



Частное учреждение профессионального образования

«Высшая школа предпринимательства»

(ЧУПО «ВШП»)

КУРСОВОЙ ПРОЕКТ

«Разработка базы данных для интернет-магазина одежды»

Выполнил:

студент 3-го курса специальности

09.02.07 «Информационные системы и программирование»

Айдарбеков Адахан Таалайбекович

подпись: _____

Проверил:

преподаватель дисциплины,

преподаватель ЧУПО «ВШП»,

к.ф.н. Ткачев П.С.

оценка: _____

подпись: _____

Содержание

Введение.....	3
Глава 1. Анализ требований и проектирование логической модели базы данных.....	6
1.1. Анализ требований к базе данных интернет-магазина одежды..	6
1.2. Анализ связей между сущностями.....	8
1.3. Разработка схемы базы данных с использованием нотации Crow's foot.....	11
1.4. Администрирование базы данных.....	12
Глава 2. Проектирование физической модели базы данных и реализация.....	14
2.1. Проектирование физической модели базы данных.....	14
2.2. Создание ролей и предоставление привилегий.....	21
2.3. Типовые запросы.....	22
2.4. Транзакции.....	24
2.5. Триггеры.....	25
2.6. Хранимые процедуры.....	26
2.7. Локальные переменные.....	28
2.8. Представления.....	29
2.9. Условия.....	30
2.10. Обработчики исключений.....	30
2.11. Пользовательские функции.....	30
Заключение.....	31
Список источников.....	32
Приложение 1. Условия.....	33
Приложение 2. Пользовательские функции.....	33
Приложение 3. Антиплагиат.....	34
Приложение 4. QR код.....	34

Введение

Актуальностью темы “Разработки базы данных для интернет-магазина одежды” и в целом разработки просто интернет-магазина, является растущая популярность и спроса на онлайн-покупки, которые в свою очередь требуют эффективную систему для его успешной работы. В настоящее время рынок интернет-торговли (бытовой техникой, одеждой и т.д.) является одним из наиболее быстрорастущих направлений. Покупатели все чаще предпочитают приобретать товары онлайн, так как это просто, удобно, экономит время и позволяет выбрать нужные вещи по душе. Перейдем к основной теме курсовой работы, в которой в качестве выбранного направления для продажи является именно одежды. На самом деле под предлогом “одежда” имеется ввиду абсолютно все товары, которые люди носят на себе (обувь, рюкзаки и т.д.). Именно эти товары являются одним из наиболее продаваемых и востребованных в онлайн-магазинах. Согласно статистике, объем продаж одежды через интернет ежегодно растет примерно от 15% до 20%.

Еще одним не менее важным аспектом, которая явно показывает актуальность разработки базы данных для онлайн магазинов, является рост конкуренции на онлайн рынке. При высокой конкуренции, качество и скорость обслуживания клиентов, а также безопасность и надежность хранения их личных данных становятся ключевыми моментами, которые с высокой долей вероятности могут повлиять на успех бизнеса. Для успешной работы нашего магазина необходимо обеспечить высокую обработку и надежное хранение крупного объема данных, связанных с клиентами, товарами, заказами и другими сущностями (таблицами).

Также, база данных кроме обеспечения хранения и обработки данных, должна предоставлять возможности для анализа о покупательском поведении клиентов, чтобы выявлять тренды и предпочтения,

формировать выгодные предложения и акции, которые в будущем поспособствуют увеличению продаж.

Кроме того, разработка базы данных для интернет-магазина одежды является важной задачей для бизнеса, рассмотрим почему же:

1. Повышению эффективности работы магазина: благодаря такой возможности управления данными, сотрудники магазина могут быстро и точно обрабатывать заказы, контролировать наличие товаров на складе и вести учет продаж.
2. Улучшению качества обслуживания клиентов: с помощью базы данных можно хранить информацию о клиентах, их предпочтениях и в целом совершенных ими действиях, что позволяет найти индивидуальный подход для каждого покупателя, обслуживать и предлагать клиентам товары и услуги, соответствующие их требованиям.
3. Росту продаж: Об этом уже упоминалось выше, но все-таки стоит еще раз напомнить про этот пункт, так как для бизнеса это очень важно.
4. Минимизирование задержек при работе: благодаря автоматизации многих бизнес-процессов, таких как учет запасов, обработка заказов, ведение отчетности, сокращается количество повторяемых операций и уменьшаются задержки, что удобно и для самих сотрудников.

В итоге, с уверенностью можно сказать, что разработка базы данных для интернет-магазина является не только актуальной, но и важной задачей для бизнеса, так как она для него способствует улучшению качества обслуживания клиентов, росту продаж и многому другому.

Цель:

Целью данной работы является разработка базы данных для интернет-магазина одежды. Для достижения этой цели необходимо выполнить нижеперечисленные задачи.

Задачи:

1. Анализ требований к базе данных интернет-магазина одежды.
2. Проектирование логической модели базы данных.
3. Проектирование физической модели базы данных.
4. Реализация базы данных с использованием системы управления базами данных (СУБД).
5. Наполнение базы данных тестовыми данными.

Объектом исследования в данной работе является база данных для интернет-магазина одежды.

Предметом исследования является системный анализ и проектирование баз данных.

Методы исследования

В качестве основных методов исследования, использованных в данной работе можно называть:

- Метод анализа - изучение требований к проектированию модели базы данных для интернет-магазина.
- Метод абстрагирования - создание схемы и принципа работы разрабатываемой базы данных.
- Метод структурных аналогий - создание базы данных при помощи общих частей, наблюдаемых в существующих базах данных.

