МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационные системы и технологии
Специальность 6-05-0612-01 Программная инженерия (профилизация Программное обеспечение информационных технологий)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Реализация базы данных интернет-магазина книг с применением репликации данных между серверами СУБД»

Выполнил студент Соленок Анастасия Александровна
(Ф.И.О.)

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

Зав. кафедрой к.т.н. Блинова Е.А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой _____

Минск 2025

Оглавление

Перечень условных обозначений и сокращений

СУБД – Система Управления Базой Данных

БД – База Данных

JSON – JavaScript Object Notation

UML – Unified Modeling Language

CLOB – Character Large Object

SQL – Structured Query Language

UTL_FILE – системный пакет UTLities for FILE operations

LPAD – функция Left PAD (выравнивание/заполнение слева)

CPU – Central Processing Unit

Введение

При проектировании базы данных интернет-магазина книг необходимо учитывать различные аспекты организации информации и управления пользователями. Основу структуры составляют роли пользователей, такие как главный администратор, продавец и покупатель. Каждая роль имеет свои функциональные обязанности и уровень доступа к данным, что обеспечивает безопасное и упорядоченное управление системой.

Создание соответствующих таблиц в базе данных является ключевым этапом проектирования. Таблицы, такие как пользователи, книги, заказы, позиции заказа и журнал действий (audit_log), формируют основу для хранения информации, обеспечивая структурированное и удобное управление данными.

Последующей важной частью разработки является реализация процедур, функций и триггеров. Триггеры позволяют автоматизировать действия при изменении данных (например, обновление количества товара на складе после оформления заказа), а процедуры и функции выполняют более сложные операции, такие как импорт/экспорт данных в формат JSON и расчет аналитических показателей.

Особое внимание уделяется репликации данных между серверами СУБД, что повышает отказоустойчивость, масштабируемость и производительность системы. Дополнительно используются современные технологии работы с данными, включая мультимедийные типы данных для хранения изображений обложек книг и хеширование паролей для безопасного хранения учетных записей пользователей.

В результате реализованная база данных обеспечивает надежное, структурированное и безопасное хранение информации, удобное управление заказами и пользователями, а также возможность масштабирования системы через репликацию данных.

Глава 1 Постановка задачи

Цель работы – разработка базы данных интернет-магазина книг с применением репликации данных между серверами СУБД. Задача состоит в предварительном проектировании UML-диаграмм и создании на их основе структуры базы данных, обеспечивающей управление пользователями, товарами и заказами, а также возможность анализа информации.

Необходимо реализовать роли пользователей (главный администратор, продавец, покупатель) с различными правами доступа, создать таблицы, индексы и представления для хранения и эффективной выборки информации о пользователях, книгах и заказах, а также журнал действий для отслеживания операций.

Для автоматизации работы с данными используются триггеры, процедуры и функции, включая импорт и экспорт данных в формате JSON и расчет статистики.

Особое внимание уделяется реализации репликации между серверами СУБД для повышения надежности и отказоустойчивости базы, а также тестированию производительности на больших таблицах и оптимизации структуры для ускорения работы запросов.

Кроме того, работа направлена на обеспечение целостности и непротиворечивости данных при одновременном доступе нескольких пользователей. Применение механизма репликации позволяет минимизировать риск потери данных и обеспечивает бесперебойную работу системы при нагрузках или сбоях на одном из серверов.

Важным аспектом является анализ пользовательских действий и ведение журнала операций, что позволяет не только отследить изменения, но и выявлять потенциальные ошибки или злоупотребления.

Внедрение процедур, функций и триггеров обеспечивает автоматизацию рутинных операций, сокращает вероятность ошибок и ускоряет выполнение типовых запросов. В результате реализованная база данных должна обеспечивать безопасное, удобное и масштабируемое управление интернет-магазином книг.

1.1 Обзор средств разработки

Для реализации курсового проекта был выбран комплекс программных средств, обеспечивающий полный цикл разработки от визуального проектирования до настройки сложных механизмов взаимодействия серверов. В качестве основной платформы для хранения данных и реализации бизнес-логики используется система управления базами данных Oracle Database 21c. Данная СУБД была выбрана благодаря своей надежности, поддержке мультимодельной архитектуры и наличию мощного процедурного языка, что позволяет эффективно обрабатывать транзакции интернет-магазина. Именно в среде Oracle создана вся структура базы данных, включая таблицы, индексы, а также программные модули — процедуры, функции и триггеры.

Для администрирования процессов репликации и управления взаимодействием с резервным контуром применяется среда SQL Server Management Studio (SSMS) версии 20. Использование этого инструмента обусловлено необходимостью настройки и мониторинга гетерогенной среды, где данные передаются между систе-

мами различных производителей. SSMS предоставляет развитые средства диагностики и визуализации, что позволяет контролировать корректность переноса информации и состояние подписчика репликации.

Проектирование инфологической и даталогической моделей выполнялось с помощью инструмента draw.io. Это кроссплатформенное средство позволило на начальном этапе разработки визуализировать структуру базы данных, определить связи между сущностями и спроектировать архитектуру потоков данных, что значительно упростило последующую физическую реализацию системы в Oracle.

1.2 Описание применяемой технологии

В ходе разработки проекта применяется ряд современных технологий, обеспечивающих надежность и функциональность системы. Фундаментальной основой является реляционная модель данных, реализованная средствами Oracle Database 21c. Данные хранятся в виде двумерных таблиц, связанных между собой ключами, что гарантирует их целостность и непротиворечивость. Для реализации сложной бизнес-логики на стороне сервера используется технология процедурного расширения языка SQL — PL/SQL. Она позволяет инкапсулировать операции с данными внутри хранимых процедур и триггеров, тем самым снижая нагрузку на сеть и повышая скорость обработки заказов.

Ключевой особенностью проекта является технология гетерогенной репликации данных. Этот механизм обеспечивает захват изменений, происходящих в основной базе данных Oracle, и их передачу на сервер, управляемый через SQL Server Management Studio. Применение репликации решает задачи повышения отказоустойчивости системы, создавая резервную копию данных в режиме реального времени, а также позволяет распределять нагрузку, выделяя отдельный узел для выполнения ресурсоемких аналитических запросов без замедления работы основного магазина.

1.3 Выводы по главе 1

В первой главе была выполнена постановка задачи на курсовой проект, определена цель работы и сформирован детализированный перечень задач, необходимых для создания эффективной базы данных интернет-магазина книг с учетом современных требований к производительности и отказоустойчивости . Был проведен комплексный анализ и обоснование выбора программных средств, включающих СУБД Oracle Database 21c, среду управления SQL Server Management Studio 20 и средство моделирования draw.io. Также описаны ключевые технологии проекта: реляционная модель, PL/SQL и гетерогенная репликация, обеспечивающая сохранность данных. Дополнительно были конкретизированы требования к подсистеме безопасности и ролевой модели, а также обоснована необходимость использования формата JSON для гибкой интеграции и хранимых процедур для автоматизации рутинных процессов. Сформулированные требования и выбранный технологический стек создают необходимую базу для дальнейшего проектирования UML-диаграмм и успешной физической реализации информационной системы.

Глава 2 Проектирование базы данных

2.1 Определение вариантов использования

В процессе анализа и разработки архитектуры проекта предполагается использование UML-диаграмм для более наглядного представления структуры, взаимодействия компонентов и логики приложения. Диаграмма представлена на рисунке 2.1.

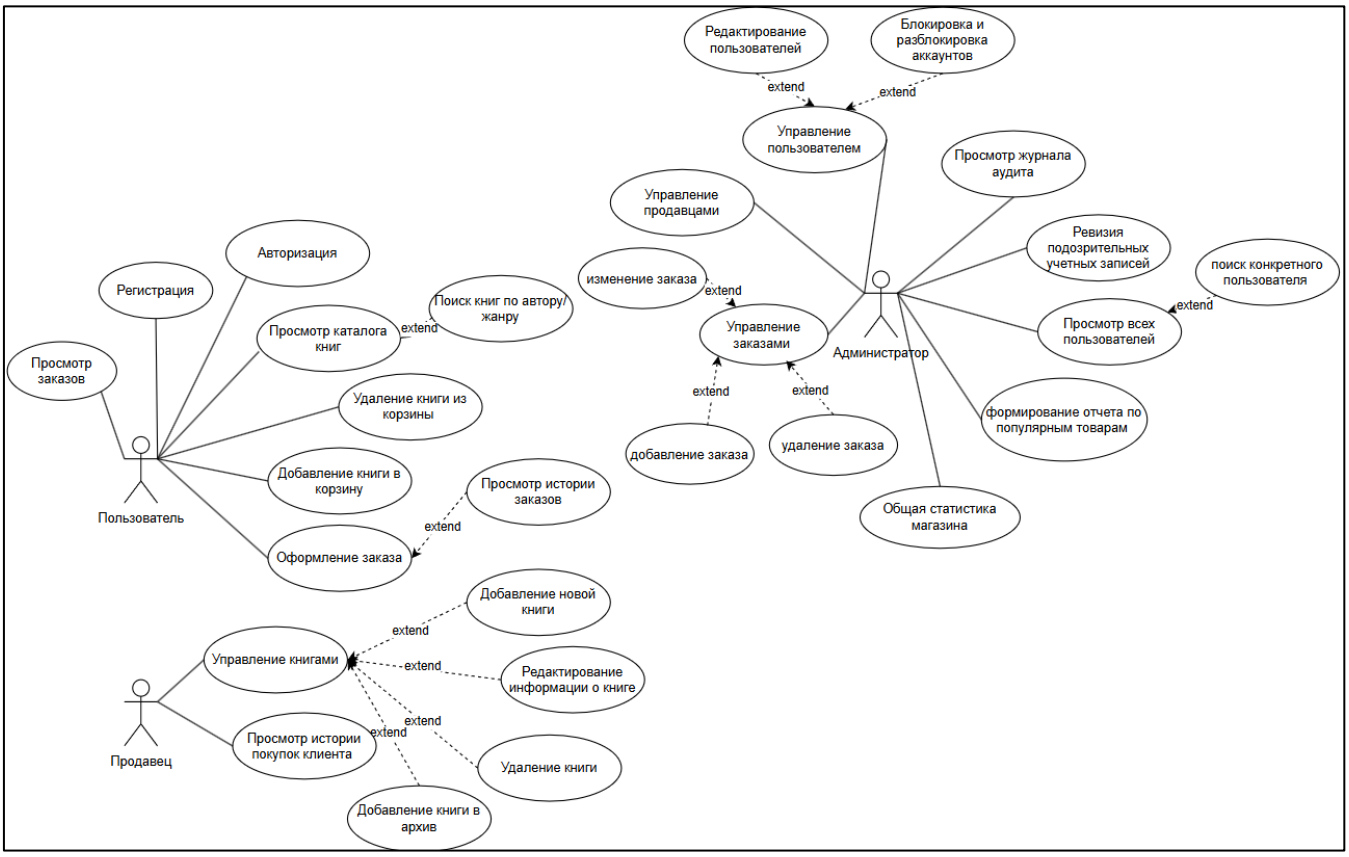


Рисунок 2.1 – UML-диаграмма

На представленной диаграмме вариантов использования визуализировано взаимодействие трех ключевых субъектов системы: Пользователя, Продавца и Администратора. Функционал Пользователя сосредоточен на клиентских операциях, включающих регистрацию, поиск литературы в каталоге, наполнение корзины и непосредственное оформление заказов. Роль Продавца предполагает управление контентом магазина, в частности добавление, удаление и редактирование информации о книгах, а также обработку поступающих заказов с обновлением их статусов. Администратор, обладая наивысшими правами доступа, осуществляет управление учетными записями всех участников системы, анализирует статистические показатели деятельности магазина и отвечает за техническую настройку репликации данных между серверами.

