');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (2, 2, SYSDATE, 5200, 'Delivered');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (3, 3, SYSDATE, 623000, 'Processing');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (4, 4, SYSDATE, 10400, 'Delivered');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (5, 5, SYSDATE, 884200, 'Canceled');

-- Вставка данных в таблицу элементов заказа

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (1, 1, 1, 1, 362000);

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (2, 1, 5, 2, 5200);

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (3, 2, 5, 1, 5200);

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (4, 3, 2, 1, 623000);

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (5, 4, 4, 1, 10400);

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (6, 5, 1, 2, 362000);

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (7, 5, 6, 1, 312000);

-- Вставка данных в таблицу платежей

INSERT INTO Payments (PaymentID, OrderID, PaymentDate, PaymentMethod, Amount, Status)

VALUES (1, 1, SYSDATE, 'Bank Card', 372200, 'Success');

INSERT INTO Payments (PaymentID, OrderID, PaymentDate, PaymentMethod, Amount, Status)

VALUES (2, 2, SYSDATE, 'QIWI', 5200, 'Success');

INSERT INTO Payments (PaymentID, OrderID, PaymentDate, PaymentMethod, Amount, Status)

VALUES (3, 3, SYSDATE, 'Bank Card', 623000, 'Error');

INSERT INTO Payments (PaymentID, OrderID, PaymentDate, PaymentMethod, Amount, Status)

VALUES (4, 4, SYSDATE, 'PayPal', 10400, 'Success');

INSERT INTO Payments (PaymentID, OrderID, PaymentDate, PaymentMethod, Amount, Status)

VALUES (5, 5, SYSDATE, 'Bank Card', 884200, 'Canceled');

-- Вставка данных в таблицу склада

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (1, 1, 48, SYSDATE);

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (2, 2, 29, SYSDATE);

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (3, 3, 15, SYSDATE);

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (4, 4, 99, SYSDATE);

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (5, 5, 197, SYSDATE);

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (6, 6, 9, SYSDATE);

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (7, 7, 70, SYSDATE);

-- Вставка данных в таблицу отзывов

INSERT INTO Reviews (ReviewID, ProductID, CustomerID, Rating, ReviewText, ReviewDate)

VALUES (1, 1, 1, 5, 'Simply the best smartphone!', SYSDATE);

INSERT INTO Reviews (ReviewID, ProductID, CustomerID, Rating, ReviewText, ReviewDate)

VALUES (2, 4, 4, 4, 'An interesting book, highly recommend.', SYSDATE);

INSERT INTO Reviews (ReviewID, ProductID, CustomerID, Rating, ReviewText, ReviewDate)

VALUES (3, 6, 5, 3, 'Good washing machine but too expensive.', SYSDATE);

INSERT INTO Reviews (ReviewID, ProductID, CustomerID, Rating, ReviewText, ReviewDate)

VALUES (4, 3, 3, 5, 'Great fridge, very economical.', SYSDATE);

INSERT INTO Reviews (ReviewID, ProductID, CustomerID, Rating, ReviewText, ReviewDate)

VALUES (5, 2, 2, 4, 'Fast laptop, but pricey.', SYSDATE);

-- Создание таблицы для логирования успешных платежей CREATE TABLE Payment\_Logs ( LogID INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, PaymentID INT, LogDate DATE, Details VARCHAR2(500) );

-- Вставка данных и триггеры для базы данных интернет-магазина

-- Триггер для проверки наличия достаточного количества товара на складе перед добавлением элемента заказа

CREATE OR REPLACE TRIGGER trg\_check\_stock

BEFORE INSERT ON Order\_Items

FOR EACH ROW

DECLARE

v\_stock INT;

BEGIN

-- Проверка количества товара на складе

SELECT Quantity

INTO v\_stock

FROM Inventory

WHERE ProductID = :NEW.ProductID;

IF v\_stock < :NEW.Quantity THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient stock available for ProductID ' || :NEW.ProductID);

END IF;

END trg\_check\_stock;

/

-- Триггер для обновления количества товара на складе после оформления заказа

CREATE OR REPLACE TRIGGER trg\_update\_stock

AFTER INSERT ON Order\_Items

FOR EACH ROW

BEGIN

-- Обновление количества товара в таблице Inventory

UPDATE Inventory

SET Quantity = Quantity - :NEW.Quantity

WHERE ProductID = :NEW.ProductID;

