



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

КУРСОВОЙ ПРОЕКТ

**«Разработка базы данных для интернет магазина по продаже строительных
материалов»**

Выполнил:
студент 3-го курса специальности
09.02.07 «Информационные системы и
программирование»
Плотников Георгий Вячеславович
подпись: _____

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

Содержание

Введение	3
Глава 1: Теоретические основы интернет-магазинов и их актуальность.....	5
1.1. Понятие интернет-магазина и его роль в современной торговле	5
1.1.1. Анализ интернет-магазина по продаже стройматериалов на примере Leroy Merlin.....	6
1.2. Анализ существующих решений и технологий	7
1.3. Требования к базе данных.....	9
1.4. Основные элементы базы данных интернет-магазина.....	10
Глава 2: Реализация базы данных для интернет-магазина по продаже стройматериалов	12
2.1. Создание и структура базы данных	12
2.1. Тестирование базы данных	14
2.2. Создание ролей и привилегий	16
2.3. Создание типовых запросов.....	18
2.4. Управление транзакциями в SQL	20
2.5. Локальные переменные в хранимых процедурах.....	22
2.6. Условия	23
2.7. Хранимые процедуры в базе данных	24
2.8. Представления в базе данных	25
2.9. Пользовательская функция в базе данных	27
2.10 Триггеры	28
2.11. Обработчик исключений	29
2.12. Создание резервных копий базы данных	30
Заключение.....	33
Источники:	35
Приложение 1. Пример транзакции для создания заказа.....	36
Приложение 2. Процедура для добавления нового товара с проверкой роли пользователя	36
Приложение 3. Хранимая процедура для добавления нового пользователя с использованием хеширования пароля:	37

Введение

В современном мире интернет-магазины занимают важное место в сфере торговли. Они предоставляют удобный и быстрый способ покупки товаров, значительно расширяя возможности выбора и экономя время потребителей. Особенно актуально это в условиях ускоряющегося ритма жизни, когда удобство и оперативность обслуживания становятся ключевыми факторами конкурентоспособности. Интернет-магазины по продаже строительных материалов не являются исключением. С ростом популярности онлайн-покупок в этой сфере возникает необходимость в создании эффективных информационных систем, которые обеспечат надежное хранение данных, удобное управление заказами и высокую степень автоматизации процессов.

Актуальность выбранной темы определяется несколькими факторами. Во-первых, рост числа пользователей, предпочитающих онлайн-покупки, требует создания специализированных баз данных, способных обрабатывать большие объемы информации быстро и безошибочно. Во-вторых, конкурентная среда обуславливает необходимость внедрения передовых технологий и решений для повышения качества обслуживания клиентов. В-третьих, базы данных являются ключевым элементом информационных систем, обеспечивая надежность и безопасность хранения данных, что особенно важно в условиях возрастающих киберугроз.

Цель:

Цель данной курсовой работы состоит в разработке базы данных для интернет-магазина по продаже строительных материалов, которая обеспечит эффективное управление информацией, связанной с товарами, заказами и клиентами.

Задачи:

1. Проведение анализа теоретических основ интернет-магазинов.

2. Обеспечение безопасности и надежности.
3. Анализ существующих решений и технологий.
4. Проектирование структуры базы данных.
5. Реализация базы данных.

Объектом исследования в данной работе является процесс управления данными в интернет-магазине по продаже строительных материалов. В рамках исследования будут рассмотрены методы проектирования и реализации баз данных, а также способы обеспечения их надежности и безопасности.

Методологической основой работы являются общепринятые методы системного анализа, проектирования баз данных и программирования. В ходе работы будут использованы методы структурного анализа, методы проектирования отношений и нормализации данных, а также технологии SQL для создания и управления базой данных.

Таким образом, данная курсовая работа направлена на разработку и внедрение эффективной информационной системы для интернет-магазина по продаже строительных материалов, что позволит оптимизировать процессы управления, улучшить качество обслуживания клиентов и обеспечить надежное хранение данных.

Глава 1: Теоретические основы интернет-магазинов и их актуальность

1.1. Понятие интернет-магазина и его роль в современной торговле

Интернет-магазин представляет собой веб-сайт или мобильное приложение, предназначенное для продажи товаров и услуг через интернет. Основные функции интернет-магазина включают отображение ассортимента товаров, управление заказами, учет складских запасов и обработку платежей. Популярность интернет-магазинов объясняется их удобством для потребителей, экономией времени и возможностью сравнения цен и характеристик товаров без необходимости посещения физических магазинов.

Плюсы интернет-магазинов:

1. Выход на глобальный рынок:

Малый бизнес не может выйти на мировой рынок обладая лишь обычным магазином. Выйти можно только через Интернет. Более 70% всего человечества имеет доступ к Интернету, более 40% из них ежедневно совершают покупки в онлайн-магазинах. Таким образом, с помощью интернет-магазина можно получить доступ к огромной глобальной аудитории клиентов.[1]

2. Высокая рентабельность:

Сегодня люди совершают больше покупок онлайн, чем оффлайн. С каждым годом количество покупок в Интернет-магазинах растёт, а вместе с ним и прибыль онлайн бизнеса. Именно поэтому розничная онлайн торговля более рентабельна, чем розничная оффлайн торговля. [1]

3. Предоставление персонализированных товаров клиентам:

Потенциальные клиенты используют Интернет для того, чтобы что-то купить или принять решение о том, что они хотят купить и где. Размещение вашей продукции в онлайн магазине увеличит шансы на продажу товара, поскольку

у вас будет возможность показать клиентам, почему им стоит купить товар именно у вас.

Основным преимуществом интернет-магазина является его удобство для потребителя. Большинство людей ведет загруженную жизнь, и когда им нужно что-то купить, они, как правило, предпочитают делать это через интернет. Это экономит их время, усилия и даже деньги. [1]

4. Интернет-магазин — гибкий инструмент продаж:

Собственный онлайн-магазин предоставляет огромное пространство для экспериментов. Вы в любое время можете что либо в нем изменить, добавить баннеры или призывы к действию, запустить акцию или email-рассылку. Любые изменения внесенные в сайт могут быть проанализированы с точки зрения эффективности. [1]

Для успешной работы интернет-магазина важным аспектом является эффективное управление данными. Здесь на первый план выходит разработка базы данных, обеспечивающей надежное хранение, управление и доступ к информации о товарах, заказах и клиентах.

1.1.1. Анализ интернет-магазина по продаже стройматериалов на примере Leroy Merlin

Для разработки базы данных интернет-магазина строительных материалов необходимо проанализировать уже существующие решения и их функциональность. Это поможет выявить ключевые особенности и потребности пользователей, которые необходимо учесть при проектировании системы.