Диаграмма облегчает понимание структуры системы, показывая, какие действия могут выполнять покупатель, продавец и администратор.

Это визуальное представление помогает пользователям и разработчикам увидеть, как каждая роль взаимодействует с системой и какие функции ей доступны.

Диаграмма также упрощает обучение и использование приложения, демонстрируя последовательность действий и возможности для каждой роли.

2.2 Диаграмма базы данных UML

На следующем этапе процесса предстоит составить наглядную картину базы данных.

Разработка программного обеспечения начинается с моделирования данных, что позволяет определить структуру системы и связи между объектами. Грамотно построенная модель обеспечивает высокую производительность и упрощает дальнейшее проектирование, визуализируя логику работы базы данных. Диаграмма базы данных представлена на рисунке 2.2.

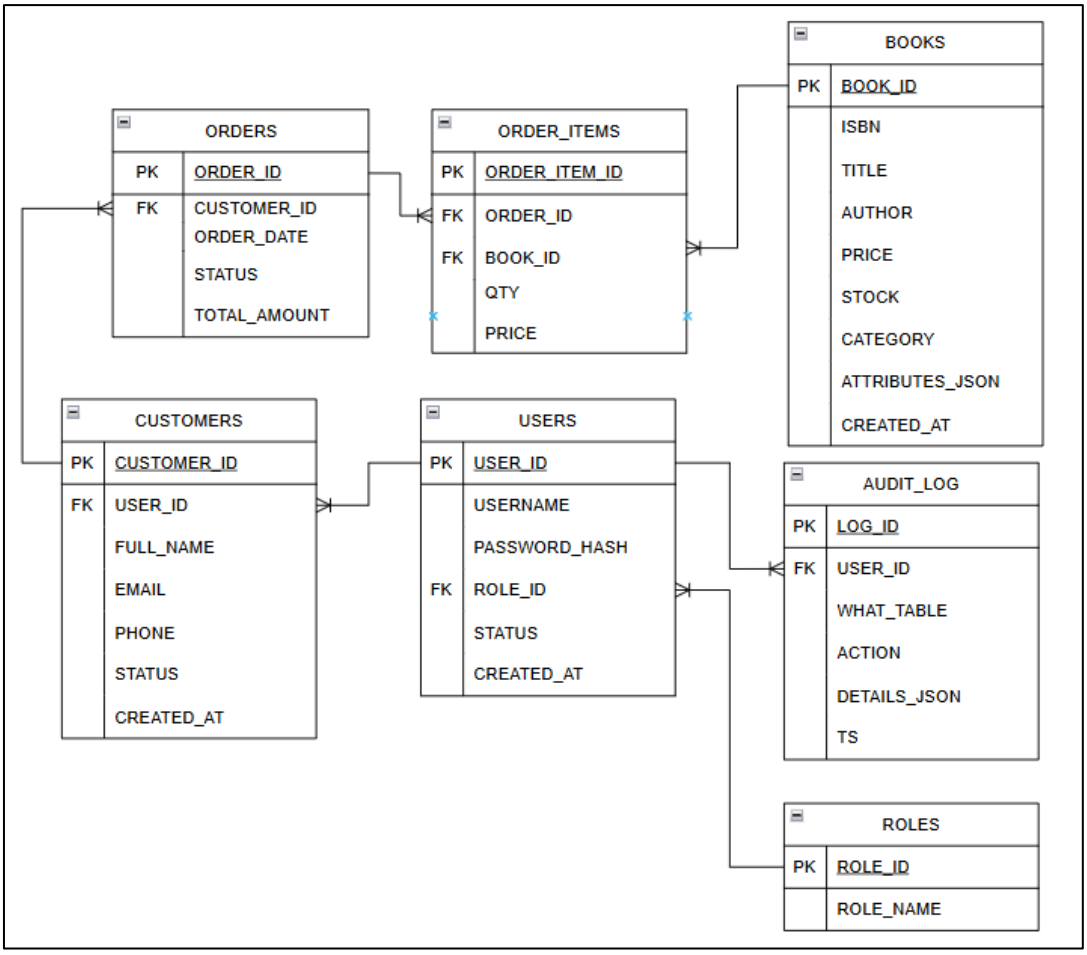


Рисунок 2.2 – Диаграмма базы данных

Таблица «User» содержит информацию о пользователях системы. Поля таблицы включают ключевые сведения, используемые для регистрации, авторизации и определения роли пользователя в системе. В ней хранятся основные учетные данные, необходимые для авторизации и разграничения прав доступа на основе назначенной роли пользователя в рамках информационной системы. Целостность данных обеспечивается первичным ключом USER_ID, а ограничение уникальности на поле USERNAME исключает дублирование логинов. Поле ROLE_ID является внешним ключом, связывающим пользователя с его правами доступа. Описание ее полей приведено в таблице 2.1.

Таблица 2.1 – Структура таблицы «User»

Название	Тип	Описание
User_id	number	Идентификатор пользователя
Username	nvarchar2	Логин пользователя
Password_hash	nvarchar2	Хэш пароля
Role_id	number	Идентификатор роли
Status	nvarchar2	Статус пользователя
Created_at	timestamp	Дата и время создания записи

Таблица «Roles» содержит информацию о ролях пользователей и используется для определения уровня доступа каждого участника системы. Поле ROLE_ID служит первичным ключом для ссылочной целостности, а на поле ROLE_NAME наложено ограничение уникальности, предотвращающее создание дублирующих наименований ролей в системе. Описание полей приведено в таблице 2.2.

Таблица 2.2 – Структура таблицы «Roles»

Название	Тип	Описание
Role_id	number	Идентификатор роли
Role_name	nvarchar2	Название роли

Таблица «Customers» содержит информацию о клиентах, связанных с пользователями: полное имя, email, телефон, статус и дату регистрации. Идентификация записей выполняется по первичному ключу CUSTOMER_ID. Связь с учетной записью реализована через внешний ключ USER_ID, а ограничение UNIQUE для поля EMAIL исключает дублирование клиентов. Описание ее полей приведено в таблице 2.3.

Таблица 2.3 – Структура таблицы «Customers»

Название	Тип	Описание
Customer_id	number	Идентификатор клиента
User_id	number	Ссылка на таблицу Users
Full_name	nvarchar2	Полное имя клиента
Email	nvarchar2	Электронная почта
Phone	nvarchar2	Телефон
Status	nvarchar2	Статус клиента
Created_at	timestamp	Дата и время создания записи

Таблица «Books» содержит информацию о книгах. Она используется для формирования актуального каталога доступных изданий и служит информационным фундаментом для операций оформления заказов, мониторинга складских запасов и финансового учета товаров. Идентификация изданий осуществляется по первичному ключу Book_id, который гарантирует уникальность каждой записи в каталоге. Описание ее полей приведено в таблице 2.4.

Таблица 2.4 – Структура таблицы «Books»

Название	Тип	Описание
Book_id	number	Идентификатор книги
Isbn	nvarchar2	Международный стандартный номер книги
Title	nvarchar2	Название книги
Author	nvarchar2	Автор книги
Price	number	Цена книги
Stock	number	Количество на складе
Category	nvarchar2	Категория книги
Attributes_json	clob	Дополнительные характеристики книги (JSON)
Created_at	timestamp	Название учёной степени

Таблица «Orders» содержит информацию о заказах, включая данные о покупателях, дате оформления, статусе и итоговой сумме. Она служит основной связующей таблицей для формирования отчетов о продажах и анализе активности пользователей. Уникальность заказов обеспечивается первичным ключом Order_id, а связь с покупателем реализована через внешний ключ Customer_id, поддерживающий ссылочную целостность данных. Описание ее полей приведено в таблице 2.5.

Таблица 2.5 – Структура таблицы «Orders»

Название	Тип	Описание
Order_id	number	Идентификатор заказа
Customer_id	date	Ссылка на таблицу Customers
Order_date	timestamp	Дата и время заказа
Status	nvarchar2	Статус заказа
Total_amount	number	Общая сумма заказа

Таблица «Order_Items» содержит информацию о позициях в заказах: какая книга была заказана, в каком количестве и по какой цене. Она обеспечивает связь между заказом и конкретными товарами, позволяя формировать структуру каждого заказа. Уникальность записей гарантируется первичным ключом ORDER_ITEM_ID, а ссылочная целостность обеспечивается внешними ключами ORDER_ID и BOOK_ID, которые жестко привязывают позицию к соответствующему заказу и изданию из каталога.

Описание ее полей приведено в таблице 2.6.

Таблица 2.6 – Структура таблицы «Order_Items»

Название	Тип	Описание
1	2	3
Order_item_id	number	Идентификатор позиции заказа
Order_id	number	Ссылка на таблицу Orders

Продолжение таблицы 2.6

1	2	3
Book_id	number	Ссылка на таблицу Books
Qty	number	Количество книг
Price	number	Цена за единицу

Таблица «Audit_Log» содержит информацию о действиях в системе: кто, с какой таблицей, какое действие, детали в формате JSON и дата-время действия. Идентификация событий осуществляется по первичному ключу LOG_ID. Для хранения подробностей операций используется поле типа CLOB, поддерживающее структуру JSON, что позволяет гибко фиксировать изменения данных без жесткой привязки к схеме. Описание ее полей приведено в таблице 2.7.

Таблица 2.7 – Структура таблицы «Audit_Log»

Название	Тип	Описание
Log_id	number	Идентификатор записи аудита
User_id	nvarchar2	Кто совершил действие
What_table	nvarchar2	Название таблицы, к которой относится
Action	nvarchar2	Тип действие (INSERT, UPDATE, DELETE)
Details_json	clob	Детали действия (JSON)
Ts	timestamp	Дата и время записи аудита

Представленные выше таблицы и их ключевые атрибуты полностью соответствуют логической структуре базы данных, отраженной на диаграмме UML, и гарантируют целостность данных через строгие ограничения.

2.3 Выводы по главе 2

Проектирование базы данных является важным этапом разработки приложения. Оно определяет, какие данные будут храниться и как они будут использоваться. Правильная структура таблиц и связи между ними обеспечивают целостность и оптимальную производительность. Такой подход повышает надёжность, безопасность и эффективность работы базы данных.