END trg\_update\_stock;

/

-- Триггер для логирования успешных платежей

CREATE OR REPLACE TRIGGER trg\_log\_successful\_payments

AFTER INSERT ON Payments

FOR EACH ROW

BEGIN

IF :NEW.Status = 'Success' THEN

INSERT INTO Payment\_Logs (PaymentID, LogDate, Details)

VALUES (:NEW.PaymentID, SYSDATE, 'Payment successful for OrderID ' || :NEW.OrderID);

END IF;

END trg\_log\_successful\_payments;

/

-- Триггер для предотвращения удаления товара, если он используется в заказах

CREATE OR REPLACE TRIGGER trg\_prevent\_delete\_product

BEFORE DELETE ON Products

FOR EACH ROW

DECLARE

v\_order\_count INT;

BEGIN

-- Проверка на наличие товара в заказах

SELECT COUNT(\*)

INTO v\_order\_count

FROM Order\_Items

WHERE ProductID = :OLD.ProductID;

IF v\_order\_count > 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Cannot delete product as it is used in orders.');

END IF;

END trg\_prevent\_delete\_product;

/

-- Триггер предотвращает создание дублирующих заказов для одного клиента

CREATE OR REPLACE TRIGGER trg\_prevent\_duplicate\_orders

BEFORE INSERT ON Orders

FOR EACH ROW

DECLARE

v\_order\_count INT;

BEGIN

-- Проверка на наличие заказов с одинаковой суммой и статусом для данного клиента

SELECT COUNT(\*)

INTO v\_order\_count

FROM Orders

WHERE CustomerID = :NEW.CustomerID AND TotalAmount = :NEW.TotalAmount AND Status = :NEW.Status;

-- Если такой заказ существует, вызывается ошибка

IF v\_order\_count > 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Duplicate order detected for the same customer with identical details.');

END IF;

END trg\_prevent\_duplicate\_orders;

/

-- Триггер логирует удаление отзывов в таблицу Review\_Logs

CREATE OR REPLACE TRIGGER trg\_log\_deleted\_reviews

AFTER DELETE ON Reviews

FOR EACH ROW

BEGIN

-- Добавление записи об удаленном отзыве в таблицу Review\_Logs

INSERT INTO Review\_Logs (LogID, ProductID, CustomerID, DeletionDate)

VALUES (Review\_Logs\_SEQ.NEXTVAL, :OLD.ProductID, :OLD.CustomerID, SYSDATE);

END trg\_log\_deleted\_reviews;

/

ТЕСТИРОВАНИЕ ТРИГГЕРА

-- Тест триггера trg\_check\_stock

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (100, 1, 1, 9999, 362000);

-- Добавляем новый элемент заказа

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (101, 1, 1, 2, 362000);

-- Проверяем остаток товара на складе

SELECT Quantity FROM Inventory WHERE ProductID = 1;

-- Создаем успешный платеж

INSERT INTO Payments (PaymentID, OrderID, PaymentDate, PaymentMethod, Amount, Status)

VALUES (100, 1, SYSDATE, 'Bank Card', 724000, 'Success');

-- Проверяем таблицу Payment\_Logs

SELECT \* FROM Payment\_Logs WHERE PaymentID = 100;

-- Тест триггера trg\_prevent\_delete\_product

DELETE FROM Products WHERE ProductID = 1;

-- Тестирование триггера trg\_prevent\_duplicate\_orders

-- Попытка вставить уникальный заказ (должно быть успешно)

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (6, 1, SYSDATE, 15000, 'Pending');

-- Попытка вставить дублирующий заказ (должно вызвать ошибку)

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (7, 1, SYSDATE, 15000, 'Pending'); -- Ошибка: Duplicate order detected

-- Тестирование триггера trg\_log\_deleted\_reviews

-- Добавление тестовых отзывов

INSERT INTO Reviews (ReviewID, ProductID, CustomerID, Rating, ReviewText, ReviewDate)

VALUES (6, 1, 1, 5, 'Test Review 1', SYSDATE);

INSERT INTO Reviews (ReviewID, ProductID, CustomerID, Rating, ReviewText, ReviewDate)

VALUES (7, 2, 2, 4, 'Test Review 2', SYSDATE);

-- Удаление одного из отзывов (должно быть записано в Review\_Logs)

DELETE FROM Reviews WHERE ReviewID = 6;