Leroy Merlin — один из крупнейших интернет-магазинов строительных материалов. Он предлагает широкий ассортимент товаров, включая строительные и отделочные материалы, инструменты, мебель и товары для дома и сада. У него есть большой выбор категорий товаров, от строительных материалов до мебели и инструментов. Подробные описания товаров, включая технические

характеристики, фото и отзывы покупателей. Удобная корзина для добавления товаров. Возможность выбора способа доставки (курьерская доставка, самовывоз из магазина). Доставка товаров на дом или стройплощадку, услуги по установке и монтажу (например, установка кухонь, монтаж окон и дверей). Регистрация и авторизация пользователей, история заказов и статус текущих заказов.

Проанализировав функциональность интернет-магазина Leroy Merlin, я понял основной функционал такого магазина, и сделал для себя выводы как должна выглядеть модель базы данных для интернет-магазина по продаже стройматериалов. На практике не все аспекты, реализованные в данном магазине, я решил включить в свою модель, а только основные, необходимые для главного функционала базы данных.

1.2. Анализ существующих решений и технологий

При разработке базы данных для интернет-магазина строительных материалов важно провести анализ существующих решений и технологий, чтобы выбрать наиболее подходящие инструменты и платформы

Реляционные СУБД

Реляционные СУБД являются классическими системами, наиболее часто используемыми для обработки транзакций в реальном времени (OLTP). Эти СУБД обеспечивают работу с большим количеством небольших транзакций, предоставляя короткое время отклика и возможность отмены изменений при необходимости.[4]

Подходят для систем с высокой нормализацией данных и для обработки большого количества коротких транзакций, среди которых значительное число операций вставки. Интернет-магазин строительных материалов требует надежного хранения данных о пользователях, товарах, категориях, заказах и корзинах, что делает реляционную СУБД идеальным выбором.

Популярные СУБД:

- Microsoft SQL Server: Хорошо интегрируется с продуктами Microsoft, обладает высокой производительностью, но также является дорогостоящим решением.
- PostgreSQL: Открытая СУБД с мощными функциями, поддерживающая сложные запросы и транзакции, но требует некоторого уровня знаний для настройки.
- MySQL: Открытая и широко используемая СУБД, легко настраиваемая и интегрируемая с различными приложениями, но может иметь ограничения при работе с очень большими объемами данных.

Когда не следует выбирать: Реляционные СУБД не подходят для хранения неструктурированных данных или простых структур «ключ-значение», а также при необходимости частого обновления значений в одних и тех же строках.

NoSQL СУБД

Модель NoSQL появилась в ответ на необходимость оперативно обрабатывать действительно огромные объёмы данных. Поэтому NoSQL по большей части заточена под масштабирование по горизонтали и работу с недостаточно структурированными или постоянно меняющимися данными.[3]

Когда следует выбирать: NoSQL СУБД подходят для приложений, требующих гибкого и масштабируемого хранилища для больших объемов неструктурированных данных, таких как веб-приложения, мобильные приложения и системы реального времени

Облачные СУБД

Облачные СУБД предлагают хранилище данных и управление ими в облаке, предоставляя масштабируемость и доступность без необходимости управлять физической инфраструктурой. Эти решения позволяют легко масштабировать

ресурсы в зависимости от потребностей приложения и обеспечивают высокую доступность данных.

Когда следует выбирать: Облачные СУБД подходят для приложений, которые требуют гибкости и масштабируемости, а также для разработчиков, желающих сосредоточиться на разработке приложений, а не на управлении инфраструктурой.

Выбор MySQL

Я выбрал MySQL для реализации базы данных интернет-магазина строительных материалов, так как у меня в MySQL больше практики, и учился создавать базы данных я на MySQL. Вот основные плюсы и минусы этого решения:

Плюсы:

- Открытый исходный код: MySQL является бесплатной и открытой СУБД, что делает ее доступной для использования без дополнительных затрат.
- Простота настройки и использования: MySQL легко устанавливается и настраивается, что позволяет быстро начать работу с базой данных.
- Широкая поддержка сообществом: благодаря большому сообществу пользователей и разработчиков, легко найти помощь и документацию по любым вопросам, связанным с MySQL.

Минусы:

- Ограничения при больших объемах данных: MySQL может испытывать трудности при работе с очень большими объемами данных и высокой нагрузкой на транзакции.
- Проблемы с масштабируемостью: MySQL может требовать дополнительных настроек и решений для обеспечения масштабируемости при увеличении объема данных и нагрузки на систему.

1.3. Требования к базе данных

При разработке базы данных для интернет-магазина строительных материалов необходимо учитывать следующие требования:

Нормализация данных:

База данных должна быть приведена как минимум к третьей нормальной форме (3NF), чтобы избежать избыточности данных и обеспечить целостность данных. Это достигается разделением данных на логические таблицы и установлением правильных связей между ними.

Безопасность:

Хеширование паролей: Пароли пользователей должны храниться в базе данных в зашифрованном виде, используя алгоритмы хеширования (например, bcrypt, SHA-256).

Контроль доступа: Разграничение прав доступа на основе ролей (клиенты, продавцы, администраторы) для защиты данных от несанкционированного доступа.

Защита от SQL-инъекций: Использование подготовленных выражений и параметризованных запросов для предотвращения атак SQL-инъекций.

Производительность и масштабируемость:

Индексация: Создание индексов по основным полям (например, product_id, category_id, price) для ускорения поиска и фильтрации данных.

Резервное копирование и восстановление:

Регулярное резервное копирование базы данных и разработка плана восстановления данных для защиты от потери данных.

1.4. Основные элементы базы данных интернет-магазина

База данных для интернет-магазина строительных материалов должна включать несколько ключевых компонентов, обеспечивающих полный цикл обработки и

управления информацией. Основные таблицы базы данных и их функциональность включают:

1. Таблица пользователей (user):

- Хранение информации о пользователях (клиенты, продавцы, администраторы).
- Поля: user_id, username, password, role.

2. Таблица категорий (category):

- Классификация товаров по категориям.
- Поля: category_id, name.

3. Таблица товаров (product):

- Хранение информации о товарах.
- Поля: product_id, name, description, price, stock, category_id, seller_id.

4. Таблица корзины (cart):

- Управление корзинами пользователей.
- Поля: cart_id, user_id, product_id, quantity, created_at.

5. Таблица заказов (order):

- Управление заказами и статусами.
- Поля: order_id, user_id, total_amount, order_status, order_date.

6. Таблица деталей заказа (order_detail):

- Хранение информации о деталях заказов.
- Поля: order_detail_id, order_id, product_id, quantity, price.[2]

Диаграмма ER

Диаграмма ER означает «Диаграмма отношений сущностей», также известная как ERD, — это диаграмма, которая отображает отношения наборов сущностей, хранящихся в базе данных. ER-диаграммы создаются на основе трех основных концепций: сущности, атрибуты и отношения.

ER-диаграммы содержат различные символы, в которых прямоугольники используются для обозначения объектов, овалы для определения атрибутов и ромбовидные формы для обозначения связей.[12]

На основе таблиц, приведённых ранее, я разработал свою ERD. Эта диаграмма наглядно отображает основные элементы базы данных интернет-магазина строительных материалов и их взаимосвязи, что помогает лучше понять и управлять логической структурой данных.

