Національний університет «Львівська політехніка»

Кафедра програмного забезпечення

**КУРСОВА РОБОТА**

**з дисципліни «Бази даних»**

на тему:

«Інформаційна система для магазину настільних ігор»

Виконала:  
студентка спеціальності 121

«Інженерія програмного забезпечення»

групи ПЗ-33

Біловус А.Ю.

Керівник:

асистентка кафедри програмного забезпечення

Майхер В.Ю.

Оцінка:

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів \_\_\_\_\_\_\_ Оцінка ECTS\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_ Білоіваненко М.В.

(підпис)

\_\_\_\_\_\_\_\_\_\_\_\_ Майхер В.Ю.

(підпис)

Львів – 2024 рік

# **РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ БАЗИ ДАНИХ ДЛЯ РОБОТИ МАГАЗИНУ НАСТІЛЬНИХ ІГОР**

## **1.1. Опис предметної області Магазину Настільних ігор**

Магазин настільних ігор — це не просто місце для продажу ігор, але і осередок спільноти, де створюється атмосфера, сприятлива для вибору, тестування та покупки настільних ігор. Основною метою магазину є забезпечення клієнтів різноманітним асортиментом настільних ігор, які відповідають їхнім інтересам, віковим категоріям і ігровим вподобанням.

У магазині настільних ігор працюють продавці-консультанти, які добре знають різні типи ігор та можуть порекомендувати клієнтам найкращі варіанти, виходячи з їхніх інтересів, досвіду та побажань. Вони також організовують демонстраційні ігри та події, щоб клієнти могли випробувати різні ігри перед покупкою. Постачальники забезпечують магазин новими релізами і класичними іграми. Адміністратори відповідають за електронну систему управління магазином, стежачи за базою даних, обліковими записами клієнтів, запасами та ігровими подіями.

Ця предметна область має високу актуальність у сучасному світі з кількох причин. По-перше, настільні ігри є інструментом соціальної взаємодії, пропонуючи можливість для спілкування, відпочинку та розвитку навичок. По-друге, ринок настільних ігор зростає завдяки зростаючому інтересу до альтернативних форм розваг, що надає великі можливості для розвитку бізнесу.

*Встановлення обсягів аналізу теми для магазину одягу*:

Для обмеження обсягу досліджень у курсовій роботі, відповідно до створеної бази даних, аналіз теми буде фокусуватися на наступних аспектах:

Обсяг предметної області для аналізу магазину настільних ігор включає:

* Ринок настільних ігор: Дослідження тенденцій в галузі, які впливають на попит, включаючи популярність жанрів ігрових систем.

## Визначення ключових понять: Ідентифікація понять, таких як клієнт, гра, жанр, вік, складність і транзакція купівлі-продажу.

## Аналіз вимог користувачів:

## Потреби клієнтів: Аналіз очікувань та потреб клієнтів, враховуючи різні вікові групи, вподобання ігрового процесу, а також частоту гри.

## Визначення функціональних і нефункціональних вимог: Встановлення функціональності системи обліку та управління магазином, яка задовольнятиме потреби клієнтів і співробітників.

## Оцінка поточного стану:

## Аналіз існуючих систем управління: Вивчення функціональності та ефективності поточних систем управління магазинами настільних ігор.

## Оцінка переваг та недоліків: Визначення переваг і недоліків існуючих систем, щоб визначити можливості для покращення.

## Визначення перспектив розвитку:

## Новітні технології: Аналіз технологій у сфері роздрібної торгівлі та інформаційних систем, що можуть оптимізувати управління магазином настільних ігор.

## Перспективи розвитку: Визначення можливих напрямків розвитку для покращення конкурентоспроможності та задоволення зростаючих потреб клієнтів.

## **1.2. Вимоги до обробки даних**

## У процесі бізнесу магазину настільних ігор відбуваються такі ключові бізнес-процеси, як обробка замовлень покупців, аналіз продажів і тенденцій, обробка повернень, а також управління користувачами та взаємодія з ними. Кожен з цих процесів має свої унікальні характеристики та вимоги до обробки даних.

## Для ефективного управління магазином настільних ігор потрібна система збору та обробки даних, яка забезпечує зручний доступ і систематичний аналіз для прийняття стратегічних рішень. Це включає збір та зберігання даних про клієнтів, товари, замовлення, а також інформацію про повернення, відгуки та транзакції. Дані мають бути ретельно організовані та захищені для забезпечення конфіденційності та безпеки.

## Постачання товарів до магазину вимагає відстеження запасів, їх розташування на складі, а також даних про виробників. Система управління запасами повинна забезпечувати ефективне відстеження товарів, їх кількість, характеристики, категорії та інші деталі.

## Обробка замовлень покупцями потребує точної та швидкої обробки інформації про вибрані товари, контактні дані клієнтів, а також деталі замовлень. Для цього потрібна надійна система обробки замовлень, яка дозволить швидко реагувати на потреби клієнтів.

## Аналіз тенденцій продажів та товарного потоку допомагає приймати обґрунтовані рішення щодо асортименту, акцій та стратегій ціноутворення. Системи збору та аналізу даних мають забезпечувати можливість відстеження продажів, рентабельності, популярності товарів, а також аналізу відгуків і повернень.

## Обробка даних про користувачів, включаючи їх електронну пошту, паролі та ролі, повинна бути безпечною та захищеною від несанкціонованого доступу. Ці дані використовуються для управління доступом до системи, встановлення ролей і взаємодії з клієнтами.

## Таким чином, ефективне управління магазином настільних ігор вимагає системи обробки даних, яка забезпечує швидкий доступ до важливої інформації, точність, безпеку та здатність до гнучкого аналізу. Ці вимоги сприяють ефективному функціонуванню магазину, задоволенню потреб клієнтів та підтримці конкурентоспроможності в сучасному ринковому середовищі.

## **1.3. Постановка завдання**

# Система управління магазином настільних ігор повинна забезпечувати ефективне ведення інформаційного обліку та управління всіма аспектами діяльності магазину. Для цього необхідно реалізувати такі основні функції:

# Ведення обліку товарів:

# Додавання, оновлення та видалення записів про настільні ігри в базі даних.

# Зберігання інформації про ціни, жанри, категорії, виробників та інші характеристики товарів.

# Управління замовленнями:

# Зберігання даних про замовлення клієнтів, включаючи інформацію про товари, кількість, ціни, дати замовлень, статуси.

# Можливість редагування замовлень та їх статусів (обробка, доставка, завершення).

# Збір та аналіз даних:

# Збір і зберігання інформації про замовлення та їхню загальну вартість за певний період часу.

# Визначення найпопулярніших настільних ігор та категорій.

# Аналіз тенденцій у продажах для прийняття стратегічних рішень щодо асортименту товарів, управління ціноутворенням та розробки маркетингових стратегій.

# Управління користувачами:

# Додавання, оновлення та видалення записів про користувачів, включаючи їх електронну пошту, пароль, ролі та пов’язані з ними клієнти.

# Зберігання даних про користувачів, їх ролі (наприклад, "Normal User" або "Admin User"), а також пов’язана з ними інформація про клієнтів.

# Звітність та відгуки:

# Можливість створення та завантаження PDF-документів, що містять інформацію про товари, замовлення, звіти щодо продажів та інші аспекти діяльності магазину.

# Можливість отримання та обробки відгуків клієнтів про настільні ігри, з можливістю публікації відгуків на веб-сайті магазину або в системі внутрішнього аналізу.

# Ці функції забезпечують комплексний підхід до управління магазином настільних ігор, забезпечуючи точне відстеження товарів, обробку замовлень, аналіз продажів, ефективне управління користувачами, а також створення різноманітної звітності для підтримки ефективної роботи магазину та прийняття стратегічних рішень.

# **РОЗДІЛ 2. РОЗРОБКА МОДЕЛЕЙ ТА ЗБЕРІГАННЯ ДАНИХ**

## **2.1. Концептуальне проектування**

Концептуальне проектування для магазину настільних ігор охоплює різні аспекти даних, що відображають основні бізнес-процеси. Ось основні елементи, які треба врахувати в контексті вашої бази даних:

1. *Постачальники (Manufaturers):* Представляють компанії або особи, які виробляють настільні ігри. Відстежування виробників важливо для управління асортиментом і контактними даними. Зв'язок з таблицею Products показує, які товари постачає кожен виробник.
2. *Продукти (Products):*Включають інформацію про настільні ігри, їхні жанри, категорії, виробників, ціни та інші характеристики. Зв'язок із таблицею Manufaturers вказує на виробника кожної гри.
3. *Замовлення (Orders):* Представляють інформацію про замовлення, зроблені клієнтами. Включають дані про замовлені товари, їхню кількість, ціни, дати замовлення, та статуси. Зв'язок з таблицею OrderDetails вказує на склад кожного замовлення.

## *Деталі замовлення (OrderDetails):* Представляють товари, які входять до складу замовлень. Зв'язок з таблицею Products показує, які ігри були замовлені.

## *Клієнти (Customers):* Відображають інформацію про клієнтів магазину, їхні контактні дані, адреси та інші особисті дані. Зв'язок з таблицею Orders показує, які замовлення зробив кожен клієнт.