-- Проверка таблицы логов

SELECT \* FROM Review\_Logs;

-- Функция для расчета общей стоимости заказа

CREATE OR REPLACE FUNCTION fn\_calculate\_order\_total(p\_order\_id INT)

RETURN NUMBER

IS

v\_total NUMBER := 0;

BEGIN

-- Подсчет общей стоимости заказа

SELECT SUM(Quantity \* Price)

INTO v\_total

FROM Order\_Items

WHERE OrderID = p\_order\_id;

RETURN v\_total;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Если данных нет, возвращается 0

END fn\_calculate\_order\_total;

/

-- Функция для получения скидки для товара

CREATE OR REPLACE FUNCTION fn\_get\_product\_discount(p\_product\_id INT)

RETURN NUMBER

IS

v\_discount NUMBER := 0;

BEGIN

-- Получение скидки из таблицы акций

SELECT COALESCE(Promotions.DiscountPercentage, 0)

INTO v\_discount

FROM Products

LEFT JOIN Promotions ON Products.PromotionID = Promotions.PromotionID

WHERE Products.ProductID = p\_product\_id;

RETURN v\_discount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Если акции нет, возвращается 0

END fn\_get\_product\_discount;

/

-- Функция для подсчета общего количества товаров на складе

CREATE OR REPLACE FUNCTION fn\_get\_total\_stock

RETURN NUMBER

IS

v\_total\_stock NUMBER := 0;

BEGIN

-- Подсчет общего количества товаров

SELECT COALESCE(SUM(Quantity), 0) -- Если NULL, возвращаем 0

INTO v\_total\_stock

FROM Inventory;

RETURN v\_total\_stock;

EXCEPTION

WHEN OTHERS THEN

RETURN 0; -- Обработка любых ошибок, возвращаем 0

END fn\_get\_total\_stock;

/

-- Функция для получения среднего рейтинга товара

CREATE OR REPLACE FUNCTION fn\_get\_average\_rating(p\_product\_id INT)

RETURN NUMBER

IS

v\_avg\_rating NUMBER := 0;

BEGIN

-- Подсчет среднего рейтинга

SELECT COALESCE(AVG(Rating), 0)

INTO v\_avg\_rating

FROM Reviews

WHERE ProductID = p\_product\_id;

RETURN v\_avg\_rating;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Если данных нет, возвращается 0

END fn\_get\_average\_rating;

/

-- Функция для получения статуса заказа

CREATE OR REPLACE FUNCTION fn\_get\_order\_status(p\_order\_id INT)

RETURN VARCHAR2

IS

v\_status VARCHAR2(20);

BEGIN

-- Получение статуса заказа

SELECT Status

INTO v\_status

FROM Orders

WHERE OrderID = p\_order\_id;

RETURN v\_status;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'Not Found'; -- Если заказ не найден, возвращается 'Not Found'

END fn\_get\_order\_status;

/

-- Функция возвращает ID продукта с самым высоким средним рейтингом

CREATE OR REPLACE FUNCTION fn\_get\_highest\_rated\_product

RETURN INT IS

v\_product\_id INT;

BEGIN

-- Поиск продукта с самым высоким средним рейтингом

SELECT ProductID

INTO v\_product\_id

FROM Reviews

GROUP BY ProductID

ORDER BY AVG(Rating) DESC

FETCH FIRST ROW ONLY;

RETURN v\_product\_id;

END fn\_get\_highest\_rated\_product;

/

-- Функция возвращает количество заказов, размещенных определенным клиентом

CREATE OR REPLACE FUNCTION fn\_get\_customer\_order\_count (

p\_customer\_id INT -- ID клиента

) RETURN INT IS

v\_order\_count INT;

BEGIN

-- Подсчет заказов для указанного клиента

SELECT COUNT(\*)

INTO v\_order\_count

FROM Orders

WHERE CustomerID = p\_customer\_id;

RETURN v\_order\_count;

END fn\_get\_customer\_order\_count;

/

-- Тест функции fn\_calculate\_order\_total

SELECT fn\_calculate\_order\_total(1) AS Total\_Order\_1 FROM DUAL;

-- Тест функции fn\_get\_product\_discount

SELECT fn\_get\_product\_discount(1) AS Discount\_Product\_1 FROM DUAL;

