Частное учреждение профессионального образования
«Высшая школа предпринимательства»
(ЧУПО «ВШП»)

КУРСОВОЙ ПРОЕКТ

«Разработка базы данных для магазина компьютерных комплектующих»

Выполнил:
студент 3-го курса специальности
09.02.07 «Информационные системы и программирование»
Колосов Кирилл Андреевич
подпись: _____

Проверил:
преподаватель дисциплины,
преподаватель ЧУПО «ВШП»,
к.ф.н. Ткачев П.С.
оценка: _____
подпись: _____

Тверь, 2024 г.

Содержание

Введение

В современном мире информационных технологий спрос на компьютерные комплектующие постоянно растет. Компьютеры стали неотъемлемой частью повседневной жизни, используемой как в домашних условиях, так и в рабочих и образовательных целях. В связи с этим, эффективное управление и хранение данных о компьютерных комплектующих становится все более важным для бизнеса.

Целью данного курсового проекта является разработка базы данных для магазина компьютерных комплектующих. Создание такой базы данных позволит оптимизировать процессы управления складом, контроль за товарным ассортиментом, а также повысить качество обслуживания клиентов.

В настоящее время существует множество магазинов, специализирующихся на продаже компьютерных комплектующих. Однако, не все из них обладают эффективной системой управления данными, что может приводить к проблемам с учетом товаров, контролем за их наличием и заказами. Разработка базы данных, специально адаптированной для потребностей магазина компьютерных комплектующих, позволит улучшить эффективность работы предприятия, и увеличить его конкурентоспособность на рынке.

В ходе выполнения курсового проекта будет рассмотрен процесс проектирования и реализации базы данных, включая определение структуры данных, выбор используемых технологий и инструментов, а также разработку интерфейса для взаимодействия с базой данных. Также будут рассмотрены вопросы безопасности данных и их резервного копирования для обеспечения надежности хранения информации.

В конечном итоге, разработка базы данных для магазина компьютерных комплектующих позволит повысить эффективность управления предприятием, улучшить обслуживание клиентов и увеличить его конкурентоспособность на рынке информационных технологий.

Разработанность

В настоящее время базы данных для магазинов, компьютерных комплектующих делятся на два основных типа. Общие базы данных представляют собой системы управления складом и продажами, которые помогают в учете товаров, заказах и клиентах. Такие системы, как Retail Pro или 1С: Управление торговлей, обрабатывают большие объемы данных и помогают магазинам эффективно управлять своим бизнесом.

Специализированные базы данных о компьютерных комплектующих содержат информацию о конкретных продуктах, таких как процессоры, видеокарты и материнские платы. Они включают технические характеристики товаров, цены, наличие на складе и отзывы клиентов. Разработка такой базы данных требует учета потребностей магазина, таких как управление запасами и обработка заказов, для эффективного управления бизнесом и обеспечения удовлетворения клиентов.

Цель

Целью данной курсовой работы является создание базы данных – для магазина компьютерных комплектующих.

Задачи

1. Провести анализ предметной области.
2. Определить требования к разрабатываемой базе данных.
3. Спроектировать схему базы данных.
4. Реализовать базу данных.
5. Создание функциональных объектов БД.
6. Назначить права пользователям базы данных.
7. Проверить работоспособность и корректность функционирования базы данных.

Объект исследования

Объектом исследования являются базы данных.

Предмет исследования

Предметом исследования являются базы данных компьютерных комплектующих для магазинов.

Методы исследования

- Метод анализа - Изучение существующих баз данных компьютерных комплектующих, используемых в магазинах.

- Метод сравнения - Сравнение различных баз данных компьютерных комплектующих между собой.

- Метод абстрагирования - Создание общей схемы и принципов работы разрабатываемой базы данных для магазина компьютерных комплектующих.

- Метод структурных аналогий - Создание базы данных компьютерных комплектующих на основе общих элементов, наблюдаемых в существующих базах данных.

Глава 1. Теоретический базис разработки БД

1.1 Анализ предметной области

База данных - упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе, существует большое множество типов баз данных. [1]

Базы данных можно разделить на несколько основных типов. Вот основные виды баз данных:

Из основных видов можно назвать такие:

- Иерархические - модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней. [3]

- Объектно-ориентированные базы данных - это постреляционная модель БД. Она включает в себя таблицы, но не ограничивается ими. В таких БД информация хранится в виде набора объектов или программных элементов многократного использования. [4]

- Реляционные - модель данных предусматривающая единственный способ предоставления информации - в виде набора двумерных таблиц и отношений между ними. [7]

- Сетевая модель данных, логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных. [9]

Сегодня существует множество СУБД, из популярных я могу выделить несколько основных:

1. **MySQL** - свободная реляционная система управления базами данных (СУБД). Под словом «свободная» подразумевается ее бесплатность, под «реляционная» – работа с базами данных, основанных на двумерных таблицах. Система выпущена в 1995 году, её разработка активно продолжается. [12]

2. **PostgreSQL** - это объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире. Имеет открытый исходный код и является альтернативой коммерческим базам данных. [13]

3. **Microsoft Access** - реляционная система управления базами данных (СУБД) корпорации Microsoft. Входит в состав пакета Microsoft Office. Имеет широкий спектр функций, включая связанные запросы, связь с внешними таблицами и базами данных. [11]

1.2 Анализ связей между сущностями.

Связи между сущностями в базе данных MySQL являются основой для создания структурированных и взаимосвязанных данных. В реляционной модели данных связи реализуются с помощью ключей и ограничений, что обеспечивает целостность и согласованность данных.

Типы связей

Один-к-одному (1:1): Связь, при которой каждая запись одной таблицы соответствует одной записи другой таблицы.

Один-ко-многим (1:n): Связь, при которой каждая запись одной таблицы соответствует нескольким записям другой таблицы.

Многие-ко-многим (n:m): Связь, при которой каждая запись одной таблицы соответствует нескольким записям другой таблицы и наоборот.

Реализация связей один-к-одному (1:1)

Связь один-к-одному реализуется с использованием внешних ключей и уникальных ограничений. Таблицы, участвующие в такой связи, содержат соответствующие первичные и внешние ключи, обеспечивающие уникальность записей.

Реализация связей один-ко-многим (1:n)

Для реализации связи один-ко-многим одна из таблиц должна содержать внешний ключ, ссылающийся на первичный ключ другой таблицы. Это позволяет одной записи в "родительской" таблице соответствовать нескольким записям в "дочерней" таблице.

Реализация связей многие-ко-многим (n:m)

Связь многие-ко-многим требует создания промежуточной таблицы, которая содержит внешние ключи, ссылающиеся на обе связанные таблицы. Эта таблица служит для установления множественных связей между записями двух таблиц. [8]

Внешние ключи и целостность данных

Внешние ключи обеспечивают целостность данных, гарантируя, что значения в одной таблице имеют соответствующие значения в другой таблице. В MySQL можно настроить каскадные действия, которые автоматически обновляют или удаляют связанные записи при изменении или удалении данных в родительской таблице.

Индексация и оптимизация связей

Для улучшения производительности запросов, связанных с внешними ключами, рекомендуется создавать индексы на столбцах, участвующих в связях. Индексация помогает ускорить поиск и соединение таблиц, что особенно важно для больших объемов данных.

1.3 Составить требования к разрабатываемой базе данных

1. Общие требования:

1.1. Надежность: База данных должна обеспечивать высокую надежность хранения и обработки данных.

1.2. Производительность: Необходимо обеспечить высокую производительность при выполнении операций добавления, обновления, удаления и поиска данных.

1.3. Масштабируемость: Система должна поддерживать возможность горизонтального и вертикального масштабирования для обработки увеличивающихся объемов данных.

1.4. Безопасность: Доступ к базе данных должен быть защищен от несанкционированного доступа. Должны быть реализованы механизмы аутентификации и авторизации пользователей.

2. Функциональные требования:

2.1. Управление товарами:

Хранение информации о товарах (название, описание, спецификация, категория, производитель, цена, количество).

Возможность добавления, обновления и удаления информации о товарах.

2.2. Управление производителями:

Хранение информации о производителях (название, страна, веб-сайт).

Возможность связывать производителя с продуктом.

2.3. Управление категориями:

Хранение информации о категориях (имя, описание).

Возможность связывать категории с продуктом.

2.4. Управление запасами:

Хранение информации о запасах (id товара, код склада, количество).

Возможность связывать запасы с продуктом.

2.5. Управление складами:

Хранение информации о складах (код склада, место расположение).

Возможность связывать склады с запасами.

2.6. Управление клиентами:

Хранение информации о клиентах (имя, фамилия, адрес доставки, email, номер телефона).

2.7. Управление заказами:

Хранение информации о заказах (id пользователя, дата заказа, статус заказа,).

Возможность связывать заказы с клиентами и товарами.

Поддержка различных статусов заказа (новый, в обработке, отправлен, доставлен, отменен).

2.8. Управление деталями заказа:

Хранение информации о деталях заказа (id заказа, id продукта, количества, со склада или нет).

Возможность связывать детали заказа с заказами.

1.4 Спроектировать схему базы данных

Диаграммы «сущность-связь» (или ERD) — неотъемлемая составляющая процесса моделирования любых систем, включая простые и сложные базы данных, однако применяемые в них фигуры и способы нотации могут ввести в заблуждение. Концептуальные модели данных дают общее представление о том, что должно входить в состав модели. Концептуальные ER-диаграммы можно брать за основу логических моделей данных. Их также можно использовать для создания отношений общности между разными ER-моделями, положив их в основу интеграции. [2]

Для создания схемы базы данных я буду использовать функцию моделирования в MySQL Workbench.