## *Програми лояльності (CustomerLoyalty):* Включають інформацію про програми лояльності для клієнтів, такі як бонусні бали та рівні лояльності. Зв'язок з таблицею Customers вказує, які клієнти мають програми лояльності.

## *Користувачі (Users):* Представляють інформацію про користувачів системи, їхні електронні адреси, паролі, ролі та дату створення акаунта. Зв'язок із таблицею Customers дозволяє вказати, до якого клієнта прив'язаний кожен користувач.

## *Відгуки (Reviews):* Включають відгуки клієнтів про настільні ігри, з рейтингом і коментарями. Зв'язок з таблицею Products показує, про які ігри залишені відгуки.

## *Транзакції (Transactions):* Інформація про фінансові транзакції, пов'язані з замовленнями. Включає суму, дату та тип транзакції. Зв'язок із таблицею Orders показує, які замовлення були оплачені через ці транзакції.

## *Повернення (ReturnRequests):* Відображають інформацію про повернення товарів, включаючи причини повернення та їх статус. Зв'язок із таблицею OrderDetails вказує, які товари були повернуті.

## *Знижки (Discounts):* Включають інформацію про знижки та акції, які пропонуються клієнтам. Зв'язок із таблицею Products показує, які товари мають знижки.

## Ці компоненти концептуального проектування забезпечують комплексний підхід до управління даними для магазину настільних ігор, дозволяючи ефективно обробляти замовлення, відстежувати товари, взаємодіяти з клієнтами, а також аналізувати продажі та маркетингові стратегії.

## **2.2. Вимоги до системи накопичення даних**

## *Оцінка обсягів даних:*

## Обсяги даних у магазині настільних ігор можуть варіюватися від помірних до великих. Таблиця Products може містити значну кількість записів, враховуючи різноманітність асортименту настільних ігор, їх категорії, виробників і жанри. Таблиці Orders і Customers також можуть бути великими, оскільки вони відображають замовлення та клієнтів відповідно. Інші таблиці, такі як CustomerLoyalty, ReturnRequests, Reviews і Transactions, можуть містити менше записів, але все одно можуть зростати разом з розвитком бізнесу.

## *Частота додавання даних:*

## Дані про настільні ігри можуть додаватися частіше, оскільки магазин регулярно оновлює свій асортимент або додає нові ігри. Інформація про замовлення також може додаватися часто, залежно від обсягу продажів і активності клієнтів. Додавання даних до таблиці CustomerLoyalty буде рідшим, оскільки вона стосується програм лояльності, а ReturnRequests залежить від кількості повернень товарів.

## *Очікувані запити та вибірки:*

## Система дозволяє виконувати різноманітні запити та вибірки для управління даними магазину настільних ігор. Можна виконувати запити на перегляд інформації про всі товари в магазині, а також отримувати дані про замовлення, їх склад та статус. Також можна додавати нові товари, оновлювати ціни та характеристики, і обробляти замовлення клієнтів. Очікувані запити можуть включати статистику продажів, кількість замовлень за певний період та вартість проданих товарів. Ці можливості допомагають керувати даними та аналізувати тенденції продажів для прийняття стратегічних рішень.

## *Обмеження доступу до даних:*

## Для забезпечення безпеки даних необхідно встановити обмеження доступу відповідно до ролей користувачів. В системі можуть бути три типи ролей:

## Гість (Guest): Мають обмежений доступ, можуть лише переглядати дані.

## Адміністратор (Admin): Мають повний доступ до системи. Вони можуть додавати, видаляти та редагувати будь-які дані, а також управляти структурою бази даних, бачити аналітику та завантажувати звітність.

## Користувач (User): Мають обмежений доступ, може переглядати дані, робити замовлення та переглядати історію своїх замовлень.

## Ці обмеження допомагають захистити дані магазину настільних ігор та контролювати доступ користувачів до різних частин системи відповідно до їхніх обов'язків і повноважень

## **2.3. Логічне проектування схеми бази даних.**

## *Таблиця Manufaturers (Виробники):*

## manufacturers\_id (INT): Унікальний ідентифікатор виробника.

## mname (VARCHAR(255)): Назва виробника.

## contact\_phone (VARCHAR(15)): Телефонний номер виробника.

## country (VARCHAR(255)): Країна походження виробника.

## web\_site (VARCHAR(255)): Веб-сайт виробника.

## *Таблиця Products (Товари):*

## product\_id (INT): Унікальний ідентифікатор товару.

## pname (VARCHAR(255)): Назва настільної гри.

## pprice (DECIMAL(10, 2)): Ціна настільної гри.

## genre (VARCHAR(50)): Жанр настільної гри.

## category (VARCHAR(50)): Категорія настільної гри.

## manufacturers\_id (INT): Зовнішній ключ, посилається на таблицю Manufaturers.

## image\_url (VARCHAR(255)): Посилання на зображення товару.

## *Таблиця Customers (Клієнти):*

## customer\_id (INT): Унікальний ідентифікатор клієнта.

## cname (VARCHAR(50)): Ім'я клієнта.

## email (VARCHAR(255)): Електронна адреса клієнта.

## phone\_number (VARCHAR(15)): Телефонний номер клієнта.

## address (VARCHAR(255)): Адреса клієнта.

## *Таблиця CustomerLoyalty (Програми лояльності):*

## loyalty\_id (INT): Унікальний ідентифікатор програми лояльності.

## customer\_id (INT): Зовнішній ключ, посилається на таблицю Customers.

## bonus\_points (INT): Кількість бонусних балів.

## level (VARCHAR(50)): Рівень лояльності.

## *Таблиця Orders (Замовлення):*

## order\_id (INT): Унікальний ідентифікатор замовлення.

## customer\_id (INT): Зовнішній ключ, посилається на таблицю Customers.

## odate (DATE): Дата оформлення замовлення.

## ostatus (VARCHAR(50)): Статус замовлення.

## *Таблиця OrderDetails (Деталі замовлення):*

## order\_details\_id (INT): Унікальний ідентифікатор деталей замовлення.

## order\_id (INT): Зовнішній ключ, посилається на таблицю Orders.

## product\_id (INT): Зовнішній ключ, посилається на таблицю Products.

## amount (INT): Кількість замовлених товарів.

## odprice (DECIMAL(6, 2)): Ціна за товар у замовленні.

## *Таблиця Reviews (Відгуки):*

## review\_id (INT): Унікальний ідентифікатор відгуку.

## product\_id (INT): Зовнішній ключ, посилається на таблицю Products.

## customer\_id (INT): Зовнішній ключ, посилається на таблицю Customers.

## rating (INT): Рейтинг настільної гри.

## rcomment (TEXT): Коментар до відгуку.

## *Таблиця ReturnRequests (Повернення):*

## return\_request\_id (INT): Унікальний ідентифікатор запиту на повернення.

## order\_id (INT): Зовнішній ключ, посилається на таблицю Orders.

## reason (VARCHAR(255)): Причина повернення.

## rstatus (VARCHAR(50)): Статус повернення.

## *Таблиця Transactions (Транзакції):*

## transaction\_id (INT): Унікальний ідентифікатор транзакції.

## order\_id (INT): Зовнішній ключ, посилається на таблицю Orders.

## amount (DECIMAL(10, 2)): Сума транзакції.

## date (DATE): Дата транзакції.

## type (VARCHAR(50)): Тип транзакції (наприклад, покупка, повернення).

## *Таблиця Users (Користувачі):*

## user\_id (INT): Унікальний ідентифікатор користувача.

## customer\_id (INT): Зовнішній ключ, посилається на таблицю Customers.

## uemail (VARCHAR(255)): Електронна адреса користувача.

## password (VARCHAR(50)): Пароль користувача.

## role (VARCHAR(20)): Роль користувача (наприклад, "Admin", "Editor").

## created\_at (TIMESTAMP): Дата створення акаунта.

## *Зв’язки між таблицями:*

## *Customers та Orders:* Кожен клієнт може мати багато замовлень (один до багатьох). Кожне замовлення належить лише одному клієнту (багато до одного).

## *Orders та OrderDetails:* Кожне замовлення може мати багато деталей (один до багатьох). Кожна деталь замовлення належить лише одному замовленню (багато до одного).

## *OrderDetails та Products:* Кожен товар може бути частиною багатьох деталей замовлення (один до багатьох). Кожна деталь замовлення належить лише одному товару (багато до одного).

## *Products та Manufaturers:* Кожен товар належить певному виробнику (багато до одного). Кожен виробник може постачати багато товарів (один до багатьох).

## *Customers та Reviews:* Кожен клієнт може залишати багато відгуків (один до багатьох). Кожен відгук належить лише одному клієнту (багато до одного).

## *Products та Reviews:* Кожен продукт може мати багато відгуків (один до багатьох). Кожен відгук стосується лише одного продукту (багато до одного).

## **2.4. Реалізація процедур бізнес-логіки.**

*Процедура UpdateOrderTotalAmount*

Оновлює загальну суму замовлення після додавання, видалення або оновлення елементів замовлення. Приймає параметр @order\_id і обчислює нову загальну суму, використовуючи деталі замовлення.