Глава 1. Анализ требований и проектирование логической модели базы данных

1.1. Анализ требований к базе данных интернет-магазина одежды

Когда мы разрабатываем базу данных, в первую очередь нужно понимать, что нужно сделать и что от нее ожидают в конце работы, затем спроектировать ее соответствующим образом. Как раз этот процесс и называют анализом требований и проектированием логической модели базы данных. Например, при создании базы данных для интернет-магазина одежды, нам обязательно нужно будет хранить информацию о товарах, которые мы продаем. Эти товары будут являться сущностями(таблицами) в нашей бд, а у этих сущностей будут свои атрибуты(столбцы), такие как: название, описание, цена, размер и т.д. Кроме товаров, у нас еще будут и другие сущности, с которыми мы будем работать, в качестве примера можно привести таблицу “Пользователь”(клиенты), которые будут покупать наш товар. У каждой из этих таблиц, также будут свои столбцы, данные которых мы будем хранить в нашей базе данных. В процессе работы, мы выясним то, как будут связаны наши таблицы и как она будет работать для удовлетворения потребностей бизнеса.

При разработке бд, мы должны помнить, что в будущем нам может понадобиться добавлять новые данные или же изменять существующие. Поэтому важно разработать нашу базу данных гибкой и масштабируемой. Представим ситуацию, в которой планируем добавлять данные в нашу базу данных, мы должны быть уверены, что она адаптируется к этим изменениям без всяких проблем. Также, что она может работать быстро даже при больших количествах нагрузок, так как будет расти клиентская база. В целом, наша база данных должна быть гибкой и масштабируемой, именно эту часть обязательно нужно учитывать.

Следующее, что может быть необходимой для полноценной работы нашей бд, это определение того, как наши клиенты будут взаимодействовать с ней, при наличии веб-приложения с профилем пользователей, мы должны реализовать функциональность редактирования профиля и добавления новых данных. Второе, что нам нужно: предоставить удобство нашим клиентам, а для этого нужно разработать фильтрацию и поиск по определенным критериям, также пагинацию, чтоб все наши товары не находились на одной странице.

Немаловажным аспектом является хранения и обработка мультимедийных файлов, таких как изображения, видео и аудиозаписи. Нам нужно рассмотреть возможность хранения изображений товаров в высоком качестве, просмотр и увеличение этих изображений. Также нужно хранить метаданные о мультимедийных файлах: название, размер, формат, описание. Это позволит нам эффективно хранить и достаточно быстро предоставлять нужные покупателям данные.

Все вышеперечисленное является не единственной, но основой при построения нужной нам базы данных для дальнейшей работы. Нужно приступать к следующим не менее важным этапам разработки.

В целом, при разработке логической модели базы данных мы рассматриваем по большей степени именно теоретическую часть. При разработке физической модели базы данных, которую мы рассмотрим в следующей главе, мы фокусируемся уже на практической части, реализовывая задуманное. Эти отличия нужно было разобрать и пояснить, чтобы в будущем все было понятно и не возникало вопросов.

1.2. Анализ связей между сущностями

Анализ связей между сущностями является важной частью при проектировании логической модели базы данных. Он позволяет определить, как сущности(таблицы) связаны друг с другом и позволяет выбрать наиболее подходящую нам связь.

В базах данных существует три основных способа связи между сущностями:

Один ко многим (one-to-many) - это способ связи, при котором одна сущность может быть связана с многими другими сущностями. Например, у одного покупателя может быть несколько отзывов, а один отзыв может принадлежать одному пользователю.

Многие ко многим (many-to-many) - это способ связи, при котором многие сущности одного типа могут быть связаны с многими сущностями другого типа. Например, у одного автора может быть несколько книг, также у одной книги может быть несколько авторов.

Один к одному (one-to-one) - это способ связи, при котором одна сущность может быть связана только с одной другой сущностью. К примеру: у одного пользователя может быть только один паспорт и наоборот, один паспорт может принадлежать только одному владельцу.

При анализе связей между сущностями нам нужно понимать, сколько объектов одной сущности может быть связано с объектами другой сущности - это называется кардинальностью связи.

Также необходимо учитывать направленность связи, то есть определить, какая сущность является владельцем связи, а какая зависимой. Например, в связи "один ко многим" между сущностями "категория товаров" и "товар" категория товаров является владельцем связи, а товар - зависимой сущностью.

Благодаря всем этим анализам связей между сущностями у нас есть возможность создать четкую и понятную структуру базы данных, который будет одновременно работать и обеспечивать нужный функционал для работы нашего магазина.

Кроме того, при разработке базы данных, нам нужно быть убежденным в том, что все данные в ней корректны и согласованы между собой. Этот процесс называется целостностью данных.

Сущностная целостность (entity integrity) гарантирует, что каждая сущность имеет уникальный идентификатор и что все атрибуты сущности не могут быть NULL, если это не предусмотрено моделью данных.

Ссылочная целостность (referential integrity) гарантирует, что все связи между сущностями корректны и что нет ссылок на несуществующие экземпляры сущностей.

При проектировании базы данных, нам также нужно помнить о производительности и масштабируемости. В эти аспекты могут быть включены оптимизации запросов к базе данных, выбор подходящей СУБД (система управления базами данных), декомпозиция таблиц и т.д

Стоит не забывать, что при разработке в целом всех баз данных необходимо учитывать требования к безопасности данных. Это может включать в себя шифрование данных, использование протоколов безопасной передачи данных, контроль доступа к базе данных и другие методы защиты данных. Нужно учитывать многие факторы, так как это не легкая работа и придется приложить не мало усилий, но с определенными инструментами можно облегчить себе работу и добиться желаемого результата.

Еще одним важным аспектом разработки логической модели базы данных является нормализация. Нормализация — это процесс разбиения

таблиц на более мелкие и более управляемые таблицы с целью устранения избыточности данных и обеспечения целостности данных. Существует несколько нормальных форм, каждая из которых имеет свои требования к структуре таблиц.

