НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

(повна назва інституту/факультету)

КАФЕДРА інформатики та програмної інженерії

(повна назва кафедри)

**КУРСОВА РОБОТА**

з дисципліни «Бази даних»

(назва дисципліни)

на тему: БАЗА ДАНИХ ДЛЯ ПІДТРИМКИ СИСТЕМИ ФІКСАЦІЇАДМІНІСТРАТИВНИХ ПРАВОПОРУШЕНЬ У СФЕРІ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДОРОЖНЬОГО РУХУ

Студента \_\_2\_\_ курсу \_ІП-33\_\_\_\_\_ групи

спеціальності 121 «Інженерія програмного забезпечення»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Соколов О. В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_Ліщук Катерина Ігорівна, доц., к.т.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

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

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

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2024 рік

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

Факультет Інформатики та обчислювальної техніки

(повна назва)

Кафедра Інформатики та програмної інженерії

(повна назва)

Дисципліна Бази даних

Курс \_\_2\_\_\_ Група \_\_ІП-33\_\_ Семестр \_\_\_3\_\_

**З А В Д А Н Н Я**

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Соколов Олександр Вячеславович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи БАЗА ДАНИХ ДЛЯ ПІДТРИМКИ СИСТЕМИ ФІКСАЦІЇ\_

АДМІНІСТРАТИВНИХ ПРАВОПОРУШЕНЬ У СФЕРІ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДОРОЖНЬОГО РУХУ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

керівник роботи Ліщук Катерина Ігорівна, доц.,к.т.н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 22.12.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до роботи створення бази даних для підтримки системи фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_1) Аналіз предметного середовища\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2) Побудова ER-моделі\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3) Побудова реляційної схеми з ER-моделі\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4) Створення бази даних, у форматі обраної системи управління базою даних\_

5) Створення користувачів бази даних\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6) Імпорт даних з використанням засобів СУБД в створену базу даних\_\_\_\_\_\_\_

7) Створення мовою SQL запитів\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8) Оптимізація роботи запитів\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Дата видачі завдання\_\_\_\_04.11.2023**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсового проекту | Строк виконання етапів проекту | Примітка |
| 1 | Аналіз предметного середовища | 2.12.2023 |  |
| 2 | Побудова ER-моделі | 5.12.2023 |  |
| 3 | Побудова реляційної схеми з ER-моделі | 8.12.2023 |  |
| 4 | Створення бази даних, у форматі обраної системи управління базою даних | 13.12.2023 |  |
| 5 | Створення користувачів бази даних | 14.12.2023 |  |
| 6 | Імпорт даних з використанням засобів СУБД в створену базу даних | 15.12.2023 |  |
| 7 | Створення запитів мовою SQL | 17.12.2023 |  |
| 8 | Оптимізація роботи запитів | 18.12.2023 |  |
| 9 | Оформлення пояснювальної записки | 22.12.2023 |  |
| 10 | Захист курсової роботи | 25.12.2024 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Студент \_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_Соколов О. В.\_\_\_\_\_\_

(підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_Ліщук К. І. \_\_\_\_\_\_

(підпис ) (прізвище та ініціали)

**Анотація**

Пояснювальна записка до курсової роботи: 102 сторінки, 55 рисунків, 16 таблиць, 4 посилань.

Об’єкт дослідження: база даних для підтримки системи фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху.

Мета роботи: розробка та реалізація бази даних для підтримки системи фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху.

Був проведений аналіз предметного середовища, на його основі побудована ER-модель та реляційна схема. Використовуючи СУБД PostgreSQL була створ ена база даних, в якій налаштовані користувачі та ролі. Дані були імпортовані до цієї бази. Також були розроблені функції, процедури, тригери, представлення та SQL-запити для ефективної роботи з базою даних. Окрім цього, була проведена оптимізація запитів для поліпшення їх продуктивності.

Здійснено успішну реалізацію бази даних для складського обліку підприємства використовуючи СУБД PostgreSQL, відповідно до вказаного варіанту завдання.