В ходе работы была построена полнофункциональная нормализованная физическая модель (третья нормальная форма), охватывающая управление пользователями, товарами и заказами. Спроектированная архитектура, гарантирующая ссылочную целостность ключами и гибкость форматом JSON, полностью учитывает требования будущей настройки гетерогенной репликации и готова к этапу программной реализации.

Выбранные архитектурные решения позволяют не только автоматизировать текущие бизнес-процессы интернет-магазина, но и создают надежный фундамент для дальнейшего масштабирования системы.

Глава 3 Разработка объектов базы данных

3.1 Разработка таблиц базы данных

При разработке приложения для курсового проекта была использована база данных Oracle с инструментами PL/SQL и Oracle SQL Developer.

Таблица – это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Она состоит из столбцов и строк [1].

Каждый столбец таблицы, или иначе говоря поле, характеризуется своим именем (названием соответствующего свойства) и типом данных, отражающих значения данного свойства. Каждое поле обладает определенным набором свойств (размер, формат и др.).

Поле базы данных – это столбец таблицы, включающий в себя значения определенного свойства.

В каждой таблице должно быть, по крайней мере, одно ключевое поле, содержимое которого уникально для любой записи в этой таблице.

Значения ключевого поля однозначно определяют каждую запись в таблице.

Для хранения информации об интернет-магазине книг были разработаны следующие таблицы: Users, Roles, Customers, Books, Orders, Order_Items, Audit_Log. Каждая таблица создается в отдельном SQL-скрипте bookstore_user.sql, что позволяет быстро развернуть базу данных на новом сервере или в другой контейнерной базе данных (например, при репликации).

Создание таблиц выполняется пользователем bookstore_user, который обладает правами на создание объектов схемы.

Скрипты создания таблиц и наложение ограничений целостности на них представлены в приложении А к данной записке.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных.

Для более удобной работы с базой данных можно создать несколько функций и процедур.

3.2 Разработка индексов базы данных

Индекс – это объект базы данных, содержащий упорядоченные значения выбранных столбцов и указатели на физическое расположение соответствующих записей. Он ускоряет операции поиска и сортировки, позволяя обращаться к данным без полного сканирования таблицы. Наиболее распространённой структурой индекса является B-дерево – сбалансированное дерево поиска, обеспечивающее быстрый доступ к нужным элементам.

Индексы работают подобно предметным указателям в книге: они содержат только ключевые столбцы и ссылки на строки данных, что облегчает выборку нужных записей.

В качестве примера можно привести индекс idx_users_role_id в таблице users, созданный на столбце role_id. Он ускоряет фильтрацию пользователей по роли и повышает эффективность соединений с таблицей roles. Создание части индексов представлено в листинге 3.1.

```
CREATE INDEX idx_users_role_id ON users(role_id);
CREATE INDEX idx_customers_user_id ON customers(user_id);
CREATE INDEX idx_orders_customer_id ON orders(customer_id);
CREATE INDEX idx_order_items_order_id ON order_items(order_id);
CREATE INDEX idx_order_items_book_id ON order_items(book_id);
CREATE INDEX idx_books_author ON books(author);
CREATE INDEX idx_books_title ON books(title);
CREATE INDEX idx_orders_order_date ON orders(order_date);
CREATE INDEX idx_users_status ON users(status);
CREATE INDEX idx_orders_status ON orders(status)
```

Листинг 3.1 – Создание индексов

Далее будут рассмотрены индексы каждой таблицы с пояснением их назначения и влияния на производительность.

idx_users_role_id – индекс для столбца role_id таблицы users.

idx_customers_user_id – индекс для столбца user_id таблицы customers.

idx_orders_customer_id – индекс для столбца customer_id таблицы orders.

idx_order_items_order_id – индекс для столбца order_id таблицы order_items.

idx_order_items_book_id – индекс для столбца book_id таблицы order_items.

idx_books_author – индекс для столбца author таблицы books.

idx_books_title – индекс для столбца title таблицы books.

idx_orders_order_date – индекс для столбца order_date таблицы orders.

idx_users_status – индекс для столбца status таблицы users.

idx_orders_status – индекс для столбца status таблицы orders.

Каждый индекс представляет собой таблицу (пусть и особый тип таблицы, но все же это таблица). Следовательно, каждый раз, когда строка добавляется в таблицу или удаляется из неё, должны быть изменены все индексы в этой таблице. При обновлении строки любые индексы для столбца (или столбцов), которые были затронуты, также должны быть изменены. Следовательно, чем больше у вас индексов, тем больше должен работать СУБД, чтобы поддерживать все объекты схемы в актуальном состоянии – что приводит к замедлению работы.

Более того, индексы занимают дополнительное место на диске и требуют внимательного управления со стороны администраторов баз данных. Поэтому оптимальным решением является создание индексов только тогда, когда это действительно необходимо. Если индекс нужен временно, например, для выполнения месячного отчёта, его можно добавить перед началом процедуры и удалить после её завершения. Важно учитывать, что избыточные индексы замедляют операции добавления и изменения данных, так как системе приходится обновлять их структуру. Поэтому необходимо соблюдать строгий баланс между скоростью чтения и производительностью записи.

В итоге, идеальный подход заключается в нахождении баланса: необходимо иметь достаточно индексов для эффективной работы, но не столько, чтобы это сказывалось на производительности. Следует учитывать, что каждый дополнительный индекс увеличивает накладные расходы при выполнении операций вставки и обновления данных, так как СУБД вынуждена перестраивать структуру индекса при каждом изменении таблицы.

3.3 Разработка процедур базы данных

Хранимая процедура – это набор операторов Transact-SQL (T-SQL), который хранится в базе данных и вызывается по имени. Чаще всего используются пользовательские процедуры, создаваемые в любой базе, кроме Resource. Также существуют временные процедуры в «tempdb» (локальные или глобальные) и системные процедуры с префиксом «sys.sp», встроенные в SQL Server [2].

Процедуры уменьшают сетевой трафик, так как вместо передачи объемного кода серверу отправляется только имя и параметры. Они также усиливают безопасность и стабильность: один и тот же блок кода всегда выполняется одинаково, что исключает ошибки при повторном написании запросов.

В качестве примера рассмотрим процедуры MANAGE_USER_ADD и MANAGE_CUSTOMER_ADD для добавления данных в таблицы users и customers. Параметры процедур позволяют передать информацию о логине, пароле, роли и статусе пользователя, а также полные данные покупателя (имя, почта, телефон) для создания связанных записей. Процедура добавления клиента представлена в листинге 3.2.

```
CREATE OR REPLACE PROCEDURE MANAGE_CUSTOMER_ADD(
    p_user_id IN NUMBER,
    p_full_name IN VARCHAR2,
    p_email IN VARCHAR2,
    p_phone IN VARCHAR2,
    p_status IN VARCHAR2 DEFAULT 'ACTIVE'
) IS
BEGIN
    INSERT INTO customers(user_id, full_name, email, phone, status)
    VALUES (p_user_id, p_full_name, p_email, p_phone, p_status);
END;
/
```

Листинг 3.2 – Процедура добавления клиента

Таким образом, в результате выполнения комплекса разработанных хранимых процедур мы получаем возможность эффективно создавать, обновлять и удалять записи в рамках строго соблюдения бизнес-логики, а также вести аудит действий в базе данных интернет-магазина книг.

MANAGE_PRODUCT_ADD – процедура создания книги.
MANAGE_USER_ADD – процедура создания пользователя.
MANAGE_CUSTOMER_ADD – процедура создания покупателя.
MANAGE_ORDER_CREATE – процедура создания заказа.
MANAGE_ORDER_ADD_ITEM – процедура добавления позиции в заказ.
ADMIN_BLOCK_USER – процедура изменения статуса пользователя.
MANAGE_ORDER_STATUS – процедура изменения статуса заказа.
MANAGE_CUSTOMER_DELETE – процедура удаления покупателя.
MANAGE_DELETE – процедура удаления заказа и связанных с ним позиций.
LOG_ACTION – процедура записи действий пользователей в журнал аудита.
Код всех разработанных процедур представлен в приложении Б.

Разработанные хранимые процедуры надежно инкапсулируют критическую бизнес-логику на сервере, что существенно повышает информационную безопасность системы за счет строгого ограничения прямого доступа к исходным таблицам. Централизация операций гарантирует целостность транзакций при многоэтапных сложных изменениях данных и обеспечивает автоматическую фиксацию ключевых действий в системном журнале аудита.

3.4 Разработка пользователей базы данных

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

При проектировании структурированной базы данных были созданы три основных типа пользователей: администратор, продавец и покупатель. Каждый пользователь обладает определёнными привилегиями в соответствии с его ролью. Администратор управляет всей информацией и данными базы данных, имея доступ ко всем процедурам и функциям. Продавец может работать с заказами и товарами, а покупатель – просматривать каталог и оформлять заказы. Доступ к системе осуществляется через приложение: ввод уникального логина и зашифрованного пароля позволяет авторизоваться под соответствующей ролью.

Дополнительно для каждого пользователя были настроены индивидуальные ограничения безопасности, включая контроль паролей и ограничение попыток входа. Для предотвращения несанкционированного доступа были реализованы политики минимизации прав: каждый пользователь получает только те разрешения, которые необходимы для выполнения его функций. Также реализован аудит действий, позволяющий отслеживать изменения данных, выполненные каждым типом пользователей. Это обеспечивает более высокий уровень надёжности и безопасности работы с системой.

Создание пользователей представлено в листинге 3.3.

```
-- ===== Роль и пользователь администратора =====
CREATE ROLE RLAdmin;
GRANT EXECUTE ON AdminPackage TO RLAdmin;
CREATE USER AdminUser IDENTIFIED BY "Qwerty12345";
GRANT RLAdmin TO AdminUser;
-- ===== Роль и пользователь продавца =====
CREATE ROLE RLSeller;
GRANT EXECUTE ON SellerPackage TO RLSeller;
CREATE USER SellerUser IDENTIFIED BY "Qwerty12345";
GRANT RLSeller TO SellerUser;
-- ===== Роль и пользователь покупателя =====
CREATE ROLE RLClient;
GRANT EXECUTE ON ClientPackage TO RLClient;
CREATE USER ClientUser IDENTIFIED BY "Qwerty12345";
GRANT RLClient TO ClientUser;
```

Листинг 3.3 – Создание пользователей и выдача привилегий

Полный код создания пользователей и выдача привилегий будет представлен в приложении Б.

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

Это позволило строго и четко разграничить функциональность зоны ответственности между администраторами, сотрудниками и клиентами. Тем самым обеспечивается комплексная защита критически важной корпоративной случайного искажения, информации от несанкционированного изменения или удаления.

3.5 Разработка представлений базы данных

В SQL представления (views) содержат строки и столбцы, аналогичные таблицам, однако без фактических данных. Представление можно рассматривать как виртуальную таблицу, созданную из одной или нескольких таблиц, чтобы упростить работу с данными [3].