*Тригер UpdateTotalAmountAfterInsert*

Спрацьовує після вставки нового запису в таблицю OrderDetails. Викликає процедуру UpdateOrderTotalAmount з order\_id нового запису.

*Тригер UpdateTotalAmountAfterDelete*

Спрацьовує після видалення запису з таблиці OrderDetails. Викликає процедуру UpdateOrderTotalAmount з order\_id видаленого запису.

*Тригер UpdateTotalAmountAfterUpdate*

Спрацьовує після оновлення запису в таблиці OrderDetails. Викликає процедуру UpdateOrderTotalAmount з order\_id оновленого запису.

*Функції для формування статистик:*

*Функція GetOrderCountsByMonth:*

Ця функція повертає кількість замовлень за кожен місяць протягом останніх 12 місяців. Вона використовується для аналізу динаміки замовлень та виявлення тенденцій. Функція приймає враховує період від поточної дати до 6 місяців тому і обчислює кількість замовлень для кожного місяця за цей період.

*Функція GetMostPurchasedProduct:*

Ця функція повертає назву товару, який найчастіше замовляли клієнти. Вона допомагає виявити популярність та попит на конкретні товари. Функція обчислює кількість замовлень для кожного товару та повертає назву товару з найбільшою кількістю замовлень.

*Функція GetTotalSalesForLastMonth:*

Ця функція повертає загальну суму продажів за останній місяць. Вона використовується для визначення обсягу продажів та доходу магазину за визначений період. Функція обчислює суму всіх продажів за попередній місяць та повертає цю суму.

*Функція GetPriceTotalAmountByMonth:*

Ця функція повертає суму ціни замовлень за кожен місяць протягом визначеного періоду. Вона використовується для аналізу обсягу продажів та доходу магазину протягом року. Функція обчислює суму всіх замовлень за кожен місяць і повертає ці дані.

Індекси

*idx\_Products\_manufacturers\_id: Індекс для швидкого пошуку товарів за виробником.*

*idx\_Orders\_customer\_id: Індекс для швидкого пошуку замовлень за клієнтами.*

*idx\_OrderDetails\_order\_id: Індекс для швидкого пошуку деталей замовлень.*

*idx\_OrderDetails\_product\_id: Індекс для швидкого пошуку деталей замовлення за товарами.*

*idx\_Reviews\_product\_id: Індекс для швидкого пошуку відгуків про певні товари.*

Ці індекси допомагають покращити продуктивність запитів, функцій та процедур, оскільки вони полегшують швидкий доступ до даних у таблицях, що мають відносини між собою через зовнішні ключі.

# **РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО КОДУ**

## **3.1. Обґрунтування обраної архітектури.**

*Тип архітектури.*

Проєкт побудовано на архітектурі REST API. REST (Representational State Transfer) — це архітектурний стиль для розробки веб-сервісів, де ресурси ідентифікуються унікальними URI, а взаємодія з ними здійснюється через стандартні операції HTTP (GET, POST, PUT, DELETE). Ось деякі ключові принципи, що використовуються в архітектурі REST для побудови веб-служб:

* Уніфікований інтерфейс: Кожен ресурс має унікальний URI, а взаємодія з ресурсами здійснюється через стандартні операції HTTP.
* Клієнт-серверна архітектура: Клієнт відповідає за інтерфейс користувача, а сервер — за зберігання та обробку даних.
* Stateless (без збереження стану): Кожен запит до сервера містить всю інформацію, необхідну для його обробки, без збереження контексту між запитами.
* Кешування: Сервер може позначати ресурси як кешовані, що дозволяє зменшити навантаження та збільшити продуктивність.
* Багаторівнева система: Архітектура може складатися з кількох рівнів, кожен з яких має певну роль у взаємодії між клієнтом і сервером.

*Мова програмування та фреймворки.*

* React: Потужна бібліотека для створення користувацьких інтерфейсів на фронтенді. Використання компонентного підходу забезпечує модульність, перевикористання та ефективність коду.
* Bootstrap 5: Популярний фреймворк для розробки інтерфейсів, який надає широкий набір стилів та компонентів для швидкого створення зручних інтерфейсів.
* ASP.NET Core: Фреймворк для розробки серверної частини на C#. ASP.NET Core надає потужний інструментарій для створення REST API, що включає маршрутизацію, обробку запитів та відповідей, аутентифікацію та авторизацію.
* Entity Framework Core: ORM-бібліотека для взаємодії з базою даних у C#. Entity Framework дозволяє працювати з базою даних на об'єктному рівні, що полегшує процес розробки та підтримки коду.
* PostgreSQL: Потужна реляційна система управління базами даних, яка забезпечує надійне зберігання та ефективне оброблення даних.

*Система баз даних.*

Для зберігання та управління даними у проекті використовується система управління базами даних PostrgeSQL. Вона надає такі переваги:

* Високу продуктивність: Оптимізований рушій баз даних дозволяє швидко виконувати запити та обробляти великі обсяги даних.
* Масштабованість: PostgreSQL підтримує горизонтальне та вертикальне масштабування, що дозволяє збільшувати обсяги даних та потужність зростання навантаження.
* Безпеку: База даних має вбудовані механізми захисту, включаючи контроль доступу, шифрування даних, та аудит.
* Транзакційність: PostgreSQL підтримує транзакції, що дозволяє забезпечити цілісність та узгодженість даних.

Підтримку ORM: PostgreSQL повністю сумісний з Entity Framework Core, що полегшує взаємодію з базою даних на об'єктному рівні.Отже, Microsoft SQL Server є потужною та надійною системою управління базами даних, яка надає широкі можливості для організації, зберігання та обробки даних.

## **3.2. Структурна модель інформаційної системи.**

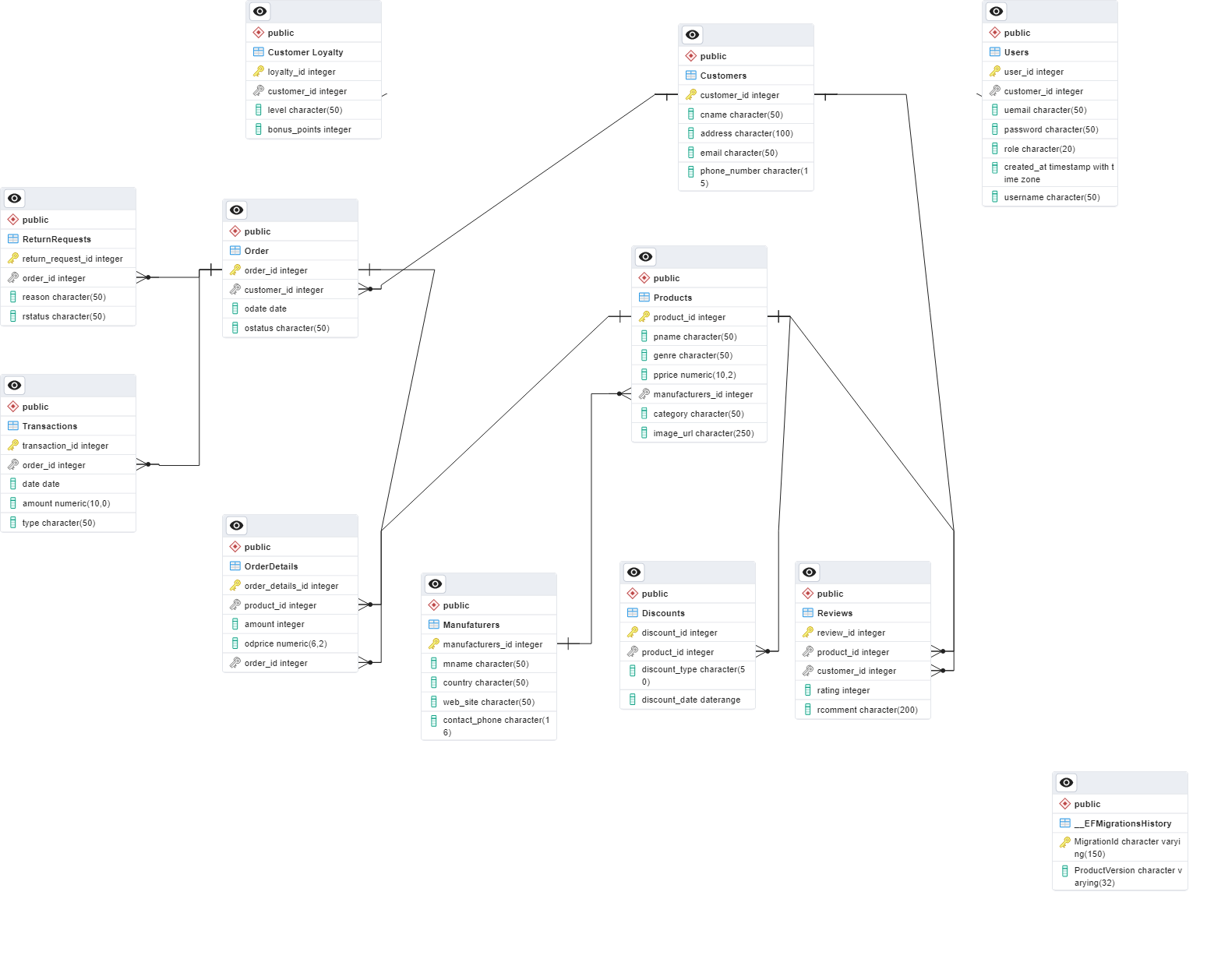


Рис. 3.2.1. Структурна модель інформаційної системи (бази даних)

## **3.3. Призначення модулів та компонентів системи.**

Загальна структура проекту поділяється на серверну частину (Backend) та клієнтську (Frontend).