**Зміст**

[ВСТУП 5](#_Toc185699979)

[1 АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА 7](#_Toc185699980)

[1.1 Опис предметного середовища 7](#_Toc185699981)

[1.2 Аналіз існуючих програмних продуктів 9](#_Toc185699982)

[1.3 Висновки 13](#_Toc185699983)

[2 ПОСТАНОВКА ЗАВДАННЯ 14](#_Toc185699984)

[2.1 Завдання 14](#_Toc185699985)

[2.2 Мета 14](#_Toc185699986)

[2.2 Вимоги до бази даних 14](#_Toc185699987)

[2.3 Висновки 15](#_Toc185699988)

**ВСТУП**

У сучасному світі забезпечення безпеки дорожнього руху є одним із найважливіших завдань суспільства. Зростаюча кількість транспортних засобів та складність дорожньої інфраструктури зумовлюють необхідність створення ефективних систем для фіксації адміністративних правопорушень. Одним із ключових елементів таких систем є база даних, яка забезпечує обробку, зберігання та аналіз великого обсягу інформації.

Актуальність теми дослідження обумовлена потребою у вдосконаленні процесів управління дорожнім рухом та посиленні контролю за виконанням правил дорожнього руху. Правильна організація зберігання даних про правопорушення, винних осіб, транспортні засоби та відповідні штрафи дозволяє не лише автоматизувати адміністративні процедури, але й підвищити ефективність роботи органів, відповідальних за безпеку на дорогах.

Метою даної курсової роботи є розробка та впровадження бази даних для підтримки системи фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху.

У рамках курсової роботи необхідно розробити базу даних для підтримки системи фіксації адміністративних правопорушень у сфері безпеки дорожнього руху. Завдання включають аналіз предметного середовища, побудову ER-моделі та реляційної схеми, нормалізацію даних, створення бази даних у СУБД PostgreSQL та налаштування багатокористувацького доступу.

Крім того, передбачено імпорт даних, написання SQL-запитів, створення функцій, процедур і тригерів, а також оптимізацію запитів для підвищення продуктивності. Усі етапи спрямовані на забезпечення цілісності даних, мінімізацію надмірності та підвищення швидкодії системи.

Сфера використання розробленої бази даних включає державні організації, які займаються моніторингом дорожнього руху, обліком адміністративних правопорушень, а також судові органи, що розглядають такі випадки.

Для реалізації курсової роботи обрана система управління базами даних (СУБД) PostgreSQL. Цей вибір обумовлений низкою переваг, які роблять PostgreSQL ідеальним інструментом для створення бази даних у рамках завдання.

По-перше, PostgreSQL є потужною реляційною СУБД з відкритим вихідним кодом, яка забезпечує високий рівень надійності, масштабованості та продуктивності. Її підтримка складних транзакцій і дотримання вимог до цілісності даних гарантують коректність роботи навіть для складних бізнес-завдань.

По-друге, PostgreSQL має багатий набір функціональних можливостей, зокрема підтримку розширень, складних типів даних, тригерів і процедур, що дозволяє реалізувати всі необхідні вимоги проєкту. Система також пропонує широкий спектр інструментів для оптимізації запитів, зокрема індекси, планувальники запитів та матеріалізовані представлення.

По-третє, ця СУБД забезпечує потужну багатокористувацьку модель доступу з гнучкими механізмами розмежування прав користувачів, що є критично важливим для забезпечення безпеки та зручності роботи із системою.

Нарешті, PostgreSQL має активну спільноту розробників, велику кількість документації та сумісність із сучасними інструментами розробки, що значно спрощує процес створення та обслуговування бази даних. Усі ці фактори роблять PostgreSQL оптимальним вибором для реалізації системи фіксації адміністративних правопорушень.

**1 АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА**

1.1 Опис предметного середовища

Об’єктом дослідження є інформаційна система для фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху. Основна мета цієї системи — автоматизувати обробку великих обсягів даних про порушення правил дорожнього руху (ПДР), реєстрацію транспортних засобів, створення та ведення протоколів і постанов, а також здійснення аналітичної обробки інформації. Завдяки цій системі забезпечується швидкий доступ до даних, їхній аналіз і можливість формування звітів, що полегшує роботу відповідних служб та підвищує ефективність управління дорожнім рухом.

Система працює з різними типами вхідних даних, які включають інформацію про громадян, транспортні засоби, порушення та адміністративні документи. Дані про громадян охоплюють персональну інформацію, таку як ім’я, прізвище, по батькові та дата народження, що є ключовими для ідентифікації особи. Система передбачає перевірку дати народження, щоб забезпечити коректний вік особи для участі в певних ролях, наприклад, для отримання водійського посвідчення або призначення поліцейським. Кожен громадянин отримує унікальний ідентифікаційний номер, що запобігає дублюванню записів.