Например, первая нормальная форма требует, чтобы каждая таблица имела уникальный первичный ключ и чтобы все атрибуты(столбцы) таблицы были атомарными, то есть неделимыми. Вторая нормальная форма требует, чтобы все неключевые столбцы зависели от всего первичного ключа, а не только от его части. Третья нормальная форма требует, чтобы все неключевые столбцы не зависели друг от друга. Неключевые столбцы - это те столбцы, которые не используются для уникальной идентификации записи в таблице. Чтобы было проще для понимания, можно привести пример: В таблице 'Пользователи' столбец с id пользователя является ключевым столбцом, так как используется для уникальной идентификации записей, а другие столбцы - нет.

Нормализация помогает избежать избыточности данных, обеспечить целостность данных и упростить модель данных. Однако в некоторых случаях нормализация может привести к снижению производительности, поэтому необходимо найти баланс между нормализацией и производительностью.

1.3. Разработка схемы базы данных с использованием нотации Crow's foot

Нотация “Crow's Foot” (“Воронья лапа”) - это графическая нотация, используемая для создания схем баз данных. Данная нотация была разработана в 1976 году Гордоном Эверестом и в настоящее время люди продолжают широко использовать ее. Нотация - система обозначения.

“Воронья лапа” использует графические символы для обозначения связей между таблицами и столбцами для определения отношений между ними. Таблицы представлены в виде прямоугольников, как в то время столбцы в виде овала, а связи представлены в виде линий, которая и выполняет роль соединения наших таблиц. В целом, существует несколько типов отношений, которые можно посмотреть в виде “Вороньей лапы”:

Один ко одному (1:1)

Один ко многим(1:M)

Многие ко многим (M)

Все предназначения этих связей мы рассматривали в прошлых подглавах, снова разбирать смысла нет. Для каждого типа связей нужно использовать уникальные графические символы. К примеру, при использовании связи “один ко многим” нужно использовать стрелку, указывающую от объекта “один” ко “многие”. Данная нотация также предоставляет инструменты для представления правил для целостности данных и ограничений, таких как первичные и внешние ключи.

Отчасти популярность и частая используемость данной нотации заключается в том, что почти всегда для полного понимания графические иллюстрации подходят больше всего. “Воронья лапка” позволяет визуализировать структуру базы данных.

1.4. Администрирование базы данных

Администрирование базы данных является неотъемлемой частью разработки, а также поддержки любого приложения и системы, которое использует базу данных. Ниже будут приведены некоторые ключевые моменты, связанные с администрированием данных:

Настройка СУБД: Настройка параметров СУБД является одним из важных частей для обеспечения работы базы данных. В этот список можно включить настройку количества соединений, размера буфера обмена, также конфигурацию кэширования и еще множество других параметров, которые могут влиять на производительность базы данных.

Теперь разберем тему про резервное копирование. Этот аспект является критически важным, так как позволяет не потерять все данные и застраховать себя от различных ситуаций. При возможности нужно протестировать и проверить восстановление данных из резервной копии, не стоит игнорировать этот случай для своего же блага в будущем.

Теперь перейдем к мониторингу. Мониторинг - анализ и сбор информации о состоянии объекта. Мониторинг производительности базы данных: эта часть является необходимой для обеспечения постоянной работы приложения, который минимизирует риски появления сбоев в системе. Также используется для мониторинга ЦП, памяти, времени отклика запросов.

Еще одним важным аспектом администрирования базы данных является управление данными. Это может включать в себя следующие задачи:

Очистка данных: Необходимо регулярно проводить очистку данных, чтобы удалять дубликаты, исправлять ошибки и устаревшие данные. Это помогает сохранить базу данных точной и эффективной.

Архивирование данных: Архивирование данных является процессом перемещения старых или редко используемых данных из активной базы данных в архивное хранилище. Это может помочь улучшить производительность и сократить расходы на хранение данных, так как она будет содержать меньшее количество данных, которые необходимо обрабатывать. Также этот процесс может помочь сократить расходы на хранение данных, по причине того, что архивные хранилища как правило дешевле, нежели активные базы данных

Если представить администрирование БД и СУБД в кратком виде, то можно их написать так:

Администрирование БД:

- Создание объектов базы данных
- Разработка структуры системы безопасности
- Контролирование целостности данных
- Отслеживание производительности

СУБД:

- Важность установки и обновления версий
- Необходимо запустить и установить службы СУБД
- Настройка СУБД
- Возможность управление учетными записями пользователей
- Создание и модификация базы данных

Мы разобрались конечно не во всех, но в частности основных моментах администрирования базы данных, которые в будущем обязательно понадобятся в работе. Это лишь малая часть из того, что нужно для

обеспечения наилучшей эффективности бд, но все эти аспекты крайне важны для обеспечения надежной и эффективной работы приложения.

Глава 2. Проектирование физической модели базы данных и реализация

2.1. Проектирование физической модели базы данных

Прежде чем начать писать о том, как будет разрабатываться физическая модель базы данных, стоит затронуть тему выбора СУБД. Заострять на этом внимание не будем и если протолковать это коротко и ясно, то в качестве СУБД был выбран PostgreSQL. Не имеет никакого большого отличия от MySQL (кроме синтаксиса), просто выбрал из-за того, что нравится интерфейс (темная тема). Также Postgre при работе с Python (Django ORM) предоставляет некоторые дополнительные функции и это тоже стало одним из факторов выбора именно его, так как в будущем будет полезно знать об этих предоставляемых функциях.

После выбора наконец уже приступаем к самой разработке базы данных.

Начнем с самого создания базы данных:

```
CREATE DATABASE shop_db;
```

После создания базы данных, приступим к самим таблицам, которые нужно разработать:

1. Таблица Product. Данная таблица хранит информацию о продуктах. Начнем описание полей таблицы и типов данных, которые были использованы:

Поле [id] - хранит идентификатор продукта, SERIAL - значит, что наш id будет автоматически инкрементироваться. Также это поле у нас PRIMARY KEY (Первичный ключ), в будущем на это поле будут ссылаться другие

таблицы с помощью внешнего ключа (FOREIGN KEY). Обычно, принято использовать такие значения как: NOT NULL (не может иметь пустое значение) и указать тип данных INTEGER, но в случае использования PostgreSQL, нам этого делать не нужно, так как SERIAL автоматически устанавливает эти значения.