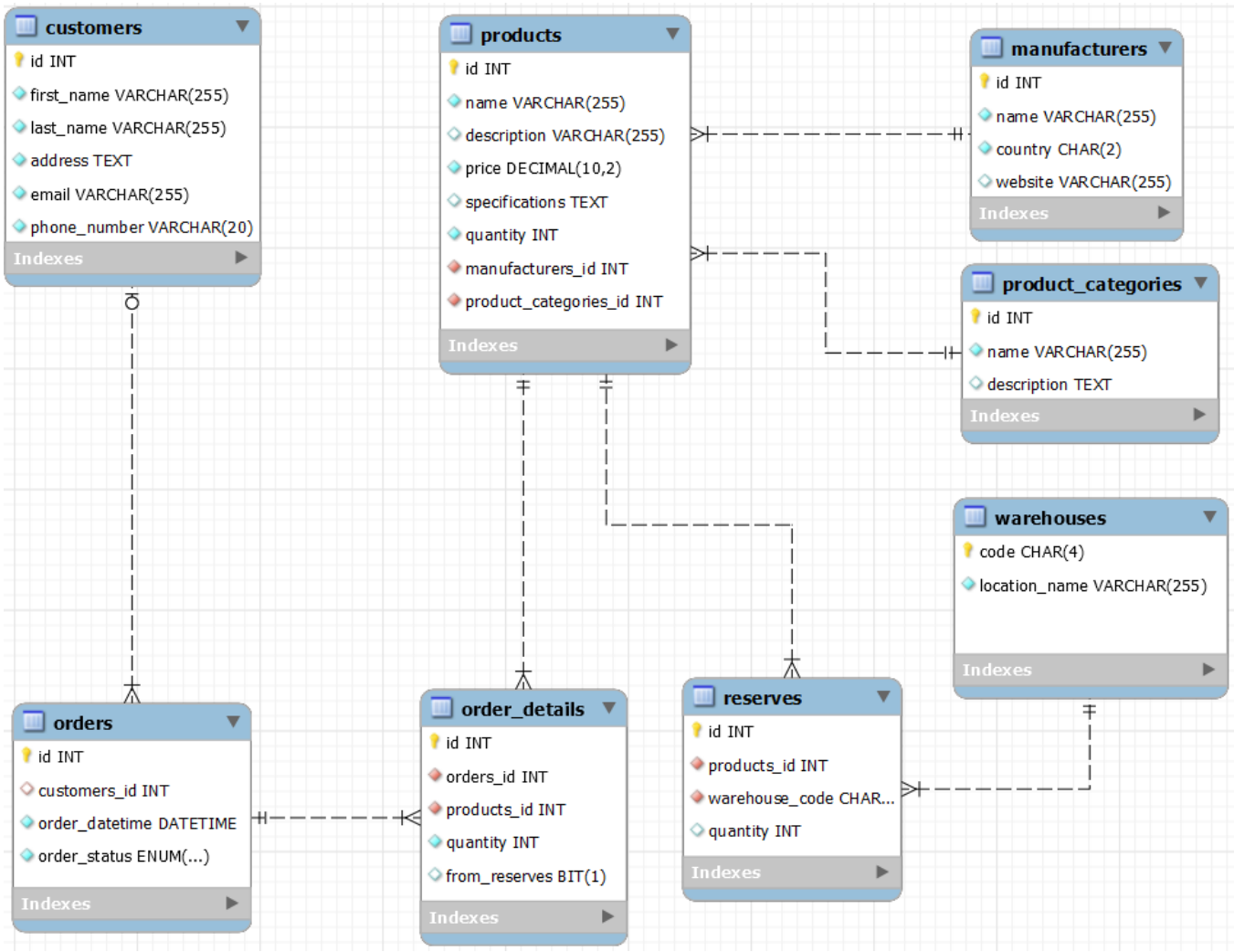

Рис. 1

На схеме (рис. 1) представлена готовая база данных, спроектированная на основе ранее сформулированных требований.

Схема включает 8 таблиц, каждая из которых выполняет или способствует выполнению поставленных задач:

1. customers [клиенты] – таблица которая содержит в себе информацию о клиентах.
2. products [продукты] – таблица которая содержит в себе информацию о продуктах.
3. orders [заказы] – таблица которая содержит в себе данные заказа.
4. orders_details [детали заказа] – таблица которая содержит в себе детали заказов.
5. manufacturers [производители] – таблица которая содержит в себе производителей.

6. product_categories [категории] – таблица которая содержит в себе информацию про категории.
7. reserves [запасы] – таблица которая содержит в себе информацию про запасы товаров.
8. warehouses [склады] – таблица которая содержит в себе информацию про склады.

Такой набор таблиц нужен, чтобы удовлетворить все требования базы данных для магазина компьютерных комплектующих.

Глава 2. Таблицы и типы значений

2.1 Создание таблицы

Теперь приступим к созданию таблиц в соответствии с разработанной схемой.

- Таблица customers содержит следующие столбцы: id, first_name, last_name, address, email, phone_number.
- Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. Тип данных INT выбран для предотвращения ограничения диапазона значений. Primary Key служит уникальным идентификатором каждой записи в таблице, а Auto Increment автоматически присваивает порядковый номер каждой записи. Атрибут Not Null применяется, поскольку id не может быть пустым.
- Для first_name используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null.
- Для last_name используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null.
- Для address используется тип данных TEXT. Тип данных TEXT выбран для хранения адреса, который может быть длиннее 255 символов. Также применяется атрибут Not Null.
- Для email используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null. Также применяется атрибут Unique Key обеспечивает уникальность значений в столбце email, предотвращая дублирование адресов электронной почты.
- Для phone_number используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null. Также применяется атрибут Unique Key обеспечивает

уникальность значений в столбце phone_number, предотвращая дублирование номеров телефонов.

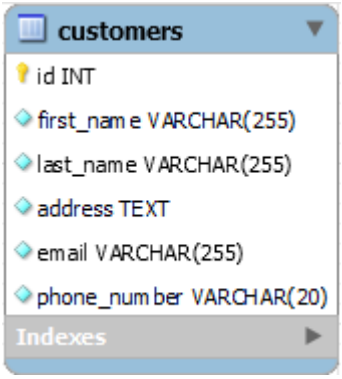

Рис. 2
таблица customers.

- Таблица products содержит столбцы: id, name, description, price, specifications, quantity, manufactures_id, product_categories_id.
- Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key.
- Для name используется тип данных VARCHAR (255). Также применяется атрибут Not Null.
- Для description используется тип данных VARCHAR (255).
- Для price используется тип данных DECIMAL (10,2) Тип данных DECIMAL (10,2) выбран чтобы обеспечить точность при хранении денежных значений, включая копейки. Также применяется атрибут Not Null.
- Для specifications используется тип данных TEXT. Тип данных TEXT выбран для хранения подробных характеристик продукта.
- Для quantity используется тип данных INT с атрибутами Not Null.
- Для manufactures_id используется тип данных INT с атрибутами Not Null и требуется для создания связи (1:n) с таблицей manufactures.
- Для product_categories_id используется тип данных INT с атрибутами Not Null и требуется для создания связи (1:n) с таблицей product_categories.

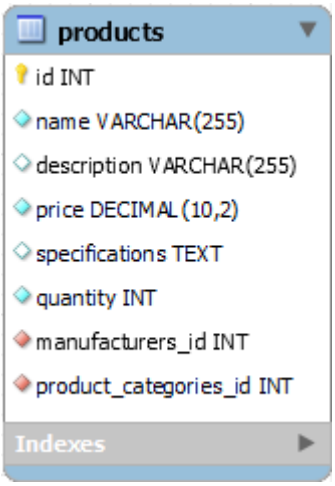

Рис. 3

таблица products.

- Таблица manufactures содержит столбцы: id, name, country, website.
- Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key.
- Для name используется тип данных VARCHAR (255). Также применяется атрибут Not Null.
- Для country используется тип данных CHAR (2) с атрибутом Not Null. Тип данных CHAR (2) выбран для хранения кода страны производителя (два символа).
- Для website используется тип данных VARCHAR (255).

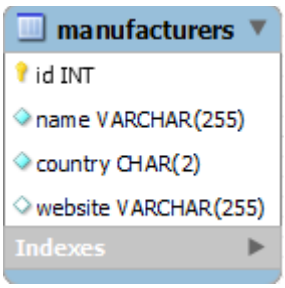

Рис. 4

таблица manufactures.

- Таблица product_categories содержит столбцы: id, name, description.
- Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key.

- Для name используется тип данных VARCHAR (255). Также применяется атрибут Not Null.
- Для description используется тип данных TEXT.

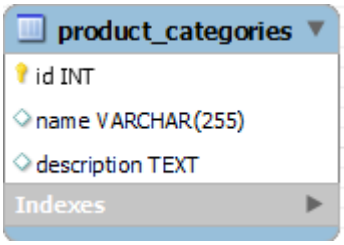

Рис. 5

таблица product_categories.

- Таблица orders содержит столбцы: id, customers_id, order_datetime, order_status.
- Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key.
- Для customers_id используется тип данных INT. Требуется для создания связи (1:n) с таблицей customers.
- Для order_datetime используется тип данных DATETIME с атрибутами Not Null. Тип данных DATETIME выбран для хранения даты и времени размещения заказа.
- Для order_status используется тип данных ENUM ('processed', 'created', 'completed', 'canceled') с атрибутами Not Null. Тип данных ENUM выбран для хранения статуса заказа, который может принимать одно из предопределенных значений.

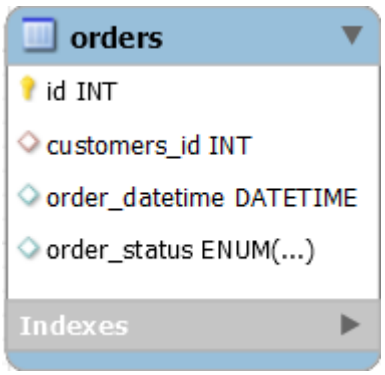

Рис. 6 таблица orders.

- Таблица orders_details содержит столбцы: id, orders_id, products_id, quantity, total_price, from_reserves.
- Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key.
- Для orders_id используется тип данных INT с атрибутами Not Null. Требуется для создания связи (1:n) с таблицей orders.
- Для products_id используется тип данных INT с атрибутами Not Null. Требуется для создания связи (1:n) с таблицей products.
- Для quantity используется тип данных INT UNSIGNED с атрибутами Not Null. Тип данных INT UNSIGNED выбран для хранения количества, так как это значение не может быть отрицательным.
- Для total_price используется тип данных DECIMAL (10,2) который не может быть NULL
- Для from_reserves используется тип данных BIT (1). Тип данных BIT (1) выбран для хранения булевого значения, указывающего, было ли взято из резерва.

Рис. 7

таблица orders_details.

- Таблица reserves содержит столбцы: id, products_id, warehouse_code, quantity.
- Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key.
- Для products_id используется тип данных INT с атрибутами Not Null. Требуется для создания связи (1:n) с таблицей products.

- Для warehouse_code используется тип данных CHAR (4) с атрибутами Not Null. Тип данных CHAR (4) выбран для хранения кода склада, на котором зарезервирован продукт. Требуется для создания связи (1: n) с таблицей warehouses.
- Для quantity используется тип данных INT.

Рис. 8

таблица reserves.

- Таблица warehouses содержит столбцы: code, location_name.
- Для code используется тип данных CHAR (4) с атрибутом Not Null. Тип данных CHAR (4) выбран для хранения кода склада, который является уникальным идентификатором id.
- Для location_name используется тип данных VARCHAR (255). Также применяется атрибут Not Null.

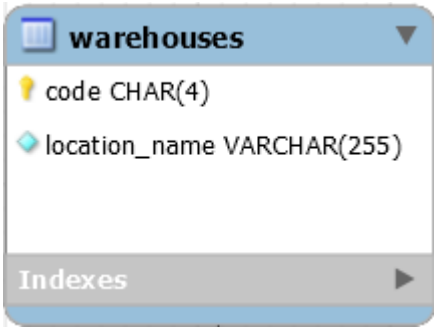

Рис. 9

таблица warehouses.

2.2 Создание хранимых процедур.

Хранимая процедура (англ. Stored procedure) — объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения. [15]

В базе данных предусмотрены две хранимые процедуры с обработкой исключений, одна хранимая процедура, использующая три локальные переменные с заданными типами данных, и одна хранимая процедура, реализующая транзакцию.