Інформація про транспортні засоби містить деталі, такі як VIN номер, номерні знаки, тип двигуна, модель, марка, колір, рік випуску та власник. Важливим елементом є перевірка на унікальність VIN, а також перевірка відповідності формату номерних знаків, типу двигуна та року випуску автомобіля. VIN та інші параметри є важливими для точного ідентифікування транспортних засобів і забезпечують достовірність даних про кожен автомобіль.

Дані про порушення ПДР містять час і місце порушення, тип порушення, номер статті КУпАП, а також транспортний засіб, що скоїв порушення, і відповідну локацію. Крім того, вказуються порушення конкретного правила дорожнього руху. Порушення реєструються в системі з обмеженням по часу: вони можуть бути зареєстровані лише за останні десять років. Ці дані допомагають не тільки в документуванні порушень, але й у подальшому аналізі порушень, а також дають змогу підтверджувати правопорушення через фото- чи відеоматеріали.

Протоколи та постанови створюються на основі даних про порушення і включають серію та номер документа, час його складання, а також інформацію про поліцейського, який складає документ, і порушника. Поліцейський, що складає протокол, не може бути причетним до самого порушення. Для забезпечення унікальності документів, система застосовує перевірку на унікальність комбінації серії та номера протоколу або постанови, що виключає можливість дублювання. Дані про протоколи і постанови допомагають ефективно адмініструвати процедуру накладення штрафів та інших санкцій.

Вихідні дані формуються у вигляді звітів і відповідей на запити користувачів. Наприклад, система дозволяє отримувати інформацію про громадян, які найчастіше порушували ПДР, виявляти місця з найбільшою кількістю порушень, аналізувати активність поліцейських, визначати найпоширеніші моделі транспортних засобів серед порушників або формувати рейтинги статей КУпАП за частотою порушень.

Основні бізнес-процеси системи включають реєстрацію громадян і транспортних засобів, фіксацію порушень ПДР, оформлення адміністративних документів, а також аналітику і звітність. Реєстрація забезпечує точність введення даних і унікальність кожного об’єкта. Фіксація порушень передбачає перевірку всіх параметрів на відповідність встановленим бізнес-правилам, що гарантує їхню коректність. Адміністративні документи створюються з дотриманням усіх вимог до часу, унікальності даних та повноти інформації. Аналітична частина системи дозволяє ефективно використовувати накопичені дані для прийняття обґрунтованих рішень і планування заходів у сфері безпеки дорожнього руху.

1.2 Аналіз існуючих програмних продуктів

Система для підтримки фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху в Україні зосереджена на наданні державним органам, таким як поліція, інструментів для ефективного моніторингу порушень правил дорожнього руху (ПДР) та здійснення адміністративних процедур. Вона включає різноманітні механізми для виявлення порушень, їх реєстрації, а також для збору доказів, що дозволяє забезпечити коректне і справедливе накладення штрафів та інші санкції. Одним з основних елементів цієї системи є інтеграція з автоматичними камерами фіксації порушень (наприклад, за перевищення швидкості, проїзд на червоне світло), а також можливість реєстрації порушень співробітниками поліції, які складають протоколи або постанови на основі свідчень, записів відеокамер або безпосереднього спостереження.

Ця система забезпечує функціонал для того, щоб правоохоронні органи могли ефективно проводити перевірку порушень, реєструвати їх в електронних базах даних та оперативно передавати для подальшої обробки, як-от накладання штрафів або направлення справ до суду. Важливо, що система підтримує збереження доказів у вигляді фото- або відеоматеріалів, що дозволяє уникнути суб'єктивності у встановленні фактів правопорушення.

Обмеження доступу для звичайних громадян та водіїв пояснюється кількома основними факторами. По-перше, з міркувань безпеки та конфіденційності даних. Інформація про правопорушення, а також інші персональні дані громадян повинні залишатися захищеними від несанкціонованого доступу, оскільки їх публічне розкриття може призвести до непотрібних маніпуляцій або порушень прав особистості. Наприклад, дозволити доступ до даних про невиплачені штрафи або про порушення ПДР безпосередньо може призвести до спроб оскарження або маніпулювання результатами.