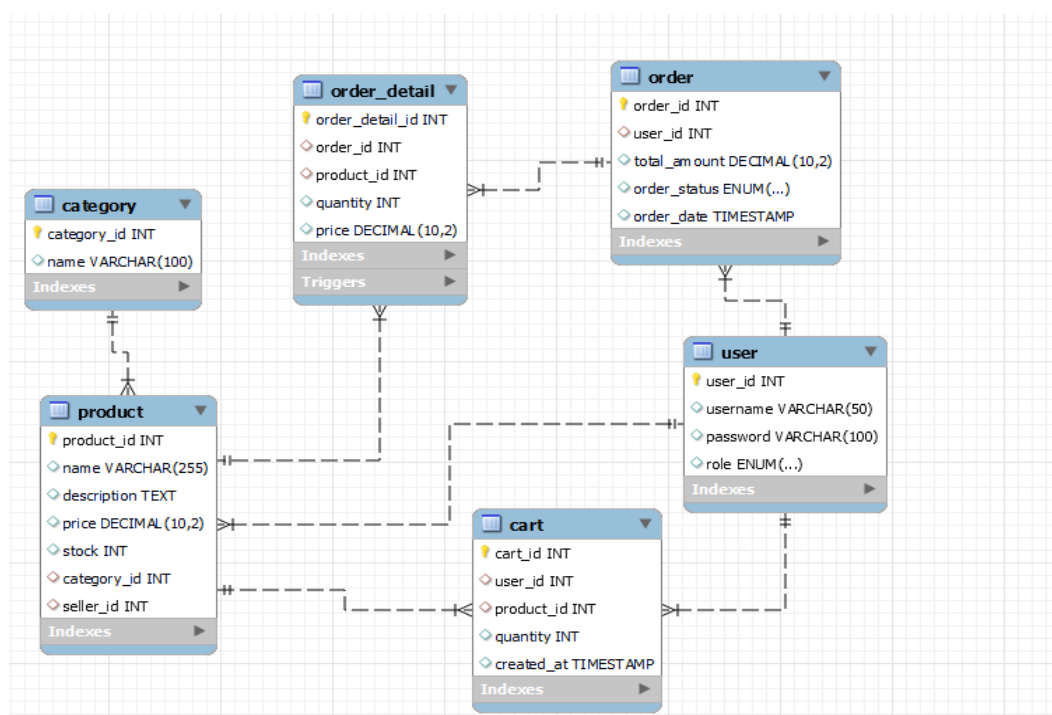


Рис.1

Глава 2: Реализация базы данных для интернет-магазина по продаже стройматериалов

2.1. Создание и структура базы данных

Создание базы данных в MySQL начинается с определения структуры базы данных, которая включает в себя таблицы, их поля и связи между ними. Для нашего

интернет-магазина строительных материалов база данных будет состоять из следующих таблиц:

- user
- category
- product
- cart
- order
- order_detail

Для создания базы данных и таблиц используется следующий SQL код:

```
CREATE DATABASE IF NOT EXISTS buildingstore;  
USE buildingstore;
```

Каждая таблица в базе данных имеет свои поля и связи с другими таблицами.

Рассмотрим каждую таблицу отдельно.

Таблица user предназначена для хранения данных пользователей интернет-магазина.

```
CREATE TABLE user (  
    user_id INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(50),  
    password VARCHAR(100),  
    role ENUM('customer', 'seller', 'admin')  
);
```

Таблица category используется для хранения категорий товаров.

```
CREATE TABLE category (  
    category_id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100)  
);
```

Таблица product содержит информацию о товарах, которые предлагаются в интернет-магазине.

```
CREATE TABLE product (  
    product_id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(255),  
    description TEXT,  
    price DECIMAL(10,2),  
    stock INT DEFAULT 0,  
    category_id INT,
```

```

        seller_id INT,
        FOREIGN KEY (category_id) REFERENCES category(category_id),
        FOREIGN KEY (seller_id) REFERENCES user(user_id)
    );

```

Таблица cart хранит информацию о товарах, которые пользователи добавили в свои корзины.

```

CREATE TABLE cart (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    product_id INT,
    quantity INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES user(user_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);

```

Таблица order предназначена для хранения данных о заказах.

```

CREATE TABLE `order` (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    total_amount DECIMAL(10,2),
    order_status ENUM('paid', 'processing', 'unpaid') DEFAULT
'processing',
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES user(user_id)
);

```

Таблица order_detail содержит подробную информацию о продуктах, включенных в заказы.

```

CREATE TABLE order_detail (
    order_detail_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    price DECIMAL(10,2),
    FOREIGN KEY (order_id) REFERENCES `order`(order_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);

```

2.2. Тестирование базы данных

Разработка базы данных для интернет-магазина строительных материалов — это лишь первый шаг. Чтобы обеспечить её корректную работу, необходимо провести всестороннее тестирование. В данной главе я расскажу о тестировании базы

данных, уделив особое внимание проверке целостности данных и CRUD-операциям (создание, чтение, обновление и удаление).

Проверка целостности данных

Первичные ключи: Убедитесь, что каждый первичный ключ уникален и не содержит NULL-значений.

Внешние ключи: Проверьте, что внешние ключи корректно ссылаются на существующие записи в связанных таблицах.

Уникальные ограничения: Убедитесь, что поля, которые должны содержать уникальные значения, не имеют дубликатов.

Не NULL ограничения: Проверьте, что поля с ограничением NOT NULL не содержат пустых значений.

Проверка CRUD операций

CRUD (Create, Read, Update, Delete) операции — это основа взаимодействия с данными в базе данных. Важно убедиться, что каждая из этих операций работает корректно на всех уровнях базы данных. Рассмотрим каждый тип операции на примере наших таблиц:

Таблица пользователей

```
INSERT INTO `user` (username, password, role) VALUES ('new_user',  
'new_password', 'customer');  
SELECT * FROM `user` WHERE user_id = 7;  
UPDATE `user` SET password = 'updated_password' WHERE user_id = 7;  
DELETE FROM `user` WHERE user_id = 7;
```

Таблица категорий

```
INSERT INTO `category` (name) VALUES ('New Category');  
SELECT * FROM `category` WHERE category_id = 6;  
UPDATE `category` SET name = 'Updated Category' WHERE category_id =  
6;  
DELETE FROM `category` WHERE category_id = 6;
```

Таблица товаров

```
INSERT INTO `product` (name, description, price, stock, category_id, seller_id) VALUES ('New Product', 'Description of new product', 99.99, 10, 1, 2);
SELECT * FROM `product` WHERE product_id = 11;
UPDATE `product` SET price = 89.99 WHERE product_id = 11;
DELETE FROM `product` WHERE product_id = 11;
```