[title] - Название продукта, тип данных varchar(255), NOT NULL (не может иметь пустое значение).

[description] - TEXT, описание продукта, не должно иметь ограниченное количество символов.

[price] - DECIMAL(10, 2) - используется для указания точных значений, в нашем случае цена будет состоять из 8 чисел до запятой и 2 после.

[quantity] - количество товаров, INT и значение по умолчанию = 0

[images] - изображения продуктов, varchar для указания пути к картинке

[category_id] - Внешний ключ, который ссылается на таблицу Category.

SQL скрипт для создания таблицы Product:

```
CREATE TABLE Product (  
id SERIAL PRIMARY KEY,  
title varchar(255) NOT NULL,  
description TEXT,  
price DECIMAL(10, 2) NOT NULL,  
quantity INT CHECK(quantity > 0) NOT NULL,  
images varchar(255) NOT NULL,  
category_id INT REFERENCES Category(id));
```

ПРИМЕЧАНИЕ!!

На этом этапе специально максимально подробно было описано каждое поле, даже самые банальные. На следующих этапах, знакомые нам уже поля не будут описываться настолько подробно, так как мы уже с ними ознакомлены.

2. Таблица Category. Данная таблица хранит информацию о категориях товара.

[id] - идентификатор категории.

[name] - название категории

[description] - описание категории (необязательное поле, но для ясности, что к чему, будет полезно знать).

SQL скрипт для создания таблицы Category:

```
CREATE TABLE Category (  
id SERIAL PRIMARY KEY,  
name VARCHAR(255),  
description TEXT  
);
```

3. Таблица Role. Данная таблица хранит информацию о ролях пользователей. Не использовал ENUM, знаю, так можно было сделать, но все же было решено создать отдельную таблицу для хранения дополнительной информации, а также гибкости управления.

Поля те же самые, что и были описаны в таблице Category, только в этом случае хранятся данные о роли пользователя

SQL скрипт для создания таблицы Role:

```
CREATE TABLE Role (  
id SERIAL PRIMARY KEY,  
name varchar(255) NOT NULL,  
description TEXT  
);
```

4. Таблица Users. Данная таблица хранит информацию о пользователях.

Описывать будем только те поля, которые нам не знакомы. Все введенные данные о пользователе будут нужны при регистрации и авторизации.

[username] - имя пользователя, тип данных varchar.

[email] - email пользователя.

[password] - Пароль пользователя, хеширование пароля еще не реализовано, в будущем будет написано уже на бэкенде со специальными фреймворками.

[first_name], [last_name] - имя и фамилия пользователя, у обоих полей тип данных varchar.

[role_id] - Foreign Key, который ссылается на таблицу Role.

SQL скрипт для создания таблицы Users:

```
CREATE TABLE Users (  
  id SERIAL PRIMARY KEY,  
  username varchar(255) NOT NULL,  
  email varchar(255) UNIQUE NOT NULL,  
  password varchar(255) NOT NULL,  
  first_name varchar(255) NOT NULL,  
  images varchar(255),  
  last_name varchar(255) NOT NULL,  
  role_id INT REFERENCES Role(id)  
);
```

5. Таблица Orders. Хранит информацию о заказах (кем и когда было создано и какой статус у заказа)

[status] - Статус заказа, тип данных varchar.

[created_at] - Дата создания, тип данных Timestamp.

[updated_at] - Дата обновления, тип данных Timestamp.

[user_id] - Внешний ключ, который ссылается на таблицу Users.

SQL скрипт для создания таблицы Orders:

```
CREATE TABLE Orders (  
  id SERIAL PRIMARY KEY,  
  status varchar(255) NOT NULL,  
  created_at TIMESTAMP NOT NULL,
```

```
updated_at TIMESTAMP NOT NULL,  
user_id INT REFERENCES Users(id) );
```

6. Таблица OrderDetails. Хранит информацию о деталях заказа (какой товар был заказан, количество, цена)

[order_id] - Foreign Key, который ссылается на таблицу Orders

[product_id] - Foreign Key, который ссылается на таблицу Product

SQL скрипт для создания таблицы OrderDetails

```
CREATE TABLE OrderDetails (  
id SERIAL PRIMARY KEY,  
quantity INT CHECK(quantity > 0),  
price DECIMAL(10, 2),  
order_id INT REFERENCES Orders(id),  
product_id INT REFERENCES Product(id)  
);
```

7. Таблица Review. Данная таблица нужна для хранения информации об отзывах на товары.

[rate] - Отзыв, тип данных Float, выбрал этот тип данных, так как мне нужно точное значение отзыва. В ином случае можно было бы выбрать Decimal. Оценка должна варьироваться от 0 до 5.

[comment] - Комментарий к отзыву.

[user_id] - Foreign Key, который ссылается на таблицу Users

[product_id] - Foreign Key, который ссылается на таблицу Product

SQL скрипт для создания таблицы Review:

```
CREATE TABLE Review (  
id SERIAL PRIMARY KEY,  
rate FLOAT CHECK(rate >= 0 AND rate <= 5) NOT NULL,  
comment TEXT,  
created_at TIMESTAMP NOT NULL,  
user_id INT REFERENCES Users(id),  
product_id INT REFERENCES Product(id));
```

8. Таблица Cart (корзина). Эта таблица хранит в себе информацию о корзине (когда и кем было создано, дата и время последнего обновления).