По-друге, обмеження доступу дозволяє запобігти маніпуляціям із самими процедурами накладання штрафів або оскарження постанов. Якщо звичайний громадянин або водій отримав би повний доступ до всієї інформації в системі, це може призвести до спроб впливу на процес (наприклад, зміни статусу штрафу чи протоколу). Таким чином, контроль доступу є важливим елементом для забезпечення справедливості та правильності процесу.

Контроль за доступом до інформації, що стосується правопорушень, є важливою частиною цієї системи. Такий контроль дозволяє забезпечити, щоб лише уповноважені особи – поліціянти, судді, представники органів державної влади – могли здійснювати відповідні дії з даними про порушення. Це дозволяє мінімізувати можливість зловживань або помилок у процесах виявлення та розгляду порушень, а також гарантує, що кожен етап процесу буде здійснюватися відповідно до закону та з урахуванням прав громадян.

Тим не менш, існують програмні продукти, доступні громадянам, які дозволяють взаємодіяти з системами фіксації порушень ПДР, такі як "Електронний кабінет водія" та інші онлайн-сервіси для перевірки штрафів. Ці продукти надають водіям зручний доступ до важливої інформації, зокрема про їхні правопорушення, нараховані штрафи та інші адміністративні питання, пов'язані з транспортними засобами. Завдяки таким сервісам водії можуть без зусиль перевіряти інформацію про існуючі порушення і сплачувати штрафи, що істотно спрощує взаємодію з органами влади та скорочує час, необхідний для виконання адміністративних процедур. Однак, ці сервіси не надають доступу до можливості реєстрації нових порушень або внутрішніх даних правоохоронних органів.

"Електронний кабінет водія" [1] є одним із таких сервісів, який дозволяє водіям отримати актуальну інформацію про їхні водійські документи, транспортні засоби та наявність штрафів. Система дозволяє перевіряти статус посвідчення водія, перевіряти наявність та суму штрафів за порушення ПДР, сплачувати ці штрафи онлайн, а також замовляти номерні знаки для транспортних засобів. Важливою перевагою є можливість отримувати сповіщення про зміни в статусі документів або наявність нових штрафів, що дозволяє водіям оперативно реагувати на ситуацію.

Сервіс перевірки адміністративних правопорушень [2] надає громадянам можливість перевірити наявність штрафів за порушення ПДР, які були зафіксовані за допомогою автоматичних камер. Для використання цього сервісу користувачам необхідно ввести державний номер транспортного засобу та серію і номер свідоцтва про реєстрацію. Після цього система надає доступ до актуальних даних про порушення, включаючи інформацію про конкретне правопорушення, його місце та час вчинення, а також поточний статус справи.

Цей сервіс має кілька переваг для водіїв. Він дозволяє швидко перевіряти наявність штрафів без необхідності звертатися до державних органів або відвідувати спеціалізовані установи. Завдяки автоматизованій системі обробки даних, користувачі отримують актуальну і точну інформацію про свої порушення, що значно полегшує процес їхнього вирішення. Однак слід зазначити, що цей сервіс обмежений лише перевіркою штрафів, зареєстрованих за допомогою автоматичних камер, і не охоплює порушення, зафіксовані іншими методами, наприклад, патрульними поліцейськими.

Застосунок "Штрафи ПДР" [3] надає водіям можливість не тільки сплачувати штрафи, але й отримувати повідомлення про нові постанови або зміни статусу вже винесених штрафів. Це дозволяє водіям бути в курсі всіх змін, що стосуються їхніх адміністративних правопорушень, а також отримувати оперативні сповіщення, що значно полегшує процес контролю за своїми штрафами та їх оплатою.

Однією з ключових функцій застосунку є інтеграція з мапою розташування камер автоматичної фіксації порушень. Це дозволяє водіям не тільки перевіряти наявність штрафів, але й бути в курсі місць, де можуть бути зафіксовані порушення правил дорожнього руху. Така інформація допомагає знижувати ризик несподіваних штрафів, оскільки водії можуть планувати свої маршрути з урахуванням місць, де встановлені камери. Серед переваг застосунку можна відзначити зручність у використанні, оперативність отримання сповіщень та можливість онлайн оплати штрафів.

Застосунок "Штрафи UA" [4] є національним сервісом для водіїв України, який дозволяє зручно перевіряти наявність штрафів та здійснювати їх оплату. Водії можуть перевірити штрафи за номером водійського посвідчення, ІПН або серією та номером штрафу. Застосунок охоплює різні типи порушень, такі як ті, що зафіксовані патрульними поліцейськими, камерами автоматичної фіксації, а також штрафи за порушення паркування в ряді міст країни.