Таблица заказов

```
INSERT INTO `order` (user_id, total_amount, order_status) VALUES (1, 250.00, 'processing');
SELECT * FROM `order` WHERE order_id = 8;
UPDATE `order` SET order_status = 'paid' WHERE order_id = 8;
DELETE FROM `order` WHERE order_id = 8;
```

Таблица деталей заказа

```
INSERT INTO `order_detail` (order_id, product_id, quantity, price) VALUES (1, 1, 2, 50.00);
SELECT * FROM `order_detail` WHERE order_detail_id = 13;
UPDATE `order_detail` SET quantity = 3 WHERE order_detail_id = 13;
DELETE FROM `order_detail` WHERE order_detail_id = 13;
```

Таблица корзины

```
INSERT INTO `cart` (user_id, product_id, quantity) VALUES (1, 1, 2);
SELECT * FROM `cart` WHERE cart_id = 6;
UPDATE `cart` SET quantity = 3 WHERE cart_id = 6;
DELETE FROM `cart` WHERE cart_id = 6;
```

2.3. Создание ролей и привилегий

Создание ролей в базе данных является важным аспектом управления доступом и обеспечения безопасности. Роли позволяют централизованно управлять правами доступа пользователей, что упрощает администрирование, особенно в больших системах с множеством пользователей. Преимущества создания ролей включают:

1. Гибкость и масштабируемость: Изменение прав доступа для группы пользователей становится проще — достаточно изменить привилегии роли.
2. Повышение безопасности: Роли помогают ограничить доступ к критически важным данным и функциям, предотвращая несанкционированные действия.

Создание ролей включает несколько шагов:

1. Создание пользователей: Создание учетных записей пользователей.

2. Назначение привилегий пользователям: Определение и назначение прав доступа для каждого пользователя или группы пользователей.

Для создания пользователей и назначения им привилегий в MySQL можно использовать следующий SQL-код:

```
CREATE USER 'customer'@'localhost' IDENTIFIED BY 'password1';
CREATE USER 'seller'@'localhost' IDENTIFIED BY 'password1';
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'password1';
GRANT ALL PRIVILEGES ON buildingstore.* TO 'admin'@'localhost';
GRANT SELECT, INSERT, UPDATE ON buildingstore.product TO
'seller'@'localhost';
GRANT SELECT ON buildingstore.category TO 'seller'@'localhost';
GRANT SELECT ON buildingstore.order TO 'seller'@'localhost';
GRANT SELECT ON buildingstore.order_detail TO 'seller'@'localhost';
GRANT SELECT ON buildingstore.product TO 'customer'@'localhost';
GRANT SELECT ON buildingstore.category TO 'customer'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON buildingstore.cart TO
'customer'@'localhost';
GRANT SELECT, INSERT ON buildingstore.order TO
'customer'@'localhost';
GRANT SELECT ON buildingstore.order_detail TO
'customer'@'localhost';
FLUSH PRIVILEGES;
```

Давайте разберем этот код по порядку, Создание пользователей:

```
CREATE USER 'customer'@'localhost' IDENTIFIED BY 'password1';
CREATE USER 'seller'@'localhost' IDENTIFIED BY 'password1';
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'password1';
```

Эти команды создают новых пользователей с именами customer, seller и admin соответственно. Все они могут подключаться только с локального компьютера (localhost). Каждый пользователь идентифицируется своим паролем password1.

Присвоение привилегий пользователям:

```
GRANT ALL PRIVILEGES ON buildingstore. TO 'admin'@'localhost';*
```

Эта команда назначает пользователю admin все привилегии на всю базу данных buildingstore. Это включает возможность создавать, изменять и удалять любые объекты базы данных.

```
GRANT SELECT, INSERT, UPDATE ON buildingstore.product TO
'seller'@'localhost';
```

```
GRANT SELECT ON buildingstore.category TO 'seller'@'localhost';  
GRANT SELECT ON buildingstore.order TO 'seller'@'localhost';  
GRANT SELECT ON buildingstore.order_detail TO 'seller'@'localhost';
```

Эти команды позволяют пользователю seller выполнять выборку, вставку и обновление записей в таблице product, а также просматривать записи в таблицах category, order и order_detail базы данных buildingstore.

```
GRANT SELECT ON buildingstore.product TO 'customer'@'localhost';  
GRANT SELECT ON buildingstore.category TO 'customer'@'localhost';  
GRANT SELECT, INSERT, UPDATE, DELETE ON buildingstore.cart TO  
'customer'@'localhost';  
GRANT SELECT, INSERT ON buildingstore.order TO 'customer'@'localhost';  
GRANT SELECT ON buildingstore.order_detail TO 'customer'@'localhost';
```

Эти команды позволяют пользователю customer просматривать записи в таблицах product, category, order и order_detail, а также выполнять выборку, вставку, обновление и удаление записей в таблице cart базы данных buildingstore.

```
FLUSH PRIVILEGES;
```

Эта команда обновляет привилегии, чтобы изменения, внесенные с помощью команд GRANT, вступили в силу.

2.4. Создание типовых запросов

Типовые запросы являются важным инструментом для быстрого и точного извлечения информации, что особенно полезно для анализа данных и принятия решений.

Получение всех продуктов категории:

```
SELECT p.name, p.description, p.price  
FROM product p  
JOIN category c ON p.category_id = c.category_id  
WHERE c.name = 'инструменты';
```

Этот запрос позволяет получить все продукты, относящиеся к категории "инструменты". Он использует соединение таблиц product и category по полю category_id, чтобы выбрать нужные данные. Данный запрос полезен для фильтрации продуктов по категориям, что облегчает пользователям поиск нужных товаров.

Получение всех заказов пользователя:

```
SELECT o.order_id, o.total_amount, o.order_status, o.order_date
FROM `order` o
WHERE o.user_id = 2;
```

Этот запрос позволяет получить все заказы, сделанные пользователем с идентификатором `user_id = 2`. Он извлекает информацию о заказах, включая их идентификатор, общую сумму, статус и дату заказа. Это полезно для мониторинга активности пользователей и управления их заказами.

Получение деталей конкретного заказа:

```
SELECT od.product_id, od.quantity, od.price
FROM order_detail od
WHERE od.order_id = 1;
```

Этот запрос извлекает детали конкретного заказа с идентификатором `order_id = 1`. Он показывает, какие продукты были включены в заказ, их количество и цену. Такой запрос помогает анализировать состав заказов и отслеживать продажи.

Получение всех продуктов продавца:

```
SELECT p.product_id, p.name, p.description, p.price
FROM product p
JOIN user u ON p.seller_id = u.user_id
WHERE u.role = 'seller' AND u.user_id = 3;
```

Этот запрос позволяет получить все продукты, добавленные продавцом с идентификатором `user_id = 3`. Он соединяет таблицы `product` и `user` по полю `seller_id` и фильтрует данные по роли пользователя (продавец). Запрос полезен для управления ассортиментом товаров каждого продавца.