Представления создаются в случаях, когда необходимо дать пользователю возможность просматривать не всю таблицу, а только ее часть, а также для объединения двух или более таблиц.

В листинге 3.4 представлены некоторые разработанные представления.

```
-- 1. Информация о заказах клиентов
CREATE OR REPLACE VIEW CustomerOrdersInfo AS
SELECT c.customer_id, c.full_name, c.email, c.phone, o.order_id,
o.order_date, o.status, o.total_amount
FROM customers c JOIN orders o ON c.customer_id = o.customer_id;

-- 2. Полная информация о покупателях (с ролями)
CREATE OR REPLACE VIEW CustomersInfo AS
SELECT c.customer_id, c.full_name, c.email, c.phone, c.status, c.cre-
ated_at, u.username, r.role_name
FROM customers c
LEFT JOIN users u ON c.user_id = u.user_id
LEFT JOIN roles r ON u.role_id = r.role_id;

-- 3. Краткая сводка по заказам
CREATE OR REPLACE VIEW OrdersInfo AS
SELECT o.order_id, o.order_date, o.status, o.total_amount, c.cus-
tomer_id, c.full_name, c.email
FROM orders o JOIN customers c ON o.customer_id = c.customer_id;

-- 4. Детализация позиций заказа с расчетом стоимости
CREATE OR REPLACE VIEW OrderItemsInfo AS
SELECT oi.order_item_id, oi.order_id, o.order_date, o.status,
b.book_id, b.title, b.author,
       oi.qty, oi.price, (oi.qty * oi.price) AS total_item_price
FROM order_items oi
JOIN orders o ON oi.order_id = o.order_id
JOIN books b ON oi.book_id = b.book_id;
```

Листинг 3.4 – Создание представлений

Представление CustomerOrdersInfo предназначено для отображения информации о покупателях и связанных с ними заказах. Оно объединяет таблицы customers и orders по идентификатору покупателя и содержит такие данные, как ФИО покупателя, контактные сведения, идентификатор заказа, дата его формирования, статус и итоговая сумма. Использование представления позволяет получать полную информацию о заказах клиентов в одном запросе, упрощая анализ активности пользователей. Кроме того, это избавляет разработчиков приложения от необходимости многократно прописывать сложные условия соединения таблиц в программном коде.

Представление CustomersInfo предоставляет обобщённые сведения о покупателях, включая данные профиля пользователя, роль, контактную информацию и статус. Оно объединяет таблицы customers, users *и* roles, что позволяет связать покупателя с аккаунтом в системе и его ролью. Представление используется для быстрого получения полной информации о зарегистрированном клиенте без необходимости выполнения нескольких соединений в каждом запросе.

Представление OrdersInfo отображает подробные данные о заказах, включая дату оформления, статус обработки, итоговую стоимость и информацию о покупателе, которому принадлежит заказ. Оно объединяет таблицы orders и customers и позволяет быстро получить весь необходимый набор данных для контроля заказов, их состояния и анализа поведения клиентов.

Представление OrderItemsInfo используется для получения детализированной информации о составе заказов. Оно объединяет данные из таблиц order_items, orders и books, предоставляя сведения о товарах в заказе, их количестве, цене, итоговой стоимости позиции, а также информации о книге. Такое представление облегчает работу с детализацией заказов, формированием чеков, отчётности и анализом продаж по товарам.

Скрипты всех представлений продемонстрированы в приложении В.

3.6 Разработка триггеров базы данных

Триггер базы данных – это часть процедурного кода, как и хранимая процедура, но который выполняется только при наступлении определенного события. Имеются различные типы событий, которые могут вызвать срабатывание триггера. Например, вставка строки в таблицу, изменение структуры таблицы или авторизация пользователя в экземпляре SQL Server. В листинге 3.5 представлен один из триггеров [4].

```
CREATE OR REPLACE TRIGGER trg_prevent_admin_user
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    IF :NEW.username = 'admin' THEN
        RAISE_APPLICATION_ERROR(-20001,      'Невозможно      создать
пользователя с username = admin');
    END IF;
END;
```

Листинг 3.4 – Создание триггера trg_prevent_admin_user

Выше приведен специальный защитный триггер, который категорически запрещает создание пользователя с уязвимым стандартным логином admin в таблице users. Если происходит любая попытка вставки таких недопустимых данных, триггер автоматически откатывает транзакцию и выводит сообщение об ошибке. Ниже представлен перечень триггеров. Скрипты представлены в приложении Г.

trg_prevent_admin_user – запрещает создание пользователя с логином admin в таблице users.

trg_check_customer_fields – проверяет обязательные поля full_name, email и phone в таблице customers.

trg_unique_customer_email – запрещает дублирование email в таблице customers.

trg_check_book_exists – проверяет наличие книги перед добавлением в заказ (order_items).

trg_check_book_stock – не позволяет заказать больше книг, чем есть на складе (books.stock).

trg_check_order_status – проверяет, что статус заказа в таблице orders соответствует допустимым значениям (Новый, В обработке, Оплачен, Отменён). trg_unique_book_isbn – запрещает вставку книги с уже существующим ISBN в таблице books.

trg_log_action – логирует изменения в таблице orders в журнале действий audit_log.

Внедрение данных триггеров позволяет реализовать автоматический контроль целостности информации непосредственно на уровне сервера базы данных. Это гарантирует, что даже в случае ошибок в клиентском приложении или при прямом доступе к таблицам, в систему не попадут некорректные данные (например, дублирующиеся email или заказы, превышающие складской остаток). Кроме того, использование триггера аудита обеспечивает прозрачность всех операций, создавая надежную историю изменений для анализа безопасности.

3.7 Выводы по главе 3

В рамках проекта разработаны высокотехнологичные и четко структурированные таблицы реляционной базы данных современного книжного магазина с использованием специальных индексов, упрощающих пользовательскую выборку данных. Дополнительные представления создают понятные виртуальные витрины, а разнообразные функции и некоторые хранимые процедуры централизуют внутреннюю бизнес-логику и оптимизируют сетевое взаимодействие. Интеллектуальные триггеры гарантируют автоматический контроль целостности и ведение детального аудита.

Четкое разделение ролей ограничивает доступ: главный администратор имеет полные права, а исполнительный продавец и клиент – лишь необходимые привилегии. Это обеспечивает многоуровневую безопасность и абсолютную сохранность данных. Данные механизмы создают надежную систему, полностью готовую к будущей репликации. Комплексный подход к проектированию позволяет эффективно обрабатывать растущие объемы информации без потери производительности.

Глава 4 Описание процедур импорта и экспорта

4.1 Описание процедуры экспорта данных

Импорт и экспорт данных из базы данных выполняется с использованием формата JSON. Данные из таблиц базы данных можно сохранять в JSON-файл, который затем используется для восстановления или переноса информации между системами. Экспорт данных осуществляется с помощью специально разработанной процедуры, которая формирует JSON-объекты на основе данных таблицы BOOKS и записывает их в файл на сервере. Импорт данных производится через процедуру, которая считывает JSON-файл, преобразует его с помощью встроенных функций Oracle и вставляет данные в соответствующую таблицу. Такой подход позволяет сохранить структуру и типы данных, включая вложенные объекты, и обеспечивает корректное восстановление информации при переносе между базами данных.

Скрипт запроса для экспорта данных представлен в листинге 4.1.

```
CREATE OR REPLACE PACKAGE json_export_pkg AS
  PROCEDURE export_books_to_json;
END json_export_pkg;
/
CREATE OR REPLACE PACKAGE BODY json_export_pkg AS
  PROCEDURE export_books_to_json IS
    l_json CLOB;
    l_file UTL_FILE.FILE_TYPE;
  BEGIN
-- Формирование JSON из таблицы
    SELECT JSON_ARRAYAGG(JSON_OBJECT(
        'book_id' VALUE book_id, 'isbn' VALUE isbn,
        'title' VALUE title, 'author' VALUE author,
        'price' VALUE price, 'stock' VALUE stock,
        'category' VALUE category,
        'created_at'    VALUE    TO_CHAR(created_at,    'YYYY-MM-DD
HH24:MI:SS'),
        'attributes_json' VALUE attributes_json FORMAT JSON
      )) INTO l_json
    FROM books;
-- Запись в файл
    l_file := UTL_FILE.FOPEN('JSON_DIR', 'books_export.json', 'w',
32767);
    UTL_FILE.PUT(l_file, l_json);
    UTL_FILE.FCLOSE(l_file);
  EXCEPTION WHEN OTHERS THEN
    IF UTL_FILE.IS_OPEN(l_file) THEN UTL_FILE.FCLOSE(l_file); END IF;
    RAISE;
  END export_books_to_json;
END json_export_pkg;
/
```

Листинг 4.1 – Процедура экспорта данных

Таким образом, пользователи базы данных могут легко экспортировать данные в формате JSON что обеспечивает высокую совместимость с внешними приложениями. Полученные файлы можно использовать для интеграции со сторонними веб-сервисами, аналитическими платформами или для передачи отчетности партнерам без необходимости сложной конвертации данных.

Результатом выполнения процедуры экспорта является текстовый файл *books_export.json*, содержащий массив JSON-объектов. Ниже приведен фрагмент содержимого этого файла с данными о нескольких книгах в листинге 4.2.

```
[
  {
    "book_id": 384192,
    "isbn": "978-3-00001",
    "title": "Book Title 1",
    "author": "Author 1",
    "price": 277.51,
    "stock": 41,
    "category": "Category 1",
    "created_at": "2025-12-15 01:25:09",
    "attributes_json": {
      "pages": 104,
      "format": "paperback"
    }
  },
  {
    "book_id": 384193,
    "isbn": "978-3-00002",
    "title": "Book Title 2",
    "author": "Author 2",
    "price": 168.54,
    "stock": 5,
    "category": "Category 2",
    "created_at": "2025-12-15 01:25:09",
    "attributes_json": {
      "pages": 305,
      "format": "paperback"
    }
  },
]
```

Листинг 4.2 – Фрагмент экспортированного файла books_export.json

Как видно из листинга, файл представляет собой массив JSON-объектов. Каждая запись содержит основные поля таблицы BOOKS, а также вложенный объект attributes_json, который хранится и передается с сохранением своей структуры. Такой текстовый формат удобен тем, что его может прочитать и человек, и компьютер. Это позволяет легко проверить выгруженные данные визуально, а также гарантирует, что при переносе на другой сервер (например, с другими настройками даты) информация загрузится без ошибок. В результате сформированный документ является готовым решением для автоматизированного обмена данными между различными компонентами информационной системы.

4.2 Описание процедуры импорта данных

Импорт выполняется процедурой, которая считывает JSON-файл и загружает записи в таблицу с сохранением типов данных. Логика экспорта реализована в PL/SQL-пакете: данные таблицы BOOKS конвертируются в JSON посредством SQL-запроса и сохраняются на диск через пакет UTL_FILE. Это обеспечивает корректную выгрузку структуры для резервного копирования и интеграции.