Основне призначення сервісу полягає в наданні водіям онлайн-можливості для перевірки наявності штрафів та швидкої оплати. Це дозволяє водіям своєчасно погашати штрафи, уникати нарахування пені та розглядів з виконавчими службами. Завдяки цьому сервісу водії можуть оперативно отримувати актуальну інформацію про нові штрафи і перевіряти історію старих, що значно полегшує управління фінансами і адміністративними питаннями. Згідно з дослідженнями команди "Штрафи UA", близько 70% водіїв, які не сплачують штрафи, роблять це через втрату протоколів або через складність процесу сплати, а також через відсутність часу чи забування про штрафи. Застосунок вирішує ці проблеми, надаючи зручний доступ до інформації і можливість швидкої оплати, що дозволяє уникнути штрафних санкцій за несвоєчасну оплату.

Telegram-бот МВС «Штрафи ПДР» [5] є офіційним інструментом для перевірки штрафів за порушення ПДР в Україні. Цей бот дозволяє користувачам швидко отримати інформацію про наявність штрафів, що були зафіксовані автоматичними системами або патрульними поліцейськими. Крім того, бот надає можливість здійснити оплату штрафів безпосередньо через додаток, що робить процес сплати зручним і швидким.

Однією з основних переваг Telegram-бота є його доступність та простота використання. Користувачі можуть без зайвих труднощів перевіряти свої штрафи та оплачувати їх, не покидаючи месенджер. Бот надає актуальні дані про порушення, що дозволяє водіям бути в курсі своїх обов'язків і виконувати їх своєчасно, мінімізуючи ризик додаткових штрафів за прострочення.

1.3 Висновки

У розділі було проведено аналіз існуючих програмних продуктів та сервісів, пов'язаних з фіксацією адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху. Вивчення цих продуктів показало важливість створення зручних і доступних для громадян інструментів, які дозволяють ефективно взаємодіяти з державними системами, перевіряти наявність штрафів, оплачувати їх та отримувати актуальні сповіщення.

Розглянуто функціонал таких сервісів, як "Електронний кабінет водія", "Штрафи UA", Telegram-боти та інші, які забезпечують водіям можливість швидко отримувати необхідну інформацію щодо адміністративних правопорушень. Це дозволяє знизити ризики забуття або затримки сплати штрафів, що, в свою чергу, допомагає уникнути нарахування пені та інших проблем.

Вивчення існуючих рішень дало змогу визначити ключові компоненти системи, які повинні бути включені в розробку, а також визначити потенційні проблеми, що потребують подальшого вдосконалення. Розробка таких сервісів важлива для підвищення ефективності процесів фіксації порушень ПДР та їх обробки.

Аналіз показав, що інтеграція сучасних технологій, зокрема мобільних додатків і чат-ботів, є важливим кроком до покращення доступу громадян до інформації та спрощення процесів сплати штрафів.

**2 ПОСТАНОВКА ЗАВДАННЯ**

2.1 Завдання

Завданням курсової роботи є розробка бази даних для підтримки фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху. База даних має бути орієнтована на ефективну обробку та зберігання інформації про правопорушення, порушників, транспортні засоби, штрафи та інші адміністративні процеси.

Основні завдання курсової роботи:

* аналіз предметного середовища;
* побудова ER-моделі;
* побудова реляційної схеми;
* створення бази даних;
* cтворення користувачів бази даних;
* імпорт даних;
* створення SQL запитів;
* оптимізація роботи запитів.

2.2 Мета

Метою роботи є створення інструменту для автоматизації збору, зберігання та обробки даних, що стосуються порушень правил дорожнього руху, та забезпечення можливості оперативного доступу до цих даних для відповідальних органів.

2.2 Вимоги до бази даних

Основні вимоги до бази даних:

* система повинна забезпечити коректне зберігання даних про порушення, порушників, штрафи, транспортні засоби, а також можливість пошуку, фільтрації та створення звітів;
* необхідно забезпечити правильність і узгодженість інформації, що зберігається в базі, шляхом введення необхідних обмежень і перевірок;
* база даних повинна ефективно зберігати тільки необхідну для виконання завдань інформацію, уникаючи дублювання даних;
* система повинна передбачати можливість використання бази кількома категоріями користувачів з різними рівнями доступу;
* база повинна мати високу швидкість обробки запитів, що дозволяє оперативно реагувати на запити користувачів.