Первая хранимая процедура используется как фильтр для поиска товара по категории. (см. Приложение 1)

Этот блок проверяет, существует ли категория с указанным именем. Если нет, то генерируется ошибка 'Категория не найдена'.

```
...
    -- Проверяем, существует ли категория с указанным именем
    IF NOT EXISTS (SELECT name FROM product_categories WHERE name = category_name) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Категория не найдена';
    END IF;
...
```

В этом блоке кода идет выборка товаров из таблицы.

```
...
    -- Если категория существует, выбираем продукты из таблицы
    SELECT p.id, p.name, p.description, p.price, p.quantity, c.name AS
category_name
    FROM products p
    JOIN product_categories c ON p.product_categories_id = c.id
    WHERE c.name = category_name;
...
```

Вторая хранимая процедура используется как фильтр для поиска товара по производителю. (см. Приложение 2)

Этот блок проверяет, существует ли производитель с указанным именем. Если нет, то генерируется ошибка 'Производитель не найден'.

```
...
-- Проверяем, существует ли производитель с указанным именем
IF NOT EXISTS (SELECT name FROM manufacturers WHERE name =
manufacturer_name) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Производитель не
найден';
    END IF;
...
```

В этом блоке кода идет выборка товаров из таблицы.

```
...
SELECT p.id, p.name, p.description, p.price, p.quantity, m.name AS
manufacturer_name
    FROM products p
    JOIN manufacturers m ON p.manufacturers_id = m.id
    WHERE m.name = manufacturer_name;
...
```

Третья хранимая используется для обновления цены товара. (см. Приложение 3)

В этом блоке создается хранимая процедура `update_product_price` с тремя локальными переменными с заданном типом данных.

```
...
CREATE PROCEDURE update_product_price(
    IN product_id INT,
    IN new_price DECIMAL(10, 2)
)
BEGIN
    DECLARE current_price DECIMAL(10, 2);
    DECLARE price_difference DECIMAL(10, 2);
    DECLARE updated_rows INT;
...
```

Эта строка выполняет запрос, который получает текущую цену продукта из таблицы products и сохраняет ее в переменной current_price.

```
...
 -- Получение текущей цену продукта
    SELECT price INTO current_price FROM products WHERE id =
product_id;
...
```

Эта строка вычисляет разницу между новой и текущей ценой и сохраняет результат в переменной price_difference.

```
...
 -- Вычисление разницы между новой и текущей ценой
    SET price_difference = new_price - current_price;
...
```

Эта строка обновляет цену продукта в таблице products до новой цены, указанной в параметре new_price, для продукта с идентификатором product_id.

```
...
   -- Обновление цены продукта
    UPDATE products SET price = new_price WHERE id = product_id;
...
```

Эта строка получает количество строк, затронутых последней командой UPDATE, и сохраняет это значение в переменной updated_rows.

```
...
  -- Получить количество обновленных строк
    SELECT ROW_COUNT() INTO updated_rows;
...
```

Этот блок проверяет, были ли обновлены строки: Если updated_rows больше 0, выводится сообщение о том, что цена продукта успешно обновлена. Если updated_rows равно 0, выводится сообщение об ошибке.

```
...
-- Выводим сообщение о выполнении операции
    IF updated_rows > 0 THEN
        SELECT CONCAT('Цена продукта ', product_id, ' успешно
обновлена.') AS message;
    ELSE
        SELECT CONCAT('Ошибка при обновлении цены продукта ',
product_id) AS message;
    END IF;
...
```

Четвертая хранимая процедура `make_order` предназначена для обработки заказов в системе. Она автоматизирует процесс создания заказа и включает в себя логику для работы с различными сценариями. (см. Приложение 4)

В этом блоке кода создается хранимая процедура с названием make_order которая принимает четыре входных параметра customer_id типа INT, product_id типа INT, order_quantity типа INT, from_reserves типа BIT (1). Также здесь объявляется локальная переменная order_id типа INT, которая будет использоваться для хранения идентификатора созданного заказа.

```
...
CREATE PROCEDURE make_order(
    IN customer_id INT,
    IN product_id INT,
    IN order_quantity INT,
    IN from_reserves BIT(1)
)
BEGIN
    DECLARE order_id INT;
...
```

Здесь объявляется обработчик исключений:

```
...
    -- Объявляем обработчик исключений
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Откатываем транзакцию в случае возникновения исключения
        ROLLBACK;
        -- Выдаем сообщение об ошибке
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Произошла ошибка
при обработке заказа';
    END;
...
```

В этом блоке кода начинается транзакция и создается новый заказ со склада:

```
...
-- Начать транзакцию
    START TRANSACTION;
```

```
    IF from_reserves = true THEN
        -- Создаем новый заказ со склада
        INSERT INTO orders (customers_id, order_datetime,
order_status)
        VALUES (customer_id, NOW(), 'processed');
...
```

Здесь мы получаем id только что созданного заказа:

```
...
-- Получаем ID только что созданного заказа
        SET order_id = LAST_INSERT_ID();
...
```

В этом блоке кода добавляется информация о деталях заказа в таблицу `order_details` и уменьшаем количество товара в таблице products.

```
...
-- Добавляем детали заказа
        INSERT INTO order_details (orders_id, products_id, quantity,
from_reserves, total_price)
        VALUES (order_id, product_id, order_quantity, 1,
order_quantity * product_price);
    ELSE
        -- Уменьшаем количество продуктов в таблице products
        UPDATE products
        SET quantity = quantity - order_quantity
        WHERE id = product_id;
...
```

Создаем новый заказ и получаем id только что созданного заказа, добавляем информацию о деталях заказа и завершаем транзакцию:

```
...
        -- Создаем новый заказ
        INSERT INTO orders (customers_id, order_datetime,
order_status)
```

```
        VALUES (customer_id, NOW(), 'processed');

        -- Получаем ID только что созданного заказа
        SET order_id = LAST_INSERT_ID();

        -- Добавляем детали заказа
        INSERT INTO order_details (orders_id, products_id, quantity,
from_reserves)
        VALUES (order_id, product_id, order_quantity, 0);
    END IF;

    -- Завершаем транзакцию
    COMMIT;
END //
```

2.3 Создание триггеров

Триггеры в MySQL 8 представляют собой специальные объекты базы данных, которые автоматически выполняются при определенных событиях или действиях с данными. Они позволяют программистам определить пользовательские действия, которые должны быть выполнены перед или после выполнения операции в базе данных. Иными словами, это определяемая пользователем SQL-команда, которая автоматически вызывается во время операций INSERT, DELETE или UPDATE. Код триггера связан с таблицей и уничтожается после удаления таблицы. Вы можете определить время действия триггера и указать, когда его нужно активировать — до или после определенного события базы данных. Поддержка триггеров в MySQL началась с версии 5.0.2. [14]

В данной базе данных триггеры используются для дополнение хранимой процедуры с транзакцией make_order

Триггер order_from_reserves_trigger автоматически обновляет таблицу запасов (reserves) при каждом добавлении новой записи в таблицу деталей заказа

(order_details). Если товар берется со склада, триггер автоматически уменьшает количество доступных товаров на складе на количество, указанное в заказе.

```
DELIMITER //

CREATE TRIGGER order_from_reserves_trigger
AFTER INSERT ON order_details
FOR EACH ROW
BEGIN
    IF NEW.from_reserves = true THEN
        UPDATE reserves
        SET quantity = quantity - NEW.quantity
        WHERE products_id = NEW.products_id;
    END IF;
END;
//
DELIMITER ;
```

2.4 Создание типовых запросов

Типовые запросы в MySQL 8 являются мощным инструментом, который помогает улучшить производительность, безопасность и удобство разработки. Они широко применяются в реальных проектах и являются важной частью современной работы с базами данных.

1. Получить среднюю стоимость заказа по каждому клиенту:

```
SELECT c.last_name AS фамилия, c.first_name AS имя,
ROUND(AVG(od.total_price)) AS средняя_стоимость_заказа
FROM customers c
LEFT JOIN orders o ON c.id = o.customers_id
LEFT JOIN order_details od ON o.id = od.orders_id
GROUP BY c.id, c.last_name, c.first_name;
```

2. Выяснить, в каких категориях продуктов есть товары с низкими остатками (менее 10):

```
SELECT pc.name AS название_категории, p.name AS название_продукта, p.quantity AS количество
FROM products p
JOIN product_categories pc ON p.product_categories_id = pc.id
WHERE p.quantity < 10;
```

3. Найти топ-5 клиентов, сделавших наибольшее количество заказов:

```
SELECT c.first_name, c.last_name, COUNT(o.id) AS общее_количество_заказов
FROM customers c
JOIN orders o ON c.id = o.customers_id
GROUP BY c.id
ORDER BY общее_количество_заказов DESC
LIMIT 5;
```

4. Узнать среднюю цену продуктов в каждой категории:

```
SELECT pc.name AS название_категории, AVG(p.price) AS средняя_цена
FROM products p
JOIN product_categories pc ON p.product_categories_id = pc.id
GROUP BY pc.name;
```

5. Получить общее количество продуктов в каждом складе:

```
SELECT w.location_name AS склад,
SUM(r.quantity) AS общее_количество_продуктов
FROM warehouses w
LEFT JOIN reserves r ON w.code = r.warehouse_code
GROUP BY w.location_name;
```

2.5 Создание представления таблиц

Представления (Views) в MySQL — это виртуальные таблицы, которые основаны на результатах выполнения запроса SELECT. Они представляют собой

логические структуры данных, которые могут быть использованы для упрощения и оптимизации запросов к базе данных.

Основная цель использования представлений в MySQL — это абстрагирование сложных запросов и создание более простых и понятных интерфейсов для работы с данными. Вместо того, чтобы каждый раз писать сложные запросы, можно создать представление, которое будет содержать уже готовый запрос. После этого можно обращаться к представлению как к обычной таблице, выполнять SELECT-запросы к нему и получать результаты. [6]

Эта виртуальная таблица (представление), показывает все заказы, соединяя данные о заказах, клиентах, деталях заказов и продуктах в одну таблицу. В результате получаем удобный способ видеть информацию о каждом заказе: когда он был сделан, кем, какие товары были заказаны, сколько, по какой цене и со склада ли они.

```
CREATE VIEW order_details_view AS
SELECT o.id AS order_id, o.order_datetime, c.first_name, c.last_name,
p.name AS product_name, od.quantity, od.from_reserves, p.price
FROM orders o
JOIN customers c ON o.customers_id = c.id
JOIN order_details od ON o.id = od.orders_id
JOIN products p ON od.products_id = p.id;
```

2.6 Создание пользовательской функции

Пользовательские функции во многом напоминают хранимые процедуры и представляют упорядоченное множество операторов T-SQL, которые заранее оптимизированы, откомпилированы и могут быть вызваны для выполнения работы в виде единого модуля. Основное различие между пользовательскими функциями и хранимыми процедурами состоит в том, как в них осуществляется возврат полученных результатов. А в связи с тем, что для обеспечения, предусмотренного в них способа возврата значений в пользовательских функциях должны

осуществляться немного другие действия, к их синтаксической структуре предъявляются более жесткие требования по сравнению с хранимыми процедурами. [5]

Пользовательская функция `calculate_order_total` вычисляет общую стоимость всех товаров определенного продукта, включая как основное количество, так и количество со склада, и умножает их суммарное количество на цену продукта. Она делает это путем получения данных из таблиц products и reserves, обработки этих данных и возврата рассчитанного значения. (см. Приложение 4)

В этом блоке кода создается пользовательская функция с названием calculate_order_total и переменные для хранения количества продуктов, количества товаров со склада, цены продукта и общей стоимости.

```
...
CREATE FUNCTION calculate_order_total(product_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE product_quantity INT;
    DECLARE reserve_quantity INT;
    DECLARE product_price DECIMAL(10,2);
    DECLARE total_price DECIMAL(10,2);
...
```

Запрашивает количество и цену продукта по заданному product_id и сохраняет их в переменные product_quantity и product_price.

```
...
-- Получаем количество и цену продукта из таблицы products
    SELECT quantity, price INTO product_quantity, product_price FROM
products WHERE id = product_id;
...
```

Запрашивает общее количество данного продукта из таблицы reserves и сохраняет его в переменную reserve_quantity. Использует SUM(quantity), чтобы получить суммарное количество товаров со склада для указанного продукта.

```
...
-- Получаем количество товара из таблицы reserves
    SELECT SUM(quantity) INTO reserve_quantity FROM reserves WHERE
product_id = product_id;
...
```

Если reserve_quantity равен NULL (то есть товара нет на складе), устанавливает его значение как 0.

```
...
-- Если товар не найден в таблице reserves, устанавливаем его
количество как 0
    IF reserve_quantity IS NULL THEN
        SET reserve_quantity = 0;
    END IF;
...
```

Вычисляет общую стоимость всех товаров (как основных, так и со склада), умножая их суммарное количество на цену продукта и возвращаем общую стоимость.

```
...
-- Вычисляем общее количество товаров и общую стоимость
    SET total_price = (product_quantity + reserve_quantity) *
product_price;

    RETURN total_price;
END $$
...
```

2.7 Создание пользователей и назначение привилегий

В информационных системах безопасность данных является одним из важнейших аспектов. При работе с базой данных MySQL, рекомендуется следовать принципу наименьших привилегий и создавать пользователей с ограниченным доступом к базе данных. Администратор MySQL должен создавать такие учетные записи для "обычных" пользователей и определять их права доступа. Это позволяет предоставлять пользователям только те привилегии, которые им действительно необходимы для работы, и обеспечивать безопасность данных. Кроме того, проведение аудита полномочий и корректировка их при необходимости также являются важными шагами для обеспечения безопасности базы данных. [10]

В современных базах данных часто используется модель управления доступом на основе ролей, где роли представляют собой совокупности прав и привилегий, назначаемые пользователям для выполнения определенных задач. Роль admin_role представляет собой одну из таких ролей, которая часто используется для предоставления полных или расширенных прав доступа к базе данных.