-- Тест функции fn\_get\_total\_stock

SELECT fn\_get\_total\_stock() AS Total\_Stock FROM DUAL;

-- Тест функции fn\_get\_average\_rating

SELECT fn\_get\_average\_rating(1) AS Avg\_Rating\_Product\_1 FROM DUAL;

-- Тест функции fn\_get\_order\_status

SELECT fn\_get\_order\_status(1) AS Order\_Status FROM DUAL;

-- Тестирование функции fn\_get\_highest\_rated\_product

-- Проверка текущих данных в таблице Reviews

SELECT ProductID, AVG(Rating) AS AvgRating

FROM Reviews

GROUP BY ProductID

ORDER BY AvgRating DESC;

-- Вызов функции для получения продукта с самым высоким рейтингом

SELECT fn\_get\_highest\_rated\_product() AS HighestRatedProduct FROM DUAL;

-- Тестирование функции fn\_get\_customer\_order\_count

-- Подсчет заказов для клиента 1 вручную

SELECT COUNT(\*) AS OrderCount FROM Orders WHERE CustomerID = 1;

-- Вызов функции для подсчета заказов клиента 1

SELECT fn\_get\_customer\_order\_count(1) AS OrderCount FROM DUAL;

-- Создание последовательностей для таблиц Orders и Order\_Items

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

CREATE SEQUENCE Order\_Items\_SEQ START WITH 1 INCREMENT BY 1;

-- Исправленная процедура для создания нового заказа

CREATE OR REPLACE PROCEDURE sp\_create\_order (

p\_customer\_id INT,

p\_order\_items SYS\_REFCURSOR

) IS

v\_order\_id INT;

v\_product\_id INT;

v\_quantity INT;

v\_price NUMBER(10, 2);

BEGIN

-- Создаем новый заказ

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (Orders\_SEQ.NEXTVAL, p\_customer\_id, SYSDATE, 0, 'Pending')

RETURNING OrderID INTO v\_order\_id;

-- Проходимся по элементам заказа и добавляем их

LOOP

FETCH p\_order\_items INTO v\_product\_id, v\_quantity, v\_price;

EXIT WHEN p\_order\_items%NOTFOUND;

INSERT INTO Order\_Items (OrderItemID, OrderID, ProductID, Quantity, Price)

VALUES (Order\_Items\_SEQ.NEXTVAL, v\_order\_id, v\_product\_id, v\_quantity, v\_price);

-- Обновляем сумму заказа

UPDATE Orders

SET TotalAmount = TotalAmount + (v\_quantity \* v\_price)

WHERE OrderID = v\_order\_id;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END sp\_create\_order;

/

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

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

-- Процедура для добавления нового товара

CREATE OR REPLACE PROCEDURE sp\_add\_product (

p\_name VARCHAR2,

p\_price NUMBER,

p\_stock\_quantity INT,

p\_category\_id INT,

p\_promotion\_id INT,

p\_description VARCHAR2

) IS

v\_product\_id INT;

BEGIN

-- Добавляем новый товар в таблицу Products

INSERT INTO Products (ProductID, Name, Price, StockQuantity, CategoryID, PromotionID, Description)

VALUES (Products\_SEQ.NEXTVAL, p\_name, p\_price, p\_stock\_quantity, p\_category\_id, p\_promotion\_id, p\_description)

RETURNING ProductID INTO v\_product\_id;

-- Добавляем запись в таблицу Inventory

INSERT INTO Inventory (InventoryID, ProductID, Quantity, LastUpdateDate)

VALUES (Inventory\_SEQ.NEXTVAL, v\_product\_id, p\_stock\_quantity, SYSDATE);

COMMIT;

END sp\_add\_product;

/

-- Процедура для обновления данных пользователя

CREATE OR REPLACE PROCEDURE sp\_update\_customer (

p\_customer\_id INT,

p\_name VARCHAR2,

p\_email VARCHAR2,

p\_phone VARCHAR2,

p\_address VARCHAR2

) IS

BEGIN

-- Обновляем данные пользователя

UPDATE Customers

SET Name = p\_name,

Email = p\_email,

Phone = p\_phone,

Address = p\_address

WHERE CustomerID = p\_customer\_id;

-- Фиксируем изменения

COMMIT;

END sp\_update\_customer;

/

BEGIN

sp\_update\_customer(

1,

'Yerassyl Sagindykov Updated',

'yerassyl.updated@gmail.com',

'+77011234568',

'Almaty, Updated Address'