Вимоги до бази даних включають забезпечення безпеки даних, захист від несанкціонованого доступу, а також підтримку стабільної роботи з великими обсягами інформації про правопорушення та порушників.

2.3 Висновки

У розділі було сформульовано постановку завдання, мету та визначено основні вимоги до розробки бази даних для підтримки фіксації адміністративних правопорушень у сфері дорожнього руху. Завдання курсової роботи полягає в створенні бази даних, що забезпечить ефективне зберігання, обробку та доступ до даних про порушення, порушників, транспортні засоби та штрафи.

Згідно з поставленими вимогами, система повинна забезпечувати коректність і цілісність даних, ефективно працювати з великими обсягами інформації та забезпечувати багатокористувацький доступ. Окрім того, важливою вимогою є мінімізація обсягу даних, що зберігаються, та оптимізація запитів для швидкої обробки даних.

Було визначено основні етапи роботи, включаючи побудову ER-моделі, реляційної схеми, створення бази даних у СУБД, розробку запитів і оптимізацію роботи запитів. З урахуванням цих завдань, на наступних етапах буде виконано детальне проектування та реалізація бази даних для ефективної підтримки процесів фіксації адміністративних правопорушень у сфері дорожнього руху.

**3 ПОБУДОВА ER-МОДЕЛІ**

3.1 Бізнес-правила

До основних бізнес-правил належить:

1. громадяни не можуть бути доросліше 120 років та не можуть бути народжені у майбутньому;
2. номери водійського посвідчення та поліцейського жетону громадян мають складатися з трьох великих літер, що накреслення яких збігається в латинській і в українській абетках, та з шести цифр;
3. номери водійського посвідчення та поліцейського жетону громадян мають бути унікальними;
4. водійське посвідчення можуть отримати лише громадяни старше 16 років;
5. поліцейськими можуть стати лише громадяни доросліше 18 років;
6. у однієї машини може бути тільки один поточний власник;
7. номерні знаки автомобілів можуть складатися з 2 лівих літер, які відповідають регіону реєстрації, 4 цифр та 2 правих літер, що відповідають серії номерного знаку, де всі де літери такі, що накреслення яких збігається в латинській і в українській абетках, або від 3 до 8 літер;
8. Vehicle Identification Number (VIN) має бути унікальним та довжиною у 17 символів;
9. автомобіль може мати страховку, номер якої має бути унікальним;
10. типи двигунів у транспортних засобів можуть бути наступними: дизельний, бензиновий, електричний, гібридний;
11. у автомобілів з електричним двигуном не може бути об’єму;
12. об’єм двигуна не електричних транспортних засобів не може бути менше 1л. і більше 20л;
13. рік вироблення автомобіля має бути після 1886 року (дата вироблення першого серійного автомобіля) та не може бути у майбутньому;
14. кількість сидячих місць має відповідати типу автомобіля;
15. пункт та частина правил дорожнього руху мають бути більше рівними 1;
16. комбінація пункту та частини правил дорожнього руху мають бути унікальними;
17. стаття, пункт та частина КУпАП мають бути більше рівними 1;
18. якщо за статтю КУпАП передбачається штраф, то він має бути більше 0 грн.;
19. комбінація статті, пункту та частини КУпАП мають бути унікальними;
20. зафіксоване порушення обов’язково має вказувати час, місце, автомобіль, порушене правило дорожнього руху та статтю КУпАП;
21. час порушення не може бути раніше ніж за 10 років від поточної дати, оскільки система буде зберігати тільки порушення за останні 10 років;
22. одне порушення може мати декілька фото та відео доказів;
23. протокол ДТП обов’язково має складатися із серії та номеру документу протоколу, часу складання, порушення, поліцейського та обвинувачуваного;
24. комбінація серії та номеру протоколу має бути унікальною;
25. час складання протоколу не може бути раніше за час скоєння відповідного порушення;
26. поліцейський не може скласти протокол сам на себе;
27. пов’язані з протоколом громадяни у вигляді свідків та жертв не можуть бути одночасно обвинувачуваними та / або поліцейськими у протоколі;
28. постанова за порушення обов’язково має вказувати на серію та номер документу постанови, час розгляду та час впровадження покарання в дію, порушення за яким складається постанова, поліцейського, який розглядає порушення та місце складання постанови;
29. за одне порушення може скластися тільки або протокол, або постанова.