Скрипт запроса для экспорта данных представлен в листинге 4.2.

```
CREATE OR REPLACE PACKAGE BODY json_import_pkg AS
  PROCEDURE import_books_from_json IS
    l_file CLOB := EMPTY_CLOB(); f UTL_FILE.FILE_TYPE; l_buf VAR-
CHAR2(32767);
  BEGIN
    f := UTL_FILE.FOPEN('JSON_DIR', 'books_export.json', 'r', 32767);
    -- Чтение файла в CLOB
    LOOP BEGIN UTL_FILE.GET_LINE(f, l_buf); l_file := l_file || l_buf;
    EXCEPTION WHEN NO_DATA_FOUND THEN EXIT; END; END LOOP;
    UTL_FILE.FCLOSE(f);
    -- Парсинг и вставка
    INSERT INTO books (book_id, isbn, title, author, price, stock,
category, attributes_json, created_at)
    SELECT book_id, isbn, title, author, price, stock, category, at-
tributes_json, TO_TIMESTAMP(created_at, 'YYYY-MM-DD HH24:MI:SS')
    FROM JSON_TABLE(l_file, '$[*]' COLUMNS(
      book_id NUMBER PATH '$.book_id', isbn VARCHAR2(20) PATH
'$.isbn',
      title VARCHAR2(400) PATH '$.title', author VARCHAR2(200) PATH
'$.author',
      price NUMBER(10,2) PATH '$.price', stock NUMBER PATH '$.stock',
      category VARCHAR2(100) PATH '$.category', attributes_json CLOB
PATH '$.attributes_json',
      created_at VARCHAR2(30) PATH '$.created_at'));
    COMMIT;
  EXCEPTION WHEN OTHERS THEN
    IF UTL_FILE.IS_OPEN(f) THEN UTL_FILE.FCLOSE(f); END IF; RAISE;
  END import_books_from_json;
END json_import_pkg;/
```

Листинг 4.3 – Процедура импорта данных

Таким образом, авторизованные пользователи базы данных могут легко и оперативно импортироватьбольшие массивы информации из формата JSON, что делает управление базой данных существенно более удобным и эффективным.

Для проверки работоспособности модуля импорта была выполнена очистка тестовых данных и запуск процедуры загрузки в рамках транзакционного блока. После выполнения пакета данные из файла *books_export.json* были успешно восстановлены в таблице BOOKS, включая корректное преобразование типов данных и проверку на соответствие установленным ограничениям первичных и уникальных ключей. Успешное завершение теста подтвердило высокую пропускную способность разработанного модуля и его готовность к логичсеким системам.

```
DELETE FROM books;
COMMIT;
SELECT COUNT(*) FROM books;
BEGIN
  json_import_pkg.import_books_from_json;
END;
/
```

Листинг 4.3 – Скрипт запуска импорта для проверки результатов

Результат выборки данных из таблицы после успешного выполнения процедуры импорта представлен на рисунке 4.4.

	BOOK_ID	ISBN	TITLE	AUTHOR	PRICE	STOCK	CATEGORY	ATTRIBUTES_JSON	CREATED_AT
1	384192	978-3-00001	Book Title 1	Author 1	277,51	41	Category 1	(null)	15.12.25 01:25:09,000000000
2	384193	978-3-00002	Book Title 2	Author 2	168,54	5	Category 2	(null)	15.12.25 01:25:09,000000000
3	384194	978-3-00003	Book Title 3	Author 3	425,53	13	Category 3	(null)	15.12.25 01:25:09,000000000
4	384195	978-3-00004	Book Title 4	Author 4	282,3	42	Category 4	(null)	15.12.25 01:25:09,000000000
5	384196	978-3-00005	Book Title 5	Author 5	205,04	21	Category 5	(null)	15.12.25 01:25:09,000000000
6	384197	978-3-00006	Book Title 6	Author 6	969,13	27	Category 6	(null)	15.12.25 01:25:09,000000000
7	384198	978-3-00007	Book Title 7	Author 7	102,48	11	Category 7	(null)	15.12.25 01:25:09,000000000
8	384199	978-3-00008	Book Title 8	Author 8	174,03	33	Category 8	(null)	15.12.25 01:25:09,000000000
9	384200	978-3-00009	Book Title 9	Author 9	831,76	20	Category 9	(null)	15.12.25 01:25:09,000000000
10	384201	978-3-00010	Book Title 10	Author 10	750,28	8	Category 0	(null)	15.12.25 01:25:09,000000000

Рисунок 4.4 – Результат выборки данных из таблицы

Как видно из результата, процедура успешно считала JSON-файл, распарсила его структуру и вставила записи в реляционную таблицу с соблюдением правил ссылочной целостности. Поле created_at корректно преобразовалось из строки обратно в тип TIMESTAMP, что обеспечивает точность временных меток для последующего хронологического анализа операции.

4.3 Выводы по главе 4

В рамках четвертой главы были успешно реализованы надежные и гибкие программные механизмы двухстороннего обмена структурированными данными с использованием стандартизированного формата JSON. Разработанные специализированные PL/SQL-пакеты позволяют выполнять оперативную выгрузку и загрузку информации непосредственно на сервере, полностью сохраняя сложную иерархическую структуру и типы данных. Использование нативных встроенных функций Oracle и системного пакета UTL_FILE гарантирует стабильно высокую производительность операций. Данное решение обеспечивает масштабируемость универсальность системы, существенно упрощая задачи резервного копирования и интеграции со сторонними веб-приложениями. Такой подход обеспечивает полную автономность модуля, поскольку для обмена данными не требуется установка дополнительного программного обеспечения или драйверов на клиентских машинах , что значительно снижает совокупную стоимость владения системой и требования к каналам связи. Кроме того, выполнение всех операций на стороне сервера позволяет централизованно контролировать права доступа и гарантирует высокий уровень защиты конфиденциальной информации при совершении экспортно-импортных операций.

Глава 5 Тестирование производительности

Тестирование отыгрывает важную роль при разработке любого программного продукта. Чем качественнее тестирование, тем лучше в итоге должен выйти конечный продукт. Выявление проблем с производительностью до того, как они повлияют на конечного пользователя, имеет жизненно важное значение для успеха любого приложения в разработке. Пользователи будут избегать приложений, которые не работают должным образом, и большинство из них предпочитают переходить к программному обеспечению, которое было должным образом протестировано.

Нагрузочное тестирование базы данных – важный шаг, который разработчики должны предпринять, чтобы обеспечить оптимизацию производительности перед развертыванием. Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Разработанная процедура позволяет добавить 100000 строк за одно выполнение. Генерация такого объема данных необходима для создания условий, максимально приближенных к реальной эксплуатации крупного интернет-магазина. Это позволяет объективно оценить время отклика системы и эффективность работы индексов под нагрузкой. Заполнение таблицы продемонстрировано в листинге 5.1.

```
BEGIN
  FOR i IN 1..100000 LOOP
    INSERT INTO books (isbn, title, author, price, stock, category,
attributes_json)
    VALUES (
      '978-3-' || LPAD(i,5,'0'),
      'Book Title ' || i,
      'Author ' || MOD(i,100),
      ROUND(DBMS_RANDOM.VALUE(100,1000),2),
      ROUND(DBMS_RANDOM.VALUE(0,50)),
      'Category ' || MOD(i,10),
      '{"pages": ' || ROUND(DBMS_RANDOM.VALUE(100,1000)) || ', "for-
mat":"paperback"}'
    );
  END LOOP;
  COMMIT;
END;
```

Листинг 5.1 – Заполнение таблицы

Программная алгоритмическая реализация генератора тестовых данных базируется на использовании стандартного цикла FOR, который выполняет итерации вставки записей. Для обеспечения уникальности каждой книги используется конкатенация строкового литерала с текущим индексом цикла i, отформатированным функцией LPAD.

В результате работы данного скрипта формируется экспериментальная база данных, необходимая для объективной оценки скорости выполнения запросов и настройки производительности.Таким образом, сформированная среда служит надежным полигоном для верификации разработанных алгоритмов.

Для получения выборки данных использовался запрос, который представлен на листинге 5.2.

```
SET STATISTICS TIME ON;
SELECT title
FROM books
WHERE category = 'Учебная литература';
```

Листинг 5.2 – Запрос к таблице

Данный запрос позволяет вывести названия всех книг, относящихся к категории «Учебная литература», что необходимо для анализа ассортимента и формирования тематических выборок.

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. В запросе мы включаем вывод статистики времени выполнения, чтобы получить информацию о времени, затраченном на анализ и компиляцию запроса, а также статистика времени выполнения, включая время CPU и общее время выполнения запроса. Включение режима отладки SET TIMING ON и SET AUTOTRACE ON позволяет зафиксировать точные метрики производительности. Это дает возможность сравнить затраты ресурсов запроса до и после оптимизации, опираясь на конкретные цифры, а не на субъективное ощущение скорости работы. Результаты запроса будут представлены на рисунке 5.1.

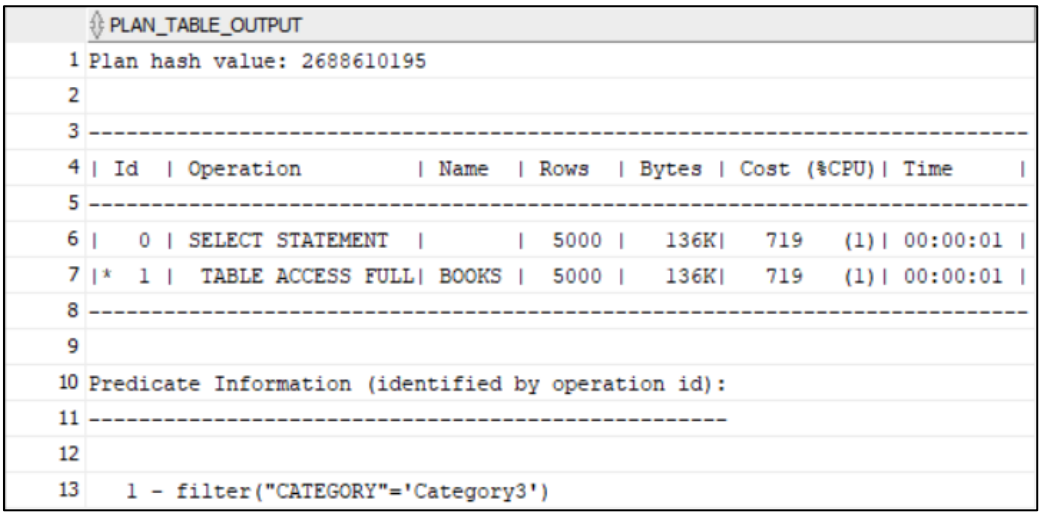
```
Elapsed: 00:00:00.007
>>Query Run In:Query Result 4
Elapsed: 00:00:00.350
```

Рисунок 5.1 – Результат выполнения запроса

Как видно из результатов, время отклика системы на тестовом объеме данных превышает допустимые пороги для интерактивных приложений. Чтобы выявить точную причину задержки и определить узкое место в обработке данных, необходимо проанализировать внутренний алгоритм действий СУБД.