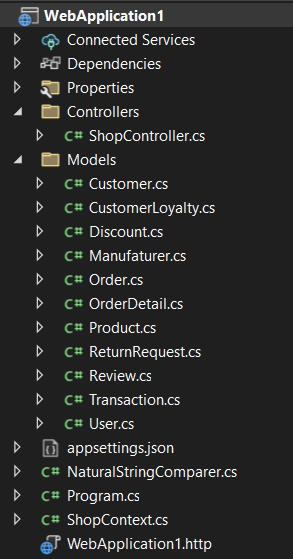


Рис. 3.3.1. Загальна структура проекту.

Бекенд, написаний на С# з допомогою ASP.NET та EntityFramework, відповідає за отримання запитів з клієнтської частини і відповідну взаємодію з базою даних – PostgreSQL. Структура серверної частини поділяється на такі основні модулі – контролер, моделі, контекст.

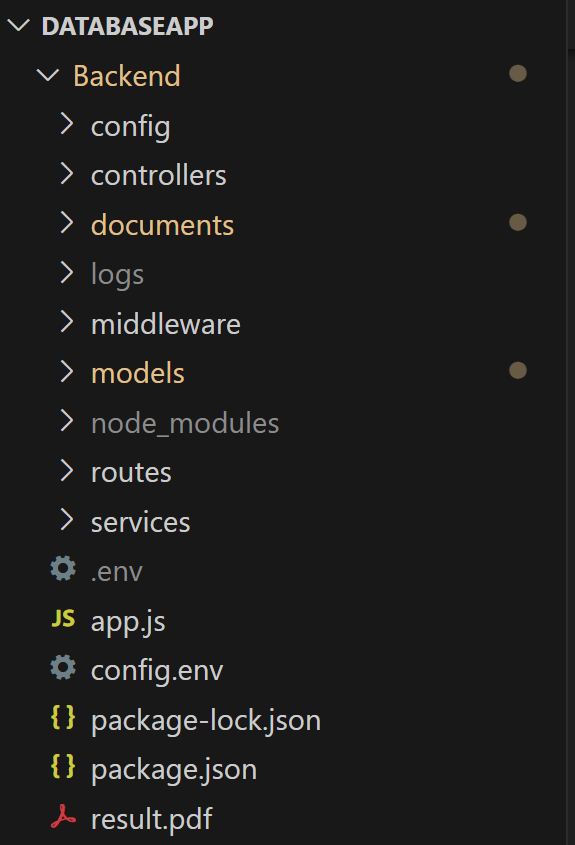


Рис. 3.3.2. Загальна структура серверної частини.

*Контролер (Controllers)* відіграє важливу роль у обробці запитів, що надходять від клієнта. Він відповідає за прийняття даних, їх обробку та подальшу передачу до відповідних частин системи. Він забезпечує взаємодію між клієнтом та сервером, виконуючи необхідні операції та відповідаючи на запити користувачів.

*Моделі (Models)* представляють собою структури даних або об'єкти, які відображають сутності в базі даних. Вони відповідають за взаємодію з базою даних та виконання різноманітних операцій з даними, таких як створення, зчитування, оновлення та видалення записів. Моделі також можуть містити логіку, пов'язану з операціями бази даних, а також валідацію даних перед їх збереженням, що робить їх важливою складовою серверної частини.

*Контекст(Context)* представляє собою клас контексту даних, який використовує Entity Framework для взаємодії з базою даних. Він виступає як місток між вашими об'єктами C# і таблицями бази даних, дозволяючи вам виконувати CRUD-операції (створення, читання, оновлення та видалення) за допомогою об'єктно-орієнтованого підходу.Фронтенд, написаний на React, відповідає за надання інтерфейсу користувача, взаємодію з користувачем, отримання запитів від користувача та взаємодію з сервером.

Структура клієнтської частини поділяється на такі основні модулі – компоненти, стилі.

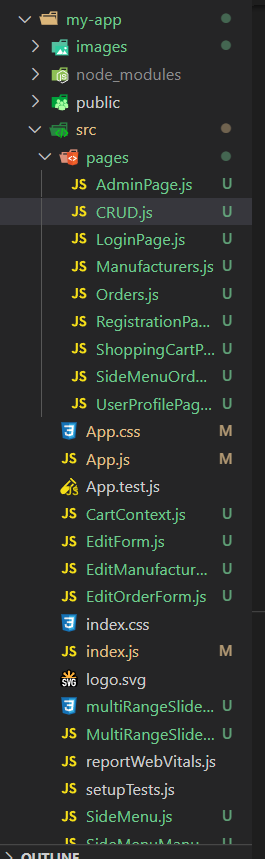


Рис. 3.3.3. Загальна структура клієнтської частини.

*Компоненти* відповідають за розділення інтерфейсу користувача на невеликі, повторно використовувані частини. Кожен компонент має свою власну логіку, розмітку та стилі, і вони використовуються для побудови складних інтерфейсів.

*Стилі* відповідають за зовнішній вигляд компонентів та інтерфейсу в цілому. Вони включають в себе визначення кольорів, шрифтів, розмірів, відступів та інших атрибутів, які впливають на вигляд елементів на сторінці.

## **3.4. Особливості реалізації та елементи інтерфейсу.**

1. Сторінка логіну:

* На сторінці логіну користувач буде зобов'язаний ввести свої облікові дані для входу в систему.
* Введені дані будуть перевірені на правильність.
* Після успішної аутентифікації користувача буде перенаправлено на головну сторінку додатку.

1. Таблиці:

* Додаток має відображати дані з бази даних у вигляді таблиць для зручного перегляду та управління.
* Адміністратор може переглядати, додавати, редагувати та видаляти записи з таблиць. Для додавання та редагування записів появлятиметься модальне вікно.

1. Статистика:

* Додаток може відображати статистику про загальну кількість замовлень, загальний прибуток від проданих товарі за певний період часу.
* Додаток відображає іншу статистику: категорія товару, яку найчастіше замовляли, та кількість користувачів що зареєструвались за кожен місяць.
* Інформація має бути представлена у вигляді графіків або інших візуальних елементів для зручного аналізу.

1. Можливість вивести дані таблиць у інші формати:

* Користувач може мати можливість експортувати дані з таблиць у різноманітні формати, такі як PDF або JSON.
* Це дозволить користувачам зберігати, обробляти та аналізувати дані за допомогою сторонніх програм.

# **РОЗДІЛ 4. НАПОВНЕННЯ БАЗИ ДАНИХ**

## **4.1. Опис джерела історичних даних.**

Для наповнення бази даних написала код що генерує бажану кількість даних для моїх таблиць.

Наприклад:

-- Генерація випадкових даних для Customers

DO $$

DECLARE

n\_rows INT := 30; -- кількість рядків для генерації

start\_id INT := 100; -- початковий ID

BEGIN

FOR i IN 0..n\_rows-1 LOOP

INSERT INTO public."Customers"

(

customer\_id,

cname,

address,

email,

phone\_number

)

VALUES

(

start\_id + i, -- ID починаючи зі 100

'Customer ' || (start\_id + i), -- ім'я

'Address ' || (start\_id + i), -- адреса

'customer' || (start\_id + i) || 'email@example.com', -- електронна пошта

'+380' || '98' || (FLOOR(RANDOM() \* 9000000) + 1000000) -- номер телефону

);

END LOOP;

END $$;

## **4.2. Опис процесів генерації тестових даних.**

Використані способи генерації даних:

1. Генерація випадкових записів: Деякі дані, такі як звертання до зовнішнього ключа, генеруються випадковим чином (певне число у відповідному діапазоні). Наприклад, для кожного нового продукту генерується випадкова назва, категорія, жанр.

## **4.3. Трансформація та первинне завантаження даних.**

1. Процес заповнення бази даних за допомогою написаного js - коду виконується шляхом зчитування файлу за допомогою модуля fs та його подальшого аналізу за допомогою csv-parser. Після цього дані обробляються та вставляються у відповідні таблиці бази даних, використовуючи підключення до бази даних, встановлене за допомогою модуля mssql. Процес також включає обробку можливих помилок та закриття з'єднання з базою даних після завершення операції для ефективного управління ресурсами.
2. Під час завантаження кожен набір даних був перевірений на унікальність, щоб уникнути дублювання записів. У випадку, якщо деякі дані виявились не унікальними, вони відкидалися або оброблялися відповідно до вимог програми.