3.2 Виділені сутності

Проаналізувавши предметне середовище та відповідно до постановленого завдання, було виділено такі сутності:

* citizen;
* driver;
* police\_officer;
* vehicle\_type;
* vehicle;
* traffic\_rule;
* administrative\_offense;
* location;
* violation;
* evidence;
* accident\_protocol;
* citizen\_on\_protocol;
* accident\_resolution;
* region.

3.3 Опис сутностей

У таблиці 3.1 наведено опис сутностей

Таблиця 3.1 – Опис сутностей

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
| citizen | репрезентує громадянина та його персональну інформацію | id | унікальний ідентифікатор сутності |
| first\_name | ім’я громадянина |
| last\_name | прізвище громадянина |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
|  |  | patronymic | по батькові громадянина |
| date\_of\_birth | дата народження громадянина |
| driver | репрезентує громадянина як водія | id | унікальний ідентифікатор сутності |
|  |  | citizen\_id | ідентифікатор громадянина |
| license\_number | номер водійського посвідчення |
| license\_issued\_time | дата та час видання водійського посвідчення |
| police\_officer | репрезентує громадянина як поліцейського | id | унікальний ідентифікатор сутності |
|  |  | citizen\_id | ідентифікатор громадянина |
| badge\_number | номер поліцейського жетону |
| rank | звання поліцейського |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
| vehicle\_type | репрезентує типи транспортних засобів | id | унікальний ідентифікатор сутності |
|  |  | name | назва транспортного засобу |
| description | опис транспортного засобу |
| min\_seating\_capacity | мінімальна кількість сидячих місць |
| max\_seating\_capacity | максимальна кількість сидячих місць |
| min\_engine\_capacity | мінімальний обʼєм двигуна |
| max\_engine\_capacity | максимальний обʼєм двигуна |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
| vehicle | репрезентує автомобілі громадян | id | унікальний ідентифікатор сутності |
|  |  | owner\_id | ідентифікатор громадянина власника |
| vehicle\_type\_id | ідентифікатор типу транспортного засобу |
| registration\_number | реєстраційний номер (номерний знак) транспортного засобу |
| vin | ідентифікаційний номер транспортного засобу |
| insurance\_policy\_number | номер страховки транспортного засобу |
| model | модель транспортного засобу |
| brand | виробник транспортного засобу |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
|  |  | color | колір транспортного засобу |
| engine\_capacity | обʼєм двигуна транспортного засобу |
| seating\_capacity | кількість сидячих місць транспортного засобу |
| year\_of\_manufacture | дата виробництва транспортного засобу |
| traffic\_rule | репрезентує правило дорожнього руху | id | унікальний ідентифікатор сутності |
| article | пункт правил дорожнього руху |
| part | частина пункту правил дорожнього руху |
| description | пункт правила дорожнього руху |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
| administrative\_offense | репрезентує статтю КУпАП | id | унікальний ідентифікатор сутності |
| article | стаття КУпАП |
| sup | пункт КУпАП |
| part | частина КУпАП |
| description | опис статті, пункту та частини КУпАП |
| penalty\_fee | штраф за порушення |
| location | репрезентує місце | id | унікальний ідентифікатор сутності |
| longitude |  |
| latitude | координата місця (ширина) |
| street | вулиця |
| building\_number | номер будівлі |
| description | опис місця |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
| violation | репрезентує порушення правил дорожнього руху | id | унікальний ідентифікатор сутності |
| time\_of\_violation | дата та час скоєння порушення |
| description | опис суті порушення |
| vehicle\_id | ідентифікатор транспортного засобу, причасного до порушення |
| location\_id | ідентифікатор місця скоєння порушення |
| administrative\_offense\_id | ідентифікатор статті КУпАП, яке відповідає порушенню |
| traffic\_rule\_id | ідентифікатор порушеного правила дорожнього руху |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
| evidence | репрезентує доказ порушення | id | унікальний ідентифікатор сутності |
| violation\_id | ідентифікатор порушення |
| type | тип доказів (фото, відео) |
| url | посилання на доказ |
| accident\_protocol | репрезентує складений протокол до порушення | id | унікальний ідентифікатор сутності |
| series | серія документу протоколу |
| number | номер документу протоколу |
| defendant\_explanation | пояснення обвинувачуваного |
| time\_of\_drawing\_up | час та дата складання протоколу |
| violation\_id | ідентифікатор порушення, за яким складався протокол |
| police\_officer\_id | ідентифікатор поліцейського, який складав протокол |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
|  |  | defendant\_id | ідентифікатор громадянина, якого обвинувачують |
| citizen\_on\_protocol | репрезентує громадянина, який повʼязаний із протоколом | id | унікальний ідентифікатор сутності |
| role | роль громадянина у протоколі (свідок, жертва) |
| citizen\_id | ідентифікатор громадянина |
| protocol\_id | ідентифікатор протоколу |
| testimony | свідчення громадянина |
| accident\_resolution | репрезентує постанову | id | унікальний ідентифікатор сутності |
| series | серія документу протоколу |
| number | номер документу протоколу |
| time\_of\_consideration | час та дата розгляду порушення та складання протоколу |