Роль администратора

Создание роли admin_role:

```
...
CREATE ROLE IF NOT EXISTS admin_role;
...
```

Предоставление полных прав администратору:

```
...
GRANT ALL PRIVILEGES ON shop.* TO admin_role;
...
```

Создание пользователя и назначение ему роли:

```
...
CREATE USER IF NOT EXISTS 'admin'@'localhost' IDENTIFIED BY
'PaSsWoRd';
```

```
GRANT admin_role TO 'admin'@'localhost';
...
```

Установка роли по умолчанию для пользователя:

```
...
SET DEFAULT ROLE admin_role TO 'admin'@'localhost';
...
```

Роль менеджера

Менеджеры в интернет-магазине выполняют ключевые задачи, связанные с управлением товарами, заказами и клиентами. Для них была создана роль manager_role, которая предоставляет полный доступ к таблицам, необходимым для выполнения их обязанностей, а также доступ к выполнению ряда хранимых процедур.

Создание роли manager_role и назначение привилегий для этой роли.

```
...
-- Создание роли manager_role
CREATE ROLE 'manager_role';

-- Назначение привилегий роли manager_role
GRANT SELECT, INSERT, UPDATE, DELETE ON shop.products TO
'manager_role';
GRANT SELECT, INSERT, UPDATE, DELETE ON shop.customers TO
'manager_role';
GRANT SELECT, INSERT, UPDATE, DELETE ON shop.orders TO 'manager_role';
GRANT SELECT, INSERT, UPDATE, DELETE ON shop.order_details TO
'manager_role';
GRANT SELECT ON shop.manufacturers TO 'manager_role';
GRANT SELECT ON shop.product_categories TO 'manager_role';
GRANT SELECT ON shop.reserves TO 'manager_role';
GRANT SELECT ON shop.warehouses TO 'manager_role';
...
```

Назначение доступа к хранимым процедурам:

```
...
-- назначение привилегий на выполнение хранимых процедур
GRANT EXECUTE ON PROCEDURE shop.get_products_by_manufacturer TO
'manager_role';
GRANT EXECUTE ON PROCEDURE shop.get_products_by_category TO
'manager_role';
GRANT EXECUTE ON PROCEDURE shop.update_product_price TO
'manager_role';
GRANT EXECUTE ON PROCEDURE shop.calculate_order_total TO
'manager_role';
GRANT EXECUTE ON PROCEDURE shop.make_order TO 'manager_role';
...
```

Создание пользователя manager и назначение ему роли manager_role и установка её по умолчанию:

```
...
-- создание пользователя manager и назначение ему роли manager_role
CREATE USER 'manager'@'localhost' IDENTIFIED BY 'PaSsWoRd';
GRANT 'manager_role' TO 'manager'@'localhost';

-- установка роли manager_role по умолчанию для пользователя manager
SET DEFAULT ROLE 'manager_role' FOR 'manager'@'localhost';
...
```

Роль покупателя

Покупатели интернет-магазина имеют ограниченный доступ к базе данных. Им необходимо просматривать доступные товары, категории и производителей, а также делать заказы. Для этих целей была создана роль customer_role.

Создание роли manager_role и назначение привилегий для этой роли.

```
...
-- создание роли customer_role
CREATE ROLE IF NOT EXISTS 'customer_role';

-- назначение привилегий роли customer_role
GRANT SELECT ON shop.products TO 'customer_role';
GRANT SELECT ON shop.product_categories TO 'customer_role';
GRANT SELECT ON shop.manufacturers TO 'customer_role';
...
```

Назначение доступа к хранимым процедурам:

```
...
-- назначение привилегий на выполнение хранимых процедур
GRANT EXECUTE ON PROCEDURE shop.make_order TO customer_role;
...
```

Создание пользователя manager и назначение ему роли customer_role и установка её по умолчанию:

```
-- создание пользователя customer и назначение ему роли customer_role
CREATE USER IF NOT EXISTS 'customer'@'localhost' IDENTIFIED BY 'SecurePassword';
GRANT 'customer_role' TO 'customer'@'localhost';

-- установка роли customer_role по умолчанию для пользователя customer
SET DEFAULT ROLE 'customer_role' FOR 'customer'@'localhost';
```

Заключение

В ходе выполнения курсового проекта на тему "Разработка базы данных для магазина компьютерных комплектующих" была достигнута поставленная цель - создание базы данных – для магазина компьютерных комплектующих. Для реализации этой цели были решены следующие задачи:

1. Проведен анализ предметной области, что позволило выявить ключевые требования к системе управления магазином компьютерных комплектующих.
2. Определены требования к разрабатываемой базе данных, что обеспечило четкое понимание необходимых функциональных возможностей и структурных элементов системы.
3. Спроектирована схема базы данных с учетом особенностей предметной области, что позволило создать логически целостную и эффективную структуру хранения данных.
4. Реализована база данных с использованием реляционной модели данных, что обеспечило надежное и гибкое хранение информации.
5. В ходе выполнения задачи по созданию функциональных объектов базы данных удалось значительно автоматизировать и оптимизировать работу с данными в системе управления магазином компьютерных комплектующих. Разработанные и интегрированные хранимые процедуры, представления, пользовательские функции и триггеры улучшили производительность и надежность системы.
6. Назначены права пользователям базы данных, что повысило уровень безопасности и обеспечило контроль доступа к данным.
7. Проверена работоспособность и корректность функционирования базы данных, что подтвердило выполнение всех функциональных требований и удовлетворение потребностей магазина.

Результаты выполненной работы показали, что разработанная база данных эффективно справляется с задачами управления складскими запасами и заказами,

обеспечивая высокую надежность и целостность данных. Реализованные хранимые процедуры и триггеры автоматизировали ключевые бизнес-процессы, что способствовало повышению эффективности работы магазина и качества обслуживания клиентов.

Таким образом, курсовая работа продемонстрировала значимость и актуальность разработки базы данных для магазина компьютерных комплектующих, а также подтвердила эффективность предложенных решений в улучшении управления бизнес-процессами и повышении конкурентоспособности магазина.

Список источников

1.База данных [Электронный ресурс] / Режим доступа: https://www.oracle.com/cis/database/what-is-database/

2.ERD диаграмма [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/op08/op08_10_lec

3.Иерархические - модель данных [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Иерархическая_модель_данных

4.Объектно - ориентированные базы данных [Электронный ресурс] / Режим доступа: https://clck.ru/3BGWZg

5.Пользовательские функции. [Электронный ресурс] / Режим доступа: https://www.dialektika.com/PDF/978-5-8459-1202-2/part.pdf

6.Представления [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/mdk1101/mdk1101_lab_23

7.Реляционные - модель данных [Электронный ресурс] / Режим доступа: https://clck.ru/3BGWrW

8.Связи между сущностями [Электронный ресурс] / Режим доступа: https://habr.com/ru/articles/488054/

9.Сетевая - модель данных [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Сетевая_модель_данных

10.Создание пользователей и назначение привилегий. [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/mdk1101/mdk1101_lab_25

11.СУБД Microsoft Access [Электронный ресурс] / Режим доступа: https://clck.ru/3BGXh5

12.СУБД MySQL [Электронный ресурс] / Режим доступа: https://clck.ru/3BGXVY

13.СУБД PostgreSQL [Электронный ресурс] / Режим доступа: https://blog.skillfactory.ru/glossary/postgresql/

14.Триггеры [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/mdk1101/mdk1101_lab_22

15.Хранимая процедура [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Хранимая_процедура

Приложение 1. QR-код на репозиторий.

https://github.com/GENADEVICH/course_work

Приложение 2. Хранимая процедура для поиска товара по категории.

```
DELIMITER //

CREATE PROCEDURE get_products_by_category(category_name VARCHAR(255))
BEGIN
    -- Проверяем, существует ли категория с указанным именем
    IF NOT EXISTS (SELECT name FROM product_categories WHERE name = category_name) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Категория не найдена';
    END IF;

    -- Если категория существует, выбираем продукты из таблицы
    SELECT p.id, p.name, p.description, p.price, p.quantity, c.name AS category_name
    FROM products p
    JOIN product_categories c ON p.product_categories_id = c.id
    WHERE c.name = category_name;
END //

DELIMITER;
```

Приложение 3. Хранимая процедура для поиска товара по производителю.

```
DELIMITER //

CREATE    PROCEDURE    get_products_by_manufacturer(manufacturer_name
VARCHAR(255))
BEGIN
    -- Проверяем, существует ли производитель с указанным именем
    IF  NOT  EXISTS  (SELECT  name  FROM  manufacturers  WHERE  name  =
manufacturer_name) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Производитель не
найден';
    END IF;

    -- Если производитель существует, выбираем продукты из таблицы
    SELECT p.id, p.name, p.description, p.price, p.quantity, m.name AS
manufacturer_name
    FROM products p
    JOIN manufacturers m ON p.manufacturers_id = m.id
    WHERE m.name = manufacturer_name;
END //

DELIMITER ;
```

Приложение 4. Хранимая процедура для обновления цены товара.

```
DELIMITER //

CREATE PROCEDURE update_product_price(
    IN product_id INT,
    IN new_price DECIMAL(10, 2)
)
BEGIN
    DECLARE current_price DECIMAL(10, 2);
    DECLARE price_difference DECIMAL(10, 2);
    DECLARE updated_rows INT;

    -- Получение текущей цену продукта
    SELECT price INTO current_price FROM products WHERE id =
product_id;

    -- Вычисление разницы между новой и текущей ценой
    SET price_difference = new_price - current_price;

    -- Обновление цены продукта
    UPDATE products SET price = new_price WHERE id = product_id;

    -- Получить количество обновленных строк
    SELECT ROW_COUNT() INTO updated_rows;

    -- Выводим сообщение о выполнении операции
    IF updated_rows > 0 THEN
        SELECT CONCAT('Цена продукта ', product_id, ' успешно
обновлена.') AS message;
    ELSE
        SELECT CONCAT('Ошибка при обновлении цены продукта ',
product_id) AS message;
    END IF;
END //
DELIMITER ;
```

