МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Реализация базы данных магазина украшений с применением технологии разработки системы мониторинга за состоянием базы данных »

Выполнил студент Дзивнель Марта Андреевна
(Ф.И.О.)

Руководитель работы асс. Харланович А. В.
(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой доцент Блинова Е.А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой _____

Минск 2024

Содержание

Введение

В современном мире магазины украшений становятся все более популярными, и эффективное управление информацией в таких предприятиях играет решающую роль. Базы данных становятся ключевым инструментом для управления запасами, заказами, клиентскими данными и другой важной информацией, обеспечивая ее структурированное хранение и быстрый доступ к ней.

На рынке представлено множество технологий доступа к данным и серверам баз данных, каждая из которых имеет свои отличительные особенности и преимущества. Современные приложения для обработки данных, разработанные для работы с большим количеством пользователей, учитывают удаленность пользователей от серверов баз данных, что позволяет им эффективно работать в различных условиях.

Темой данного курсового проекта является разработка база данных «Магазин украшений».

Цель данного курсового проекта – создание базы данных для хранения данных магазина украшений, ознакомление с технологией разработки системы мониторинга за состоянием базы данных и её применение в базе данных.

Основными задачами курсовой работы являются:

– возможность поиска изделия по различным параметрам, таким как: наименование, артикул, вид материала, категория изделия и т.д;

– реализовать механизм внесения изменений в каталог изделий, таких как добавление, изменение и удаление записей;

– определение четырех ролей: администратор, продавец, покупатель и менеджер.

Для проектирования базы данных используется СУБД «PostgreSQL». В базе данных применяется технология разработки системы мониторинга за состоянием базы данных.

1. Постановка задачи

1.1. Основные задачи и цели проекта

Целью данного курсового проекта является создание базы данных, которая позволит эффективно управлять всеми аспектами деятельности магазина украшений.

Задача проекта: разработка структуры базы данных, включая таблицы для хранения информации о продуктах (украшениях), заказах, клиентах и дополнительной информацией. Корректно спроектированная структура данных позволит удобно хранить и организовывать информацию о продуктах и связанных с ними сущностях. Реализация процедур и функций для управления данными, включая добавление новых продуктов, обновление информации, удаление устаревших записей, обработка запросов и извлечение необходимых данных. Создание пользовательских ролей и привилегий для различных уровней доступа к данным (для администратора, продавцов, покупателей и менеджеров). Реализация механизмов импорта и экспорта данных в формате XML.

Функционально должны быть выполнены следующие задачи:

– поиска изделия по различным параметрам, таким как: наименование, артикул, вид материала, категория изделия и т.д.;

– внесение изменений в каталог изделий, таких как добавление, изменение и удаление занятий;

– определение четырех ролей: администратор, продавец, покупатель, менеджер. Администратор будет иметь возможность изменять каталог изделий, управлять их стоимостью, обрабатывать запросы о пополнении количества товаров. Продавец будет рассчитывать итоговую стоимость изделий с учетом скидок, создавать запись о покупке и соответственно обновлять информацию о количестве товаров. Покупатель может просматривать товары, стоимость и их количество. Менеджер может просматривать информацию о изделиях, следить за количеством товара и отправлять запрос на добавление товара, управлять информацией о времени и дате поставок.

Должны быть выполнены следующие требования:

− база данных должна быть спроектирована в СУБД «PostgreSQL»;

− доступ к данным должен осуществляться только через соответствующие процедуры и функции;

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

− необходимо протестировать производительность базы данных (на таблицах, содержащих не менее 100 000 строк) и внести изменения в структуру в случае необходимости;

Также необходимо применить технологию базы данных согласно выбранной теме: рассказать про актуальность выбранной технологии и сфере ее применения в разрабатываемой базе данных, подробно описать применяемые системные пакеты, утилиты, технологии или расширения; показать применение указанной технологии в базе данных.

1.2. Обзор аналогов

В современном мире магазины украшений становятся все более популярными. Одной из главных задач таких магазинов является надежная продажа и обслуживание украшений, чтобы удовлетворить потребности и ожидания клиентов.

Магазины украшений предлагают широкий ассортимент украшений различных видов. Клиенты могут выбрать украшение, соответствующее их предпочтениям и потребностям.

1.2.1. Аналог «Pandora»

В качестве первого аналога рассмотрим сайт «Pandora» [1], который представлен на рисунке 1.1.

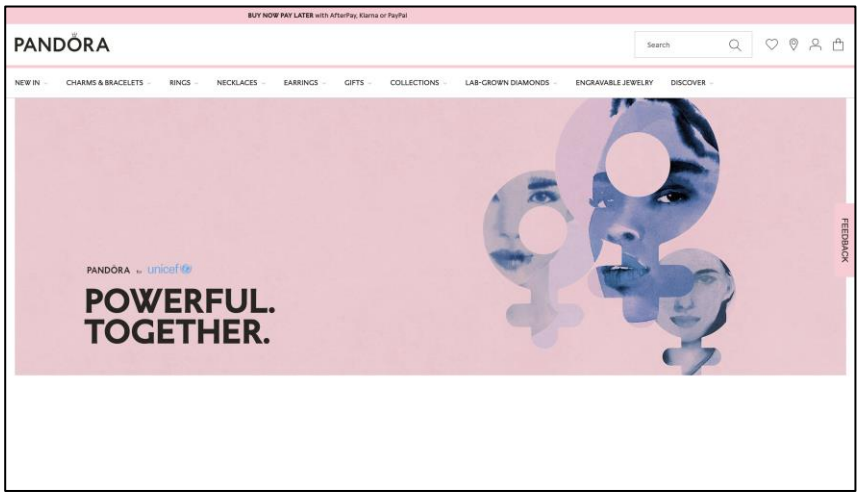

Рисунок 1.1 – Сайт «Pandora»

Сайт «Pandora» предлагает широкий ассортимент ювелирных изделий, от колец и подвесок до очков. Дизайн сайта минималистичный, с акцентом на изображения изделий. В ходе анализа были выделены некоторые плюсы и минусы данного приложения.

Плюсы:

– удобная навигация: простой и интуитивно понятный интерфейс, пользователи могут легко находить товары по категориям;

– фильтры поиска: возможность фильтрации по материалу, цене и популярности;

– мобильная версия: хорошо адаптирован для мобильных устройств.

Минусы:

– ограниченная информация о товаре: некоторые страницы товаров содержат недостаточно подробной информации о характеристиках (рис. 1.2);

– сложности с загрузкой: временами страница грузится медленно из-за большого объема изображений.

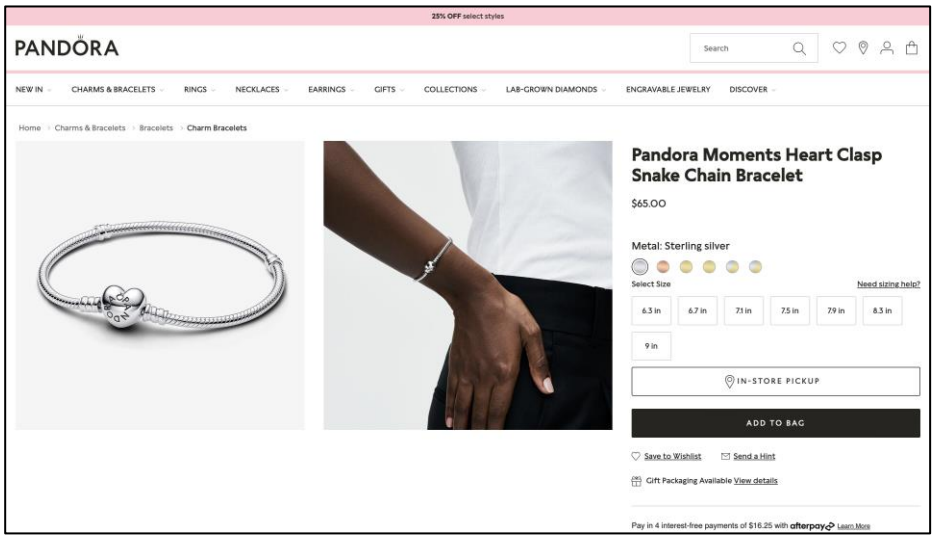

Рисунок 1.2 – Карточка товара на сайте «Pandora»

База данных реализована следующим образом:

– используется реляционная база данных для хранения информации о пользователях, товарах и заказах;

– информация о товарах организована по категориям, что упрощает администрирование и поиск.

1.2.2. Аналог «Cartier»

Еще одним аналогом является сайт «Cartier» [2], предоставляющий услуги по покупке украшений по всему миру. Сайт продемонстрирован на рисунке 1.3.