[user_id] - Foreign Key, который ссылается на таблицу Users.

```
CREATE TABLE Cart (  
  Id SERIAL PRIMARY KEY,  
  user_id INT REFERENCES Users(id),  
  created_at TIMESTAMP NOT NULL,  
  updated_at TIMESTAMP NOT NULL  
);
```

9. Таблица CartDetails, таблица, которая хранит в себе информацию о деталях корзины (товар, количество товаров в корзине, цена)

[product_id] - Foreign Key, который ссылается на таблицу Product

[cart_id] - Foreign Key, который ссылается на таблицу Cart

```
CREATE TABLE CartDetails (  
  id SERIAL PRIMARY KEY,  
  quantity INT CHECK(quantity > 0),  
  price DECIMAL(10, 2),  
  product_id INT REFERENCES Product(id),  
  cart_id INT REFERENCES Cart(id)  
);
```

В итоге, мы создаем базу данных из 9 таблиц. ERD схема для данных таблиц будет предоставлена позже.

ERD - это графическое представление структуры базы данных, который предоставляет возможность детально рассмотреть и изучить все сущности и связи между ними. Сущности(таблицы) в ERD-схеме представляются в виде прямоугольников с названиями этих сущностей. Атрибуты(столбцы) - это свойства наших сущностей, они могут быть уникальными или неуникальными, а также иметь разные типы данных. В теории, читая все эти определения, может казаться, что ничего не понятно, поэтому лучше все рассматривать с практической части, так как такой подход больше

позволяет вникать в суть. На следующей странице как раз таки находится изображение ERD моей базы данных.

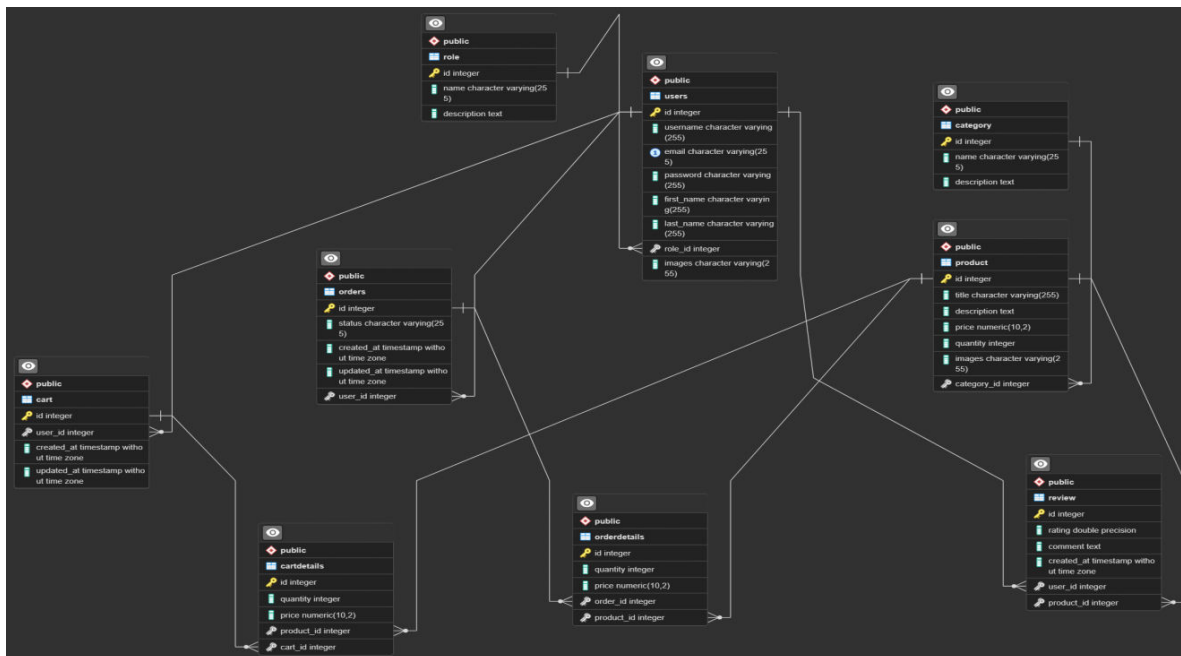


Рис 1.

В данной картинке мы имеем 9 таблиц связанных между собой линиями и элементами похожими на вороньи лапки. В теории мы обсуждали нотацию Crow's foot (вороньи лапки), которые уже использованы в нашей ERD-схеме для обозначения связей.

Как видно из нашего ERD, между всеми таблицами у меня реализовано связь один ко многим. Это значит, что одна запись из одной таблицы может иметь несколько записей из второй таблицы, но одна запись из второй таблицы может иметь только одну запись из первой.

Примеры: Отношение Users к Role (one-to-many): у одной роли может быть несколько пользователей, но у одного пользователя может быть только одна роль. Разберем немного сложное отношение в данной картине, отношение Cart к CartDetails (one-to-many): у одной корзины может быть несколько позиций(записей) из CartDetails, но у одной позиции может быть

только одна корзина. Аналогичная ситуация между отношениями Orders и OrderDetails.

2.2 Создание ролей и предоставление привилегий

Создание ролей играет важную роль в безопасности при разработке базы данных, так как можно контролировать доступ пользователей к определенным ресурсам и их возможности совершать какие-либо действия. Также, вообще можно создать роль 'read-only', в котором пользователь может просто читать данные и не более. В разработанной мною базе данных, было создано 3 роли: администратор, менеджер, покупатель. Всем троим были предоставлены разные привилегии над таблицами. Посмотрим на примере как создаются роли и как предоставляются привилегии.

Создание ролей:

```
CREATE ROLE admin WITH LOGIN PASSWORD '*****';  
CREATE ROLE customer WITH LOGIN PASSWORD '*****';  
CREATE ROLE manager WITH LOGIN PASSWORD '*****';
```

Вместо * каждому установлен свой пароль.

Предоставление доступа

```
Админу: GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Product  
TO admin;
```

...