Приложение 5. Хранимая процедура для обработки заказа.

```
DELIMITER ;;

DROP PROCEDURE IF EXISTS make_order;;

CREATE PROCEDURE make_order(
    IN customer_id INT,
    IN product_id INT,
    IN order_quantity INT,
    IN from_reserves BIT(1)
)
BEGIN
    DECLARE order_id INT;
    DECLARE product_price DECIMAL(10,2);

    -- Объявляем обработчик исключений
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Откатываем транзакцию в случае возникновения исключения
        ROLLBACK;
        -- Выдаем сообщение об ошибке
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Произошла ошибка
при обработке заказа';
    END;

    -- Начать транзакцию
    START TRANSACTION;

    -- Получаем цену продукта
    SELECT price INTO product_price FROM products WHERE id =
product_id;

    -- Если заказ из резерва
    IF from_reserves = 1 THEN
        -- Создаем новый заказ
```

```
        INSERT    INTO    orders    (customers_id,    order_datetime,
order_status)
        VALUES (customer_id, NOW(), 'processed');

        -- Получаем ID только что созданного заказа
        SET order_id = LAST_INSERT_ID();

        -- Добавляем детали заказа
        INSERT INTO order_details (orders_id, products_id, quantity,
from_reserves, total_price)
        VALUES    (order_id,    product_id,    order_quantity,    1,
order_quantity * product_price);
    ELSE
        -- Уменьшаем количество продуктов в таблице products
        UPDATE products
        SET quantity = quantity - order_quantity
        WHERE id = product_id;

        -- Создаем новый заказ
        INSERT    INTO    orders    (customers_id,    order_datetime,
order_status)
        VALUES (customer_id, NOW(), 'processed');

        -- Получаем ID только что созданного заказа
        SET order_id = LAST_INSERT_ID();

        -- Добавляем детали заказа с total_price
        INSERT INTO order_details (orders_id, products_id, quantity,
from_reserves, total_price)
        VALUES    (order_id,    product_id,    order_quantity,    0,
order_quantity * product_price);
    END IF;
    COMMIT;
END;;
DELIMITER ;
```

Приложение 6. Пользовательская функция.

```
DELIMITER $$

CREATE FUNCTION calculate_order_total(product_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE product_quantity INT;
    DECLARE reserve_quantity INT;
    DECLARE product_price DECIMAL(10,2);
    DECLARE total_price DECIMAL(10,2);

    -- Получаем количество и цену продукта из таблицы products
    SELECT quantity, price INTO product_quantity, product_price FROM
products WHERE id = product_id;

    -- Получаем количество товара из таблицы reserves
    SELECT SUM(quantity) INTO reserve_quantity FROM reserves WHERE
product_id = product_id;

    -- Если товар не найден в таблице reserves, устанавливаем его
количество как 0
    IF reserve_quantity IS NULL THEN
        SET reserve_quantity = 0;
    END IF;

    -- Вычисляем общее количество товаров и общую стоимость
    SET total_price = (product_quantity + reserve_quantity) *
product_price;

    RETURN total_price;
END $$

DELIMITER ;
```

Уважаемый пользователь!
Обращаем ваше внимание, что система Антиплагиус отвечает на вопрос, является тот или иной фрагмент текста заимствованным или нет. Ответ на вопрос, является ли заимствованный фрагмент именно плагиатом, а не законной цитатой, система оставляет на ваше усмотрение.

Отчет о проверке № 8872137

Дата выгрузки: 2024-06-14 19:53:48
Пользователь: kirikpo@bk.ru, ID: 8872137

Отчет предоставлен сервисом «Антиплагиат»
на сайте antiplagius.ru/

Информация о документе

№ документа: 8872137
Имя исходного файла: Колосов К - курсовой проект.pdf
Размер файла: 0.82 МБ
Размер текста: 39949
Слов в тексте: 5810
Число предложений: 731

Информация об отчете

Дата: 2024-06-14 19:53:48 - Последний готовый отчет
Оценка оригинальности: 90%
Заимствования: 10%

90.49% 9.51%

Источники:

Доля в тексте	Ссылка
30.40%	https://javastudy.ru/interview/javaee-sql-jdbc-interview/
24.90%	https://wiki2.org/ru/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%...
24.90%	https://руни.рф/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%D1%8F...
24.90%	https://ru.ruwiki.ru/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D...
24.30%	https://stydopedya.ru/2_45241_oharakterizuyte-ponyatie-indeksi-p...
23.60%	https://otherreferats.allbest.ru/programming/00199801_0.html
19.90%	https://selectel.ru/blog/tutorials/how-to-create-a-new-user-and-...
19.10%	https://sky.pro/wiki/sql/tipy-dannyh-v-sql/
14.20%	https://pandia.ru/text/86/064/31235.php
14.10%	https://codechick.io/tutorials/sql/sql-create-table
12.70%	https://habr.com/ru/articles/310328/
12.60%	https://infourok.ru/poyasnitelnaya-zapiska-k-kursovoj-rabote-raz...
12.30%	https://otherreferats.allbest.ru/programming/00624944_0.html
11.60%	https://pro-prof.com/forums/topic/sql-lesson-7
9.65%	https://www.bibliofond.ru/view.aspx?id=792117
9.40%	https://firstvds.ru/technology/sozdanie-polzovatelya-i-nastroyka...

Доля в тексте	Ссылка
8.00%	https://timeweb.cloud/tutorials/mysql/kak-sozdavat-tablitsy-v-my...
7.80%	https://selectel.ru/blog/tutorials/how-to-create-tables-in-mysql...
7.50%	https://nsportal.ru/npo-spo/informatika-i-vychislitelnaya-tekhni...
7.30%	https://appmaster.io/ru/blog/preimushchestva-sistem-upravleniia-...
6.90%	https://itonboard.ru/analysis/data_analysis/434-agregatnye_funkc...
6.90%	https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%...

Информация о документе:

Частное учреждение профессионального образования "Высшая школа предпринимательства" (ЧУПО "ВШП") КУРСОВОЙ ПРОЕКТ "Разработка базы данных для магазина компьютерных комплектующих" Выполнил: студент 3-го курса специальности 09.02.07 "Информационные системы и программирование" Колосов Кирилл Андреевич подпись: _____ Проверил: преподаватель дисциплины, преподаватель ЧУПО "ВШП", к.ф.н. Ткачев П.С. оценка: _____ подпись: _____ Тверь, 2024 г. Содержание Глава 1. Теоретический базис разработки БД 6 1.1 Анализ предметной области 6 1.2 Анализ связей между сущностями 7 1.3 Составить требования к разрабатываемой базе данных 9 1.4 Спроектировать схему базы данных 10 Глава 2. Таблицы и типы значений 13 2.1 Создание таблицы 13 2.2 Создание хранимых процедур 19 2.3 Создание триггеров 25 2.4 Создание типовых запросов 26 2.5 Создание представления таблиц 27 2.6 Создание пользовательской функции 28 2.7 Создание пользователей и назначение привилегий 31 Заключение 36 Список источников 38 Приложение 1. QR-код на репозиторий 39 Приложение 2. Хранимая процедура для поиска товара по категории 40 Приложение 3. Хранимая процедура для поиска товара по производителю 41 Приложение 4. Хранимая процедура для обновления цены товара 42 Приложение 5. Хранимая процедура для обработки заказа 43 Приложение 6. Пользовательская функция 45 Введение В современном мире информационных технологий спрос на компьютерные комплектующие постоянно растет. Компьютеры стали неотъемлемой частью повседневной жизни, используемой как в домашних условиях, так и в рабочих и образовательных целях. В связи с этим, эффективное управление и хранение данных о компьютерных комплектующих становится все более важным для бизнеса. Целью данного курсового проекта является разработка базы данных для магазина компьютерных комплектующих. Создание такой базы данных позволит оптимизировать процессы управления складом, контроль за товарным ассортиментом, а также повысить качество обслуживания клиентов. В настоящее время существует множество магазинов, специализирующихся на продаже компьютерных комплектующих. Однако, не все из них обладают эффективной системой управления данными, что может приводить к проблемам с учетом товаров, контролем за их наличием и заказами. Разработка базы данных, специально адаптированной для потребностей магазина компьютерных комплектующих, позволит улучшить эффективность работы предприятия, и увеличить его конкурентоспособность на рынке. В ходе выполнения курсового проекта будет рассмотрен процесс проектирования и реализации базы данных, включая определение структуры данных, выбор используемых технологий и инструментов, а также разработку интерфейса для взаимодействия с базой данных. Также будут рассмотрены вопросы безопасности данных и их резервного копирования для обеспечения надежности хранения информации. В конечном итоге, разработка базы данных для магазина компьютерных комплектующих позволит повысить эффективность управления предприятием, улучшить обслуживание клиентов и увеличить его конкурентоспособность на рынке информационных технологий. Разработанность В настоящее время базы данных для магазинов, компьютерных комплектующих делятся на два основных типа. Общие базы данных представляют собой системы управления складом и продажами, которые помогают в учете товаров, заказах и клиентах. Такие системы, как Retail Pro или 1С: Управление торговлей, обрабатывают большие объемы данных и помогают магазинам эффективно управлять своим бизнесом. Специализированные базы данных о компьютерных комплектующих содержат информацию о конкретных продуктах, таких как процессоры, видеокарты и материнские платы. Они включают технические характеристики товаров, цены, наличие на складе и отзывы клиентов. Разработка такой базы данных требует учета потребностей магазина, таких как управление запасами и обработка заказов, для эффективного управления бизнесом и обеспечения удовлетворения клиентов. Цель Целью данной курсовой работы является создание базы данных - для магазина компьютерных комплектующих. Задачи 1. Провести анализ предметной области. 2. Определить требования к разрабатываемой базе данных. 3. Спроектировать схему базы данных. 4. Реализовать базу данных. 5. Создание функциональных объектов БД. 6. Назначить права пользователям базы данных. 7. Проверить работоспособность и корректность функционирования базы данных. Объект исследования Объектом исследования являются базы данных. Предмет исследования Предметом исследования являются базы данных компьютерных комплектующих для магазинов. Методы исследования • Метод анализа - Изучение существующих баз данных компьютерных комплектующих, используемых в магазинах. • Метод сравнения - Сравнение различных баз данных компьютерных комплектующих между собой. • Метод абстрагирования - Создание общей схемы и принципов работы разрабатываемой базы данных для магазина компьютерных комплектующих. • Метод структурных аналогий - Создание базы данных компьютерных комплектующих на основе общих элементов, наблюдаемых в существующих