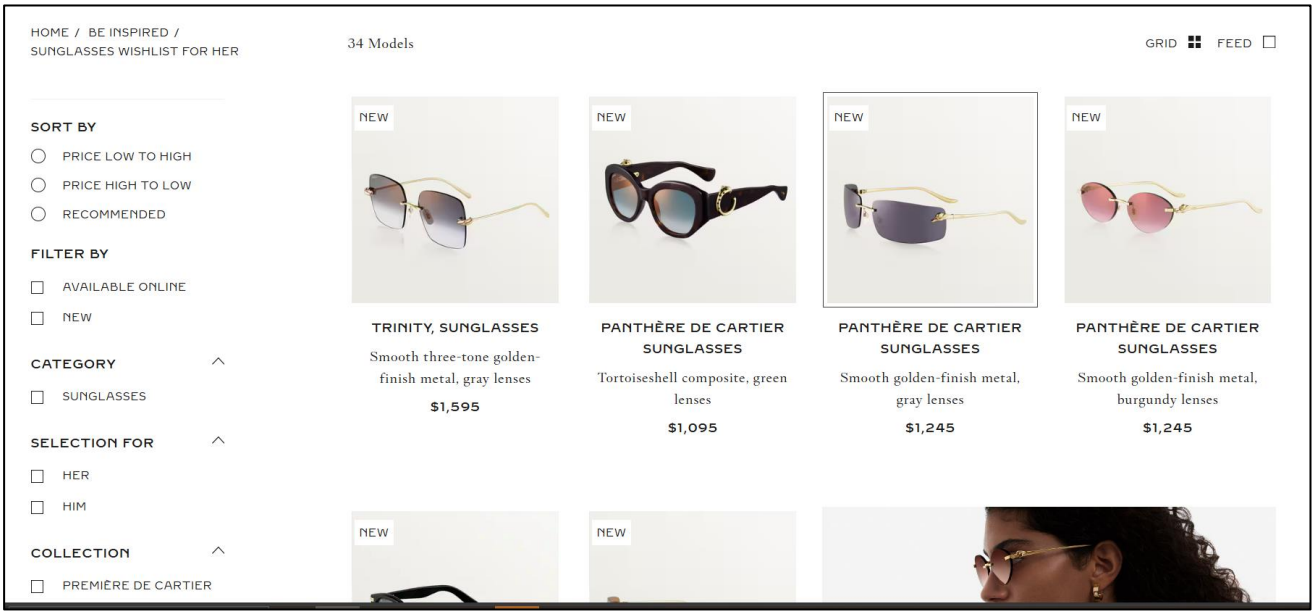

Рисунок 1.3 – Выбор украшения «Cartier»

Сайт «Cartier» демонстрирует роскошный и элегантный дизайн. Он фокусируется на высококачественных ювелирных изделиях и часах, что прекрасно

видно на сайте. В ходе анализа также выделены плюсы и минусы организации и устройства сайта.

Плюсы:

– эстетика и дизайн: высококачественные изображения и стильный интерфейс создают атмосферу роскоши;

– интерактивные элементы: возможность виртуального примерки ювелирных изделий;

– подробные описания: каждое изделие сопровождается детальной информацией и историей (рис 1.6).

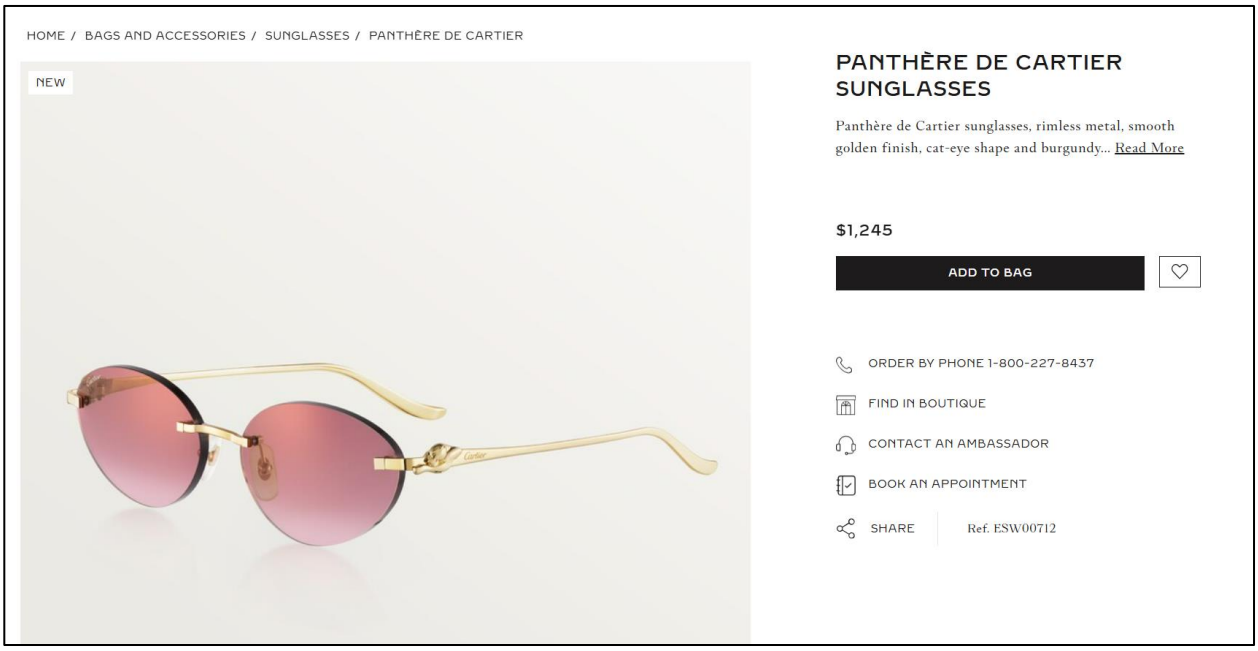

Рисунок 1.4 – Карточка товара на сайте «Cartier»

Минусы:

– сложная навигация: из-за большого количества информации пользователи могут потеряться;

– медленная загрузка: страницы могут загружаться медленно из-за большого объема графики.

Работа с БД:

– сложная структура базы данных, которая включает в себя не только товары, но и историю бренда, что требует более глубокой организации данных;

– использует системы управления контентом для обновления информации о новых коллекциях и событиях.

1.2.3. Аналог «Ziko»

Последним аналогом будет рассмотрен белорусский аналог ювелирных магазинов – «Ziko» [3]. Изображение главной страницы представлено на рисунке 1.5.

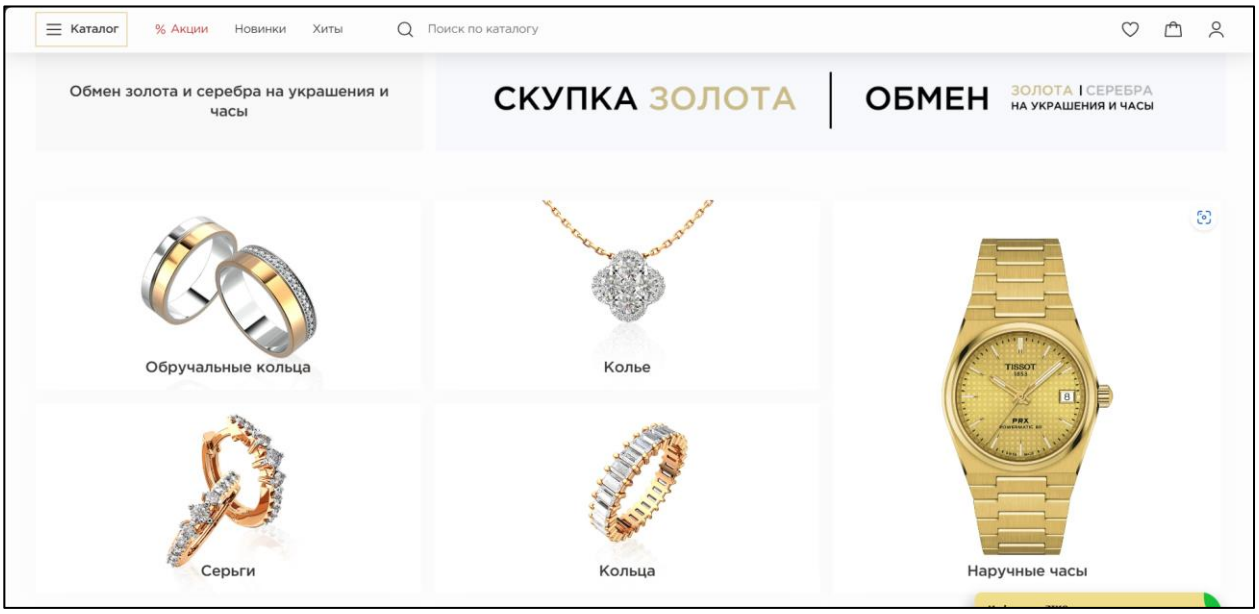

Рисунок 1.5 – Главная страница сайта «Ziko»

Сайт Ziko предлагает доступные ювелирные изделия и ориентирован на широкий круг потребителей. Дизайн более простой и ориентирован на практичность.

Плюсы:

– доступность информации: легкий доступ к информации о товарах и ценах;

– акции и скидки: ярко выделенные разделы с акциями и специальными предложениями;

– простота использования: удобное меню и фильтры, которые помогают быстро находить нужные товары.

Минусы:

– ограниченный дизайн: меньше акцентов на визуальные элементы, что может снизить привлекательность;

– недостаток уникальности: нет впечатляющих интерактивных функций, как у конкурентов.

Работа с БД:

– использует простую реляционную базу данных для управления товарами и пользователями;

– функционал базы данных сосредоточен на поддержке акций и скидок, что требует регулярного обновления информации.

Карточка товара и отображаемые данные организованы подобно другим сайтам, однако информации о товаре предоставлено больше, чем было на странице у «Cartier», что может говорить о том, что база данных данного сайта хранит в себе больше информации (рис 1.6).