План выполнения – это большое дерево, на котором много объектов. Это дерево – инструкция, что нужно сделать, чтобы получить результат запроса. Операторы в плане вызываются по порядку слева направо (с левого верхнего угла), а данные в плане перемещаются справа налево.

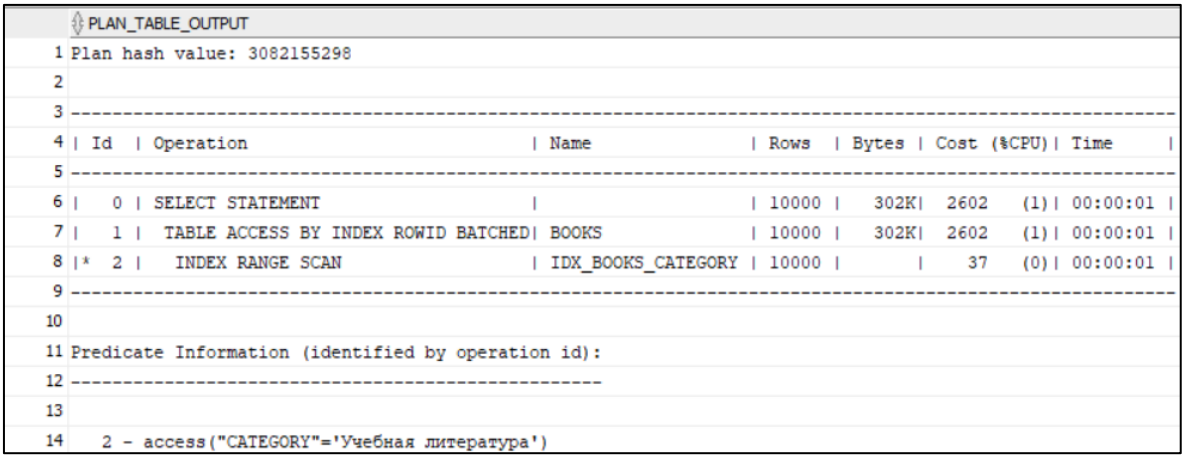
Детальное изучение плана позволяет глубоко понять внутреннюю логику работы системного оптимизатора СУБД, оценить предполагаемую стоимость выполнения операций и точно определить, почему выполнение запроса занимает недопустимо много времени. Например, можно наглядно увидеть, где происходит сортировка большого объема данных или применяется неэффективное соединение таблиц. Это дает возможность принять обоснованное решение о необходимости создания новых индексов, сбора актуальной статистики по объектам или переписывании SQL-кода с использованием более оптимальных предикатов фильтрации. Результат будет представлен на рисунке 5.2.



```
PLAN_TABLE_OUTPUT
Plan hash value: 2688610195

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  5000 |   136K|   719   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| BOOKS |  5000 |   136K|   719   (1)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("CATEGORY"='Category3')
```

Рисунок 5.2 – Результат плана выполнения

Для сокращения времени выполнения запроса на выборку данных был добавлен индекс idx_books_category для таблицы books, позволяющий заменить операцию полного сканирования таблицы на эффективный поиск по дереву, это показано на рисунке 5.3.



```
PLAN_TABLE_OUTPUT
Plan hash value: 3082155298

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                    | 10000 |   302K|  2602   (1)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOOKS              | 10000 |   302K|  2602   (1)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_BOOKS_CATEGORY | 10000 |       |    37   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("CATEGORY"='Учебная литература')
```

Рисунок 5.3 – План выполнения с индексом

Рисунок показывает, что индекс подключился и работает. Мы видим операцию INDEX RANGE SCAN по индексу IDX_BOOKS_CATEGORY. База данных не тратит время на просмотр лишних строк, а сразу берет только те, которые нужны по условию запроса, что значительно экономит ресурсы.

5.1 Выводы по главе 5

Нагрузочное тестирование на массиве из 100 000 записей выявило критическую ресурсоемкость запросов без оптимизации, выраженную в экспоненциальном росте времени ожидания и избыточном количестве операции дискового чтения. Анализ статистики и планов выполнения подтвердил необходимость внедрения индексов для обеспечения быстродействия базы данных. В частности, это позволило заменить ресурсоемкое полное сканирование таблицы на эффективный поиск по ключам, многократно сократив нагрузку на процессор.

Глава 6 Описание технологии и ее применения в базе данных

В процессе разработки проекта активно применялась технология репликации данных между серверами системы управления базами данных (СУБД). Концепция репликации в данном контексте заключается в том, чтобы поддерживать несколько наборов данных в состоянии взаимной согласованности. Основная цель — обеспечить сохранность информации и распределение нагрузки. В рамках курсового проекта реализована схема односторонней гетерогенной репликации, при которой изменения, внесенные в основной источник данных (Oracle Database), автоматически переносятся в базу данных-приемник (SQL Server) [5].

Поскольку система объединяет СУБД от разных производителей (Oracle и Microsoft), стандартные механизмы репликации, присущие одной платформе, неприменимы. Для решения этой задачи была выбрана технология Связанных серверов (Linked Server) [6].

Такой метод репликации обеспечивает важные преимущества, включая независимость от версии основной базы данных и удалённой реплики, а также возможность контролировать процесс передачи данных и выполнять необходимые преобразования в соответствии с требованиями проекта.

Среди функциональных задач, которые успешно решаются через репликацию, можно выделить поддержку резервной базы данных для обеспечения надёжной защиты от потери основного набора данных [7].

Для организации прямого взаимодействия между СУБД был настроен системный объект Linked Server («Связанный сервер») на стороне SQL Server. Данная конфигурация выступает в роли шлюза, обеспечивая прозрачный доступ к объектам удаленной базы данных Oracle через OLE DB провайдер. Скрипт создания и настройки подключения представлен в листинге 6.1 [8].

```
-- 1. Создание подключения к Oracle (Linked Server)
EXEC sp_addlinkedserver
   @server = 'ORACLE_LINK',            -- Имя сервера в системе
   @srvproduct = 'Oracle',             -- Тип продукта
   @provider = 'OraOLEDB.Oracle',      -- Используемый драйвер
   @datasrc = '//localhost:1521/PDB1'; -- Адрес базы данных Oracle

-- 2. Настройка безопасности (Маппинг логина)
EXEC sp_addlinkedsrvlogin
   @rmtsrvname = 'ORACLE_LINK',
   @useself = 'FALSE',
   @locallogin = NULL,
   @rmtuser = 'bookstore_user',        -- Логин пользователя Oracle
   @rmtpassword = '********';        -- Пароль (скрыт для безопасности)

-- 3. Включение опций RPC (для выполнения сложных запросов)
EXEC sp_serveroption 'ORACLE_LINK', 'rpc', 'true';
EXEC sp_serveroption 'ORACLE_LINK', 'rpc out', 'true';
```

Листинг 6.1 – Скрипт создания связанного сервера

Далее в проекте была реализована логика репликации для ключевых таблиц базы данных. Синхронизация выполняется посредством сценариев на языке T-SQL, которые автоматически переносят изменения, произошедшие в источнике (Oracle), в локальную базу данных. В отличие от триггерного подхода, нагружающего транзакции, обмен данными реализован через периодические пакетные запросы с использованием шлюза Linked Server. Пример реализации скрипта синхронизации для таблицы BOOKS представлен в листинге 6.2.

Пример синхронизации таблицы Book представлен в листинге 6.2.

```
-- Вставка новых записей из Oracle в SQL Server
INSERT INTO books (book_id, isbn, title, author, price, stock, cate-
gory, attributes_json, created_at)
SELECT book_id, isbn, title, author, price, stock, category, attrib-
utes_json, created_at
FROM OPENQUERY(OracleServer,
    'SELECT book_id, isbn, title, author, price, stock, category, at-
tributes_json, created_at FROM books');

-- Обновление существующих записей
UPDATE B
SET B.isbn = O.isbn,
    B.title = O.title,
    B.author = O.author,
    B.price = O.price,
    B.stock = O.stock,
    B.category = O.category,
    B.attributes_json = O.attributes_json,
    B.created_at = O.created_at
FROM books AS B
INNER JOIN OPENQUERY(OracleServer,
    'SELECT book_id, isbn, title, author, price, stock, category, at-
tributes_json, created_at FROM books') AS O
ON B.book_id = O.book_id;

-- Удаление записей, которых больше нет на удалённом сервере
DELETE FROM books
WHERE book_id NOT IN (
    SELECT book_id
    FROM OPENQUERY(OracleServer, 'SELECT book_id FROM books')
);
```

Листинг 6.2 – Скрипт синхронизации данных таблицы Books

Для обеспечения непрерывной и актуальной репликации данных был разработан механизм автоматического планирования. В связи с ограничениями используемой редакции СУБД (SQL Server Express), которая не включает службу SQL Server Agent, была реализована имитация работы планировщика на уровне исполняемого сценария T-SQL.

Сценарий запускает хранимую процедуру dbo.Sync_Books (а также процедуры для других таблиц) в бесконечном цикле. Цикл содержит обязательный тайм-

аут (задержку) в 5 секунд, что предотвращает избыточную нагрузку на сервер и сеть при непрерывном опросе источника, что представлено в листинге 6.3.

```
/* Фрагмент скрипта, имитирующий работу планировщика */
WHILE (1 = 1)
BEGIN
    -- 1. Запуск синхронизации для всех объектов
    EXEC dbo.Sync_Books;
    EXEC dbo.Sync_Orders
    -- 2. Обязательный таймаут перед следующим циклом (5 секунд)
    WAITFOR DELAY '00:00:05';
END;
```

Листинг 6.3 – Сценарий автоматического планирования

Такой подход гарантирует, что данные в базе-приемнике обновляются с задержкой не более 5 секунд, полностью решая задачу поддержки резервной копии в актуальном состоянии.