# **РОЗДІЛ 5. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ КОРИСТУВАЧА**

## **5.1. Опис інтерфейсу у відповідності до бізнес-процесів.**

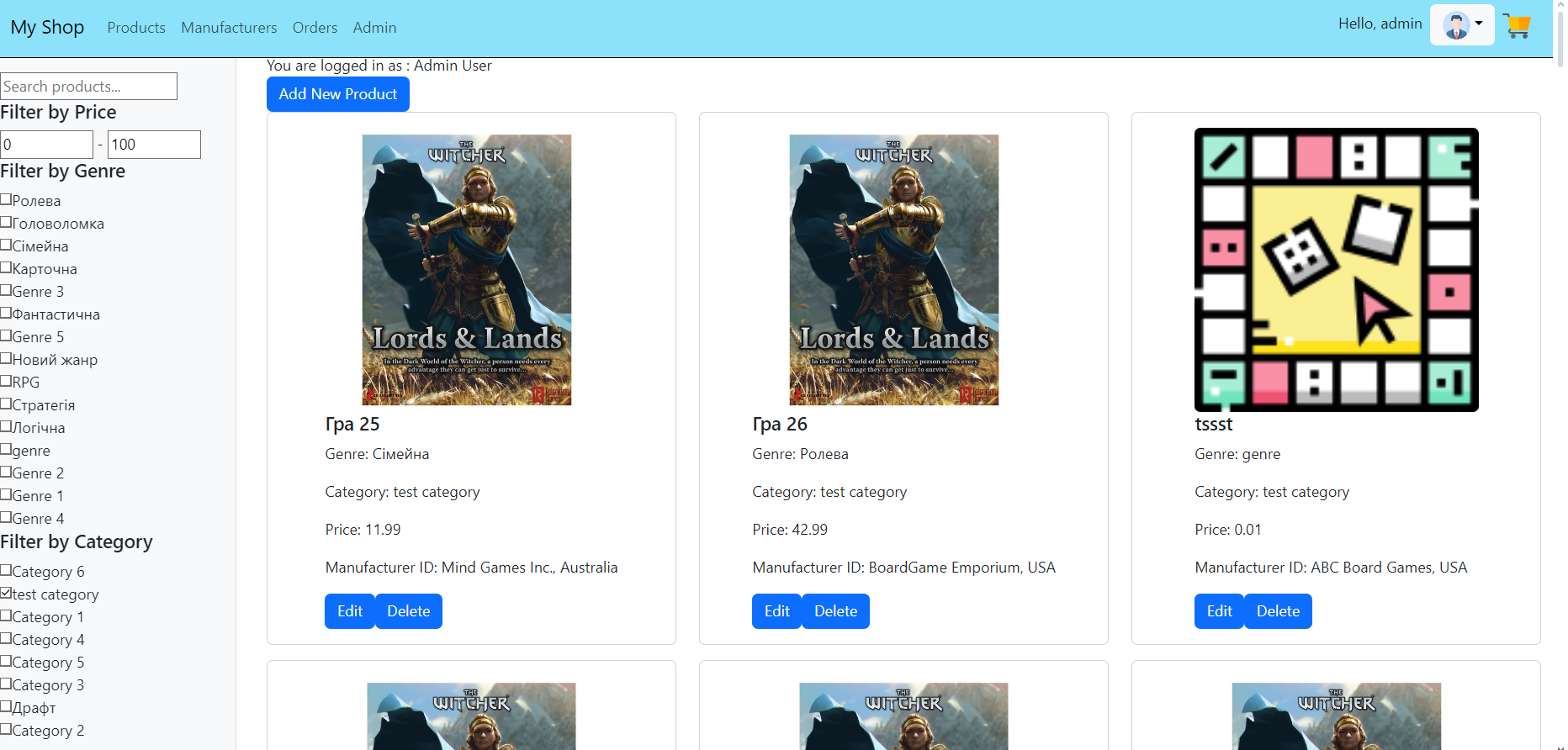


Рис. 5.1.1. Загальна структура клієнтської частини.

Інтерфейс програми складається з різних елементів, які відображають ключові бізнес-процеси системи обліку магазину. Основною частиною інтерфейсу є табличні представлення даних, які дозволяють користувачеві переглядати, редагувати та керувати інформацією.

## **5.2. Засоби аналітичного представлення даних.**

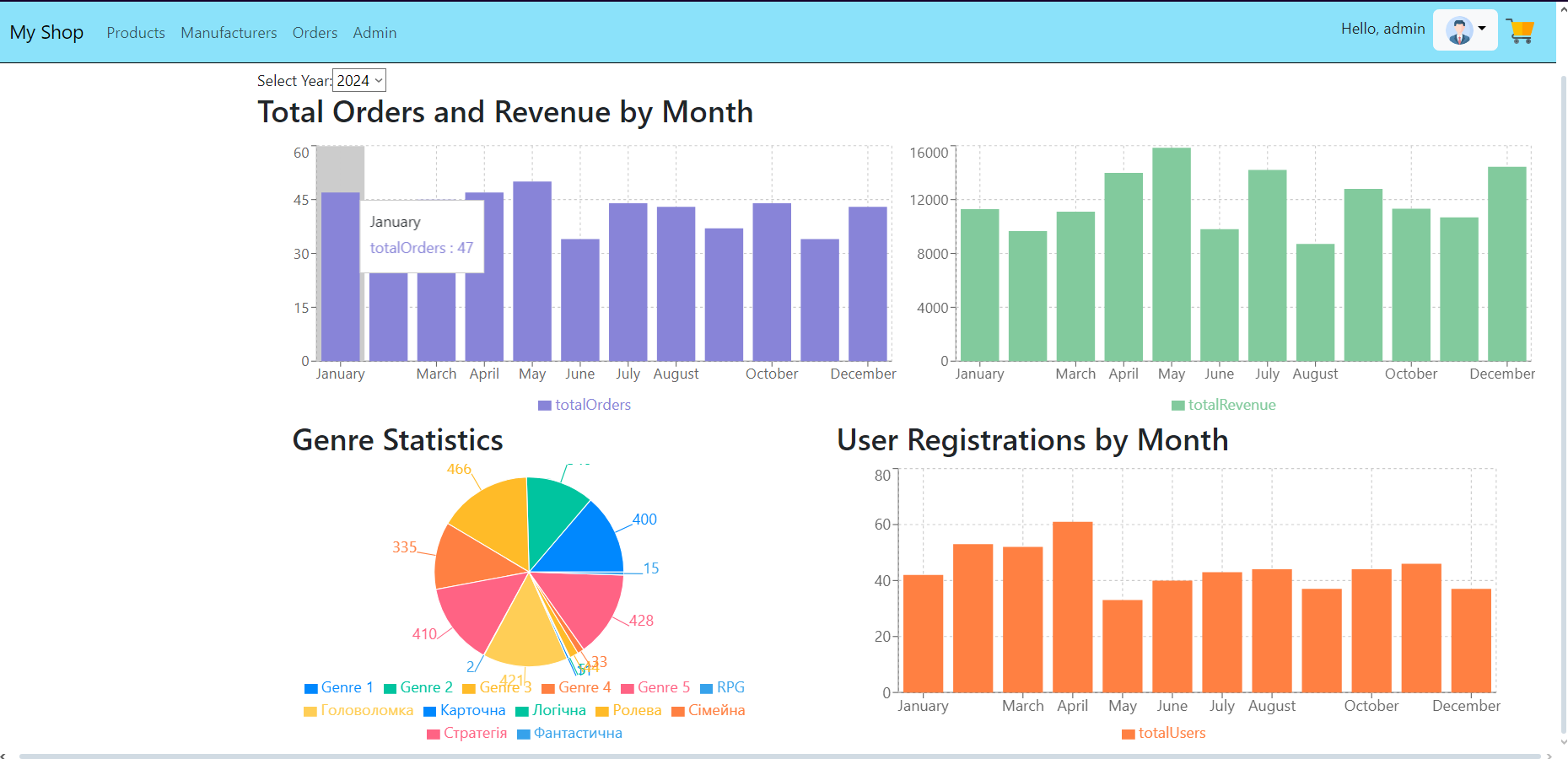


Рис. 5.2.1. Вигляд засобу аналітичного представлення даних

Засіб аналітичного представлення даних це інструмент, який дозволяє візуалізувати та аналізувати інформацію про кількість замовлень за кожен місяць, загальний прибуток проданих товарів протягом кожного місяця, категорії, яка є найпопулярнішою, користувачів, що зареєструвались протягом кожного місяця. Ці графіки і статистика надає користувачеві можливість прийняття стратегічних рішень щодо формування асортименту товарів, управління ціноутворенням, та розробки маркетингових акцій.