базах данных. Глава 1. Теоретический базис разработки БД 1.1 Анализ предметной области База данных - упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе, существует большое множество типов баз данных. [1] Базы данных можно разделить на несколько основных типов. Вот основные виды баз данных: Из основных видов можно назвать такие: • Иерархические - модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней. [3] • Объектно-ориентированные базы данных - это постреляционная модель БД. Она включает в себя таблицы, но не ограничивается ими. В таких БД информация хранится в виде набора объектов или программных элементов многократного использования. [4] • Реляционные - модель данных предусматривающая единственный способ предоставления информации - в виде набора двумерных таблиц и отношений между ними. [7] • Сетевая модель данных, логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных. [9] Сегодня существует множество СУБД, из популярных я могу выделить несколько основных: 1. MySQL - свободная реляционная система управления базами данных (СУБД). Под словом "свободная" подразумевается ее бесплатность, под "реляционная" - работа с базами данных, основанных на двумерных таблицах. Система выпущена в 1995 году, её разработка активно продолжается. [12] 2. PostgreSQL - это объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире. Имеет открытый исходный код и является альтернативой коммерческим базам данных. [13] 3. Microsoft Access - реляционная система управления базами данных (СУБД) корпорации Microsoft. Входит в состав пакета Microsoft Office. Имеет широкий спектр функций, включая связанные запросы, связь с внешними таблицами и базами данных. [11] 1.2 Анализ связей между сущностями. Связи между сущностями в базе данных MySQL являются основой для создания структурированных и взаимосвязанных данных. В реляционной модели данных связи реализуются с помощью ключей и ограничений, что обеспечивает целостность и согласованность данных. Типы связей Один-к-одному (1:1): Связь, при которой каждая запись одной таблицы соответствует одной записи другой таблицы. Один-ко-многим (1:n): Связь, при которой каждая запись одной таблицы соответствует нескольким записям другой таблицы. Многие-ко-многим (n:m): Связь, при которой каждая запись одной таблицы соответствует нескольким записям другой таблицы и наоборот. Реализация связей один-к-одному (1:1) Связь один-к-одному реализуется с использованием внешних ключей и уникальных ограничений. Таблицы, участвующие в такой связи, содержат соответствующие первичные и внешние ключи, обеспечивающие уникальность записей. Реализация связей один-ко-многим (1:n) Для реализации связи один-ко-многим одна из таблиц должна содержать внешний ключ, ссылающийся на первичный ключ другой таблицы. Это позволяет одной записи в "родительской" таблице соответствовать нескольким записям в "дочерней" таблице. Реализация связей многие-ко-многим (n:m) Связь многие-ко-многим требует создания промежуточной таблицы, которая содержит внешние ключи, ссылающиеся на обе связанные таблицы. Эта таблица служит для установления множественных связей между записями двух таблиц. [8] Внешние ключи и целостность данных Внешние ключи обеспечивают целостность данных, гарантируя, что значения в одной таблице имеют соответствующие значения в другой таблице. В MySQL можно настроить каскадные действия, которые автоматически обновляют или удаляют связанные записи при изменении или удалении данных в родительской таблице. Индексация и оптимизация связей Для улучшения производительности запросов, связанных с внешними ключами, рекомендуется создавать индексы на столбцах, участвующих в связях. Индексация помогает ускорить поиск и соединение таблиц, что особенно важно для больших объемов данных. 1.3 Составить требования к разрабатываемой базе данных 1. Общие требования 1.1. Надежность: База данных должна обеспечивать высокую надежность хранения и обработки данных. 1.2. Производительность: Необходимо обеспечить высокую производительность при выполнении операций добавления, обновления, удаления и поиска данных. 1.3. Масштабируемость: Система должна поддерживать возможность горизонтального и вертикального масштабирования для обработки увеличивающихся объемов данных. 1.4. Безопасность: Доступ к базе данных должен быть защищен от несанкционированного доступа. Должны быть реализованы механизмы аутентификации и авторизации пользователей. 2. Функциональные требования 2.1. Управление товарами: Хранение информации о товарах (название, описание, спецификация, категория, производитель, цена, количество). Возможность добавления, обновления и удаления информации о товарах. 2.2. Управление производителями: Хранение информации о производителях (название, страна, веб-сайт). Возможность связывать производителя с продуктом. 2.3. Управление категориями: Хранение информации о категориях (имя, описание). Возможность связывать категории с продуктом. 2.4. Управление запасами: Хранение информации о запасах (id товара, код склада, количество). Возможность связывать запасы с продуктом. 2.5. Управление складами: Хранение информации о складах (код склада, место расположение). Возможность связывать склады с запасами. 2.6. Управление клиентами: Хранение информации о клиентах (имя, фамилия, адрес доставки, email, номер телефона). 2.7. Управление заказами: Хранение информации о заказах (id пользователя, дата заказа, статус заказа,). Возможность связывать заказы с клиентами и товарами. Поддержка различных статусов заказа (новый, в обработке, отправлен, доставлен, отменен). 2.8. Управление деталями заказа: Хранение информации о деталях заказа (id заказа, id продукта, количества, со склада или нет). Возможность связывать детали заказа с заказами. 1.4 Спроектировать схему базы данных Диаграммы "сущность-связь" (или ERD) - неотъемлемая составляющая процесса моделирования любых систем, включая простые и сложные базы данных, однако применяемые в них фигуры и способы нотации могут ввести в заблуждение. Концептуальные модели данных дают общее представление о том, что должно входить в состав модели. Концептуальные ER-диаграммы можно брать за основу логических моделей данных. Их также можно использовать для создания отношений общности между разными ER-моделями, положив их в основу интеграции. [2] Для создания схемы базы данных я буду использовать функцию моделирования в MySQL Workbench. Рис. 1 На схеме (рис. 1) представлена готовая база данных, спроектированная на основе ранее сформулированных требований. Схема включает 8 таблиц, каждая из которых выполняет или способствует выполнению поставленных задач: 1. customers [клиенты] - таблица которая содержит в себе информацию о клиентах. 2. products [продукты] - таблица которая содержит в себе информацию о продуктах. 3. orders [заказы] -

таблица которая содержит в себе данные заказа. 4. orders_details [детали заказа] - таблица которая содержит в себе детали заказов. 5. manufacturers [производители] - таблица которая содержит в себе производителей. 6. product_categories [категории] - таблица которая содержит в себе информацию про категории. 7. reserves [запасы] - таблица которая содержит в себе информацию про запасы товаров. 8. warehouses [склады] - таблица которая содержит в себе информацию про склады. Такой набор таблиц нужен, чтобы удовлетворить все требования базы данных для магазина компьютерных комплектующих. Глава 2. Таблицы и типы значений 2.1 Создание таблицы Теперь приступим к созданию таблиц в соответствии с разработанной схемой. • Таблица customers содержит следующие столбцы: id, first_name, last_name, address, email, phone_number. • Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. Тип данных INT выбран для предотвращения ограничения диапазона значений. Primary Key служит уникальным идентификатором каждой записи в таблице, а Auto Increment автоматически присваивает порядковый номер каждой записи. Атрибут Not Null применяется, поскольку id не может быть пустым. • Для first_name используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null. • Для last_name используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null. • Для address используется тип данных TEXT. Тип данных TEXT выбран для хранения адреса, который может быть длиннее 255 символов. Также применяется атрибут Not Null. • Для email используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null. Также применяется атрибут Unique Key обеспечивает уникальность значений в столбце email, предотвращая дублирование адресов электронной почты. • Для phone_number используется тип данных VARCHAR (255) который позволяет хранить строки длиной до 255 символов. Также применяется атрибут Not Null. Также применяется атрибут Unique Key обеспечивает уникальность значений в столбце phone_number, предотвращая дублирование номеров телефонов. Рис. 2 таблица customers. • Таблица products содержит столбцы: id, name, description, price, specifications, quantity, manufactures_id, product_categories_id. • Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. • Для name используется тип данных VARCHAR (255). Также применяется атрибут Not Null. • Для description используется тип данных VARCHAR (255). • Для price используется тип данных DECIMAL (10,2) Тип данных DECIMAL (10,2) выбран чтобы обеспечить точность при хранении денежных значений, включая копейки. Также применяется атрибут Not Null. • Для specifications используется тип данных TEXT. Тип данных TEXT выбран для хранения подробных характеристик продукта. • Для quantity используется тип данных INT с атрибутами Not Null. • Для manufactures_id используется тип данных INT с атрибутами Not Null и требуется для создания связи (1:n) с таблицей manufactures. • Для product_categories_id используется тип данных INT с атрибутами Not Null и требуется для создания связи (1:n) с таблицей product_categories. Рис. 3 таблица products. • Таблица manufactures содержит столбцы: id, name, country, website. • Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. • Для name используется тип данных VARCHAR (255). Также применяется атрибут Not Null. • Для country используется тип данных CHAR (2) с атрибутом Not Null. Тип данных CHAR (2) выбран для хранения кода страны производителя (два символа). • Для website используется тип данных VARCHAR (255). Рис. 4 таблица manufactures. • Таблица product_categories содержит столбцы: id, name, description. • Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. Для name используется тип данных VARCHAR (255). Также применяется атрибут Not Null. • Для description используется тип данных TEXT. Рис. 5 таблица product_categories. • Таблица orders содержит столбцы: id, customers_id, order_datetime, order_status. • Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. • Для customers_id используется тип данных INT. Требуется для создания связи (1:n) с таблицей customers. • Для order_datetime используется тип данных DATETIME с атрибутами Not Null. Тип данных DATETIME выбран для хранения даты и времени размещения заказа. • Для order_status используется тип данных ENUM ('processed', 'created', 'completed', 'canceled') с атрибутами Not Null. Тип данных ENUM выбран для хранения статуса заказа, который может принимать одно из предопределенных значений. Рис. 6 таблица orders. Таблица orders_details содержит столбцы: id, orders_id, products_id, quantity, total_price, from_reserves. • Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. • Для orders_id используется тип данных INT с атрибутами Not Null. Требуется для создания связи (1:n) с таблицей orders. • Для products_id используется тип данных INT с атрибутами Not Null. Требуется для создания связи (1:n) с таблицей products. • Для quantity используется тип данных INT UNSIGNED с атрибутами Not Null. Тип данных INT UNSIGNED выбран для хранения количества, так как это значение не может быть отрицательным. • Для total_price используется тип данных DECIMAL (10,2) который не может быть NULL • Для from_reserves используется тип данных BIT (1). Тип данных BIT (1) выбран для хранения булевого значения, указывающего, было ли взято из резерва. Рис. 7 таблица orders_details. • Таблица reserves содержит столбцы: id, products_id, warehouse_code, quantity. • Для id используется тип данных INT с атрибутами Not Null, Auto Increment и Primary Key. • Для products_id используется тип данных INT с атрибутами Not Null. Требуется для создания связи (1:n) с таблицей products. Для warehouse_code используется тип данных CHAR (4) с атрибутами Not Null. Тип данных CHAR (4) выбран для хранения кода склада, на котором зарезервирован продукт. Требуется для создания связи (1: n) с таблицей warehouses. • Для quantity используется тип данных INT. Рис. 8 таблица reserves. • Таблица warehouses содержит столбцы: code, location_name. • Для code используется тип данных CHAR (4) с атрибутом Not Null. Тип данных CHAR (4) выбран для хранения кода склада, который является уникальным идентификатором id. • Для location_name используется тип данных VARCHAR (255). Также применяется атрибут Not Null. Рис. 9 таблица warehouses. 2.2 Создание хранимых процедур. Хранимая процедура (англ. Stored procedure) - объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления