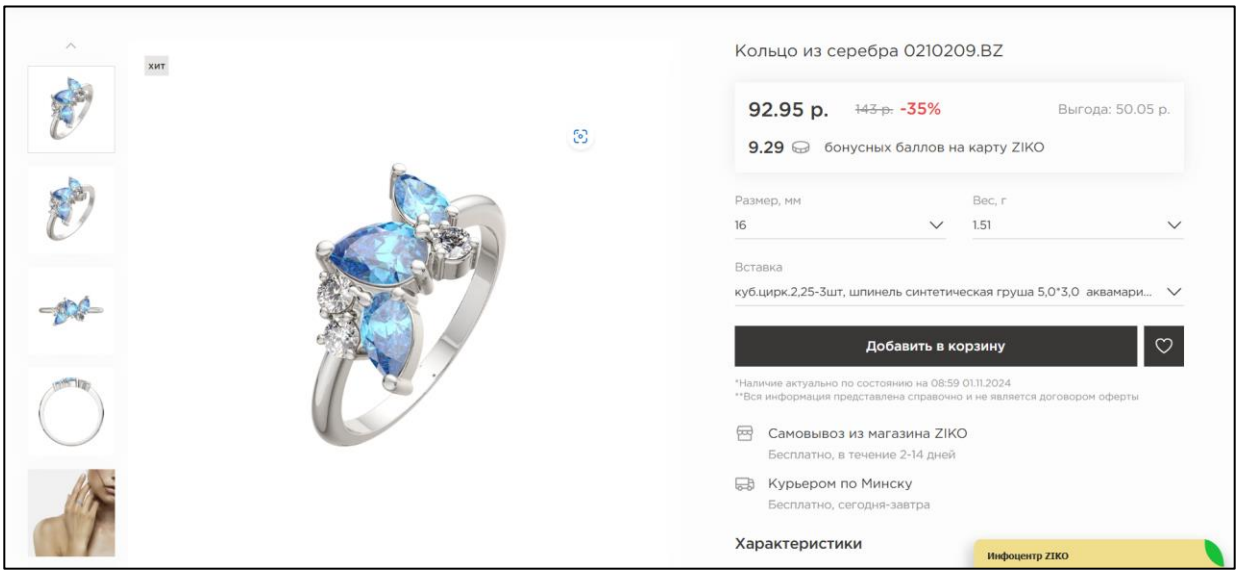

Рисунок 1.6 – Карточка товара на сайте «Ziko»

Анализ показывает, что при разработке схемы базы данных для ювелирного магазина стоит учитывать как визуальные, так и функциональные аспекты. Успешный сайт должен сочетать в себе привлекательный дизайн, удобную навигацию и эффективное управление данными. Учитывая плюсы и минусы каждого из представленных сайтов, можно создать эффективную и удобную базу данных для пользователей, что в конечном итоге повысит уровень продаж и удовлетворенности клиентов.

1.3. Описание функциональных требований

В ходе изучения требований было выявлено, что база данных магазина украшений должна включать информацию об украшениях, клиентах, заказах, отзывах, запросах на пополнение товара. Были определены следующие варианты использования:

- хранение информации об украшениях, менеджерах, клиентах, продавцах, администраторах, заказах, отзывах, запросах на пополнение товара и просмотр информации;
- поиск и сортировка украшений по различным критериям;
- оформление заказов на украшения;
- отслеживание статуса запросов на пополнение товара;
- просмотр истории покупок клиентов;
- оставление отзывов о приобретенных товарах и обслуживании;
- добавление и обновление информации об украшениях;
- удаление украшений из базы данных;
- ведение учета проданных украшений;
- обеспечение защиты данных и авторизации доступа к базе данных;
- мониторинг состояния базы данных и обнаружение проблем в работе.

1.4. Описание нефункциональных требований

В качестве нефункциональных требований, можно выделить:

− безопасность: база данных должна обеспечивать безопасность данных и защиту от несанкционированного доступа, в том числе использование механизмов авторизации и аутентификации;

− производительность: база данных должна иметь высокую производительность и обеспечивать быстрый доступ к данным для обеспечения эффективной работы магазина украшений;

− надежность: база данных должна быть надежной и стабильной, обеспечивать целостность данных и иметь возможность быстрого восстановления после сбоев;

− масштабируемость: база данных должна быть масштабируемой и гибкой, чтобы обеспечивать эффективное управление растущим объемом данных магазина украшений;

− удобство использования: база данных должна быть удобной и простой в использовании для обеспечения эффективной работы пользователей.

1.5. Вывод по разделу

В данном разделе проведен аналитический обзор аналогичных проектов, включающий три конкретных примера: "Pandora", "Cartier" и "Ziko". Это позволило выявить достоинства и недостатки существующих решений и использовать их при разработке нового проекта.

Важным этапом является изучение требований и определение основных функциональных и нефункциональных требований к базе данных. Это позволило выявить ключевые аспекты, которые следует учесть при проектировании и разработке проекта

2. Проектирование базы данных

2.1. Определение вариантов использования

В этом разделе представлена диаграмма, демонстрирующая сценарии взаимодействия пользователей с системой. Она выполняет роль визуального инструмента для понимания функционала проекта, выделяя ключевые роли и доступные действия. Каждый сценарий описывает определённые задачи, которые система должна реализовать, что способствует более точному формулированию требований и ожиданий пользователей. Изучение этих сценариев будет полезно для дальнейшего проектирования и настройки системы в соответствии с потребностями целевой аудитории. Диаграмма, касающаяся разрабатываемой базы данных, представлена на рисунке 2.1.

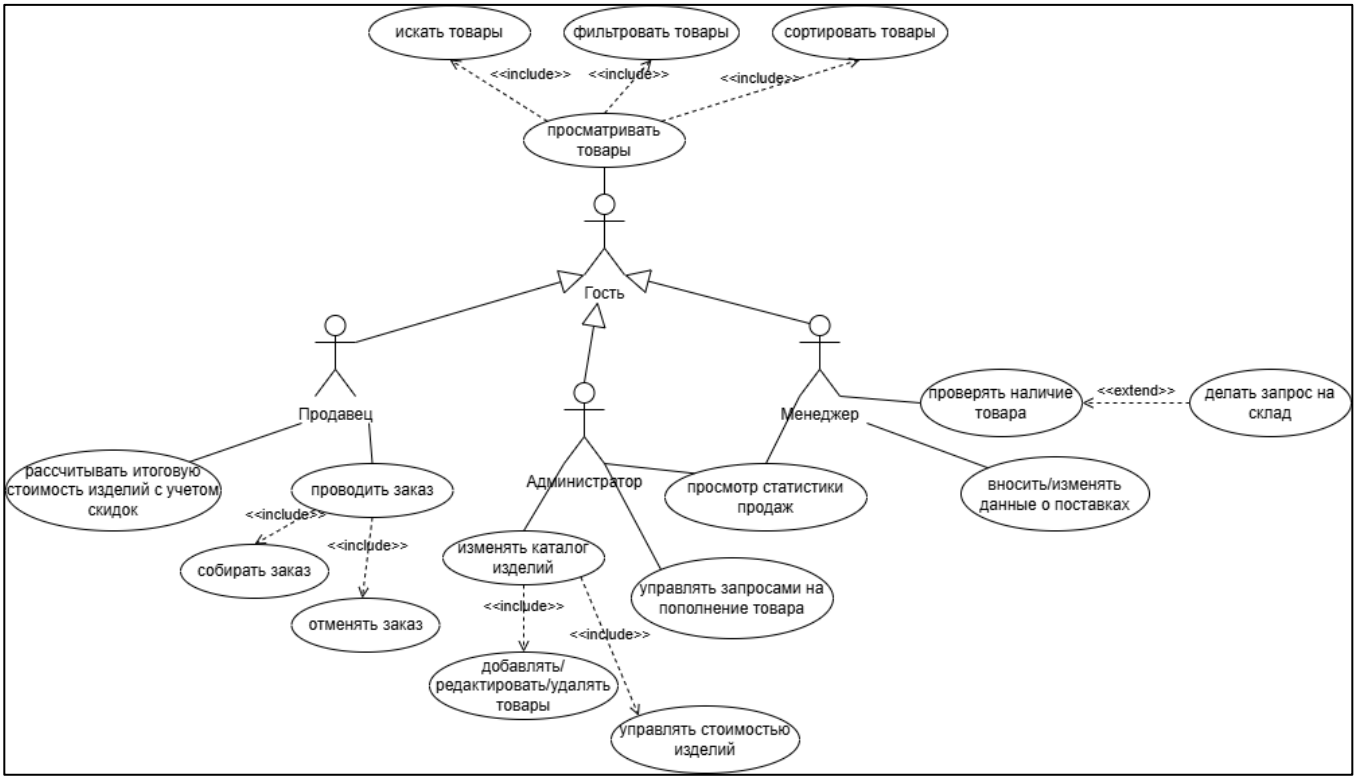

Рисунок 2.1 – Диаграмма вариантов использования

Актерами на диаграмме являются продавец, администратор, менеджер и гость. Варианты использования включают в себя проверка наличия товара, изменение/добавление/удаление товаров, просмотр статистики и другое.

2.2. Схема базы данных

Схема базы данных — это структурированное представление организации данных в базе. Она описывает таблицы, поля, типы данных и связи между таблицами.

Схема помогает определить, как данные будут храниться, организовываться и взаимодействовать друг с другом.

Основные элементы схемы включают:

- таблицы: основные структуры, где хранятся данные.
- поля: атрибуты таблиц, определяющие типы данных (текст, число, дата).
- связи: определяют, как таблицы связаны друг с другом.

Схема базы данных является ключевым компонентом при проектировании и разработке базы данных, обеспечивая её целостность и эффективность. Схема базы данных для проекта представлена на рисунке 2.2.