## **5.3. Засоби експорту даних.**

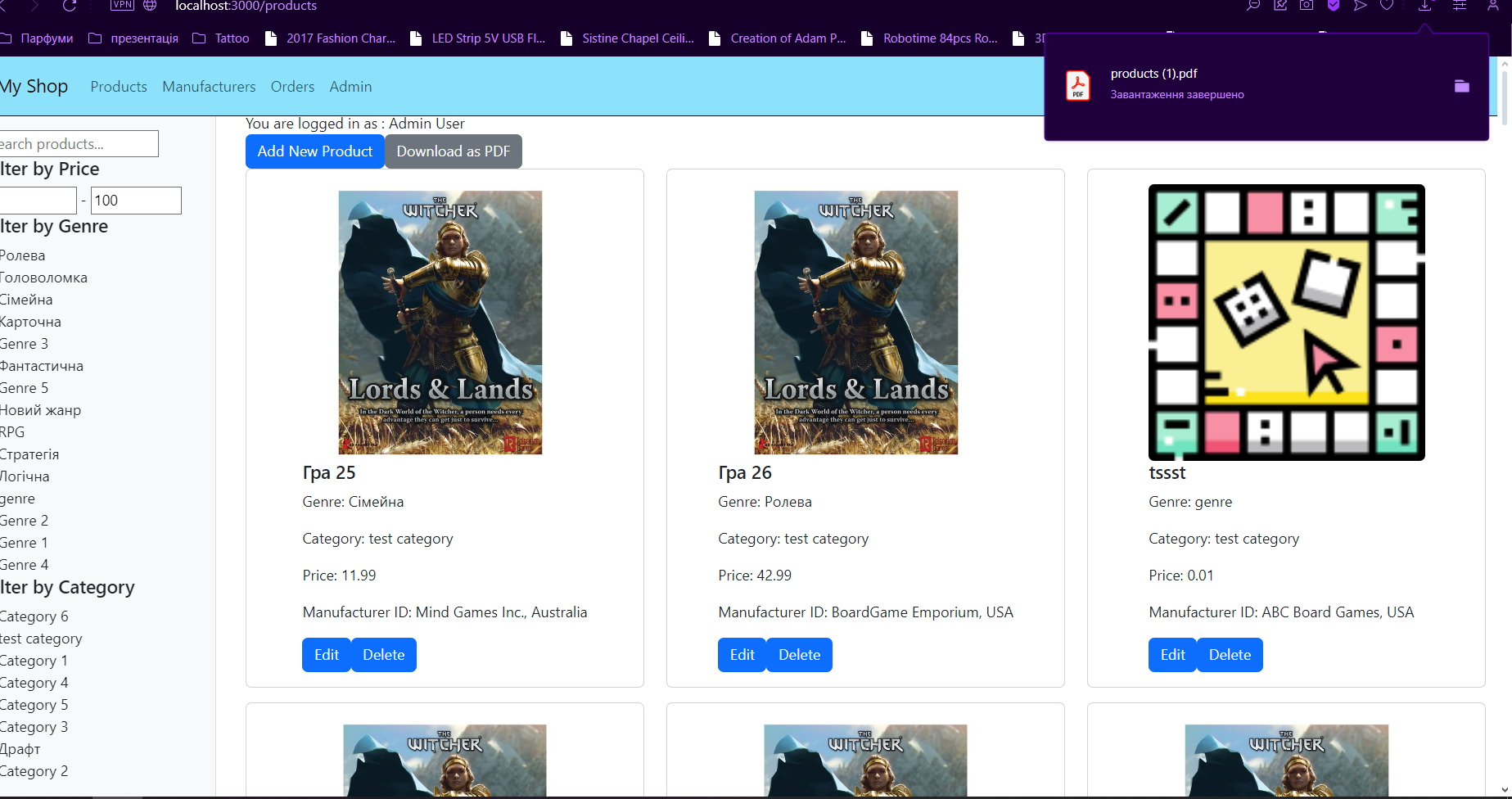


Рис. 5.3.1. Вигляд засобу експорту даних

Засіб експорту даних у форматі та PDF - це інструмент, який дозволяє користувачам експортувати табличні дані з програми у різні формати файлів для подальшого використання.

## **5.4. Засоби програмного інтерфейсу.**

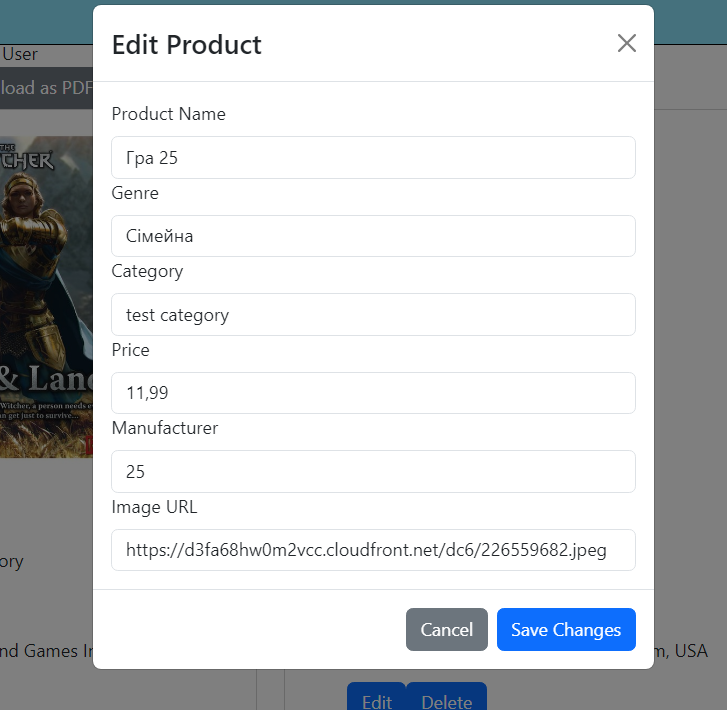


Рис. 5.4.1. Вигляд засобу редагування даних

Програмний інтерфейс для додавання, редагування та видалення даних в базі даних надає розширений набір можливостей для взаємодії з даними.

## **5.5. Засоби фільтрації на інтерфейсі.**

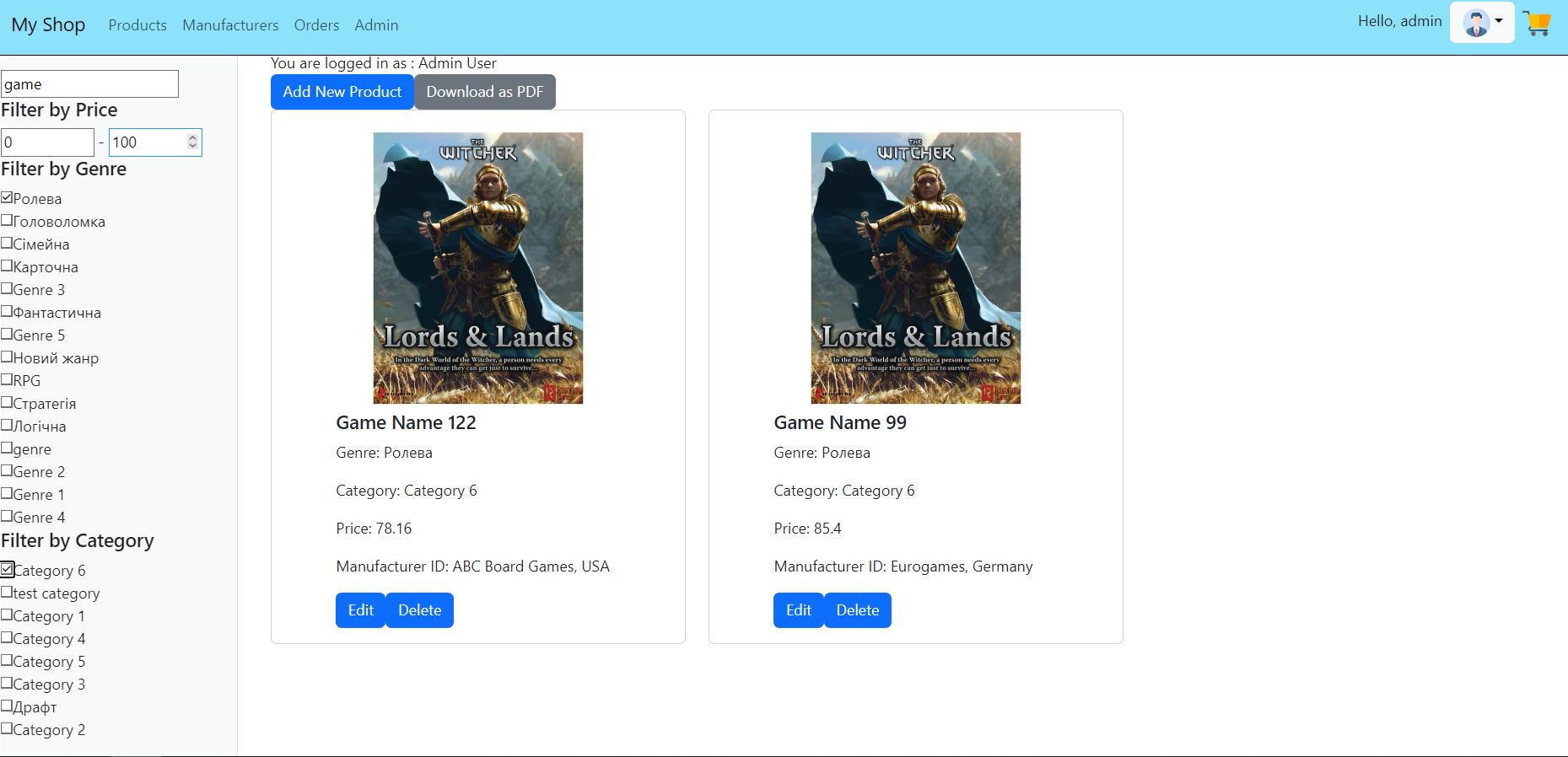


Рис. 5.4.2. Вигляд засобу редагування даних

Програмний інтерфейс дозволяє шукати дані за динамічно створеними фільтрами для легшого пошуку потрібних даних.