Обновление статуса заказа:

```
UPDATE `order`
SET order_status = 'paid'
WHERE order_id = 1;
```

Этот запрос обновляет статус заказа с идентификатором `order_id = 1` на "paid". Это важно для отслеживания статуса заказов и их обработки в системе.

Типовые запросы являются неотъемлемой частью эффективного управления базой данных, обеспечивая быстрый и точный доступ к необходимой информации и улучшая общую производительность системы.

2.5. Управление транзакциями в SQL

Транзакции SQL – это группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Иными словами, транзакции позволяют нам контролировать процессы сохранения и изменения в базах данных, обеспечивая целостность и согласованность данных.[5]

Я написал хранимую процедуру CreateOrder для выполнения комплекса операций, связанных с созданием заказа. Основная цель этой процедуры — обработка заказа пользователя, включая вычисление общей суммы заказа, вставку данных о заказе и деталях заказа, обновление запасов продуктов и очистку корзины. Внутри процедуры используется транзакция, чтобы гарантировать целостность данных.

Начало транзакции:

```
START TRANSACTION;
```

Транзакция начинается с команды START TRANSACTION. Это позволяет сгруппировать все последующие операции в одну транзакцию, обеспечивая атомарность выполнения.

```
SELECT IFNULL(SUM(p.price * c.quantity), 0) INTO total  
FROM cart c  
JOIN product p ON c.product_id = p.product_id  
WHERE c.user_id = p_user_id;
```

Этот запрос вычисляет общую сумму заказа пользователя, умножая цену каждого продукта на его количество в корзине. IFNULL(expression, 0): Функция IFNULL используется для замены значения NULL на указанное значение (в данном случае на 0). Это важно, потому что если в корзине нет товаров, то SUM(p.price * c.quantity) вернет NULL. Использование IFNULL гарантирует, что результат будет 0 вместо NULL.

```
INSERT INTO `order` (user_id, total_amount, order_status, order_date)  
VALUES (p_user_id, total, 'processing', NOW());
```

Вставка новой записи в таблицу order, содержащей информацию о пользователе, общей сумме заказа, статусе и дате заказа.

```
SET last_order_id = LAST_INSERT_ID();
```

Я использую команду LAST_INSERT_ID() для получения последнего вставленного идентификатора заказа. MySQL функция LAST_INSERT_ID возвращает первое значение AUTO_INCREMENT, которое было установлено с помощью самой последней инструкции INSERT или UPDATE, которая повлияла на столбец AUTO_INCREMENT. Это значение затем присваивается переменной last_order_id.[7]

```
INSERT INTO order_detail (order_id, product_id, quantity, price)
SELECT last_order_id, c.product_id, c.quantity, p.price
FROM cart c
JOIN product p ON c.product_id = p.product_id
WHERE c.user_id = p_user_id;
UPDATE product p
JOIN cart c ON p.product_id = c.product_id
SET p.stock = p.stock - c.quantity
WHERE c.user_id = p_user_id;
```

Вставка деталей заказа в таблицу order_detail и обновление количества продуктов на складе.

```
DELETE FROM cart WHERE user_id = p_user_id;
```

Удаление всех товаров из корзины пользователя после оформления заказа.

```
COMMIT;
```

Завершение транзакции командой COMMIT, что означает подтверждение всех изменений, сделанных в ходе транзакции.

Преимущества использования транзакций

Хотя каждый SQL-оператор и так является атомарной операцией (то есть сам по себе представляет транзакцию), использование транзакционного контекста для каждого оператора позволяет обеспечить проверку целостности данных, контролировать выполнение и предотвращать конфликтные ситуации, например, гонки состояний.[6]

Допустим, у нас есть триггеры, которые могут вызвать изменения и, если они не выполнены полностью, привести к несогласованности данных. Транзакции помогут предотвратить такую ситуацию.[6]

2.6. Локальные переменные в хранимых процедурах

Локальные переменные в хранимых процедурах позволяют сохранять значения внутри процедуры и использовать их в различных операциях. Это удобно для временного хранения промежуточных результатов или для выполнения сложных вычислений. Для объявления и использования локальных переменных используется ключевое слово `DECLARE`, за которым следует имя переменной и ее тип данных.[8]

Процедура `CalculateTotalPrice` принимает два входных параметра: `productId` (идентификатор продукта) и `quantity` (количество продукта). Процедура вычисляет общую стоимость указанного количества продукта и возвращает это значение.

```
DELIMITER //
CREATE PROCEDURE CalculateTotalPrice(IN productId INT, IN quantity
INT)
BEGIN
    DECLARE total_price DECIMAL(10, 2);
    DECLARE product_price DECIMAL(10, 2);
    -- Получение цены продукта
    SELECT price INTO product_price FROM product WHERE product_id =
productId;
    -- Вычисление общей стоимости
    SET total_price = product_price * quantity;
    -- Возвращение общей стоимости
    SELECT total_price;
END //
DELIMITER ;
```

В начале процедуры объявляются две локальные переменные `total_price` и `product_price`, обе типа `DECIMAL(10, 2)`. Эти переменные будут использоваться для хранения промежуточных результатов.

```
DECLARE total_price DECIMAL(10, 2);
DECLARE product_price DECIMAL(10, 2);
```

Получение цены продукта:

```
SELECT price INTO product_price FROM product WHERE product_id = productId;
```

Запрос выбирает цену продукта из таблицы product по заданному productId и сохраняет результат в переменной product_price.

Вычисление общей стоимости:

```
SET total_price = product_price * quantity;
```

После получения цены продукта, общая стоимость вычисляется путем умножения product_price на quantity. Результат сохраняется в переменной total_price.

Возвращение общей стоимости SELECT total_price;

В конце процедуры общее значение стоимости продукта возвращается с помощью команды SELECT.

2.7. Условия

Рассмотрим условие в хранимой процедуре AddProduct. Эта процедура добавляет новый продукт в базу данных, но только если пользователь, выполняющий добавление, имеет роль seller. Если роль пользователя отличается от seller, процедура генерирует ошибку: (см. Приложение 2).

```
SELECT role INTO user_role  
FROM user  
WHERE user_id = p_seller_id;
```

Сначала процедура извлекает роль пользователя, который пытается добавить товар, и сохраняет её в локальную переменную user_role.

Проверка роли пользователя:

```
IF user_role = 'seller' THEN  
    INSERT INTO product (name, description, price, stock,  
category_id, seller_id)  
VALUES (p_name, p_description, p_price, p_stock, p_category_id,  
p_seller_id);  
ELSE  
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Only sellers can add  
products';  
END IF;
```

Условия являются неотъемлемой частью хранимых процедур и позволяют эффективно управлять потоком выполнения кода. В примере с процедурой `AddProduct` условие используется для проверки роли пользователя и обеспечения того, что только пользователи с ролью `seller` могут добавлять новые товары в базу данных.