И так дальше предоставляем лишь такие привилегии для нашего админа над всеми остальными таблицами. Решил не предоставлять права для CREATE, ALTER, DROP, TRUNCATE админу. Такие права остаются за суперпользователем.

Покупателю:

...

```
GRANT SELECT ON TABLE Product TO customer;  
GRANT SELECT ON TABLE Category TO customer;
```

...

Покупатель может делать только выборку данных из вышеуказанных таблиц.

Теперь перейдем к предоставлению определенных привилегий к роли менеджера. Для таблиц Product и Category manager может выполнять действия SELECT, INSERT, UPDATE, DELETE. А для других таблиц:

...

```
GRANT SELECT, UPDATE ON TABLE Orders TO manager;  
GRANT SELECT, UPDATE ON TABLE OrderDetails TO manager;  
GRANT SELECT ON TABLE Cart TO manager;  
GRANT SELECT ON TABLE CartDetails TO manager;
```

2.3. Типовые запросы

Типовые запросы - это запросы, которые используются для вывода определенной информации из базы данных. Для моей базы данных были написаны 8 типовых запросов. Все запросы рассматривать не будем, так как их много, разберем только самые интересные.

1. Вывод общей суммы заказов за определенный период.

В этом запросе мы используем SELECT для выборки данных, агрегатную функцию SUM для подсчета общей суммы, все эти данные мы берем из таблицы OrderDetails. Дату и время созданного заказа мы должны взять из таблицы Orders используя подзапрос. В подзапросе мы получаем id заказа который был создан в период от 4 апреля до 7 апреля, после получения срабатывает наш основной запрос.

```
SELECT SUM(quantity * price) as total_sum  
FROM OrderDetails  
WHERE order_id IN (SELECT id FROM Orders WHERE created_at  
                    BETWEEN '2024-04-01' AND '2024-04-07');
```

2. Вывод всех товаров в корзине пользователя. Выводим все товары и количество товаров из корзины с помощью SELECT, связываем таблицы Product и CartDetails с помощью JOIN. Связываться они будут по Primary Key в таблице Product и Foreign Key в таблице CartDetails. Теперь чтобы узнать корзину пользователя, нам нужно с помощью WHERE получить cart_id равный 1, который связан с определенным пользователем

```
SELECT Product.*, CartDetails.quantity  
FROM CartDetails  
JOIN Product ON Product.id = CartDetails.product_id  
WHERE CartDetails.cart_id = 1;
```

3. Вывод списка товаров, которые не были заказаны за определенный период. Выводим все товары с таблицы Product, где будут извлекаться все идентификаторы, которые не совпадают со значением внутри списка и созданные в период от 16 до 31 марта.

IN - условие, в котором будем получать все совпадающие со списком объекты.

NOT IN - условие, в котором будем получать все не совпадающие со списком объекты.

```
SELECT * FROM Product  
WHERE id NOT IN (SELECT product_id FROM OrderDetails  
WHERE order_id IN (SELECT id FROM Orders WHERE created_at  
BETWEEN '2024-03-16' AND '2024-03-31'));
```

4. Вывод списка пользователей, сделавших заказ в определенный период. Выводим всех пользователей с помощью SELECT и используя команды WHERE и IN получаем именно тех пользователей, которые совпадают указанным требованиям внутри скобок. В нашем случае мы должны вывести пользователей сделавших заказ с 1 по 7 апреля.

Разберем важные коман подзапрос: DISTINCT - ключевое слово, которое используется для получения только уникальных значений из столбцов

таблицы. BETWEEN - оператор, который позволяет указывать нужный нам диапазон для получения данных.

```
SELECT * FROM Users WHERE id IN (SELECT DISTINCT user_id FROM Orders WHERE created_at BETWEEN '2024-04-01' AND '2024-04-07');
```

2.4. Транзакции

Транзакции - предоставляют собой операции, которые выполняются как единое целое. Эти операции используются для обеспечения согласованности и целостности.

Транзакции имеют такие свойства как:

Атомарность (Atomicity) - Либо в транзакции выполнится все, либо не выполнится ничего.

Согласованность(Consistency) - Транзакция переводит базу данных из одного согласованного состояния в другое.

Изоляция(Isolation) - транзакция должна выполняться независимо от других транзакций выполняющихся в это же время.

Долговечность(Durability) - изменения, которые были сделаны транзакцией должны быть сохранены в базе данных и в случае сбоя или системного отказа не должны быть потеряны.

Пришло время рассмотреть транзакцию выполненную для моей базы данных. Рассматривать будем только одну транзакцию, чтобы сократить количество страниц

1. Создание заказа и добавление товаров в него. DO \$\$ является частью синтаксиса PostgreSQL для объявления блоков кода. DECLARE используем для объявления переменной order_id, которая будет хранить id нового заказа. BEGIN - начинаем транзакцию. Создаем заказ для таблицы

Orders, возвращаем id этого заказа и присваиваем order_id, затем добавляем нашу переменную и данные для OrderDetails.

```
DO $$
DECLARE
    order_id INTEGER;
BEGIN
    -- Создание нового заказа
    INSERT INTO Orders (status, user_id, created_at, updated_at)
    VALUES ('Не оплачено', 1, NOW(), NOW())
    RETURNING id INTO order_id;
    -- Добавление товаров в заказ
    INSERT INTO OrderDetails (order_id, product_id, quantity,
    price)
    VALUES
        (order_id, 1, 2, 1299.00),
        (order_id, 2, 1, 5000.00);
COMMIT;
END;
$;
```

2.5. Триггеры

Триггеры - это процедуры, которые автоматически срабатывают в ответ на определенные события в базе данных. В качестве событий могут выступать удаление, создание, обновления данных в таблицах.

Разберем триггер написанный для моей базы данных.