# **ВИСНОВКИ**

# *Недоліки системи:*

# Проблеми з оптимізацією: Система потребує покращення в області продуктивності та швидкості роботи. Проблеми, як-от повільне завантаження даних чи затримки при їх обробці, вказують на необхідність перегляду алгоритмів та оптимізації архітектури системи для забезпечення більш плавної та стабільної роботи.

# Обмежена функціональність інтерфейсу: Інтерфейс програмування додатків (API), який використовується для взаємодії з базою даних, має деякі обмеження, що ускладнює ефективне керування даними. Наприклад, відсутність підтримки реєстрації користувачів може бути проблемою для розширення та покращення взаємодії з системою.

# *Перспективи покращення:*

# Оптимізація системи: Провести аналіз процесів завантаження та обробки даних, щоб знайти і усунути вузькі місця. Впровадити механізми кешування для швидшої відповіді на запити, а також поліпшити алгоритми, щоб підвищити загальну продуктивність системи.

# Розширення функціональності інтерфейсу: Додати можливість реєстрації користувачів і переглянути інтерфейс API, щоб забезпечити більше можливостей для керування даними. Впровадження функцій сортування та пошуку може значно покращити зручність використання та дозволить більш гнучко працювати з даними.

# **СПИСОК ЛІТЕРАТУРИ**

1. Crud App: URL - https://www.youtube.com/watch?v=OrHO7UeDwZc
2. PostgreSQL: URL - https://www.postgresql.org/docs/
3. Bootstrap5: URL - https://getbootstrap.com/docs/4.1/getting-started/introduction/
4. React: URL - <https://uk.legacy.reactjs.org/>
5. Troubleshooting: URL - <https://stackoverflow.com/questions/>
6. Chat-gpt: URL - <https://chat.openai.com/>

# **ДОДАТКИ**

## **8.1. Скрипт створення бази даних.**

USE NikeShop;

CREATE TABLE Suppliers (

supplier\_id INT PRIMARY KEY IDENTITY,

supplier\_name VARCHAR(255) NOT NULL,

phone VARCHAR(15),

email VARCHAR(255),

address VARCHAR(255)

);

CREATE TABLE Colors (

color\_id INT PRIMARY KEY IDENTITY,

color\_name VARCHAR(50) UNIQUE

);

CREATE TABLE Types (

type\_id INT PRIMARY KEY IDENTITY,

type\_name VARCHAR(255) UNIQUE

);

CREATE TABLE Sizes (

size\_id INT PRIMARY KEY IDENTITY,

size\_name VARCHAR(50) UNIQUE

);

CREATE TABLE ProductSpecifics (

product\_specific\_id INT PRIMARY KEY IDENTITY,

price DECIMAL(10,2),

color\_id INT,

size\_id INT,

type\_id INT,

FOREIGN KEY (color\_id) REFERENCES Colors(color\_id),

FOREIGN KEY (size\_id) REFERENCES Sizes(size\_id),

FOREIGN KEY (type\_id) REFERENCES Types(type\_id)

);

CREATE TABLE Products (

product\_id INT PRIMARY KEY IDENTITY,

product\_name VARCHAR(255) UNIQUE,

amount INT,

product\_specific\_id INT,

supplier\_id INT,

FOREIGN KEY (product\_specific\_id) REFERENCES ProductSpecifics(product\_specific\_id),

FOREIGN KEY (supplier\_id) REFERENCES Suppliers(supplier\_id)

);

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY IDENTITY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(255) UNIQUE,

phone VARCHAR(15) UNIQUE

);

CREATE TABLE Employees (

employee\_id INT PRIMARY KEY IDENTITY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(255) UNIQUE,

position VARCHAR(100),

salary DECIMAL(10,2)

);

CREATE TABLE Orders (

order\_id INT PRIMARY KEY IDENTITY,

customer\_id INT,

employee\_id INT,

total\_amount DECIMAL(10,2),

order\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),

FOREIGN KEY (employee\_id) REFERENCES Employees(employee\_id)

);

CREATE TABLE OrderItems (

order\_item\_id INT PRIMARY KEY IDENTITY,

order\_id INT,

product\_id INT,

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

CREATE TABLE Returns (

return\_id INT PRIMARY KEY IDENTITY,

order\_item\_id INT,

returned\_date DATE,

reason TEXT,

FOREIGN KEY (order\_item\_id) REFERENCES OrderItems(order\_item\_id)

);

CREATE TABLE Seasons (

season\_id INT PRIMARY KEY IDENTITY,

season\_name VARCHAR(255) UNIQUE,

product\_specific\_id INT,

FOREIGN KEY (product\_specific\_id) REFERENCES ProductSpecifics(product\_specific\_id)

);

CREATE TABLE Materials (

material\_id INT PRIMARY KEY IDENTITY,

material\_name VARCHAR(255) UNIQUE,

percentage\_in\_product DECIMAL(5,2),

product\_specific\_id INT,

FOREIGN KEY (product\_specific\_id) REFERENCES ProductSpecifics(product\_specific\_id)

);

CREATE TABLE Storage (

storage\_id INT PRIMARY KEY IDENTITY,

row\_number INT,

shelf\_number INT,

product\_id INT,

quantity INT,

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

## **8.2. Скрипти генерування даних**

DO $$

DECLARE

n\_rows INT := 10; -- кількість рядків для генерації

start\_id INT := 55; -- початковий айді, оскільки значення мають бути більші за 54

BEGIN

FOR i IN 0..n\_rows-1 LOOP

INSERT INTO public."Products"

(

product\_id,

pname,

genre,

pprice,

manufacturers\_id,

category,

image\_url

)

VALUES

(

start\_id + i,

'Game Name ' || (start\_id + i), -- ім'я гри

'Genre ' || (start\_id + i), -- жанр гри

ROUND((1 + (RANDOM() \* 99))::numeric, 2), -- ціна в діапазоні від 1 до 100

(1 + (RANDOM() \* 49)), -- manufacturers\_id від 1 до 50

'Category ' || (start\_id + i), -- категорія гри

'https://d3fa68hw0m2vcc.cloudfront.net/dc6/226559682.jpeg' -- URL зображення

);

END LOOP;

END $$;

DO $$

DECLARE

n\_rows INT := 100; -- кількість рядків для генерації

start\_id INT := 65; -- початковий айді, оскільки значення мають бути більші за 54

BEGIN

FOR i IN 0..n\_rows-1 LOOP

INSERT INTO public."Products"

(

product\_id,

pname,

genre,

pprice,

manufacturers\_id,

category,

image\_url

)

VALUES

(

start\_id + i,

'Game Name ' || (start\_id + i), -- ім'я гри

'Genre ' || ((i % 7) + 1), -- жанр гри, де значення не більше 7

ROUND((1 + (RANDOM() \* 99))::numeric, 2), -- ціна в діапазоні від 1 до 100

(1 + (RANDOM() \* 49)), -- manufacturers\_id від 1 до 50

'Category ' || ((i % 7) + 1), -- категорія гри, де значення не більше 7

'https://d3fa68hw0m2vcc.cloudfront.net/dc6/226559682.jpeg' -- URL зображення

);

END LOOP;

END $$;

UPDATE public."Products"

SET

genre = 'Головоломка'

WHERE

genre ~ '\d\*7\d\*';

ALTER TABLE IF EXISTS public."Manufaturers"

OWNER to postgres;

-- Функція для вибору випадкової країни з можливих варіантів

CREATE OR REPLACE FUNCTION random\_country()

RETURNS TEXT AS $$

DECLARE

countries TEXT[] := ARRAY['USA', 'Canada', 'UK', 'Germany', 'France', 'Spain', 'Italy'];

BEGIN

RETURN countries[FLOOR(RANDOM() \* ARRAY\_LENGTH(countries, 1)) + 1];

END;

$$ LANGUAGE plpgsql;

-- Генерація випадкових даних для Manufacturers

DO $$

DECLARE

n\_rows INT := 20; -- кількість рядків для генерації

start\_id INT := 51; -- початковий ID, оскільки він має бути більше за 50

BEGIN

FOR i IN 0..n\_rows-1 LOOP

INSERT INTO public."Manufaturers"

(

manufacturers\_id,

mname,

country,

web\_site,

contact\_phone

)

VALUES

(

start\_id + i, -- ID більше 50

'Manufacturer ' || (start\_id + i), -- назва виробника

random\_country(), -- випадкова країна

'www.manufacturer' || (start\_id + i) || '.com', -- вебсайт

'+1-777-333-' || LPAD((RANDOM() \* 10000)::INT::TEXT, 4, '0') -- номер телефону

);