2.8. Хранимые процедуры в базе данных

В моей разработанной базе данных я использовал хранимые процедуры для выполнения различных задач, связанных с управлением данными. Хранимые процедуры позволяют мне повторно использовать код, когда это необходимо, помогая упростить разработку приложений и уменьшить количество ошибок в операторах. Разработчикам не приходится писать сложные запросы для каждого требования приложения, и команда QA тратит меньше времени на проверку запросов при тестировании приложений.[9]

В моем проекте я использовал следующие хранимые процедуры:

Процедура для получения деталей заказа по его идентификатору.

Процедура для создания заказа и удаления записи из корзины с обновлением количества товара в таблице `product`.

Процедура для добавления нового товара с проверкой роли пользователя.

Локальные переменные в процедуре для расчета общей стоимости определенного количества товара.

Хранимые процедуры в MySQL объявляются с помощью оператора `CREATE PROCEDURE`, за которым следует имя процедуры и параметры, если они необходимы. Для обработки блоков SQL-кода внутри процедур используется ключевое слово `BEGIN` и завершается блок ключевым словом `END`. Это позволяет выполнять несколько операторов SQL как одну логическую единицу.

DELIMITER //

```
CREATE PROCEDURE ProcedureName (IN parameterName DataType)
BEGIN
    -- Тело процедуры
    -- SQL операторы
END //
DELIMITER ;
```

Использование хранимых процедур в моей базе данных позволяет централизовать и стандартизировать операции, что облегчает поддержку и масштабирование системы. Кроме того, хранимые процедуры обеспечивают безопасность и целостность данных, так как все операции выполняются в контролируемой среде базы данных.

2.9. Представления в базе данных

В моей разработанной базе данных я использовал представления для упрощения доступа к данным и создания абстракций, которые скрывают сложность базовых запросов. Представления позволяют мне создавать виртуальные таблицы, которые содержат результаты выполнения сложных запросов, что делает работу с данными более удобной и безопасной.

Что такое представления?

Представление (view) в базе данных — это виртуальная таблица, сформированная на основе результатов выполнения запроса. Представление не содержит данных непосредственно, оно отображает данные из одной или нескольких таблиц, представляя их в удобном для использования виде. Представления могут включать данные, полученные из различных таблиц, и позволяют фильтровать, сортировать и агрегировать эти данные.

Использование представлений имеет несколько преимуществ:

Упрощение запросов: Представления позволяют скрыть сложность базовых запросов и предоставить пользователям более простой интерфейс для работы с данными.

Повышение безопасности: Представления могут ограничивать доступ к определенным столбцам или строкам данных, предоставляя пользователям доступ только к необходимой информации.

Удобство управления данными: Изменения в базовых таблицах автоматически отражаются в представлениях, что упрощает поддержание данных в актуальном состоянии.

В моей разработанной базе данных я создал представление для отображения деталей заказов. Это представление объединяет информацию из таблиц `order` и `order_detail`, предоставляя полный набор данных о каждом заказе, включая его детали.

```
CREATE VIEW orderdetailsview AS
SELECT
    o.order_id,
    o.user_id,
    o.total_amount,
    o.order_status,
    o.order_date,
    od.product_id,
    od.quantity,
    od.price
FROM `order` o
JOIN order_detail od ON o.order_id = od.order_id;
```

Разбор представления

CREATE VIEW orderdetailsview AS: Объявление нового представления с именем `orderdetailsview`.

SELECT ... FROM order o JOIN order_detail od ON o.order_id = od.order_id: Запрос, который объединяет таблицы `order` и `order_detail` по идентификатору заказа (`order_id`).

o.order_id, o.user_id, o.total_amount, o.order_status, o.order_date: Поля из таблицы `order`, которые включают идентификатор заказа, идентификатор пользователя, общую сумму заказа, статус заказа и дату заказа.

od.product_id, od.quantity, od.price: Поля из таблицы `order_detail`, которые включают идентификатор товара, количество и цену.

Это представление позволяет легко получить всю необходимую информацию о заказах и их деталях, выполняя один простой запрос. Вместо выполнения сложных

соединений каждый раз, когда требуется эта информация, пользователи могут просто обратиться к представлению `orderdetailsview`.

Использование представлений в моей базе данных способствует повышению удобства работы с данными, улучшению безопасности и упрощению управления данными. Представления позволяют создать более структурированный и понятный интерфейс для взаимодействия с базой данных, что значительно упрощает разработку и поддержку приложений.

2.10. Пользовательская функция в базе данных

Пользовательские функции в базе данных — это специальные объекты базы данных, которые позволяют выполнять определенные операции и возвращать одно значение. Они похожи на хранимые процедуры, но с важным отличием: функции всегда возвращают значение и могут использоваться в SQL-запросах.

Использование пользовательских функций имеет несколько преимуществ:

Повторное использование кода: Функции позволяют инкапсулировать часто повторяющиеся операции, что облегчает их повторное использование.

Упрощение запросов: Сложные операции можно спрятать внутри функций, что делает SQL-запросы более читабельными.

Поддержка и масштабируемость: Функции облегчают управление и модификацию кода, поскольку изменения в логике нужно внести только в одном месте.

Пример пользовательской функции в моей базе данных

В моей базе данных я создал пользовательскую функцию `GetOrderTotal`, которая рассчитывает общую сумму заказа по его идентификатору. Эта функция полезна для получения общей стоимости заказа без необходимости писать сложные запросы каждый раз.

```
DELIMITER //
```

```
CREATE FUNCTION `GetOrderTotal`(p_order_id INT) RETURNS
decimal(10,2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10,2) DEFAULT 0.00;
    -- Подсчитываем общую сумму заказа
    SELECT SUM(price * quantity) INTO total
    FROM order_detail
    WHERE order_id = p_order_id;
    RETURN total;
END //
DELIMITER ;
```

CREATE FUNCTION GetOrderTotal(p_order_id INT) RETURNS decimal(10,2) DETERMINISTIC: Объявление функции GetOrderTotal, которая принимает один параметр p_order_id типа INT и возвращает значение типа DECIMAL(10,2). Ключевое слово DETERMINISTIC указывает, что функция всегда возвращает одно и то же значение для одного и того же набора входных данных.

DECLARE total DECIMAL(10,2) DEFAULT 0.00: Объявление локальной переменной total для хранения общей суммы заказа с начальным значением 0.00.

SELECT SUM(price * quantity) INTO total FROM order_detail WHERE order_id = p_order_id: Запрос, который рассчитывает сумму всех элементов заказа, умножая цену на количество и записывая результат в переменную total.

RETURN total: Возвращает рассчитанную общую сумму заказа.

Эта функция позволяет легко и быстро получать общую сумму любого заказа, просто вызвав функцию GetOrderTotal с нужным идентификатором заказа. Использование такой функции упрощает и стандартизирует процесс получения суммарной стоимости заказа, делая код более чистым и легким для поддержки.

2.11. Триггеры

В моей базе данных я использовал триггеры для автоматического выполнения определенных действий в ответ на добавление данных в таблице order_detail. Триггеры представляют собой специальные процедуры, которые выполняются автоматически при наступлении определенных событий, таких как вставка, обновление или удаление записей.