);

END;

SELECT \* FROM Customers WHERE CustomerID = 1;

-- Процедура для удаления устаревших акций с обновлением связей

CREATE OR REPLACE PROCEDURE sp\_delete\_expired\_promotions IS

BEGIN

-- Обновляем связанные записи в таблице Products

UPDATE Products

SET PromotionID = NULL

WHERE PromotionID IN (

SELECT PromotionID

FROM Promotions

WHERE EndDate < SYSDATE

);

-- Удаляем устаревшие акции

DELETE FROM Promotions

WHERE EndDate < SYSDATE;

-- Фиксируем изменения

COMMIT;

END sp\_delete\_expired\_promotions;

/

BEGIN

sp\_delete\_expired\_promotions;

END;

SELECT \* FROM Promotions WHERE EndDate < SYSDATE;

SELECT \* FROM Products WHERE PromotionID IS NULL;

-- Процедура для генерации отчета о продажах

CREATE OR REPLACE PROCEDURE sp\_generate\_sales\_report (

p\_start\_date DATE,

p\_end\_date DATE

) IS

v\_total\_income NUMBER := 0;

v\_order\_count INT := 0;

BEGIN

-- Считаем общий доход за период

SELECT COALESCE(SUM(TotalAmount), 0)

INTO v\_total\_income

FROM Orders

WHERE OrderDate BETWEEN p\_start\_date AND p\_end\_date;

-- Считаем количество заказов за период

SELECT COUNT(\*)

INTO v\_order\_count

FROM Orders

WHERE OrderDate BETWEEN p\_start\_date AND p\_end\_date;

-- Выводим отчет в лог

DBMS\_OUTPUT.PUT\_LINE('Sales Report:');

DBMS\_OUTPUT.PUT\_LINE('Period: ' || TO\_CHAR(p\_start\_date, 'YYYY-MM-DD') || ' to ' || TO\_CHAR(p\_end\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Total Income: ' || v\_total\_income || ' KZT');

DBMS\_OUTPUT.PUT\_LINE('Total Orders: ' || v\_order\_count);

END sp\_generate\_sales\_report;

BEGIN

sp\_generate\_sales\_report(

TO\_DATE('2024-01-01', 'YYYY-MM-DD'),

TO\_DATE('2024-12-31', 'YYYY-MM-DD')

);

END;

-- Заголовок пакета

CREATE OR REPLACE PACKAGE OnlineStore\_PKG AS

PROCEDURE sp\_create\_order(p\_customer\_id INT, p\_order\_items SYS\_REFCURSOR);

PROCEDURE sp\_add\_product(p\_name VARCHAR2, p\_price NUMBER, p\_stock\_quantity INT, p\_category\_id INT, p\_promotion\_id INT, p\_description VARCHAR2);

PROCEDURE sp\_update\_customer(p\_customer\_id INT, p\_name VARCHAR2, p\_email VARCHAR2, p\_phone VARCHAR2, p\_address VARCHAR2);

PROCEDURE sp\_delete\_expired\_promotions;

PROCEDURE sp\_generate\_sales\_report(p\_start\_date DATE, p\_end\_date DATE);

FUNCTION fn\_calculate\_order\_total(p\_order\_id INT) RETURN NUMBER;

FUNCTION fn\_get\_product\_discount(p\_product\_id INT) RETURN NUMBER;

FUNCTION fn\_get\_total\_stock RETURN NUMBER;

FUNCTION fn\_get\_average\_rating(p\_product\_id INT) RETURN NUMBER;

FUNCTION fn\_get\_order\_status(p\_order\_id INT) RETURN VARCHAR2;

END OnlineStore\_PKG;

-- Процедура для очистки таблицы

CREATE OR REPLACE PROCEDURE sp\_clear\_table(p\_table\_name VARCHAR2) IS

BEGIN

EXECUTE IMMEDIATE 'DELETE FROM ' || p\_table\_name;

COMMIT;

END sp\_clear\_table;

BEGIN

sp\_clear\_table('Order\_Items');

END;