END LOOP;

END $$;

-- Функція для вибору випадкового статусу замовлення

CREATE OR REPLACE FUNCTION random\_order\_status()

RETURNS TEXT AS $$

DECLARE

statuses TEXT[] := ARRAY['Shipped', 'Delivered', 'Processing', 'Pending'];

BEGIN

RETURN statuses[FLOOR(RANDOM() \* ARRAY\_LENGTH(statuses, 1)) + 1];

END;

$$ LANGUAGE plpgsql;

-- Функція для вибору випадкової дати в діапазоні

CREATE OR REPLACE FUNCTION random\_order\_date()

RETURNS DATE AS $$

DECLARE

start\_date DATE := '2023-01-01';

end\_date DATE := '2024-12-31';

days\_difference INT := (end\_date - start\_date);

BEGIN

RETURN start\_date + INTERVAL '1 day' \* (FLOOR(RANDOM() \* days\_difference));

END;

$$ LANGUAGE plpgsql;

-- Генерація випадкових даних для Order

DO $$

DECLARE

n\_rows INT := 30; -- кількість рядків для генерації

start\_id INT := 57; -- початковий ID, оскільки він має бути більше 56

BEGIN

FOR i IN 0..n\_rows-1 LOOP

INSERT INTO public."Order"

(

order\_id,

customer\_id,

odate,

ostatus

)

VALUES

(

start\_id + i, -- ID більше 56

(FLOOR(RANDOM() \* 1000)), -- customer\_id в діапазоні 0-1000

random\_order\_date(), -- випадкова дата

random\_order\_status() -- випадковий статус

);

END LOOP;

END $$;

-- Генерація випадкових даних для Customers

DO $$

DECLARE

n\_rows INT := 30; -- кількість рядків для генерації

start\_id INT := 100; -- початковий ID

BEGIN

FOR i IN 0..n\_rows-1 LOOP

INSERT INTO public."Customers"

(

customer\_id,

cname,

address,

email,

phone\_number

)

VALUES

(

start\_id + i, -- ID починаючи зі 100

'Customer ' || (start\_id + i), -- ім'я

'Address ' || (start\_id + i), -- адреса

'customer' || (start\_id + i) || 'email@example.com', -- електронна пошта

'+380' || '98' || (FLOOR(RANDOM() \* 9000000) + 1000000) -- номер телефону

);

END LOOP;

END $$;

-- Генерація випадкових даних для OrderDetails

DO $$

DECLARE

n\_rows INT := 30; -- кількість рядків для генерації

start\_id INT := 53; -- початковий ID

product\_id\_range INT := 1164; -- максимальне значення product\_id

order\_id\_max INT := 1055; -- максимальне значення order\_id

product\_id\_var INT; -- змінна для зберігання product\_id

product\_price NUMERIC(6, 2); -- змінна для зберігання ціни продукту

amount INT; -- змінна для зберігання кількості

BEGIN

FOR i IN 0..n\_rows-1 LOOP

-- Випадкове значення product\_id

product\_id\_var := (FLOOR(RANDOM() \* product\_id\_range) + 1);

-- Отримуємо ціну для цього product\_id

SELECT pprice INTO product\_price

FROM public."Products"

WHERE product\_id = product\_id\_var -- явне вказування змінної

LIMIT 1;

-- Призначаємо випадкову кількість товарів

amount := (FLOOR(RANDOM() \* 10) + 1);

-- Вставка нових рядків у таблицю "OrderDetails"

INSERT INTO public."OrderDetails"

(

order\_details\_id,

product\_id,

amount,

odprice,

order\_id

)

VALUES

(

start\_id + i, -- ID починаючи з 53

product\_id\_var, -- використання явного імені змінної

amount, -- випадкова кількість

product\_price \* amount, -- ціна продукту помножена на кількість

(FLOOR(RANDOM() \* order\_id\_max) + 1) -- випадковий order\_id

);

END LOOP;

END $$;

-- Генерація випадкових даних для Users

DO $$

DECLARE

n\_rows INT := 30; -- кількість рядків для генерації

start\_id INT := 31; -- ID більше 30

customer\_id INT; -- оголошення змінної

customer\_id\_range INT := 1100; -- максимальний діапазон customer\_id

role TEXT; -- оголошення змінної для ролі

roles TEXT[] := ARRAY['Normal User', 'Admin User']; -- можливі ролі

created\_at TIMESTAMP WITH TIME ZONE; -- оголошення змінної для created\_at

BEGIN

FOR i IN 0..n\_rows-1 LOOP

-- Випадковий customer\_id

customer\_id := (FLOOR(RANDOM() \* customer\_id\_range) + 1);

-- Випадкова роль

role := roles[FLOOR(RANDOM() \* ARRAY\_LENGTH(roles, 1))];

-- Випадкова дата створення в діапазоні 2023.01.01 - 2024.12.31

created\_at := '2023-01-01'::DATE + INTERVAL '1 day' \* FLOOR(RANDOM() \* 730); -- 730 днів між 2023-2024

-- Вставка даних у таблицю "Users"

INSERT INTO public."Users"

(

user\_id,

customer\_id,

uemail,

password,

role,

created\_at,

username

)

VALUES

(

start\_id + i, -- ID більше 30

customer\_id, -- випадковий customer\_id

'User' || (start\_id + i) || '@example.com', -- email

'1234', -- пароль

role, -- випадкова роль

created\_at, -- випадкова дата створення

'user' || (start\_id + i) -- username

);

END LOOP;

END $$;

UPDATE public."Users"

SET

role = 'Admin User' -- нове значення

WHERE

role IS NULL;

## **8.2. Інший програмний код.**

Посилання на Git-репозиторій з кодом проекту:

<https://github.com/luff3/DataBaseApp>

**8.3. Інший код створених моделей.**

Код створення процедур, тригерів та індексів:

CREATE INDEX idx\_Products\_manufacturers\_id

ON Products (manufacturers\_id);

CREATE INDEX idx\_Orders\_customer\_id

ON Orders (customer\_id);

CREATE INDEX idx\_OrderDetails\_order\_id

ON OrderDetail (order\_id);

CREATE INDEX idx\_OrderDetails\_product\_id

ON OrderDetail (product\_id);

CREATE INDEX idx\_Reviews\_product\_id

ON Reviews (product\_id);

CREATE FUNCTION GetOrderCountsByMonth()

RETURNS TABLE(month INT, order\_count INT)

LANGUAGE SQL

AS $$

SELECT EXTRACT(MONTH FROM odate) AS month,

COUNT(\*) AS order\_count

FROM Orders

WHERE odate > NOW() - INTERVAL '12 months'

GROUP BY month

ORDER BY month;

$$;

CREATE FUNCTION GetMostPurchasedProduct()

RETURNS VARCHAR(255)

LANGUAGE SQL

AS $$

SELECT p.pname

FROM OrderDetail od

JOIN Products p ON p.product\_id = od.product\_id

GROUP BY p.pname

ORDER BY COUNT(\*) DESC

LIMIT 1;

$$;

CREATE FUNCTION GetTotalSalesForLastMonth()

RETURNS DECIMAL(10, 2)

LANGUAGE SQL

AS $$

SELECT SUM(od.odprice \* od.amount)

FROM OrderDetail od

JOIN Orders o ON o.order\_id = od.order\_id

WHERE EXTRACT(MONTH FROM o.odate) = EXTRACT(MONTH FROM NOW() - INTERVAL '1 month');

$$;

CREATE FUNCTION GetTopSellingEmployee()

RETURNS VARCHAR(255)

LANGUAGE SQL

AS $$

SELECT e.first\_name || ' ' || e.last\_name

FROM Orders o

JOIN Employees e ON e.employee\_id = o.employee\_id

JOIN OrderDetail od ON od.order\_id = o.order\_id

GROUP BY e.first\_name, e.last\_name

ORDER BY SUM(od.odprice \* od.amount) DESC

LIMIT 1;

$$;

CREATE TRIGGER UpdateTotalAmountAfterUpdate

AFTER UPDATE ON OrderDetail

FOR EACH ROW

EXECUTE PROCEDURE UpdateOrderTotalAmount(NEW.order\_id);

CREATE TRIGGER UpdateTotalAmountAfterDelete

AFTER DELETE ON OrderDetail

FOR EACH ROW

EXECUTE PROCEDURE UpdateOrderTotalAmount(OLD.order\_id);

CREATE TRIGGER UpdateTotalAmountAfterInsert

AFTER INSERT ON OrderDetail

FOR EACH ROW

EXECUTE PROCEDURE UpdateOrderTotalAmount(NEW.order\_id);

CREATE OR REPLACE PROCEDURE UpdateOrderTotalAmount(order\_id INT)

LANGUAGE plpgsql

AS $$

BEGIN

-- Обчислити нову загальну суму замовлення

UPDATE Orders

SET total\_amount = (

SELECT SUM(odprice \* amount)

FROM OrderDetail

WHERE order\_id = order\_id

)

WHERE order\_id = order\_id;

END;

$$;