В моем проекте я разработал триггер, который автоматически обновляет общую сумму заказа после вставки новой записи в таблицу order_detail. Этот триггер

гарантирует, что сумма заказа всегда будет актуальной и отражать все изменения, связанные с добавлением новых товаров в заказ.

```
DELIMITER //
CREATE TRIGGER UpdateOrderTotalAmount
AFTER INSERT ON order_detail
FOR EACH ROW
BEGIN
    DECLARE new_total DECIMAL(10, 2);
    SELECT SUM(price * quantity) INTO new_total
    FROM order_detail
    WHERE order_id = NEW.order_id;
    UPDATE `order`
    SET total_amount = new_total
    WHERE order_id = NEW.order_id;
END //
DELIMITER ;
```

AFTER INSERT ON order_detail: Указывает, что триггер должен срабатывать после вставки новой записи в таблицу order_detail. **FOR EACH ROW:** Определяет, что триггер должен выполняться для каждой вставленной строки. **SELECT SUM(price * quantity) INTO new_total FROM order_detail WHERE order_id = NEW.order_id:** Вычисляет новую общую сумму заказа, суммируя стоимость всех товаров в заказе, и сохраняет результат в переменной new_total. **UPDATE order SET total_amount = new_total WHERE order_id = NEW.order_id:** Обновляет общую сумму заказа в таблице order, используя вычисленное значение new_total.

2.12. Обработчик исключений

В процессе разработки базы данных важно учитывать возможность возникновения ошибок и корректно их обрабатывать. При возникновении ошибки внутри хранимой процедуры необходимо принять соответствующие меры, например, продолжить или остановить выполнение операции и выдать сообщение об ошибке.

MySQL предоставляет простой способ определения обработчиков, которые обрабатывают ошибки, исходя из общих условий, таких как предупреждения или исключения из условий, например, конкретные коды ошибок.[10]

В моей разработанной базе данных обработчик исключений используется в процедуре для добавления нового товара. Если пользователь, пытающийся

добавить товар, не являясь продавцом, процедура выдает ошибку. Мы уже рассматривали его на примере условия процедуры AddProduct.

```
IF user_role = 'seller' THEN
    INSERT INTO product (name, description, price, stock,
category_id, seller_id)
    VALUES (p_name, p_description, p_price, p_stock,
p_category_id, p_seller_id);
ELSE
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Only sellers can
add products';
END IF;
```

Использование обработчика исключений позволяет эффективно управлять ошибками и обеспечивать целостность данных, предотвращая выполнение недопустимых операций и информируя пользователя о причинах ошибок. Это значительно упрощает разработку и поддержку приложений, делая их более надежными и удобными в использовании.

2.13. Создание резервных копий базы данных

Резервное копирование базы данных (БД) является одной из важнейших задач для обеспечения её надежности и безопасности. В процессе работы интернет-магазина по продаже строительных материалов накапливается большой объем данных, которые необходимо защищать от потерь. Потеря данных может произойти по различным причинам, таким как сбой оборудования, ошибки программного обеспечения, атаки злоумышленников и человеческие ошибки. В этой главе описывается процесс создания резервных копий БД и меры для обеспечения их безопасности и надежности.

Типы резервных копий:

Полное резервное копирование: включает копирование всей базы данных целиком. Это наиболее полная форма резервного копирования, но требует значительных ресурсов и времени.

Дифференциальное резервное копирование: копирует только изменения, произошедшие с момента последнего полного резервного копирования. Это позволяет уменьшить объем данных, подлежащих копированию, и ускорить процесс восстановления.

Частота резервного копирования:

Регулярное выполнение резервного копирования необходимо для минимизации потерь данных. В зависимости от объема данных и частоты их изменений, резервное копирование может выполняться ежедневно, еженедельно или по другому расписанию.

Процесс создания резервных копий:

Вот основные шаги, которые используются для резервного копирования базы данных в MySQL версии 8:

Для регулярного резервного копирования используется `mysqldump`, который позволяет экспортировать всю базу данных или отдельные таблицы в файл. Например, команда для полного резервного копирования выглядит следующим образом:

```
mysqldump -u root -p12345678 buildingstore > D:\backups\db_backup.sql
```

Для дифференциального резервного копирования используются встроенные функции MySQL для ведения журналов транзакций (binary logs), что позволяет копировать только измененные данные. Пример команды:

```
mysqlbinlog --read-from-remote-server --raw --stop-never --  
host=127.0.0.1 --port=3306 --user=root --password=12345678 mysql-  
bin.000001 > D:\backups\mysql-bin.000001
```

Хранение и защита резервных копий:

Для повышения защиты данных рекомендуется настроить резервное копирование на удаленные сервера и облачные хранилища, такие как Amazon S3, Google Cloud Storage и другие. Пример команды для копирования резервной копии в Amazon S3:

```
aws s3 cp D:\backups\db_backup1.sql s3://mybucket/backups/
```


Заключение

В ходе выполнения курсовой работы была достигнута поставленная цель — разработана база данных для интернет-магазина по продаже строительных материалов. В процессе работы были выполнены следующие задачи:

1. Проведение анализа теоретических основ интернет-магазинов.

Был проведен подробный анализ теоретических аспектов функционирования интернет-магазинов, включающий изучение их структуры, принципов работы и основных функциональных возможностей. Это дало возможность лучше понять требования к базе данных и необходимые компоненты для успешной работы интернет-магазина.

2. Обеспечение безопасности и надежности.

Были рассмотрены и внедрены меры по обеспечению безопасности и надежности базы данных. Это включало в себя обеспечение целостности данных и создание резервных копий. Были использованы механизмы шифрования данных (см. Приложение 3), База данных была приведена к третьей нормальной форме

3. Анализ существующих решений и технологий.

Был проведен анализ существующих решений и технологий для разработки баз данных интернет-магазинов. Рассматривались различные системы управления базами данных (СУБД), их преимущества и недостатки, а также инструменты для их реализации. Это позволило выбрать наиболее подходящую СУБД и технологии для данного проекта.

4. Проектирование структуры базы данных.

На основе проведенного анализа была спроектирована структура базы данных. Были определены основные сущности и их взаимосвязи, создана модель данных, описывающая таблицы, их поля и отношения между ними. Особое внимание было

уделено нормализации данных для устранения избыточности и обеспечения целостности информации.

5. Реализация базы данных.

На завершающем этапе работы была реализована база данных в выбранной СУБД. Созданы таблицы, установлены связи между ними, определены индексы и ограничения. Была проведена настройка производительности и тестирование базы данных на предмет корректности работы и соответствия поставленным требованиям.

В процессе работы возникло несколько трудностей, которые были успешно преодолены. Одной из сложных задач было обеспечение безопасности данных. Внедрение надежных механизмов защиты данных потребовало изучения современных методов шифрования и безопасного хранения данных.