процессом исполнения. [15] В базе данных предусмотрены две хранимые процедуры с обработкой исключений, одна хранимая процедура, использующая три локальные переменные с заданными типами данных, и одна хранимая процедура, реализующая транзакцию. Первая хранимая процедура используется как фильтр для поиска товара по категории. (см. Приложение 1) Этот блок проверяет, существует ли категория с указанным именем. Если нет, то генерируется ошибка 'Категория не найдена'. ... -- Проверяем, существует ли категория с указанным именем IF NOT EXISTS (SELECT name FROM product_categories WHERE name = category_name) THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Категория не найдена'; END IF; ... В этом блоке кода идет выборка товаров из таблицы. ... -- Если категория существует, выбираем продукты из таблицы SELECT p.id, p.name, p.description, p.price, p.quantity, c.name AS category_name FROM products p JOIN product_categories c ON p.product_categories_id = c.id WHERE c.name = category_name; ... Вторая хранимая процедура используется как фильтр для поиска товара по производителю. (см. Приложение 2) Этот блок проверяет, существует ли производитель с указанным именем. Если нет, то генерируется ошибка 'Производитель не найден'. ... -- Проверяем, существует ли производитель с указанным именем IF NOT EXISTS (SELECT name FROM manufacturers WHERE name = manufacturer_name) THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Производитель не найден'; END IF; ... В этом блоке кода идет выборка товаров из таблицы. ... SELECT p.id, p.name, p.description, p.price, p.quantity, m.name AS manufacturer_name FROM products p JOIN manufacturers m ON p.manufacturers_id = m.id WHERE m.name = manufacturer_name; ... Третья хранимая используется для обновления цены товара. (см. Приложение 3) В этом блоке создается хранимая процедура update_product_price с тремя локальными переменными с заданном типом данных. ... CREATE PROCEDURE update_product_price(IN product_id INT, IN new_price DECIMAL(10, 2)) BEGIN DECLARE current_price DECIMAL(10, 2); DECLARE price_difference DECIMAL(10, 2); DECLARE updated_rows INT; ... Эта строка выполняет запрос, который получает текущую цену продукта из таблицы products и сохраняет ее в переменной current_price. ... -- Получение текущей цену продукта SELECT price INTO current_price FROM products WHERE id = product_id; ... Эта строка вычисляет разницу между новой и текущей ценой и сохраняет результат в переменной price_difference. ... -- Вычисление разницы между новой и текущей ценой SET price_difference = new_price - current_price; ... Эта строка обновляет цену продукта в таблице products до новой цены, указанной в параметре new_price, для продукта с идентификатором product_id. ... -- Обновление цены продукта UPDATE products SET price = new_price WHERE id = product_id; ... Эта строка получает количество строк, затронутых последней командой UPDATE, и сохраняет это значение в переменной updated_rows. ... -- Получить количество обновленных строк SELECT ROW_COUNT() INTO updated_rows; ... Этот блок проверяет, были ли обновлены строки: Если updated_rows больше 0, выводится сообщение о том, что цена продукта успешно обновлена. Если updated_rows равно 0, выводится сообщение об ошибке. ... -- Выводим сообщение о выполнении операции IF updated_rows > 0 THEN SELECT CONCAT('Цена продукта ', product_id, ' успешно обновлена.') AS message; ELSE SELECT CONCAT('Ошибка при обновлении цены продукта ', product_id) AS message; END IF; ... Четвертая хранимая процедура make_order предназначена для обработки заказов в системе. Она автоматизирует процесс создания заказа и включает в себя логику для работы с различными сценариями. (см. Приложение 4) В этом блоке кода создается хранимая процедура с названием make_order которая принимает четыре входных параметра customer_id типа INT, product_id типа INT, order_quantity типа INT, from_reserves типа BIT (1). Также здесь объявляется локальная переменная order_id типа INT, которая будет использоваться для хранения идентификатора созданного заказа. ... CREATE PROCEDURE make_order(IN customer_id INT, IN product_id INT, IN order_quantity INT, IN from_reserves BIT(1)) BEGIN DECLARE order_id INT; ... Здесь объявляется обработчик исключений: ... -- Объявляем обработчик исключений DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN -- Откатываем транзакцию в случае возникновения исключения ROLLBACK; -- Выдаем сообщение об ошибке SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Произошла ошибка при обработке заказа'; END; ... В этом блоке кода начинается транзакция и создается новый заказ со склада: ... -- Начать транзакцию START TRANSACTION; Здесь мы получаем id только что созданного заказа: ... -- Получаем ID только что созданного заказа SET order_id = LAST_INSERT_ID(); ... В этом блоке кода добавляется информация о деталях заказа в таблицу order_details и уменьшаем количество товара в таблице products. Создаем новый заказ и получаем id только что созданного заказа, добавляем информацию о деталях заказа и завершаем транзакцию: ... -- Создаем новый заказ INSERT INTO orders (customers_id, order_datetime, order_status) VALUES (customer_id, NOW(), 'processed'); -- Получаем ID только что созданного заказа SET order_id = LAST_INSERT_ID(); -- Добавляем детали заказа INSERT INTO order_details (orders_id, products_id, quantity, from_reserves) VALUES (order_id, product_id, order_quantity, 0); END IF; -- Завершаем транзакцию COMMIT; END // 2.3 Создание триггеров Триггеры в MySQL 8 представляют собой специальные объекты базы данных, которые автоматически выполняются при определенных событиях или действиях с данными. Они позволяют программистам определить пользовательские действия, которые должны быть выполнены перед или после выполнения операции в базе данных. Иными словами, это определяемая пользователем SQL-команда, которая автоматически вызывается во время операций INSERT, DELETE или UPDATE. Код триггера связан с таблицей и уничтожается после удаления таблицы. Вы можете определить время действия триггера и указать, когда его нужно активировать - до или после определенного события базы данных. Поддержка триггеров в MySQL началась с версии 5.0.2. [14] В данной базе данных триггеры используются для дополнение хранимой процедуры с транзакцией make_order Триггер order_from_reserves_trigger автоматически обновляет таблицу запасов (reserves) при каждом добавлении новой записи в таблицу деталей заказа (order_details). Если товар берется со склада, триггер автоматически уменьшает количество доступных товаров на складе на количество, указанное в заказе. DELIMITER // CREATE TRIGGER order_from_reserves_trigger AFTER INSERT ON order_details FOR EACH ROW BEGIN IF NEW.from_reserves = true THEN UPDATE reserves SET quantity = quantity - NEW.quantity WHERE products_id = NEW.products_id; END IF; END; // DELIMITER ; 2.4 Создание типовых запросов Типовые запросы в MySQL 8 являются мощным инструментом, который помогает улучшить производительность, безопасность и удобство разработки. Они широко применяются в реальных проектах и являются важной частью современной работы с базами данных. 1. Получить среднюю стоимость заказа по каждому клиенту: SELECT c.last_name AS фамилия, c.first_name AS имя, ROUND(AVG(od.total_price)) AS средняя_стоимость_заказа FROM customers c LEFT JOIN orders o ON c.id = o.customers_id LEFT JOIN order_details od ON o.id =

od.orders_id GROUP BY c.id, c.last_name, c.first_name; 2. Выяснить, в каких категориях продуктов есть товары с низкими остатками (менее 10): 3. Найти топ-5 клиентов, сделавших наибольшее количество заказов: SELECT c.first_name, c.last_name, общее_количество_заказов FROM customers c JOIN orders o ON c.id = o.customers_id GROUP BY c.id ORDER BY общее_количество_заказов DESC LIMIT 5; COUNT(o.id) AS 4. Узнать среднюю цену продуктов в каждой категории: 5. Получить общее количество продуктов в каждом складе: 2.5 Создание представления таблиц Представления (Views) в MySQL - это виртуальные таблицы, которые основаны на результатах выполнения запроса SELECT. Они представляют собой логические структуры данных, которые могут быть использованы для упрощения и оптимизации запросов к базе данных. Основная цель использования представлений в MySQL - это абстрагирование сложных запросов и создание более простых и понятных интерфейсов для работы с данными. Вместо того, чтобы каждый раз писать сложные запросы, можно создать представление, которое будет содержать уже готовый запрос. После этого можно обращаться к представлению как к обычной таблице, выполнять SELECTзапросы к нему и получать результаты. [6] Эта виртуальная таблица (представление), показывает все заказы, соединяя данные о заказах, клиентах, деталях заказов и продуктах в одну таблицу. В результате получаем удобный способ видеть информацию о каждом заказе: когда он был сделан, кем, какие товары были заказаны, сколько, по какой цене и со склада ли они. CREATE VIEW order_details_view AS SELECT o.id AS order_id, o.order_datetime, c.first_name, c.last_name, p.name AS product_name, od.quantity, od.from_reserves, p.price FROM orders o JOIN customers c ON o.customers_id = c.id JOIN order_details od ON o.id = od.orders_id JOIN products p ON od.products_id = p.id; 2.6 Создание пользовательской функции Пользовательские функции во многом напоминают хранимые процедуры и представляют упорядоченное множество операторов T-SQL, которые заранее оптимизированы, откомпилированы и могут быть вызваны для выполнения работы в виде единого модуля. Основное различие между пользовательскими функциями и хранимыми процедурами состоит в том, как в них осуществляется возврат полученных результатов. А в связи с тем, что для обеспечения, предусмотренного в них способа возврата значений в пользовательских функциях должны осуществляться немного другие действия, к их синтаксической структуре предъявляются более жесткие требования по сравнению с хранимыми процедурами. [5] Пользовательская функция calculate_order_total вычисляет общую стоимость всех товаров определенного продукта, включая как основное количество, так и количество со склада, и умножает их суммарное количество на цену продукта. Она делает это путем получения данных из таблиц products и reserves, обработки этих данных и возврата рассчитанного значения. (см. Приложение 4) В этом блоке кода создается пользовательская функция с названием calculate_order_total и переменные для хранения количества продуктов, количества товаров со склада, цены продукта и общей стоимости. ... CREATE FUNCTION calculate_order_total(product_id INT) RETURNS DECIMAL(10,2) DETERMINISTIC BEGIN DECLARE product_quantity INT; DECLARE reserve_quantity INT; DECLARE product_price DECIMAL(10,2); DECLARE total_price DECIMAL(10,2); ... Запрашивает количество и цену продукта по заданному product_id и сохраняет их в переменные product_quantity и product_price. ... -- Получаем количество и цену продукта из таблицы products SELECT quantity, price INTO product_quantity, product_price FROM products WHERE id = product_id; ... Запрашивает общее количество данного продукта из таблицы reserves и сохраняет его в переменную reserve_quantity. Использует SUM(quantity), чтобы получить суммарное количество товаров со склада для указанного продукта. ... -- Получаем количество товара из таблицы reserves SELECT SUM(quantity) INTO reserve_quantity FROM reserves WHERE product_id = product_id; ... Если reserve_quantity равен NULL (то есть товара нет на складе), устанавливает его значение как 0. 2.7 Создание пользователей и назначение привилегий В информационных системах безопасность данных является одним из важнейших аспектов. При работе с базой данных MySQL, рекомендуется следовать принципу наименьших привилегий и создавать пользователей с ограниченным доступом к базе данных. Администратор MySQL должен создавать такие учетные записи для "обычных" пользователей и определять их права доступа. Это позволяет предоставлять пользователям только те привилегии, которые им действительно необходимы для работы, и обеспечивать безопасность данных. Кроме того, проведение аудита полномочий и корректировка их при необходимости также являются важными шагами для обеспечения безопасности базы данных. [10] В современных базах данных часто используется модель управления доступом на основе ролей, где роли представляют собой совокупности прав и привилегий, назначаемые пользователям для выполнения определенных задач. Роль admin_role представляет собой одну из таких ролей, которая часто используется для предоставления полных или расширенных прав доступа к базе данных. Роль администратора Создание роли admin_role: ... CREATE ROLE IF NOT EXISTS admin_role; ... Предоставление полных прав администратору: ... GRANT ALL PRIVILEGES ON shop.* TO admin_role; ... Создание пользователя и назначение ему роли: ... CREATE USER IF NOT EXISTS 'admin'@'localhost' IDENTIFIED BY 'PaSsWoRd'; GRANT admin_role TO 'admin'@'localhost'; ... Установка роли по умолчанию для пользователя: ... SET DEFAULT ROLE admin_role TO 'admin'@'localhost'; ... Перезагрузка привилегий: ... FLUSH PRIVILEGES; ... Роль менеджера Менеджеры в интернет-магазине выполняют ключевые задачи, связанные с управлением товарами, заказами и клиентами. Для них была создана роль manager_role, которая предоставляет полный доступ к таблицам, необходимым для выполнения их обязанностей, а также доступ к выполнению ряда хранимых процедур. Создание роли manager_role и назначение привилегий для этой роли. ... -- Создание роли manager_role CREATE ROLE 'manager_role'; -- Назначение привилегий роли manager_role GRANT SELECT, INSERT, UPDATE, DELETE ON shop.products TO 'manager_role'; GRANT SELECT, INSERT, UPDATE, DELETE ON shop.customers TO 'manager_role'; GRANT SELECT, INSERT, UPDATE, DELETE ON shop.orders TO 'manager_role'; GRANT SELECT, INSERT, UPDATE, DELETE ON shop.order_details TO 'manager_role'; GRANT SELECT ON shop.manufacturers TO 'manager_role'; GRANT SELECT ON shop.product_categories TO 'manager_role'; GRANT SELECT ON shop.reserves TO 'manager_role'; GRANT SELECT ON shop.warehouses TO 'manager_role'; ... Назначение доступа к хранимым процедурам: ... -- назначение привилегий на выполнение хранимых процедур GRANT EXECUTE ON PROCEDURE shop.get_products_by_manufacturer TO 'manager_role'; GRANT EXECUTE ON PROCEDURE shop.get_products_by_category TO 'manager_role'; GRANT EXECUTE ON PROCEDURE shop.update_product_price TO 'manager_role'; GRANT EXECUTE ON PROCEDURE shop.calculate_order_total TO 'manager_role'; GRANT EXECUTE ON PROCEDURE shop.make_order TO 'manager_role'; ... Создание пользователя manager и