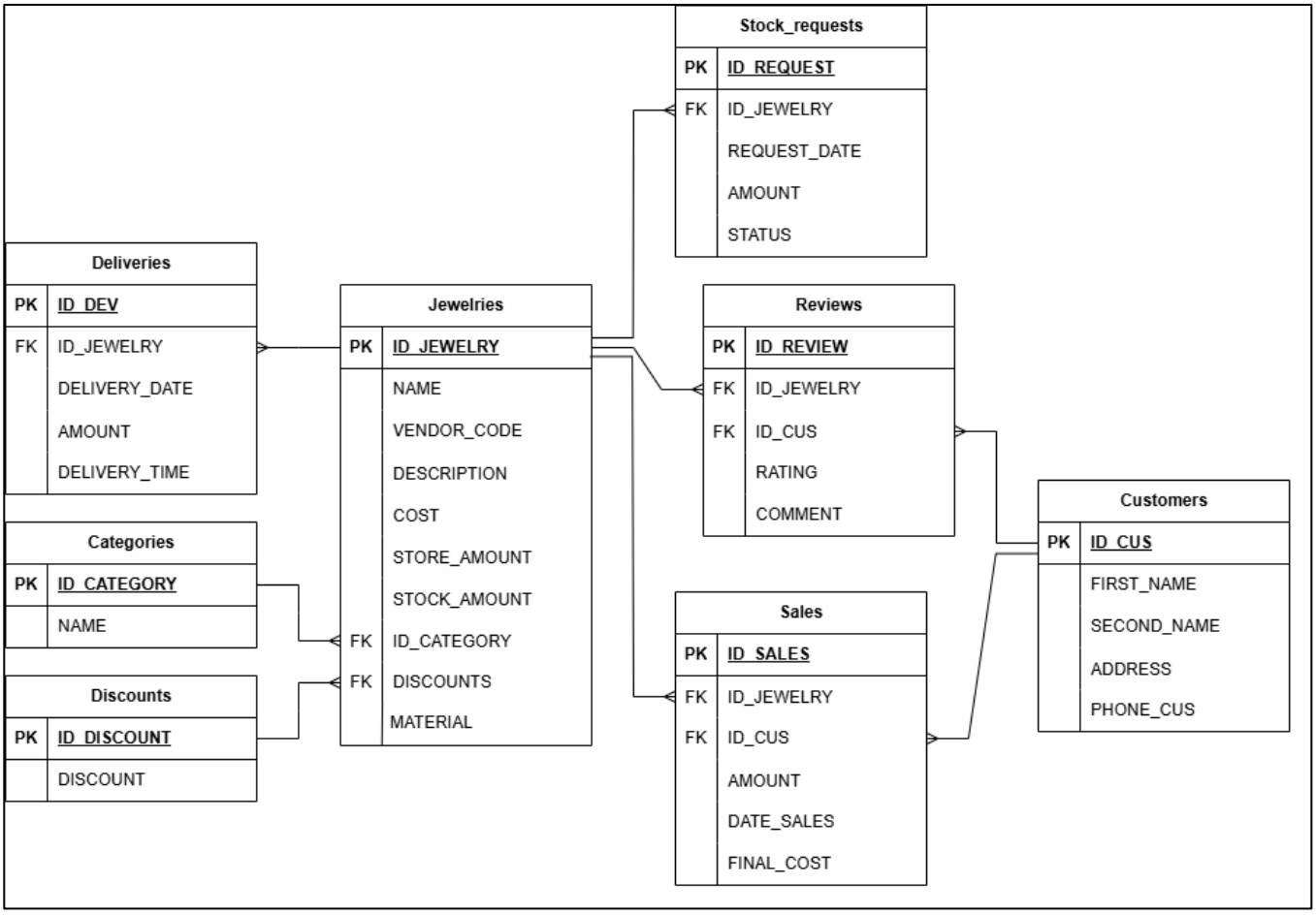

Рисунок 2.2 – Схема базы данных магазина украшений

Данная схема будет использована при разработке объектов БД.

2.3. Вывод по разделу

Разработка архитектуры проекта важна для определения структуры и функционала приложения. Обобщенная структура управления помогает выявить необходимые компоненты и их взаимодействие. Описание информационных объектов также играет ключевую роль, поскольку позволяет понять, какие данные будут использоваться в приложении и как они будут храниться и обрабатываться.

3. Разработка объектов базы данных

3.1. Таблицы

Для хранения сущностей в БД создано 8 таблиц.

Таблица JEWELRIES (для хранения украшений):

- ID_JEWELRY. Уникальный идентификатор украшения. Содержит ограничение первичного ключа;
- NAME. Название украшения. Не может быть NULL;
- VENDOR_CODE. Артикул украшения. Не может быть NULL;
- WEIGHT. Вес украшения. Не может быть NULL;
- METALL. Количество товара. Не может быть NULL;
- DESCRIPTION. Описание украшения;
- COST. Стоимость украшения. Не может быть NULL;
- AMOUNT. Количество украшений. Не может быть NULL;
- ID_SUP. Стоимость украшения. Не может быть NULL. Ссылается на таблицу Suppliers;
- ID_CATEGORY. Стоимость украшения. Не может быть NULL. Ссылается на таблицу Categories;
- DISCOUNTS. Стоимость украшения. Не может быть NULL. Ссылается на таблицу Discounts;

Таблица DELIVERIES (для хранения информации о доставках):

- ID_DEV. Уникальный идентификатор поставки. Содержит ограничение первичного ключа;
- ID_SUP. Уникальный идентификатор поставщика. Ссылается на таблицу Suppliers;
- DELIVERY_DATE. Дата поставки товара;
- DELIVARY_TIME. Время поставки товара;
- AMOUNT. Количество доставленного товара.

Таблица CUSTOMERS (для хранения информации о покупателях):

- ID_CUS. Уникальный идентификатор покупателя. Содержит ограничение первичного ключа;
- FIRST_NAME. Имя покупателя. Не может быть NULL;
- SECOND_NAME VARCHAR. Фамилия покупателя;
- ADDRESS. Адрес покупателя;
- PHONE_CUS. Мобильный телефон покупателя.

Таблица SALES (для хранения информации о продажах):

- ID_SALES. Уникальный идентификатор продажи. Содержит ограничение первичного ключа;
- ID_JEWELRY. Уникальный идентификатор украшения. Не может быть NULL. Ссылается на таблицу Jewelries;
- ID_CUS. Уникальный идентификатор покупателя, который приобрел украшение. Не может быть NULL. Ссылается на таблицу Customers;
- AMOUNT. Количество проданного товара;
- DATE_SALES. Дата продажи;

− FINAL_COST. Стоимость проданного товара.

Таблица REVIEWS (для хранения информации об отзывах):

− ID_REVIEW. Уникальный идентификатор отзыва. Содержит ограничение первичного ключа;

− ID_JEWELRY. Уникальный идентификатор украшения, на которое оставили отзыв. Не может быть NULL. Ссылается на таблицу Jewelries;

− ID_CUS. Уникальный идентификатор пользователя, который оставил отзыв. Не может быть NULL. Ссылается на таблицу Customers;

− RATING. Оценка украшения;

− COMMENTT. Комментарий к украшению.

Таблица CATEGORIES (для хранения категорий):

− ID_CATEGORY. Уникальный идентификатор категории. Содержит ограничение первичного ключа;

− NAME. Название категории;

Таблица DISCOUNTS (для хранения информации о скидках):

− ID_DISCOUNT. Уникальный идентификатор скидки. Содержит ограничение первичного ключа;

− DISCOUNT. Скидка. Не может быть NULL.

Таблица STOCK_REQUESTS (для хранения информации о запросах на склад):

− ID_REQUST. Уникальный идентификатор запроса. Содержит ограничение первичного ключа;

− ID_JEWELRY. Уникальный идентификатор запрашиваемого украшения. Ссылается на таблицу Jewelries;

− REQUEST_DATE. Дата запроса;

− AMOUNT. Количество запрашиваемого украшения;

− STATUS. Статус запроса.

Листинги создания таблиц приведены в приложении А.

3.2. Роли и пользователи

Роли и пользователи в базе данных играют важную роль в управлении доступом и безопасностью данных.

Пользователи – это индивидуальные учетные записи, которые имеют доступ к базе данных. Роли – это группы привилегий, которые могут быть назначены пользователям. Они позволяют упростить управление доступом, так как можно определить набор разрешений для группы пользователей, а не для каждого отдельно.

В БД курсового проекта разработано 6 ролей:

− programmer (роль для разработчика);

− base_user (базовая роль для всех пользователей);

− administrator (управляющий магазином);

− manager (менеджер);

− salesman (продавец);

− customer (покупатель).

Для всех ролей, кроме base_user, были разработаны пользователи.

3.3. Процедуры и функции

Процедура – это блок кода в СУБД, который может быть вызван многократно для выполнения определенной операции или последовательности операций. Использование процедур для доступа к данным может ускорить обработку большого объема данных, а также обеспечить более безопасное и удобное взаимодействие с БД, позволяя выполнять сложные операции и обеспечивая целостность данных.

Были разработаны следующие процедуры:

- add_jewelry. Добавить украшение;
- add_review. Добавить отзыв;
- calculate_final_cost. Высчитать конечную стоимость украшения;
- cancel_order. Отменить заказ;
- delete_jewelry. Удалить украшение;
- delete_review. Удалить отзыв;
- exportdeliveriestoxml. Экспортировать доставки в XML;
- generate_db_report. Сгенерировать отчет по базе данных;
- getjewelryinfobymetal. Получить информацию об украшениях по металлу;
- getjewelryinfobycategory. Получить информацию об украшениях по категории;
- get_sorted_jewelry_by_rating. Получить отсортированные украшения по рейтингу;
- process_order. Обработать заказ;
- request_restock. Запросить пополнение запасов;
- sort_jewelries. Отсортировать украшения;
- update_jewelry. Обновить информацию об украшении;
- update_jewelry_amount. Обновить количество украшений;
- update_jewelry_cost. Обновить стоимость украшения;
- update_request_status. Обновить статус запроса;
- updatedeliverydatetimebyjewelry. Обновить дату и время доставки по украшению.

Пример создания процедуры удаления товара представлен в листинге 3.1.

```
create or replace procedure delete_jewelry(jewelry_id int)security
definer
as $$ begin
    delete from jewelries where id_jewelry = jewelry_id;
    if not found then raise notice 'нет украшения с id: %', jewelry_id;
    else raise notice 'украшение было успешно удалено:id %', jewelry_id;
    end if;
exception when others then
        raise exception 'ошибка при удалении украшения: %', sqlerrm;
end;$$ language plpgsql;
```

Листинг 3.1. – Пример создания процедуры удаления товара

Функции – это блоки кода, которые могут быть вызваны для выполнения определенной операции и возвращают результат. Они могут принимать параметры, иметь локальные переменные и содержать операторы SQL или других языков программирования. Функции позволяют выполнить определенные операции над данными и вернуть результат.