-- Процедура для отчета о продажах по категориям

CREATE OR REPLACE PROCEDURE sp\_sales\_report\_by\_category IS

CURSOR sales\_cursor IS

SELECT pc.CategoryName, SUM(o.TotalAmount) AS TotalIncome

FROM Orders o

JOIN Order\_Items oi ON o.OrderID = oi.OrderID

JOIN Products p ON oi.ProductID = p.ProductID

JOIN Product\_Categories pc ON p.CategoryID = pc.CategoryID

GROUP BY pc.CategoryName;

v\_category\_name Product\_Categories.CategoryName%TYPE;

v\_total\_income NUMBER;

BEGIN

OPEN sales\_cursor;

LOOP

FETCH sales\_cursor INTO v\_category\_name, v\_total\_income;

EXIT WHEN sales\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Category: ' || v\_category\_name || ', Total Income: ' || v\_total\_income || ' KZT');

END LOOP;

CLOSE sales\_cursor;

END sp\_sales\_report\_by\_category;

BEGIN

sp\_sales\_report\_by\_category;

END;

/

-- Процедура применяет глобальную скидку ко всем продуктам в выбранной категории

CREATE OR REPLACE PROCEDURE sp\_apply\_global\_discount (

p\_category\_id INT, -- ID категории

p\_discount\_percentage NUMBER -- Размер скидки в процентах

) IS

BEGIN

-- Обновление цен продуктов в выбранной категории

UPDATE Products

SET Price = Price - (Price \* (p\_discount\_percentage / 100))

WHERE CategoryID = p\_category\_id;

COMMIT;

END sp\_apply\_global\_discount;

/

-- Процедура генерирует отчет о заказах для указанного клиента

CREATE OR REPLACE PROCEDURE sp\_generate\_customer\_order\_summary (

p\_customer\_id INT -- ID клиента

) IS

CURSOR order\_cursor IS

-- Получение данных о заказах клиента

SELECT OrderID, OrderDate, TotalAmount, Status

FROM Orders

WHERE CustomerID = p\_customer\_id;

v\_order\_id INT;

v\_order\_date DATE;

v\_total\_amount NUMBER;

v\_status VARCHAR2(20);

BEGIN

-- Открытие курсора и перебор заказов

OPEN order\_cursor;

LOOP

FETCH order\_cursor INTO v\_order\_id, v\_order\_date, v\_total\_amount, v\_status;

EXIT WHEN order\_cursor%NOTFOUND;

-- Вывод информации о заказах в консоль

DBMS\_OUTPUT.PUT\_LINE('Order ID: ' || v\_order\_id || ', Date: ' || v\_order\_date || ', Amount: ' || v\_total\_amount || ', Status: ' || v\_status);

END LOOP;

CLOSE order\_cursor;

END sp\_generate\_customer\_order\_summary;

/

-- Тестирование процедуры sp\_apply\_global\_discount

-- Проверка текущих цен продуктов в категории 1

SELECT \* FROM Products WHERE CategoryID = 1;

-- Применение 10% скидки ко всем продуктам в категории 1

BEGIN

sp\_apply\_global\_discount(1, 10);

END;

-- Проверка обновленных цен

SELECT \* FROM Products WHERE CategoryID = 1;

-- Тестирование процедуры sp\_generate\_customer\_order\_summary

-- Добавление тестового заказа для клиента 1

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount, Status)

VALUES (8, 1, SYSDATE, 20000, 'Delivered');

-- Вызов процедуры для генерации отчета по клиенту 1

BEGIN

sp\_generate\_customer\_order\_summary(1);

END;

CREATE SEQUENCE Customer\_Audit\_SEQ

START WITH 1

INCREMENT BY 1

NOCACHE;

-- Триггер для аудита изменений

CREATE OR REPLACE TRIGGER trg\_audit\_customers

AFTER UPDATE ON Customers

FOR EACH ROW

BEGIN

INSERT INTO Customer\_Audit (AuditID, CustomerID, Operation, ChangeDate, OldValue, NewValue)

VALUES (

Customer\_Audit\_SEQ.NEXTVAL,

:OLD.CustomerID,

'UPDATE',

SYSDATE,

:OLD.Name || '; ' || :OLD.Email || '; ' || :OLD.Phone || '; ' || :OLD.Address,

:NEW.Name || '; ' || :NEW.Email || '; ' || :NEW.Phone || '; ' || :NEW.Address

);

END trg\_audit\_customers;

UPDATE Customers

SET Name = 'Yerassyl Updated', Phone = '+77011234567'

WHERE CustomerID = 1;

SELECT \* FROM Customer\_Audit;