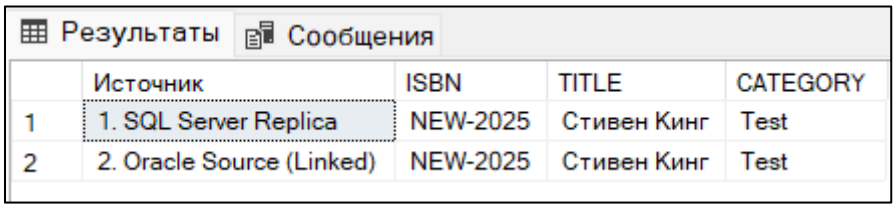
Для наглядной демонстрации функционирования механизма репликации был проведен контрольный эксперимент. В базу данных-источник Oracle была добавлена тестовая запись (ISBN: NEW-2025, Название: «Стивен Кинг»), после чего была запущена процедура автоматической синхронизации. Контрольный запрос согласованности данных представлен в листинге 6.4.

```
/*-- Запрос, сравнивающий одну и ту же тестовую запись в двух базах:
SELECT
    '1. SQL Server Replica' AS Источник, ISBN, TITLE, CATEGORY
FROM
    dbo.BOOKS
WHERE ISBN = 'NEW-2025'

UNION ALL

SELECT
    '2. Oracle Source (Linked)' AS Источник, T.ISBN, T.TITLE, T.CATE-
GORY
FROM
    OPENQUERY(ORACLE_LINK,
    'SELECT ISBN, TITLE, CATEGORY FROM BOOKS WHERE ISBN = ''NEW-
2025''') AS T;
```

Листинг 6.4 – Контрольный запрос согласованности данных

Результат выполнения контрольного запроса Листинг 6.3 подтверждает, что запись с ISBN NEW-2025 присутствует в обеих системах: в источнике (2. Oracle Source Linked) и в реплике (1. SQL Server Replica) с полной идентичностью всех данных. Факт успешного переноса тестовой записи, совпадающей по всем ключевым полям, валидирует работу всего механизма синхронизации, включая функцию OPENQUERY и логику предотвращения дубликатов (NOT EXISTS). Таким образом, данный эксперимент полностью доказывает, что гетерогенная репликация

настроена корректно, а данные между СУБД Oracle и SQL Server поддерживаются в согласованном и актуальном состоянии, что наглядно представлено на рисунке 6.5.


Результаты Сообщения

	Источник	ISBN	TITLE	CATEGORY
1	1. SQL Server Replica	NEW-2025	Стивен Кинг	Test
2	2. Oracle Source (Linked)	NEW-2025	Стивен Кинг	Test

Рисунок 6.5 – Контрольный запрос согласованности данных

Таким образом, проведенный контрольный эксперимент наглядно подтвердил работоспособность настроенного механизма гетерогенной репликации: данные успешно переносятся из Oracle в SQL Server и поддерживаются в согласованном и актуальном состоянии с минимальной задержкой. Все поставленные задачи по интеграции и автоматизации были успешно решены.

6.1 Выводы по главе 6

Глава 6 подтвердила полную работоспособность системы гетерогенной репликации.

Была успешно реализована логика синхронизации на основе хранимых процедур, использующих OPENQUERY для эффективного доступа к удаленной базе данных Oracle. Применение конструкции WHERE NOT EXISTS гарантировало целостность данных, предотвращая дублирование при переносе.

Для обеспечения непрерывности процесса, несмотря на ограничения SQL Server Express, была создана автоматизированная система на базе цикла WHILE (1=1), поддерживающая синхронизацию с задержкой не более 5 секунд. Финальный контрольный эксперимент с запросом UNION ALL убедительно доказал, что тестовая запись полностью совпадает в обеих СУБД.

Таким образом, была создана полностью функционирующая и автоматизированная система односторонней гетерогенной репликации, выполнившая все требования проекта по отказоустойчивости и актуальности данных.

Заключение

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

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

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

Благодаря внедрению индексов, процедур, и оптимизированным структурам таблиц, система способна эффективно обрабатывать запросы, в том числе на больших объемах данных, что существенно повышает производительность системы. В целом, разработанная база данных представляет собой мощный инструмент, сочетающий в себе гибкость, производительность и безопасность. Технология репликации через Linked Server обеспечивает автоматическую синхронизацию данных между серверами, поддерживая их согласованность и целостность.

Список используемых источников

1 Microsoft Learn. Таблицы (SQL Server) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/tables/tables. – Дата доступа: 01.10.2025.

2 Microsoft Learn. Хранимые процедуры (ядро СУБД) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/stored-procedures/stored-procedures-database-engine.– Дата доступа: 26.10.2025.

3 Microsoft Learn. Представления (Views) в SQL Server [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/views/views. – Дата доступа: 12.11.2025.

4 Oracle Help Center. PL/SQL Triggers [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/21/lnpls/plsql-triggers.html. – Дата доступа: 12.11.2025.

5 SQL Server Central (или MSDN). Running Scheduled Tasks in SQL Server Express Edition (Запуск запланированных задач в SQL Server Express) [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/primenenie-znaniy-dlya-sinhronizatsii-agentov-v-parallelnom-diskretno-sobytiynom-modelirovanii. – Дата доступа: 09.12.2025.

6 SQL Shack (или аналогичный профильный ресурс). Replication between Oracle and SQL Server using Linked Servers (Репликация между Oracle и SQL Server с использованием связанных серверов) [Электронный ресурс]. – Режим доступа: https://support.dbagenesis.com/goldengate/ms-sql-server-to-oracle-replication-using-golden-gate. – Дата доступа: 01.10.2025.

7 Microsoft Learn. Использование конструкции NOT EXISTS для исключения дубликатов при вставке данных [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/answers/questions/344590/insert-where-not-exists-or-ignore-dup-key. – Дата доступа: 26.10.2025.

8 Microsoft Learn. Устранение неполадок с репликацией Active Directory [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows-server/identity/ad-ds/manage/troubleshoot/troubleshooting-active-directory-replication-problems. – Дата доступа: 12.11.2025.

9 Официальный сайт Oracle. Руководство по настройке сетевого взаимодействия и TNSNAMES.ORA [Электронный ресурс]. – Режим доступа: https://oracle-base.com/articles/misc/oracle-network-configuration. – Дата доступа: 12.11.2025.

ПРИЛОЖЕНИЕ А
Листинг создания таблиц

```
CREATE TABLE roles (
  role_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  role_name VARCHAR2(50) NOT NULL UNIQUE);
CREATE TABLE users (
  user_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  username VARCHAR2(100) NOT NULL UNIQUE,
  password_hash VARCHAR2(200) NOT NULL,
  role_id NUMBER NOT NULL REFERENCES roles(role_id));
CREATE TABLE customers (
  customer_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  user_id NUMBER REFERENCES users(user_id),
  full_name VARCHAR2(200),
  email VARCHAR2(200) UNIQUE,
  phone VARCHAR2(50));
CREATE TABLE books (
  book_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  isbn VARCHAR2(20),
  title VARCHAR2(400),
  author VARCHAR2(200),
  price NUMBER(10,2),
  stock NUMBER DEFAULT 0,
  category VARCHAR2(100),
  attributes_json CLOB CHECK (attributes_json IS JSON));
CREATE TABLE orders (
  order_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  customer_id NUMBER REFERENCES customers(customer_id),
  order_date TIMESTAMP DEFAULT SYSTIMESTAMP,
  status VARCHAR2(50) NOT NULL,
  total_amount NUMBER(12,2));
CREATE TABLE order_items (
  order_item_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  order_id NUMBER REFERENCES orders(order_id),
  book_id NUMBER REFERENCES books(book_id),
  qty NUMBER NOT NULL,
  price NUMBER(10,2) NOT NULL);
CREATE TABLE audit_log (
  log_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  who VARCHAR2(200),
  what_table VARCHAR2(200),
  action VARCHAR2(50),
  details_json CLOB CHECK (details_json IS JSON),
  ts TIMESTAMP DEFAULT SYSTIMESTAMP
);
```