Были разработаны следующие функции:

- getjewelrycostbyname. Получить стоимость украшения по имени;
- getjewelrycostbyvendorcode. Получить стоимость украшения по коду поставщика;
- getjewelrycountbyname. Получить количество украшений по имени;
- getjewelrycountbyvendorcode. Получить количество украшений по коду поставщика;
- get_db_reports. Получить отчеты по базе данных;
- check_status_update. Проверить обновление статуса;
- adding_jewelry_trigger. Триггер для добавления украшения;
- get_sales_statistics. Получить статистику продаж;
- get_top_selling_jewelry. Получить список самых продаваемых украшений;
- getjewelryinfo. Получить информацию об украшениях;
- view_stock_requests. Просмотреть запросы на запас;
- get_newest_reviews. Получить самые новые отзывы;
- getjewelryinfobyvendorcode. Получить информацию об украшениях по коду поставщика;
- getjewelryinfobyname. Получить информацию об украшениях по имени.

Пример создания функции для вывода всех покупателей представлен в листинге 3.2.

```
create or replace function get_all_customers()
returns table (
    id_cus int,
    first_name varchar,
    second_name varchar,
    address varchar,
    phone_cus varchar
) security definer
as $$
begin
    return query select * from customers;
end;
$$ language plpgsql;
```

Листинг 3.2. – Пример создания функции для вывода всех покупателей

Листинги некоторых процедур и функций представлены в приложении Б и В соответственно.

3.4. Триггеры

Триггер – хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE, или изменением UPDATE данных в определённой таблице реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики.

Были разработаны следующие триггеры:

– stock_request_status_update. Триггер для обновления даты поставки при пополнении товара со склада представлен в листинге 3.3;

– insert_jewelry_trigger. Триггер после добавления украшения, оповещает об успешном добавлении.

```
create or replace function check_status_update()
returns trigger
security definer
as $$
begin
    if new.status = 'Successful' then
        call           updatedeliverydatetimebyjewelry(new.id_jewelry,
new.requested_amount, current_date, current_time::time);
    end if;
    return new;
end;
$$ language plpgsql;
--
create trigger stock_request_status_update
after update of status on stock_requests
for each row
execute function check_status_update();
```

Листинг 3.3 – Пример создания триггера для обновления даты поставки

Листинги других созданных триггеров представлены в приложении Г.

3.5. Вывод по разделу

В данном разделе было описаны созданные необходимые объекты для работы с разрабатываемой базой данных. Были разработаны такие объекты БД как процедуры, функции, пользователи, роли, таблицы и триггеры.

4. Описание процедур импорта и экспорта

4.1. Процедуры импорта и экспорта

XML (eXtensible Markup Language) – это расширяемый язык разметки, предназначенный для хранения и передачи данных. Он разработан для того, чтобы быть как человеко-, так и машиночитаемым, что делает его популярным для обмена информацией между различными системами.

Экспорт и импорт данных между базой данных и форматом XML — это распространенные операции, которые позволяют передавать данные между различными системами и приложениями.

Процедуры применяются для таблицы Customers, чтобы не потерять данные о покупателях или подгрузить данные о новых клиентах.

Код создания процедуры представлен на листинге 4.1.

```
    CREATE OR REPLACE PROCEDURE ExportCustomersToXML file_path TEXT)
AS $$
    DECLARE
        xml_data TEXT := '<?xml version="1.0" encoding="UTF-
8"?><customers>';
        customer_rec RECORD;
    BEGIN
        FOR customer_rec IN SELECT * FROM CUSTOMERS LOOP
            xml_data := xml_data || '<customer>';
            xml_data := xml_data || '<ID_CUS>' || customer_rec.ID_CUS
|| '</ID_CUS>';
            xml_data    :=    xml_data    ||    '<FIRST_NAME>'    ||
COALESCE(customer_rec.FIRST_NAME, '') || '</FIRST_NAME>';
            xml_data    :=    xml_data    ||    '<SECOND_NAME>'    ||
COALESCE(customer_rec.SECOND_NAME, '') || '</SECOND_NAME>';
            xml_data     :=     xml_data     ||     '<ADDRESS>'     ||
COALESCE(customer_rec.ADDRESS, '') || '</ADDRESS>';
            xml_data    :=    xml_data    ||    '<PHONE_CUS>'    ||
COALESCE(customer_rec.PHONE_CUS, '') || '</PHONE_CUS>';
            xml_data := xml_data || '</customer>';
        END LOOP;
        xml_data := xml_data || '</customers>';
        PERFORM pg_write_file(file_path, xml_data);
        RAISE NOTICE 'Данные клиентов успешно экспортированы в файл %',
file_path;
    EXCEPTION WHEN OTHERS THEN
            RAISE NOTICE 'Произошла ошибка при экспорте данных в
XML: %', SQLERRM;
    END;
    $$ LANGUAGE plpgsql;
```

Листинг 4.1. – Процедура экспорта в формат XML

Процедура импорта представлена в листинге 4.2.

```
CREATE OR REPLACE PROCEDURE ImportCustomersFromXML(file_path VARCHAR)
AS $$
DECLARE
    xml_data TEXT;
BEGIN
    xml_data := pg_read_file(file_path);
    IF xml_data IS NULL THEN
        RAISE EXCEPTION 'Не удалось прочитать данные из файла %',
file_path;
    END IF;
    RAISE INFO 'Прочитанные данные из файла: %', xml_data;
    CREATE
        TEMP TABLE tmp_customers (
        FIRST_NAME VARCHAR,
        SECOND_NAME VARCHAR,
        ADDRESS VARCHAR,
        PHONE_CUS VARCHAR
     );
    BEGIN
        EXECUTE 'INSERT INTO tmp_customers (FIRST_NAME, SECOND_NAME,
ADDRESS,PHONE_CUS)
     SELECT
     unnest(xpath(''/customers/customer/FIRST_NAME/text()'',
xmlparse(document ''' || xml_data || '''))::text AS first_name,

unnest(xpath(''/customers/customer/SECOND_NAME/text()'',
xmlparse(document ''' || xml_data || '''))::text AS second_name,

unnest(xpath(''/customers/customer/ADDRESS/text()'', xmlparse(document
''' || xml_data || '''))::text AS address,

unnest(xpath(''/customers/customer/PHONE_CUS/text()'',
xmlparse(document ''' || xml_data || '''))::text AS phone_cus';
    EXCEPTION
        WHEN OTHERS THEN
            RAISE EXCEPTION 'Произошла ошибка при импорте данных из
XML: %', SQLERRM;
    END;
    RAISE
       INFO 'Данные клиентов успешно импортированы из файла % во
временную таблицу tmp_customers', file_path;
END;
$$
LANGUAGE plpgsql;
```

Листинг 4.2. – Процедура импорта из формата XML

4.2. Вывод по разделу

В данном разделе были описаны примеры разработанных процедур импорта и экспорта данных в таблице «Customers». Также были продемонстрированы листинги кода процедур.

5. Тестирование производительности

5.1. Тестирование производительности на таблице Jewerlies

Тестирование производительности базы данных является важным для выявления и решения проблем, которые могут негативно сказываться на работе приложений. Его целью является обеспечение быстрой и эффективной работы базы данных и удовлетворение потребностей пользователей. Для тестирования производительности в таблицу Jewelries были добавлены 100 000 записей.

Код добавления 100 000 строк представлен в листинге 6.1.

```
    DO $$
    DECLARE
        i INTEGER := 1;
        metals VARCHAR[] := ARRAY['Gold', 'Silver', 'Platinum',
'Titanium', 'Rose Gold'];
        cat_count INTEGER;
        cat_id INTEGER;
    BEGIN
        -- Получаем количество записей в таблице CATEGORIES_TEST
        SELECT COUNT(*) INTO cat_count FROM CATEGORIES_TEST;
            WHILE i <= 100000 LOOP

            SELECT ID_CATEGORY INTO cat_id FROM CATEGORIES_TEST OFFSET
floor(random() * cat_count) LIMIT 1;
            INSERT INTO TEST_JEWELRIES (NAME, VENDOR_CODE, WEIGHT,
METALL, DESCRIPTION, COST, STORE_AMOUNT, ID_CATEGORY, DISCOUNTS)
            VALUES (
                'Jewelry ' || i,                          -- NAME
                'VC' || i,                               -- VENDOR_CODE
                RANDOM() * 100,                             -- WEIGHT
(генерация случайного числа от 0 до 100)
                metals[i % 5 + 1],                          -- METALL
(периодическое повторение строковых значений)
                'Description ' || i,                     -- DISCRIPTION
                ROUND((RANDOM() * 1000)::NUMERIC, 2),       -- COST
(генерация случайной цены от 0 до 1000)
                (i % 100 + 1),                              -- AMOUNT
(периодическое повторение значений от 1 до 100)
                cat_id,                                  -- ID_CATEGORY
(периодическое повторение значений от 1 до 20)
                (i % 3 + 1)                               -- DISCOUNTS
(периодическое повторение значений от 1 до 3)
            );
            i := i + 1;
        END LOOP; END $$;
```

Листинг 6.1. – Код добавления 100 000 строк в таблицу TEST_JEWELRIES

Для того, чтобы оценить производительность, необходимо сделаь запросы к заполненной таблице с индексом и без (использование индексов уменьшает время обработки запроса). На рисунке 6.1. представлен результат select-запроса с оператором where к таблице Test_Jewelries до добавления индекса. Время выполнения составляет 0.012 с.