назначение ему роли manager_role и установка её по умолчанию: ... -- создание пользователя manager и назначение ему роли manager_role CREATE USER 'manager'@'localhost' IDENTIFIED BY 'PaSsWoRd'; GRANT 'manager_role' TO 'manager'@'localhost'; -- установка роли manager_role по умолчанию для пользователя manager SET DEFAULT ROLE 'manager_role' FOR 'manager'@'localhost'; ... Перезагрузка привилегий: ... FLUSH PRIVILEGES; ... Роль покупателя Покупатели интернет-магазина имеют ограниченный доступ к базе данных. Им необходимо просматривать доступные товары, категории и производителей, а также делать заказы. Для этих целей была создана роль customer_role. Создание роли manager_role и назначение привилегий для этой роли. ... -- создание роли customer_role CREATE ROLE IF NOT EXISTS 'customer_role'; -- назначение привилегий роли customer_role GRANT SELECT ON shop.products TO 'customer_role'; GRANT SELECT ON shop.product_categories TO 'customer_role'; GRANT SELECT ON shop.manufacturers TO 'customer_role'; ... Назначение доступа к хранимым процедурам: ... -- назначение привилегий на выполнение хранимых процедур GRANT EXECUTE ON PROCEDURE shop.make_order TO customer_role; ... Создание пользователя manager и назначение ему роли customer_role и установка её по умолчанию: -- создание пользователя customer и назначение ему роли customer_role CREATE USER IF NOT EXISTS 'customer'@'localhost' IDENTIFIED BY 'SecurePassword'; GRANT 'customer_role' TO 'customer'@'localhost'; -- установка роли customer_role по умолчанию для пользователя customer SET DEFAULT ROLE 'customer_role' FOR 'customer'@'localhost'; Перезагрузка привилегий: ... FLUSH PRIVILEGES; ... Заключение В ходе выполнения курсового проекта на тему "Разработка базы данных для магазина компьютерных комплектующих" была достигнута поставленная цель - создание базы данных - для магазина компьютерных комплектующих. Для реализации этой цели были решены следующие задачи: 1. Проведен анализ предметной области, что позволило выявить ключевые требования к системе управления магазином компьютерных комплектующих. 2. Определены требования к разрабатываемой базе данных, что обеспечило четкое понимание необходимых функциональных возможностей и структурных элементов системы. 3. Спроектирована схема базы данных с учетом особенностей предметной области, что позволило создать логически целостную и эффективную структуру хранения данных. 4. Реализована база данных с использованием реляционной модели данных, что обеспечило надежное и гибкое хранение информации. 5. В ходе выполнения задачи по созданию функциональных объектов базы данных удалось значительно автоматизировать и оптимизировать работу с данными в системе управления магазином компьютерных комплектующих. Разработанные и интегрированные хранимые процедуры, представления, пользовательские функции и триггеры улучшили производительность и надежность системы. 6. Назначены права пользователям базы данных, что повысило уровень безопасности и обеспечило контроль доступа к данным. 7. Проверена работоспособность и корректность функционирования базы данных, что подтвердило выполнение всех функциональных требований и удовлетворение потребностей магазина. Результаты выполненной работы показали, что разработанная база данных эффективно справляется с задачами управления складскими запасами и заказами, обеспечивая высокую надежность и целостность данных. Реализованные хранимые процедуры и триггеры автоматизировали ключевые бизнес-процессы, что способствовало повышению эффективности работы магазина и качества обслуживания клиентов. Таким образом, курсовая работа продемонстрировала значимость и актуальность разработки базы данных для магазина компьютерных комплектующих, а также подтвердила эффективность предложенных решений в улучшении управления бизнес-процессами и повышении конкурентоспособности магазина. Список источников 1.База данных [Электронный ресурс] / Режим доступа: https://www.oracle.com/cis/database/what-is-database/ 2.ERD диаграмма [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/op08/op08_10_lec 3.Иерархические - модель данных [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Иерархическая_модель_данных 4.Объектно - ориентированные базы данных [Электронный ресурс] / Режим доступа: https://clck.ru/3BGWZg 5.Пользовательские функции. [Электронный ресурс] / Режим доступа: https://www.dialektika.com/PDF/978-5-8459-1202-2/part.pdf 6.Представления [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/mdk1101/mdk1101_lab_23 7.Реляционные - модель данных [Электронный ресурс] / Режим доступа: https://clck.ru/3BGWrW 8.Связи между сущностями [Электронный ресурс] / Режим доступа: https://habr.com/ru/articles/488054/ 9.Сетевая - модель данных [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Сетевая_модель_данных 10.Создание пользователей и назначение привилегий. [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/mdk1101/mdk1101_lab_25 11.СУБД Microsoft Access [Электронный ресурс] / Режим доступа: https://clck.ru/3BGXh5 12.СУБД MySQL [Электронный ресурс] / Режим доступа: https://clck.ru/3BGXVY 13.СУБД PostgreSQL [Электронный ресурс] / Режим доступа: https://blog.skillfactory.ru/glossary/postgresql/ 14.Триггеры [Электронный ресурс] / Режим доступа: https://it.vshp.online/#/pages/mdk1101/mdk1101_lab_22 15.Хранимая процедура [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Хранимая_процедура Приложение 1. QR-код на репозиторий. https://github.com/GENADEVICH/course_work Приложение 2. Хранимая процедура для поиска товара по категории. DELIMITER // CREATE PROCEDURE get_products_by_category(category_name VARCHAR(255)) BEGIN -- Проверяем, существует ли категория с указанным именем IF NOT EXISTS (SELECT name FROM product_categories WHERE name = category_name) THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Категория не найдена'; END IF; -- Если категория существует, выбираем продукты из таблицы SELECT p.id, p.name, p.description, p.price, p.quantity, c.name AS category_name FROM products p JOIN product_categories c ON p.product_categories_id = c.id WHERE c.name = category_name; END // DELIMITER; Приложение 3. Хранимая процедура для поиска товара по производителю. DELIMITER // CREATE PROCEDURE get_products_by_manufacturer(manufacturer_name VARCHAR(255)) BEGIN -- Проверяем, существует ли производитель с указанным именем IF NOT EXISTS (SELECT name FROM manufacturers WHERE name = manufacturer_name) THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Производитель не найден'; END IF; -- Если производитель существует, выбираем продукты из таблицы SELECT p.id, p.name, p.description, p.price, p.quantity, m.name AS manufacturer_name FROM products p JOIN manufacturers m ON p.manufacturers_id = m.id WHERE m.name = manufacturer_name; END // DELIMITER ; DELIMITER // CREATE PROCEDURE update_product_price(IN product_id INT, IN new_price DECIMAL(10, 2)) BEGIN DECLARE current_price DECIMAL(10, 2); DECLARE price_difference DECIMAL(10, 2); DECLARE updated_rows INT; -- Получение текущей цену

продукта SELECT price INTO current_price FROM products WHERE id = product_id; -- Вычисление разницы между новой и текущей ценой SET price_difference = new_price - current_price; -- Обновление цены продукта UPDATE products SET price = new_price WHERE id = product_id; -- Получить количество обновленных строк SELECT ROW_COUNT() INTO updated_rows; -- Выводим сообщение о выполнении операции IF updated_rows > 0 THEN SELECT CONCAT('Цена продукта ', product_id, ' успешно обновлена.') AS message; ELSE SELECT CONCAT('Ошибка при обновлении цены продукта ', product_id) AS message; END IF; END // DELIMITER ; Приложение 4. Хранимая процедура для обновления цены товара. Приложение 5. Хранимая процедура для обработки заказа. DELIMITER ;; DROP PROCEDURE IF EXISTS make_order;; CREATE PROCEDURE make_order(IN customer_id INT, IN product_id INT, IN order_quantity INT, IN from_reserves BIT(1)) BEGIN DECLARE order_id INT; DECLARE product_price DECIMAL(10,2); -- Объявляем обработчик исключений DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN -- Откатываем транзакцию в случае возникновения исключения ROLLBACK; -- Выдаем сообщение об ошибке SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Произошла ошибка при обработке заказа'; END; -- Начать транзакцию START TRANSACTION; -- Получаем цену продукта SELECT price INTO product_price FROM products WHERE id = product_id; -- Если заказ из резерва IF from_reserves = 1 THEN -- Создаем новый заказ INSERT INTO orders (customers_id, order_datetime, order_status) VALUES (customer_id, NOW(), 'processed'); -- Получаем ID только что созданного заказа SET order_id = LAST_INSERT_ID(); -- Добавляем детали заказа INSERT INTO order_details (orders_id, products_id, quantity, from_reserves, total_price) VALUES (order_id, product_id, order_quantity, 1, order_quantity * product_price); ELSE -- Уменьшаем количество продуктов в таблице products UPDATE products SET quantity = quantity - order_quantity WHERE id = product_id; -- Создаем новый заказ INSERT INTO orders (customers_id, order_datetime, order_status) VALUES (customer_id, NOW(), 'processed'); -- Получаем ID только что созданного заказа SET order_id = LAST_INSERT_ID(); -- Добавляем детали заказа с total_price INSERT INTO order_details (orders_id, products_id, quantity, from_reserves, total_price) VALUES (order_id, product_id, order_quantity, 0, order_quantity * product_price); END IF; COMMIT; END;; DELIMITER ; Приложение 6. Пользовательская функция. DELIMITER $$ CREATE FUNCTION calculate_order_total(product_id INT) RETURNS DECIMAL(10,2) DETERMINISTIC BEGIN DECLARE product_quantity INT; DECLARE reserve_quantity INT; DECLARE product_price DECIMAL(10,2); DECLARE total_price DECIMAL(10,2); -- Получаем количество и цену продукта из таблицы products SELECT quantity, price INTO product_quantity, product_price FROM products WHERE id = product_id; -- Получаем количество товара из таблицы reserves SELECT SUM(quantity) INTO reserve_quantity FROM reserves WHERE product_id = product_id; -- Если товар не найден в таблице reserves, устанавливаем его количество как 0 IF reserve_quantity IS NULL THEN SET reserve_quantity = 0; END IF; -- Вычисляем общее количество товаров и общую стоимость SET total_price = (product_quantity + reserve_quantity) * product_price; RETURN total_price; END $$ DELIMITER ; 2 2 □ □ 2 2 □ 2 2 2 2