ПРИЛОЖЕНИЕ Б
Листинг создания процедур

```
-- Процедура логирования (Автономная: сохранит запись, даже если ос-
новная операция откатится)
CREATE OR REPLACE PROCEDURE LOG_ACTION(p_u VARCHAR2, p_t VARCHAR2,
p_a VARCHAR2, p_d VARCHAR2) IS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO AUDIT_LOG (who, what_table, action, details_json, ts)
    VALUES (p_u, UPPER(p_t), UPPER(p_a), p_d, CURRENT_TIMESTAMP);
    COMMIT;
END;
-- Процедура авторизации с проверкой жизненного цикла аккаунта
CREATE OR REPLACE PROCEDURE LOGIN(p_un VARCHAR2, p_pw VARCHAR2, p_res
OUT VARCHAR2) IS
    v_s VARCHAR2(20); v_id NUMBER;
BEGIN
    SELECT user_id, status INTO v_id, v_s FROM USERS
    WHERE username = p_un AND password_hash = p_pw;
    IF v_s != 'ACTIVE' THEN
        p_res := 'ACCESS_DENIED_STATUS_' || v_s;
        LOG_ACTION(p_un, 'AUTH', 'LOGIN_FAIL', 'Reason: ' || v_s);
    ELSE
        p_res := 'SUCCESS';
        LOG_ACTION(p_un, 'AUTH', 'LOGIN_SUCCESS', 'User_ID: ' ||
v_id);
    END IF;
EXCEPTION WHEN NO_DATA_FOUND THEN p_res := 'INVALID_CREDENTIALS';
END;
-- Регистрация клиента (двухэтапная транзакция)
CREATE OR REPLACE PROCEDURE MANAGE_CUSTOMER_ADD(p_un VARCHAR2, p_pw
VARCHAR2, p_fn VARCHAR2, p_em VARCHAR2, p_ph VARCHAR2) IS
    v_uid NUMBER;
BEGIN
    -- Создаем системную учетную запись
    INSERT INTO users (username, password_hash, role_id, status)
    VALUES (p_un, p_pw, (SELECT role_id FROM roles WHERE role_name =
'Client'), 'ACTIVE')
    RETURNING user_id INTO v_uid;
    -- Привязываем персональные данные
    INSERT INTO customers (user_id, full_name, email, phone)
    VALUES (v_uid, p_fn, p_em, p_ph);

    LOG_ACTION(USER, 'CUSTOMERS', 'REGISTRATION', 'New client: ' ||
p_un);
    COMMIT;
END;
-- Редактирование товара с пересчетом маркетинговых полей
CREATE OR REPLACE PROCEDURE MANAGE_PRODUCT_EDIT(p_id NUMBER, p_pr
NUMBER, p_st NUMBER, p_disc NUMBER DEFAULT 0) IS
BEGIN
```

```
    UPDATE BOOKS SET
        price = NVL(p_pr, price),
        stock = NVL(p_st, stock),
        discount_percent = p_disc,
        price_after_discount = NVL(p_pr, price) * (1 - p_disc/100)
    WHERE book_id = p_id;

    IF SQL%ROWCOUNT > 0 THEN
        LOG_ACTION(USER, 'BOOKS', 'UPDATE', 'ID: ' || p_id || ' Stock:
' || p_st);
    END IF;
    COMMIT;
END;
-- Создание нового заказа (корзины)
CREATE OR REPLACE PROCEDURE MANAGE_ORDER_CREATE(p_cid NUMBER, p_oid
OUT NUMBER) IS
BEGIN
    INSERT INTO ORDERS (customer_id, order_date, total_amount, sta-
tus)
    VALUES (p_cid, SYSDATE, 0, 'Новый') RETURNING order_id INTO p_oid;
    LOG_ACTION(USER, 'ORDERS', 'CREATE_EMPTY', 'Order: ' || p_oid);
END;
-- Добавление товара в заказ с контролем склада
CREATE OR REPLACE PROCEDURE MANAGE_ORDER_ADD_ITEM(p_oid NUMBER, p_bid
NUMBER, p_q NUMBER) IS
    v_cur_pr NUMBER; v_stock NUMBER;
BEGIN
    SELECT price_after_discount, stock INTO v_cur_pr, v_stock FROM
BOOKS WHERE book_id = p_bid;
    IF v_stock >= p_q THEN
        -- Добавляем в позиции заказа
        INSERT INTO ORDER_ITEMS (order_id, book_id, qty, price) VAL-
UES (p_oid, p_bid, p_q, v_cur_pr);
        -- Списываем со склада
        UPDATE BOOKS SET stock = stock - p_q WHERE book_id = p_bid;
        -- Пересчитываем итоговую сумму заказа
        UPDATE ORDERS SET total_amount = (SELECT SUM(qty * price)
FROM ORDER_ITEMS WHERE order_id = p_oid)
        WHERE order_id = p_oid;

        LOG_ACTION(USER, 'ORDER_ITEMS', 'ADD', 'Book: ' || p_bid ||
' Qty: ' || p_q);
        COMMIT;
    ELSE
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient stock for book
ID: ' || p_bid);
    END IF;
END;
-- Динамический поиск по каталогу для UI
CREATE OR REPLACE PROCEDURE CLIENT_SEARCH_BOOKS(p_k VARCHAR2, p_cat
VARCHAR2, p_cur OUT SYS_REFCURSOR) IS
BEGIN
    OPEN p_cur FOR
```

```
        SELECT book_id, title, author, price_after_discount, stock
        FROM BOOKS
        WHERE (p_k IS NULL OR LOWER(title) LIKE '%'||LOWER(p_k)||'%')
          AND (p_cat IS NULL OR category = p_cat)
          AND stock > 0;
END;
-- Формирование сводного отчета для Dashboard
CREATE OR REPLACE PROCEDURE GET_GENERAL_STATS(p_o OUT NUMBER, p_rev
OUT NUMBER, p_c OUT NUMBER) IS
BEGIN
    SELECT COUNT(*), SUM(total_amount) INTO p_o, p_rev FROM ORDERS
WHERE status = 'Оплачен';
    SELECT COUNT(DISTINCT customer_id) INTO p_c FROM ORDERS;
END;
-- Создание заказа с проверкой безопасности
CREATE OR REPLACE PROCEDURE BOOKSTORE_USER.MANAGE_ORDER_CREATE(p_cus-
tomer_id IN NUMBER, p_order_id OUT NUMBER) IS
    v_status VARCHAR2(50); v_uid NUMBER;
BEGIN
    SELECT user_id INTO v_uid FROM CUSTOMERS WHERE customer_id =
p_customer_id;
    SELECT status INTO v_status FROM USERS WHERE user_id = v_uid;

    IF v_status IN ('BLOCKED', 'INACTIVE') THEN
        LOG_ACTION('System',          'ORDERS',          'DENIED',
'{"cust_id":'||p_customer_id||'}');
        p_order_id := NULL; RETURN;
    END IF;
    INSERT INTO ORDERS (customer_id, order_date, total_amount, sta-
tus)
    VALUES (p_customer_id, SYSDATE, 0, 'Новый') RETURNING order_id
INTO p_order_id;

    LOG_ACTION(USER, 'ORDERS', 'INSERT', '{"id":'||p_order_id||'}');
    COMMIT;
END;
/
-- Автоматическая покупка (Поиск корзины + Добавление)
CREATE OR REPLACE PROCEDURE BOOKSTORE_USER.CLIENT_BUY_BOOK_AUTO(
    p_username IN VARCHAR2, p_isbn IN VARCHAR2, p_qty IN NUMBER DE-
FAULT 1
) IS
    v_cid NUMBER; v_bid NUMBER; v_stock NUMBER; v_oid NUMBER;
BEGIN
    SELECT c.customer_id INTO v_cid FROM CUSTOMERS c JOIN USERS u ON
c.user_id = u.user_id WHERE u.username = p_username;
    SELECT book_id, stock INTO v_bid, v_stock FROM BOOKS WHERE
TRIM(isbn) = TRIM(p_isbn) AND is_archived = 0;
    IF v_stock < p_qty THEN RETURN; END IF;
    BEGIN
        SELECT order_id INTO v_oid FROM ORDERS WHERE customer_id =
v_cid AND status = 'Новый' FETCH FIRST 1 ROWS ONLY;
    EXCEPTION
```

```
        WHEN NO_DATA_FOUND THEN MANAGE_ORDER_CREATE(v_cid, v_oid);
    END;
    MANAGE_ORDER_ADD_ITEM(v_oid, v_bid, p_qty);
    COMMIT;
END;
/
-- Добавление книги
CREATE OR REPLACE PROCEDURE BOOKSTORE_USER.MANAGE_PRODUCT_ADD(
    p_isbn IN VARCHAR2, p_title IN VARCHAR2, p_author IN VARCHAR2,
    p_price IN NUMBER, p_stock IN NUMBER, p_category IN VARCHAR2,
p_url IN VARCHAR2 DEFAULT NULL
) IS
BEGIN
    IF p_price < 0 OR p_stock < 0 THEN RETURN; END IF;
    INSERT INTO BOOKS (isbn, title, author, price, stock, category,
image_url)
    VALUES (p_isbn, p_title, p_author, p_price, p_stock, p_category,
p_url);

    LOG_ACTION(USER, 'BOOKS', 'INSERT', '{"isbn":"'||p_isbn||'"}');
    COMMIT;
END;
/
-- Мягкое удаление книги
CREATE OR REPLACE PROCEDURE MANAGE_PRODUCT_DELETE(p_book_id IN NUM-
BER) IS
    v_cnt NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_cnt FROM order_items WHERE book_id =
p_book_id;

    IF v_cnt > 0 THEN
        UPDATE books SET stock = 0, is_archived = 1, title = '[АРХИВ]
' || title WHERE book_id = p_book_id;
        LOG_ACTION(USER,           'BOOKS',           'SOFT_DELETE',
'{"id":'||p_book_id||'}');
    ELSE
        DELETE FROM books WHERE book_id = p_book_id;
        LOG_ACTION(USER,             'BOOKS',             'DELETE',
'{"id":'||p_book_id||'}');
    END IF;
    COMMIT;
END;/
```

ПРИЛОЖЕНИЕ В
Листинг создания пользователей и ролей

```
-- 1. Присвоение ролей
GRANT RLAdmin TO AdminUser;
GRANT RLSeller TO SellerUser;
GRANT RLClient TO ClientUser;
-- 2. Права на вход
GRANT CREATE SESSION TO AdminUser;
GRANT CREATE SESSION TO SellerUser;
GRANT CREATE SESSION TO ClientUser;
-- 3. Привилегии админа
GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKSTORE_USER.BOOKS TO
RLAdmin;
GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKSTORE_USER.USERS TO
RLAdmin;
GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKSTORE_USER.CUSTOMERS TO
RLAdmin;
GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKSTORE_USER.ORDERS TO
RLAdmin;
GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKSTORE_USER.ORDER_ITEMS TO
RLAdmin;
GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKSTORE_USER.AUDIT_LOG TO
RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.CREATE_BOOK TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.CREATE_USER TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.CREATE_CUSTOMER TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.CREATE_ORDER TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.ADD_ORDER_ITEM TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.UPDATE_USER_STATUS TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.UPDATE_ORDER_STATUS TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.DELETE_CUSTOMER TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.DELETE_ORDER TO RLAdmin;
GRANT EXECUTE ON BOOKSTORE_USER.LOG_ACTION TO RLAdmin;
--4. привилегии продавца
GRANT SELECT, INSERT, UPDATE ON BOOKSTORE_USER.BOOKS TO RLSeller;
GRANT SELECT, INSERT, UPDATE ON BOOKSTORE_USER.ORDERS TO RLSeller;
GRANT SELECT, INSERT, UPDATE ON BOOKSTORE_USER.ORDER_ITEMS TO
RLSeller;
GRANT EXECUTE ON BOOKSTORE_USER.CREATE_ORDER TO RLSeller;
GRANT EXECUTE ON BOOKSTORE_USER.ADD_ORDER_ITEM TO RLSeller;
GRANT EXECUTE ON BOOKSTORE_USER.UPDATE_ORDER_STATUS TO RLSeller;
GRANT SELECT ON BOOKSTORE_USER.BOOKS TO RLClient;
GRANT SELECT ON BOOKSTORE_USER.ORDERS TO RLClient;
GRANT SELECT ON BOOKSTORE_USER.ORDER_ITEMS TO RLClient;
GRANT SELECT ON BOOKSTORE_USER.CUSTOMERS TO RLClient;
```

ПРИЛОЖЕНИЕ Г
Листинг создания триггеров

```
--Запрет вставки пользователя с username = 'admin'
CREATE OR REPLACE TRIGGER trg_prevent_admin_user
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    IF :NEW.username = 'admin' THEN
        RAISE_APPLICATION_ERROR(-20001,     'Невозможно     создать
пользователя с username = admin');
    END IF;
END;/
--Обязательные поля для клиентов
CREATE OR REPLACE TRIGGER trg_check_customer_fields
BEFORE INSERT ON customers
FOR EACH ROW
BEGIN
    IF :NEW.full_name IS NULL
       OR :NEW.email IS NULL
       OR :NEW.phone IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002,    'Заполните    обязательные
поля: full_name, email, phone');
    END IF;
END;/
--Проверка уникальности email у клиентов
CREATE OR REPLACE TRIGGER trg_unique_customer_email
BEFORE INSERT OR UPDATE ON customers
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM customers
    WHERE email = :NEW.email
      AND customer_id != NVL(:NEW.customer_id, 0);

    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Email клиента должен быть
уникальным');
    END IF;
END;/
--Проверка количества на складе перед добавлением позиции заказа
CREATE OR REPLACE TRIGGER trg_check_stock
BEFORE INSERT ON order_items
FOR EACH ROW
DECLARE
    v_stock NUMBER;
BEGIN
    SELECT stock INTO v_stock
    FROM books
    WHERE book_id = :NEW.book_id;
```

```
    IF v_stock < :NEW.qty THEN
        RAISE_APPLICATION_ERROR(-20005,   'Недостаточно   книг   на
складе');
    END IF;
END;
/
--Автоматическое обновление статуса заказа
CREATE OR REPLACE TRIGGER trg_auto_order_status
BEFORE INSERT ON orders
FOR EACH ROW
BEGIN
    IF :NEW.status IS NULL THEN
        :NEW.status := 'Новый';
    END IF;
END;
/
BEGIN
    -- Попытка создать обычного пользователя
    create_user('ivanov_test2', '12345', 1);
    DBMS_OUTPUT.PUT_LINE('Пользователь создан успешно');
    -- Попытка создать пользователя admin (если есть проверка внутри
триггера/процедуры)
    create_user('admin', '12345', 1);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Ошибка: ' || SQLERRM);
END;
/
SELECT * FROM users WHERE username = 'ivanov_test';
SELECT * FROM customers WHERE user_id = 2;
```