2. Обновление количества товаров в корзине после добавления или удаления товаров. Сначала создаем функцию, которая не имеет никаких параметров. RETURNS TRIGGER - мы будем возвращать функцию-триггер. \$\$ - начало блока кода. Объявляем переменную. Выводим общее количество из таблицы CartDetails, где id корзины равен новому значению

cart_id и все это присваиваем нашей переменной. Обновляем нашу таблицу, изменяя количество товаров и возвращаем новое значение. Создаем триггер, который будет срабатывать после вставки или удаления данных для каждой строки, в конце вызываем нашу функцию.

```
CREATE OR REPLACE FUNCTION update_product_quantity_in_cart()  
RETURNS TRIGGER AS $$  
DECLARE  
new_product_quantity INTEGER;  
BEGIN  
SELECT SUM(cd.quantity) INTO new_product_quantity  
FROM CartDetails AS cd  
WHERE cd.cart_id = NEW.cart_id;  
  
UPDATE CartDetails SET quantity = new_product_quantity  
WHERE cart_id = NEW.cart_id;  
RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER update_quantity  
AFTER INSERT OR DELETE ON CartDetails  
FOR EACH ROW  
EXECUTE FUNCTION update_product_quantity_in_cart();
```

2.6. Хранимые процедуры

Хранимые процедуры имеют возможность содержать в себе готовый код, хранящиеся в базе данных, которую при необходимости можно вызвать. Также они позволяют выполнять сложные операции с данными. Хранимые процедуры похожи на функции, могут также принимать параметры и возвращать значения, но основное различие между ними в том, что хранимые процедуры могут изменять данные в базе данных, выполняя такие операции INSERT, DELETE, UPDATE. Функции же возвращают результаты вычислений и не могут изменять данные. Есть еще ряд преимуществ использования хранимых процедур, но пора уже перейти к практике. В разработанной мною базе данных 3 хранимых функций(процедур), но рассмотрим только одну и самую интересную из них. Перед тем как приступить, нужно обратить внимание на то, что в PostgreSQL хоть и есть команда CREATE PROCEDURE, но ее на практике часто не используют. И в целом, вместо хранимых процедур используют хранимые функции, которые могут выполнять те же самые задачи.

1. Создаем хранимую функцию add_to_cart, который в качестве параметров имеет p_user_id – id пользователя, p_product_id – id товара и p_quantity - количество товара. RETURNS VOID - никакого значения наша функция возвращать не будет. \$\$ - начало блока кода.

Добавление товаров в корзину:

```
CREATE OR REPLACE FUNCTION add_to_cart(p_user_id INTEGER, p_product_id  
INTEGER, p_quantity INTEGER)
```

```
RETURNS VOID AS $$
```

```
...
```

2. С помощью DECLARE объявляем переменные cart_id и product_price. Первая переменная будет хранить в себе id корзины, а вторая цену товара.

BEGIN - начинаем работу. Получаем id из корзины, где user_id = текущему пользователю, который будет передавать аргументы в параметры, затем результат присваиваем переменной cart_id. Если корзины не существует, создаем новую.

```
DECLARE
cart_id INTEGER;
product_price DECIMAL(10, 2);
BEGIN
SELECT id INTO cart_id FROM Cart WHERE user_id =
    p_user_id;
IF cart_id IS NULL THEN
INSERT INTO Cart(user_id, created_at, updated_at)
VALUES (p_user_id, NOW(), NOW()) RETURNING id INTO
    cart_id;
END IF;
...
```

3. Получаем цену товара, где id нашего товара = id товару, который мы получим из параметра. После этого создаем корзину. Сохраняем изменения, завершаем нашу хранимую функцию и закрываем блок кода.

```
SELECT price INTO product_price FROM Product WHERE id =
p_product_id;
INSERT INTO CartDetails (quantity, price, product_id, cart_id)
VALUES (p_quantity, product_price, p_product_id, cart_id);
COMMIT;
END;
$$ LANGUAGE plpgsql;
```

2.7. Локальные переменные

Локальные переменные - используют для хранения временных значений, которые используются внутри блока хранимых процедур, триггеров и функций. Работают такие переменные только внутри блока кода, после завершения выполнения кода все удаляется. При разработке базы данных мною было создано 7 локальных переменных (total_price, product_count,

user_role, rating, comment, user_id, product_id), выполняющие определенные действия внутри блока кода

.

2.8. Представления

Представления позволяют создавать таблицы, на основе полученных результатов запроса. В моей базе данных создано 3 представления. Рассмотрим 1 из них.

1. Вывод всех заказов пользователя. Выводим данные из все[столбцов таблицы Orders, имя пользователя и количество товаров в заказе. Затем соединяем таблицы Orders и OrderDetails для получения данных о заказе. Затем Product с OrderDetails с целью получения информации о товаре. Наконец, Users с Orders для получения данных о пользователе, которому принадлежит определенное количество заказов.

```
CREATE VIEW user_orders AS
SELECT
o.id, o.user_id, o.created_at, o.updated_at, o.status,
p.id AS product_id, p.title, p.price,
u.username,
od.quantity
FROM Orders AS o
JOIN OrderDetails AS od ON od.order_id = o.id
JOIN Product AS p ON od.product_id = p.id
JOIN Users AS u ON u.id = o.user_id;
```

2.9. Условия

Условия в sql нужны для получения данных в запросе, которые соответствуют определенным критериям и с полученными данными позволяют выполнять разные операции. В процессе написания условий, я создал новые транзакции, не стал пользоваться старыми. В базе данных написанной мною, есть 2 транзакции, с условиями внутри блока кода. В первой транзакции мы проверяем наличия товаров на складе перед добавлением их в заказ, во второй проверяю наличие товара на складе перед добавлением его в корзину. В первой транзакции, если на складе есть товар, мы создаем заказ, иначе получаем ошибку 'Недостаточно товаров на складе'. Во второй транзакции такая же логика, если товар существует, создаем корзину, иначе получаем ошибку. см. Приложение 1

2.10. Обработчики исключений

Обработчики исключений в sql используют для того, чтобы отслеживать и обрабатывать ошибки, которые могут возникнуть во время выполнения кода. В рамках разработки базы данных для интернет-магазина, было написано 2 функции с обработчиком исключения в каждой из них. В первом случае, мы получим исключение при отсутствии товара на складе, а во втором, при добавлении отзыва, если такой отзыв уже существует.