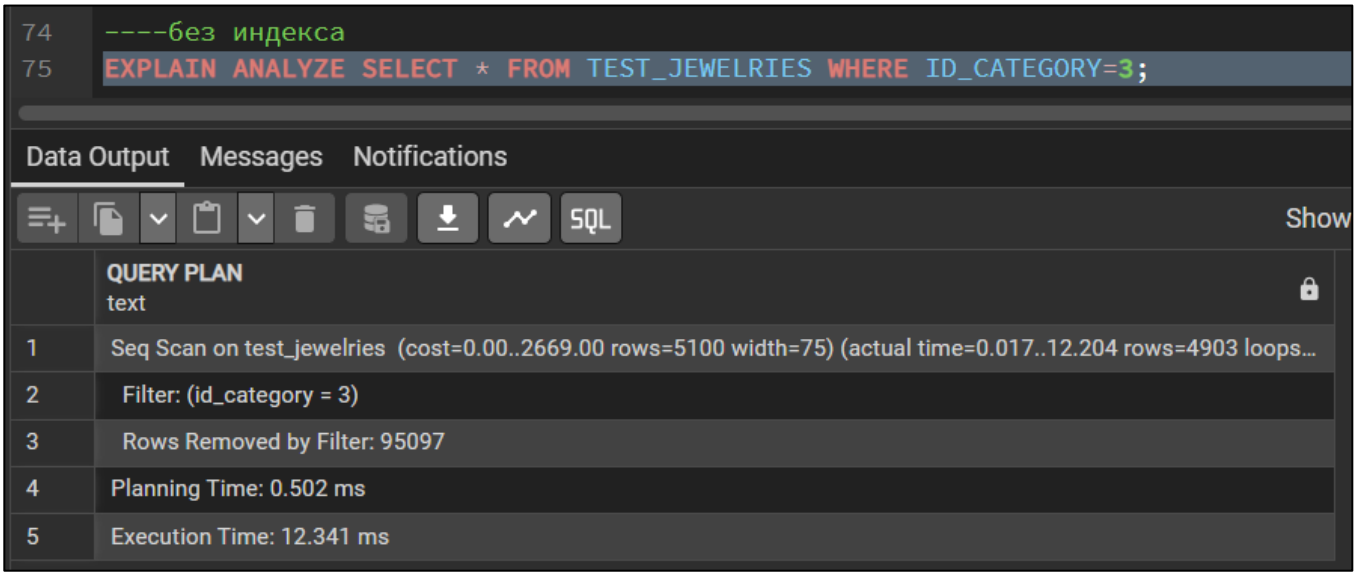

Рисунок 6.1. – Результат select-запроса с оператором where к таблице Test_Jewelries без индекса

На рисунке 6.2 представлен результат select-запроса с оператором where к таблице Test_Jewelries после добавления индекса. Время выполнения уже составляет 0.002 с, что показывает, что добавление индекса повышает производительность таблицы, уменьшая время запроса.

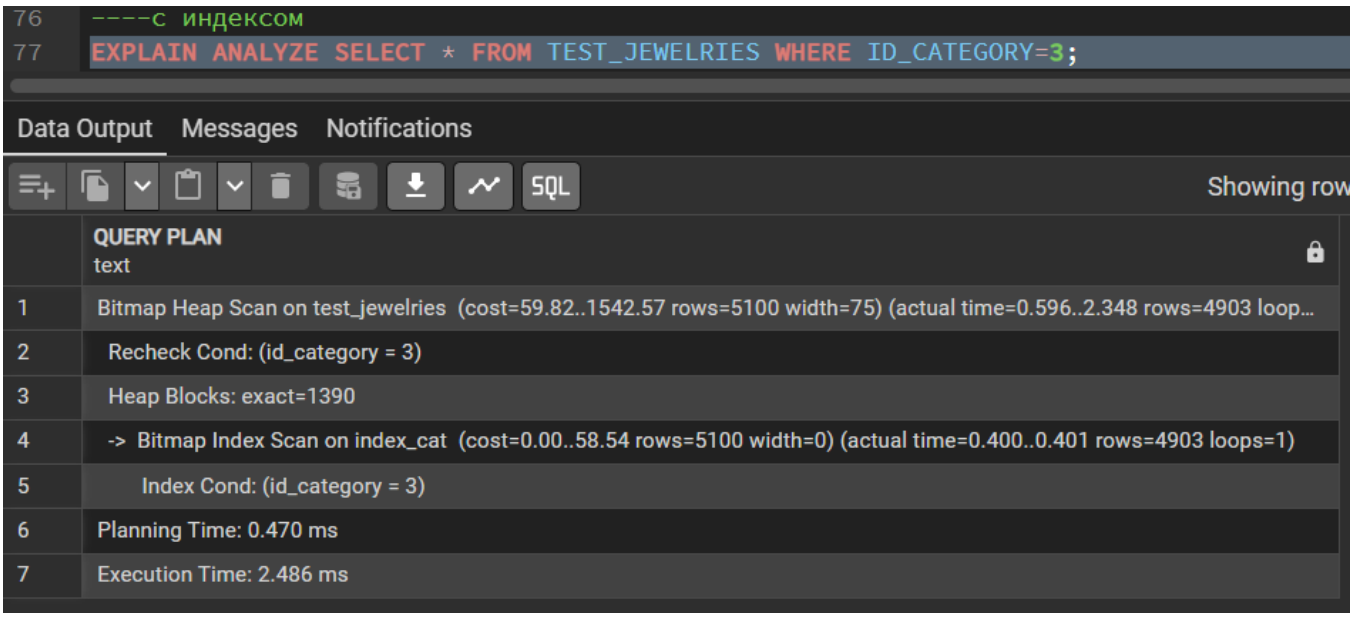

Рисунок 6.2. – Результат select-запроса с оператором where к таблице Test_Jewelries с индексом

5.2. Вывод по разделу

В данном разделе было описано тестирование производительности разрабатываемой базы данных. Были выполнены select-запросы и продемонстрировано время их выполнения до и после добавления индекса.

6. Описание технологии и ее применение в базе данных

6.1. Технология разработки системы мониторинга за состоянием базы данных

Технология мониторинга за состоянием базы данных — это комплекс средств и методов, предназначенных для непрерывного отслеживания и анализа различных параметров и метрик работы базы данных с целью обеспечения её бесперебойной работы, оптимальной производительности и своевременного выявления и устранения потенциальных проблем. Основные задачи мониторинга включают наблюдение за нагрузкой на сервер, выполнением запросов, состоянием соединений, обнаружением аномалий. В результате, мониторинг базы данных способствует поддержанию её стабильности, безопасности и эффективности работы.

Для мониторинга состояния базы данных PostgreSQL в данном курсовом проекте используется PgAdmin 4. PgAdmin — популярное графическое средство для администрирования PostgreSQL. Программа упрощает основные задачи администрирования, отображает объекты баз данных, позволяет выполнять запросы SQL. На рисунке 6.1. представлен Dashboard, который предоставляет метрики по БД.

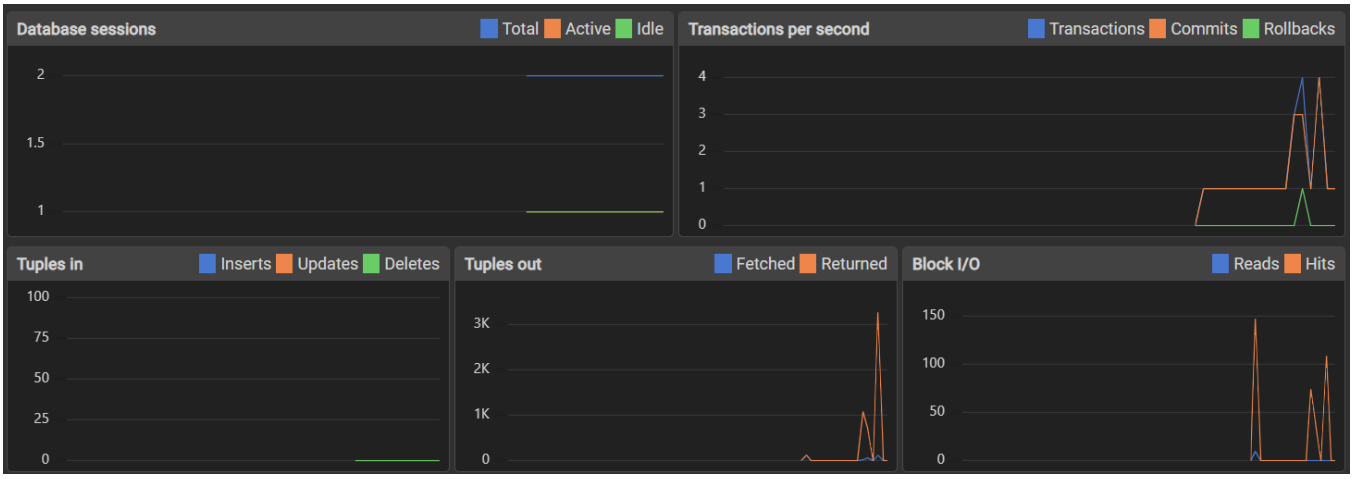

Рисунок 6.1. – Dashboard pgAdmin4

График Database sessions отображает количество активных сессий базы данных во времени. Зависимость:

– вверх: график идет вверх, когда увеличивается количество подключений к базе данных. Это может происходить в периоды пиковой нагрузки, когда много пользователей или приложений одновременно обращаются к базе данных;

– вниз: график идет вниз, когда сессии закрываются или завершаются. Это может быть результатом окончания рабочего дня, уменьшения активности пользователей или оптимизации запросов.

График Transactions per second (TPS) показывает количество транзакций, обрабатываемых базой данных в секунду. Зависимость:

– вверх: график идет вверх, когда увеличивается количество выполняемых транзакций. Это может быть связано с повышенной активностью пользователей, выполнением массовых обновлений или загрузкой данных;

– вниз: график идет вниз, когда количество транзакций уменьшается. Это может быть вызвано уменьшением активности пользователей, завершением массовых операций или проблемами с производительностью базы данных.

График Tuples in показывает количество вставленных и обновленных кортежей (строк) в базу данных. Зависимость:

– вверх: график идет вверх, когда в базу данных вставляется или обновляется большое количество данных. Это происходит при массовых операциях вставки, обновлениях данных или миграциях данных;

– вниз: график идет вниз, когда активность по вставке и обновлению данных снижается.

График Tuples out показывает количество извлеченных кортежей (строк) из базы данных. Зависимость:

– вверх: график идет вверх, когда выполняется много операций чтения данных, например, при выполнении сложных запросов, отчетов или выборок данных пользователями;

– вниз: график идет вниз, когда количество операций чтения данных уменьшается.