В результате выполнения курсовой работы была создана эффективная и надежная база данных для интернет-магазина по продаже строительных материалов, которая отвечает всем основным требованиям по управлению информацией о товарах, заказах и клиентах. В будущем можно будет расширить данную базу данных, добавить в нее новые таблицы для реализации функционала доставки, и можно будет прописать новые хранимые процедуры, которые будут отвечать за логику.

Источники:

1. Плюсы интернет-магазинов [Электронный ресурс] / – Режим доступа: <https://vc.ru/marketing/146426-internet-magazin-put-k-uspehu-ili-5-prichin-pochemu-vashemu-biznesu-nuzhen-onlain-magazin>
2. Пример связей между таблицами, строение таблиц для базы данных [Электронный ресурс] / – Режим доступа: <https://habr.com/ru/articles/194714/>
3. Что такое модель NoSQL СУБД [Электронный ресурс] / – Режим доступа: https://yandex.cloud/ru/blog/posts/2022/10/nosql?utm_referrer=https%3A%2F%2Fyandex.ru%2F
4. Типы СУБД и выбор правильного [Электронный ресурс] / – Режим доступа: <https://folko.gitbook.io/podgotovka-k-sobesedovaniyu/bd/vidy-bd>
5. Транзакция определение [Электронный ресурс] / – Режим доступа: <https://loftschool.com/blog/posts/5-tranzakcii/>
6. Преимущества транзакций [Электронный ресурс] / – Режим доступа: <https://sky.pro/wiki/sql/tranzaktsii-v-ms-sql-plyusy-i-minusy-oborachivaniya-zaprosov/>
7. Команда LAST_INSERT_ID [Электронный ресурс] / – Режим доступа: https://oracleplsql.ru/mysql-function-last_insert_id.html?ysclid=lxj9koc3ve137016562
8. Локальные переменные [Электронный ресурс] / – Режим доступа: https://it.vshp.online/#!/pages/mdk1101/mdk1101_lab_19
9. Хранимые процедуры [Электронный ресурс] / – Режим доступа: https://sql-ex.ru/blogs/?/Rabota_s_hranimymi_procedurami_v_MySQL.html
10. Обработчик исключений [Электронный ресурс] / – Режим доступа: <https://www.internet-technologies.ru/articles/obrabotka-oshibok-mysql-v-hranimyh-procedurah.html>

11. Определение ERD [Электронный ресурс] / – Режим доступа:
<https://www.guru99.com/ru/er-diagram-tutorial-dbms.html>

Приложение 1. Пример транзакции для создания заказа

```
DELIMITER //
CREATE PROCEDURE CreateOrder(IN p_user_id INT)
BEGIN
    DECLARE total DECIMAL(10,2);
    DECLARE last_order_id INT;
    -- Начинаем транзакцию
    START TRANSACTION;
    -- Вычисляем общую сумму заказа
    SELECT IFNULL(SUM(p.price * c.quantity), 0) INTO total
    FROM cart c
    JOIN product p ON c.product_id = p.product_id
    WHERE c.user_id = p_user_id;
    -- Вставляем запись в таблицу заказов
    INSERT INTO `order` (user_id, total_amount, order_status,
order_date)
VALUES (p_user_id, total, 'processing', NOW());
    -- Получаем последний вставленный идентификатор заказа
    SET last_order_id = LAST_INSERT_ID();
    -- Вставляем детали заказа и обновляем запасы товара
    INSERT INTO order_detail (order_id, product_id, quantity, price)
SELECT last_order_id, c.product_id, c.quantity, p.price
FROM cart c
JOIN product p ON c.product_id = p.product_id
WHERE c.user_id = p_user_id;
    -- Обновляем запасы товаров
    UPDATE product p
    JOIN cart c ON p.product_id = c.product_id
    SET p.stock = p.stock - c.quantity
    WHERE c.user_id = p_user_id;
    -- Очищаем корзину
    DELETE FROM cart WHERE user_id = p_user_id;
    -- Подтверждаем транзакцию
    COMMIT;
END //
DELIMITER ;
CALL CreateOrder(1);
```

Приложение 2. Процедура для добавления нового товара с проверкой роли пользователя

```

DELIMITER //
CREATE PROCEDURE AddProduct(
    IN p_name VARCHAR(255),
    IN p_description TEXT,
    IN p_price DECIMAL(10,2),
    IN p_stock INT,
    IN p_category_id INT,
    IN p_seller_id INT
)
BEGIN
    DECLARE user_role ENUM('customer', 'seller', 'admin');
    SELECT role INTO user_role
    FROM user
    WHERE user_id = p_seller_id;
    IF user_role = 'seller' THEN
        INSERT INTO product (name, description, price, stock,
category_id, seller_id)
        VALUES (p_name, p_description, p_price, p_stock,
p_category_id, p_seller_id);
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Only sellers can
add products';
    END IF;
END //
DELIMITER ;

```


Приложение 3. Хранимая процедура для добавления нового пользователя с использованием хеширования пароля:

```

DELIMITER //
CREATE PROCEDURE AddUser (
    IN p_username VARCHAR(255),
    IN p_password VARCHAR(255),
    IN p_role ENUM('customer', 'seller', 'admin')
)
BEGIN
    DECLARE hashed_password VARCHAR(255);
    -- Хеширование пароля
    SET hashed_password = SHA2(p_password, 256);
    -- Добавление нового пользователя в таблицу
    INSERT INTO user (username, password, role)
    VALUES (p_username, hashed_password, p_role);
END //
DELIMITER ;

```

Приложение 4. Антиплагиат

**Antiplagius**
№1 в России

Антиплагиат 2.0, Проверка и повышение
уникальности текста за 2 минуты

antiplagius.ru

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 № 8896301

Дата выгрузки: 2024-06-19 21:31:19
Пользователь: bbd3372005@gmail.com, ID: 8896301

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

Информация о документе

№ документа: 8896301
Имя исходного файла: Разработка базы данных для интернет
магазина по продаже строительных материалов.docx
Размер файла: 0.13 МБ
Размер текста: 43119
Слов в тексте: 5996
Число предложений: 568

Информация об отчете

Дата: 2024-06-19 21:31:19 - Последний готовый отчет
Оценка оригинальности: 90%
Заемствования: 10%

90.86%

9.14%

Источники:

Доля в тексте	Ссылка
56.90%	https://vc.ru/marketing/146426-internet-magazin-put-k-uspehu-ili...
54.20%	https://opencart-cms.ru/nuzhen_sayt_internet_magazina/
18.70%	https://autopolka.com/
7.70%	https://www.oracle.com/cis/database/nosql/what-is-nosql/
6.60%	https://bpium.ru/blog/chem-baza-dannyh-luchshe-elektronnoy-tabli...

Приложение 5. Ссылка на репозиторий проекта

Ссылка на репозиторий: <https://github.com/Platina1337/DataBaseForBuildingStore.git>