2.11. Пользовательские функции

Это функции созданные пользователем для выполнения определенных задач, которые могут принимать аргументы и возвращать значения, а также выполнять сложные вычисления. В моей базе данных есть 2 пользовательских функций. Первая для расчета общей суммы заказа, вторая для расчета среднего рейтинга. Но рассмотрим только одну из них.

см. Приложение 2

Заключение

Подведем итоги - в процессе работы была разработана база данных для интернет-магазина одежды, который содержит необходимые свойства для полноценной работы. Решены задачи, которые были необходимы для достижения поставленной цели. Для проверки работоспособности базы данных, были добавлены тестовые данные, в ходе которых были получены ожидаемые результаты. Моя база данных спроектирована хорошо, для учебных целей, дополнительной практики, получения опыта и знаний, она подходит, но при этом приходит понимание того, что этого недостаточно, так как бизнес-логика на этом не заканчивается. К примеру, как минимум должна быть добавлена таблица с данными об оплате, так как для покупки, пользователи должны платить деньги, но пока из-за недостаточности знаний и практики(временно), реализовать не получилось. Не все разработано идеально, но моим ожиданиям соответствует. Еще многое нужно узнать и продолжать получать еще больше информации в этой сфере. Есть потенциальные возможности для развития и улучшения бизнес-процессов этой базы данных, а также при возможности использовать уже на реальной практике. По поводу безопасности бд - были созданы отдельные роли для каждого пользователя, чтобы определенный пользователь имел право совершать какие-либо действия, которые были дозволены ему. Такой подход защищает от непредвиденных действий злоумышленников. В процессе разработки, вспомнил многие забытые мною темы, что конечно же было полезно, также получил хороший багаж знаний и практический опыт.

Список источников

1. Документация PostgreSQL. Вся необходимая информация для написания второй главы, было взято из документации [Электронный ресурс] / - <https://postgrespro.ru/docs/postgresql/16/index>
2. Актуальность интернет-магазинов в наше время [Электронный ресурс] / - <https://www.ds77.ru/news/1063845/>
https://www.cossa.ru/synergy_academy/308100/
3. Пример базы данных интернет-магазина [Электронный ресурс] / - <https://fabric.inc/blog/commerce/ecommerce-database-design-example>
4. Отличия между физической и логической моделью базы данных [Электронный ресурс] / - <https://aws.amazon.com/ru/compare/the-difference-between-logical-and-physical-data-model/>
5. Связи между таблицами базы данных [Электронный ресурс] / - <https://habr.com/ru/articles/488054/>
6. Нотация “Crow’s foot” [Электронный ресурс] / - <https://vertabelo.com/blog/crow-s-foot-notation/>
7. Презентация на тему “Администрирование базы данных” [Электронный ресурс] / - <https://ppt-online.org/157132>
8. Что такое ERD [Электронный ресурс] / - <https://www.lucidchart.com/pages/ru/erd-%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0>
9. Что такое ACID [Электронный ресурс] / - <https://habr.com/ru/articles/555920/>
10. Нормализация данных и нормальные формы [Электронный ресурс] / - <https://www.kaznu.kz/content/files/news/folder22759/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%205%20.pdf> HYPERLINK
"https://www.kaznu.kz/content/files/news/folder22759/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%205%20.pdf" HYPERLINK
"https://www.kaznu.kz/content/files/news/folder22759/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%205%20.pdf" HYPERLINK
"https://www.kaznu.kz/content/files/news/folder22759/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%205%20.pdf"/


Приложение 1. Условия. Проверка наличия товара на складе перед добавлением его в заказ:

```
DO $$
DECLARE
    product_quantity INTEGER;
BEGIN
    SELECT quantity INTO product_quantity FROM Product WHERE
id = 1;
    IF product_quantity >= 1 THEN
        INSERT INTO OrderDetails (quantity, price, order_id,
product_id)
            VALUES (2, 1299.00, 2, 1);
        UPDATE Product SET quantity = quantity - 2 WHERE id=1;
    ELSE
        RAISE NOTICE 'Недостаточно товаров на складе';
    END IF;
    COMMIT;
END $$;
```

Приложение 2. Пользовательские функции. Создание функции для расчета общей суммы заказа:

```
CREATE OR REPLACE FUNCTION calculate_total_price(order_id
INTEGER)
RETURNS DECIMAL(10, 2) AS $$
DECLARE
    total_price DECIMAL(10, 2);
BEGIN
    SELECT SUM(od.quantity * od.price) INTO total_price
    FROM OrderDetails AS od
    WHERE od.order_id = order_id;
    RETURN total_price;
END;
$$ LANGUAGE plpgsql;
```

Приложение 3. 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 № 8759857

Дата загрузки: 2024-05-26 21:07:10
Пользователь: aidarbekovadahan8@gmail.com, ID: 8759857

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

Информация о документе
№ документа: 8759857
Имя исходного файла: Айдарбеков Адахан. Курсовая работа.pdf
Размер файла: 0.58 МБ
Размер текста: 0
Слов в тексте: 0
Число предложений: 0

Информация об отчете
Дата: 2024-05-26 21:07:10 - Последний готовый отчет
Оценка оригинальности: 98%
Заимствования: 2%

98.19%

1.81%

Источники:

Доля в тексте	Ссылка
10.30%	https://habr.com/ru/articles/488054/
6.70%	https://www.internet-technologies.ru/articles/rukovodstvo-po-raz...

Приложение 4. QR код на репозиторий



Ссылка на репозиторий: https://github.com/adeqoou/pgdb_work