График Block I/O отображает количество операций ввода-вывода блоков, выполняемых базой данных. Зависимость:

– вверх: график идет вверх, когда увеличивается активность чтения и записи данных на диске. Это может происходить при выполнении интенсивных запросов, больших операций вставки/обновления или при выполнении резервного копирования данных;

– вниз: график идет вниз, когда активность ввода-вывода блоков снижается, что может быть связано с уменьшением рабочей нагрузки или улучшением производительности за счет кэширования данных.

Также для ведения статистики необходимо включить расширение pg_stat_statements и настроить log_statement_stats. Код приведен в листинге 6.1.

```
ALTER SYSTEM SET log_statement_stats TO on;
CREATE EXTENSION pg_stat_statements;
```

Листинг 6.1. – Код настроек для статистики

Пример использования – вывод самых времязатратных запросов к БД, что помогает разработчикам оптимизировать работу. Вывод приведен на рисунке 6.2.

	query text
1	CREATE TEMP TABLE pga_tmp_zombies(jagpid int4)
2	SELECT set_config($1,$2,$3) FROM pg_show_all_settings() WHERE name = $4
3	/*pga4dash*/
4	SELECT J.jobid FROM pgagent.pga_job J WHERE jobenabled AND jobagentid IS NULL AND jobnextrun <= now() AND (jobhostagent = $1 OR jobhostagent = $2) ORDER BY jobnextrun
5	SELECT set_config($1,$2,$3) FROM pg_show_all_settings() WHERE name = $4
6	SELECT
7	DROP TABLE pga_tmp_zombies
8	SELECT DISTINCT att.attname as name, att.attnum as OID, pg_catalog.format_type(ty.oid,$1) AS datatype,
9	SELECT * FROM pg_stat_statements
10	SELECT

Рисунок 6.2. – Вывод запросов

Для того, чтобы сохранять данные о периодических проверках состояния БД, была создана таблица db_mon_stats, в которой хранится данные проверки и дата проверки. Код создания приведен в листинге 6.2.

```
create table db_mon_stats(
    id SERIAL PRIMARY KEY,
    database_name VARCHAR(255),
    total_size VARCHAR(50),
    table_size VARCHAR(50),
    index_size VARCHAR(50),
    used_space VARCHAR(50),
    active_connections INTEGER,
    idle_connections INTEGER,
    total_connections INTEGER,
    total_commits INTEGER,
    total_rollbacks INTEGER,
    deadlocks INTEGER,
    critical_errors INTEGER DEFAULT 0,
    warning_errors INTEGER DEFAULT 0,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

Листинг 6.2. – Создание таблицы для отчетов

Для удобства периодической проверки производительности и состояния БД можно установить расширение pgAgent, с помощью которого можно выполнять периодические работы (Jobs). Скриншот приведен на рисунке 6.3.

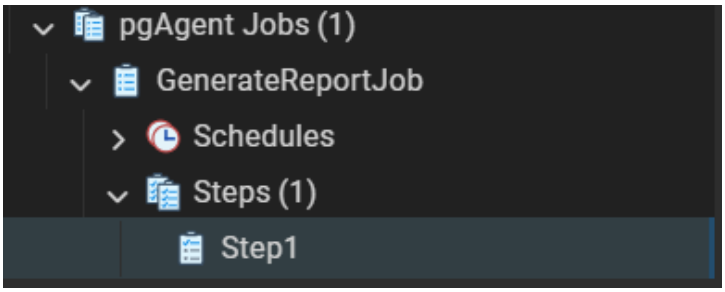

Рисунок 6.3. – pgAgent Jobs

Выполнение работ настраивается через интерфейс, где указываются параметры (база данных, пользователь), исполняемый код (рис. 6.4), расписание и другие необходимые настройки.

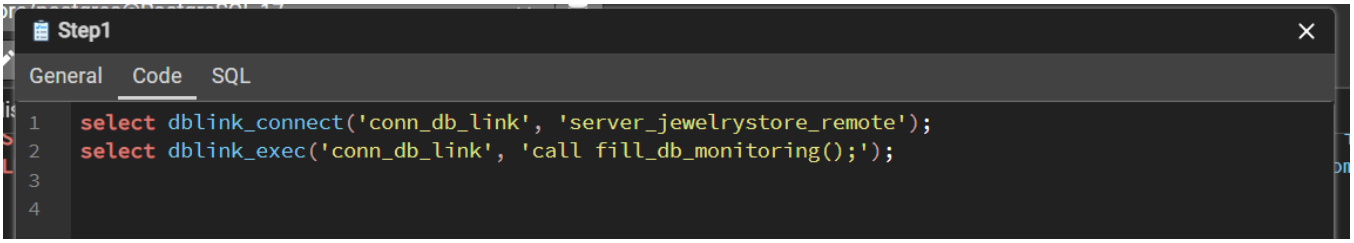

Рисунок 6.4. – Настройка выполнения периодических проверок.

Пример содержания таблицы отчетов приведен на рисунке 6.5.

id integer	database_name character varying	total_size character var	table_size character va	index_si charact	used_sp charact	active_c integer	idle_cor integer	total_coni integer	total_comi integer	total_rollba integer	deadloc integer	critical_ integer	warning_ integer	cur_timestamp timestamp without time zone
2	jewelrystore	9435 kB	9720 kB	0 by...	0 by...	1	7	8	2190	76	0	0	0	2024-12-16 22:32:15.710792
4	jewelrystore	9435 kB	9720 kB	0 by...	0 by...	1	7	8	2224	81	0	0	0	2024-12-16 22:42:05.744604
5	jewelrystore	9435 kB	9720 kB	0 by...	0 by...	1	7	8	2241	84	0	0	0	2024-12-16 22:45:02.486029
6	jewelrystore	9435 kB	9720 kB	0 by...	0 by...	1	7	8	2250	85	0	0	0	2024-12-16 22:47:13.438553
7	jewelrystore	9435 kB	9720 kB	0 by...	0 by...	1	6	7	2253	85	0	0	0	2024-12-16 22:47:29.802712
8	jewelrystore	9547 kB	9832 kB	0 by...	0 by...	1	3	4	2889	91	0	0	0	2024-12-17 16:10:11.814235
9	jewelrystore	9547 kB	9832 kB	0 by...	0 by...	1	3	4	2891	91	0	0	0	2024-12-17 16:11:05.249567
10	jewelrystore	9547 kB	9832 kB	0 by...	0 by...	1	3	4	2895	91	0	0	0	2024-12-17 16:12:05.584844
11	jewelrystore	9547 kB	9832 kB	0 by...	0 by...	1	3	4	2899	91	0	0	0	2024-12-17 16:13:06.097768

Рисунок 6.5. – Содержание таблицы отчетов

Код создания процедуры для создания отчета представлен в приложении Д.

6.2. Вывод по разделу

Технологии мониторинга состояния баз данных являются критически важными инструментами для эффективного управления и оптимизации производительности баз данных. Анализ использования системных ресурсов различными запросами помогает более рационально распределять ресурсы, предотвращая перегрузки и улучшая стабильность системы.

7. Руководство пользователя

7.1. Сценарий использования для администратора

Сценарий работы с базой данных (БД) представляет собой последовательность шагов или действий, необходимых для взаимодействия с БД. Этот сценарий описывает типичные операции, которые могут выполняться с БД.

Для данного проекта были разработаны сценарии, которые отражают функциональные возможности каждого пользователя. Пример сценария использования для пользователя administrator представлен на рисунке 7.1.

```
---------admin
--проверка запросов от менеджера
select  view_stock_requests();
--обновление статуса запроса
call update_request_status(6, 'Successful');
--получение общей статистики
select * from get_sales_statistics();
--получение статистики о самых продаваемых товарах
select * from get_top_selling_jewelry(12, 2024);
--вывод товаров по рейтингу
DO $$
DECLARE
    sorted_cursor REFCURSOR;
    sorted_record RECORD;
BEGIN
    CALL get_sorted_jewelry_by_rating('desc', sorted_cursor);
```

Рисунок 7.1. – Пример возможностей администратора

Данный сценарий использования показывает, что пользователь administrator имеет доступ к функциям для получения данных о запросах на склад, для получения базовой статистики и статистики продаваемости товаров. Также представлена процедура для вывода товаров согласно их рейтингу (как по возрастанию или убыванию).

Ниже на рисунке 7.2 продемонстрирован пример вызова функции для получения самых продаваемых товаров.

```
126  --получение статистики о самых продаваемых товарах
127  select * from get_top_selling_jewelry(12, 2024);
```

Data Output | Messages | Notifications

SQL

	id_jewelry integer	name character varying	total_sales integer
1	1	Gold Necklace	15
2	5	Emerald Pendant	11
3	7	Sapphire Ring	10
4	9	Gemstone Earrings	5
5	10	Gold Cufflinks	3

Рисунок 7.2. – Вывод статистики

Пример изменения цены товара приведен на рисунке 7.3.

```
148  -----
149  call update_jewelry_cost(1, 1);
150  --
```

Data Output | Messages | Notifications

```
ЗАМЕЧАНИЕ:  Стоимость ювелирного изделия ID 1 успешно обновлена на 1.
CALL
```

Рисунок 7.3. – Изменение цены товара

7.2. Вывод по разделу

В данном разделе были представлены функция и процедура, подтверждающая работоспособность базы данных, а также продемонстрирован сценарий.

Заключение

В ходе выполнения курсового проекта была успешно разработана база данных для магазина украшений, которая отвечает современным требованиям управления информацией. Создание структуры базы данных позволило оптимизировать процесс хранения и обработки данных о товарах, клиентах и заказах, что, в свою очередь, обеспечивает быстрый доступ к необходимой информации.

Реализация функционала поиска изделий по различным параметрам значительно улучшила пользовательский опыт, позволяя клиентам быстро находить нужные товары. Механизм внесения изменений в каталог изделий гарантирует актуальность данных, что является ключевым аспектом в динамично меняющемся бизнесе.

Определение ролей пользователей, таких как администратор, продавец, покупатель и менеджер, создало четкую структуру доступа к информации, что повышает безопасность и управляемость системы. Внедрение системы мониторинга состояния базы данных обеспечивает контроль за производительностью и надежностью работы, что критично для поддержания высоких стандартов обслуживания.

Таким образом, выполненный проект не только продемонстрировал возможности СУБД PostgreSQL, но и стал основой для дальнейшего развития системы управления данными в магазине украшений. В будущем стоит рассмотреть возможность интеграции дополнительных функций, таких как аналитика продаж и прогнозирование спроса, что позволит предприятию более эффективно адаптироваться к изменениям на рынке.

Список используемых источников

1. Сайт магазина ювелирных украшений «Pandora» [Электронный ресурс]. – Режим доступа: https://ie.pandora.net/;

2. Сайт магазина ювелирных украшений «Cartier» [Электронный ресурс]. – Режим доступа: https://www.cartier.com/ru-ru/;

3. Сайт магазина ювелирных украшений «Ziko» [Электронный ресурс]. – Режим доступа: https://ziko.by/;

4. Сайт документации расширения pgAgent [Электронный ресурс]. – Режим доступа: https://www.pgadmin.org/docs/pgadmin4/development/pgagent.html;

Приложение А

Листинг создания таблиц

```
CREATE TABLE JEWELRIES (
  ID_JEWELRY SERIAL PRIMARY KEY,
  NAME VARCHAR(255) NOT NULL,
  VENDOR_CODE VARCHAR(30) NOT NULL,
  WEIGHT FLOAT NOT NULL,
  METALL VARCHAR(80) NOT NULL,
  DESCRIPTION TEXT,
  COST NUMERIC(10,2) NOT NULL,
  STORE_AMOUNT INTEGER NOT NULL,
  ID_CATEGORY INTEGER,
  DISCOUNTS INTEGER,
  FOREIGN KEY (DISCOUNTS) REFERENCES DISCOUNTS (ID_DISCOUNT) ON DELETE
SET NULL,
  FOREIGN KEY (ID_CATEGORY) REFERENCES CATEGORIES (ID_CATEGORY) ON
DELETE CASCADE
);
CREATE TABLE DELIVERIES(
  ID_DEV SERIAL PRIMARY KEY,
  DELIVERY_DATE DATE NOT NULL,
  DELIVERY_TIME INTEGER NOT NULL,
  AMOUNT INTEGER,
  ID_JEWELRY SERIAL,
  FOREIGN KEY (ID_JEWELRY) REFERENCES JEWELRIES (ID_JEWELRY) ON DELETE
SET NULL);
ALTER TABLE DELIVERIES
ALTER COLUMN DELIVERY_TIME TYPE TIME USING
  TO_TIMESTAMP(DELIVERY_TIME::TEXT, 'HH24MI');
CREATE TABLE CUSTOMERS(
  ID_CUS SERIAL PRIMARY KEY,
  FIRST_NAME VARCHAR(255) NOT NULL,
  SECOND_NAME VARCHAR(255),
  ADDRESS VARCHAR(255),
  PHONE_CUS VARCHAR(20)
);
CREATE TABLE SALES (
    ID_SALES SERIAL PRIMARY KEY,
    ID_JEWELRY INTEGER NOT NULL,
    ID_CUS INTEGER NOT NULL,
    AMOUNT INTEGER,
    DATE_SALES DATE,
    FINAL_COST NUMERIC(10,2),
    FOREIGN KEY (ID_JEWELRY) REFERENCES JEWELRIES(ID_JEWELRY) ON DELETE
SET NULL,
    FOREIGN KEY (ID_CUS) REFERENCES CUSTOMERS(ID_CUS) ON DELETE SET
NULL);
```

```
CREATE TABLE REVIEWS(
  ID_REVIEW SERIAL PRIMARY KEY,
  ID_JEWELRY  INTEGER NOT NULL,
  ID_CUS  INTEGER  NOT NULL,
  RATING  INTEGER,
  COMMENT TEXT,
  FOREIGN KEY (ID_JEWELRY) REFERENCES JEWELRIES(ID_JEWELRY) ON DELETE
CASCADE,
  FOREIGN KEY (ID_CUS) REFERENCES CUSTOMERS(ID_CUS) ON DELETE SET NULL
);
CREATE TABLE CATEGORIES(
   ID_CATEGORY SERIAL PRIMARY KEY,
   NAME VARCHAR(255) UNIQUE
);
CREATE TABLE DISCOUNTS(
  ID_DISCOUNT SERIAL PRIMARY KEY,
  DISCOUNT INTEGER UNIQUE
);
CREATE TABLE STOCK_REQUESTS (
    ID_REQUEST SERIAL PRIMARY KEY,
    ID_JEWELRY INTEGER NOT NULL,
    REQUEST_DATE DATE NOT NULL,
    REQUESTED_AMOUNT INTEGER NOT NULL,
    STATUS VARCHAR(50) DEFAULT 'Pending',
    FOREIGN KEY (ID_JEWELRY) REFERENCES JEWELRIES(ID_JEWELRY)
);
```

Приложение Б

Листинг созданных процедур

```
create or replace procedure update_request_status(
    request_id int,
    new_status varchar
)
language plpgsql
security definer as $$
declare
    requested_amount integer;
    jewelry_id integer;
    old_status varchar(50);
begin
    select    r.requested_amount,    r.id_jewelry,    r.status    into
requested_amount, jewelry_id, old_status
    from stock_requests as r
    where r.id_request = request_id;

    if not found then
        raise exception 'запрос с id % не найден.', request_id;
    end if;

    if old_status = 'Successful' then
        raise notice 'статус запроса id % не может быть обновлен на %,
т.к. текущий статус %.',
            request_id, new_status, old_status;
    else
        update stock_requests
        set status = new_status
        where id_request = request_id;

        if new_status = 'Successful' then
            update jewelries
            set store_amount = store_amount + requested_amount
            where id_jewelry = jewelry_id;

            raise notice 'статус запроса id % обновлен на % и количество
товара увеличено на %.',
                        request_id, new_status, requested_amount;
        else
            raise  notice  'статус  запроса  id  %  обновлен  на  %.',
request_id, new_status;
        end if;
    end if;
end;
$$;
```

Приложение В

Листинг созданных функций

```
create or replace function getjewelryinfobyvendorcode(
    p_vendor_code varchar
)
returns table (
    id_jewelry integer,
    name varchar,
    cost numeric,
    vendor_code varchar,
    store_amount integer
)
security definer
as $$
begin
    return query
    select * from jewelries where vendor_code = p_vendor_code;
end;
$$ language plpgsql;
create or replace function getjewelrycountbyvendorcode(
    f_vendor_code varchar
)
returns integer as $$
declare
    product_count integer;
begin
    select count(*) into product_count from jewelries where vendor_code
= f_vendor_code;
    return product_count;
end;
$$ language plpgsql;
create or replace function view_stock_requests()
returns table (
    id_request integer,
    id_jewelry integer,
    request_date date,
    status varchar(50)
)
security definer
as $$
begin
    return query select r.id_request, r.id_jewelry, r.request_date,
r.status
                 from stock_requests as r;
end;
$$ language plpgsql;
```

Приложение Г

Листинг созданных триггеров

```
create or replace function adding_jewelry_trigger()
returns trigger
security definer
as $$
begin
    if tg_op = 'insert' and tg_when = 'after' and exists(select 1 from
pg_trigger where tgname = tg_name and tgenabled = 'o') then
        raise notice 'данные успешно вставлены в таблицу jewelries.';
    end if;
    return new;
end;
$$ language plpgsql;

create trigger insert_jewelry_trigger
after insert on jewelries
for each statement
execute function adding_jewelry_trigger();
```

Приложение Д

Листинг создания процедуры для выполнения технологии

```
CREATE OR REPLACE PROCEDURE fill_db_monitoring()
LANGUAGE plpgsql
AS $$
DECLARE
    db_name VARCHAR(255) := current_database();
    total_size VARCHAR(50);
    table_size VARCHAR(50);
    index_size VARCHAR(50);
    used_space VARCHAR(50);
    active_conn INTEGER;
    idle_conn INTEGER;
    total_conn INTEGER;
    total_commits INTEGER;
    total_rollbacks INTEGER;
    deadlock_count INTEGER;
    critical_errors INTEGER := 0;
    warning_errors INTEGER := 0;
BEGIN
    SELECT
        pg_size_pretty(pg_database_size(db_name)) INTO total_size;
    SELECT
        pg_size_pretty(SUM(pg_total_relation_size(oid)))          INTO
table_size
    FROM
        pg_class
    WHERE
        relkind = 'r';

    SELECT
        pg_size_pretty(SUM(pg_indexes_size(oid))) INTO index_size
    FROM
        pg_class
    WHERE
        relkind = 'i';
    SELECT
pg_size_pretty(pg_total_relation_size('pg_catalog.pg_stat_activity'))
INTO used_space;

    SELECT
        COUNT(*) FILTER (WHERE state = 'active') INTO active_conn
    FROM
        pg_stat_activity
    WHERE
        datname = db_name;
    SELECT
```

```
        COUNT(*) FILTER (WHERE state = 'idle') INTO idle_conn
    FROM
        pg_stat_activity
    WHERE
        datname = db_name;

    SELECT
        numbackends,
        xact_commit,
        xact_rollback,
        deadlocks INTO
        total_conn,
        total_commits,
        total_rollbacks,
        deadlock_count
    FROM
        pg_stat_database
    WHERE
        datname = db_name;

    INSERT INTO db_mon_stats (database_name, total_size, table_size,
index_size,    used_space,    active_connections,    idle_connections,
total_connections,    total_commits,    total_rollbacks,    deadlocks,
critical_errors, warning_errors)
    VALUES (db_name, total_size, table_size, index_size, used_space,
active_conn, idle_conn, total_conn, total_commits, total_rollbacks,
deadlock_count, critical_errors, warning_errors);
END;
$$;
```