Продовження таблиці 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Назва сутності | Опис сутності | Атрибути | Опис атрибутів |
|  |  | time\_of\_entry\_into\_force | час та дата надбання постанови законної сили |
| violation\_id | ідентифікатор порушення, за яким складалася постанова |
| police\_officer\_id | ідентифікатор поліцейського, який складав постанову |
| location\_id | ідентифікатор місця складання постанови |
| region | репрезентує область України | id | унікальний ідентифікатор сутності |
| region\_name | назва області |
| code\_2004 | код регіону реєстрації ТЗ у 2004 |
| code\_2013 | код регіону реєстрації ТЗ у 2013 |
| code\_2021 | код регіону реєстрації ТЗ у 2021 |

3.4 Опис зв’язків між сутностями

Для подальшої реалізації реляційної бази даних, варто визначити зв’язки між сутностями системи. Зв’язки мають повністю відповідати визначеним бізнес-правилам.

Зв’язки між сутностями:

* citizen – driver: один до одного (1:1) – громадянин може бути водієм, і кожен водій обов’язково має відповідного громадянина;
* citizen – police\_officer: один до одного (1:1) – кожен громадянин може бути поліцейським, і кожен поліцейський має відповідного громадянина;
* citizen – vehicle: один до багатьох (1:n) – один громадянин може володіти кількома транспортними засобами, але кожен транспортний засіб має лише одного власника;
* vehicle\_type – vehicle: один до багатьох (1:n) – один тип транспортного засобу може бути пов’язаний з багатьма транспортними засобами, але кожен транспортний засіб має лише один тип;
* traffic\_rule – violation: один до багатьох (1:n) – кожне порушення правил дорожнього руху пов’язане з одним правилом, але одне правило може бути порушене багаторазово;
* administrative\_offense – violation: один до багатьох (1:n) – кожне адміністративне порушення може бути застосоване до кількох порушень, але одне порушення пов’язане лише з одним адміністративним правопорушенням;
* location – violation: один до багатьох (1:n) – одне місце розташування може бути пов’язане з багатьма порушеннями, але кожне порушення відбувається в одному місці;
* violation – evidence: один до багатьох (1:n) – кожне порушення може мати кілька доказів, але кожен доказ належить до одного порушення;
* violation – accident\_protocol: один до одного (1:1) – кожне порушення може бути задокументоване лише одним протоколом, і кожен протокол стосується одного порушення;
* police\_officer – accident\_protocol: один до багатьох (1:n) – один поліцейський може складати кілька протоколів, але кожен протокол оформляє лише один поліцейський;
* citizen – accident\_protocol: один до одного (1:1) – кожен протокол стосується одного порушника, який є громадянином;
* accident\_protocol – citizen\_on\_protocol: один до багатьох (1:n) – один протокол може включати кілька громадян у різних ролях (наприклад, свідків чи учасників), але кожен запис ролі стосується одного протоколу;
* violation – accident\_resolution: один до одного (1:1) – кожне порушення може мати лише одну постанову, яке його стосується;
* police\_officer – accident\_resolution: один до багатьох (1:n) – один поліцейський може складати кілька постанов, але кожну постанову ухвалює лише один поліцейський;
* location – accident\_resolution: один до багатьох (1:n) – одне місце розташування може бути пов’язане з кількома постановами, але кожна постановастосується одного місця.

3.5 ER-модель

Побудована ER-модель зображена на рис. 3.1.

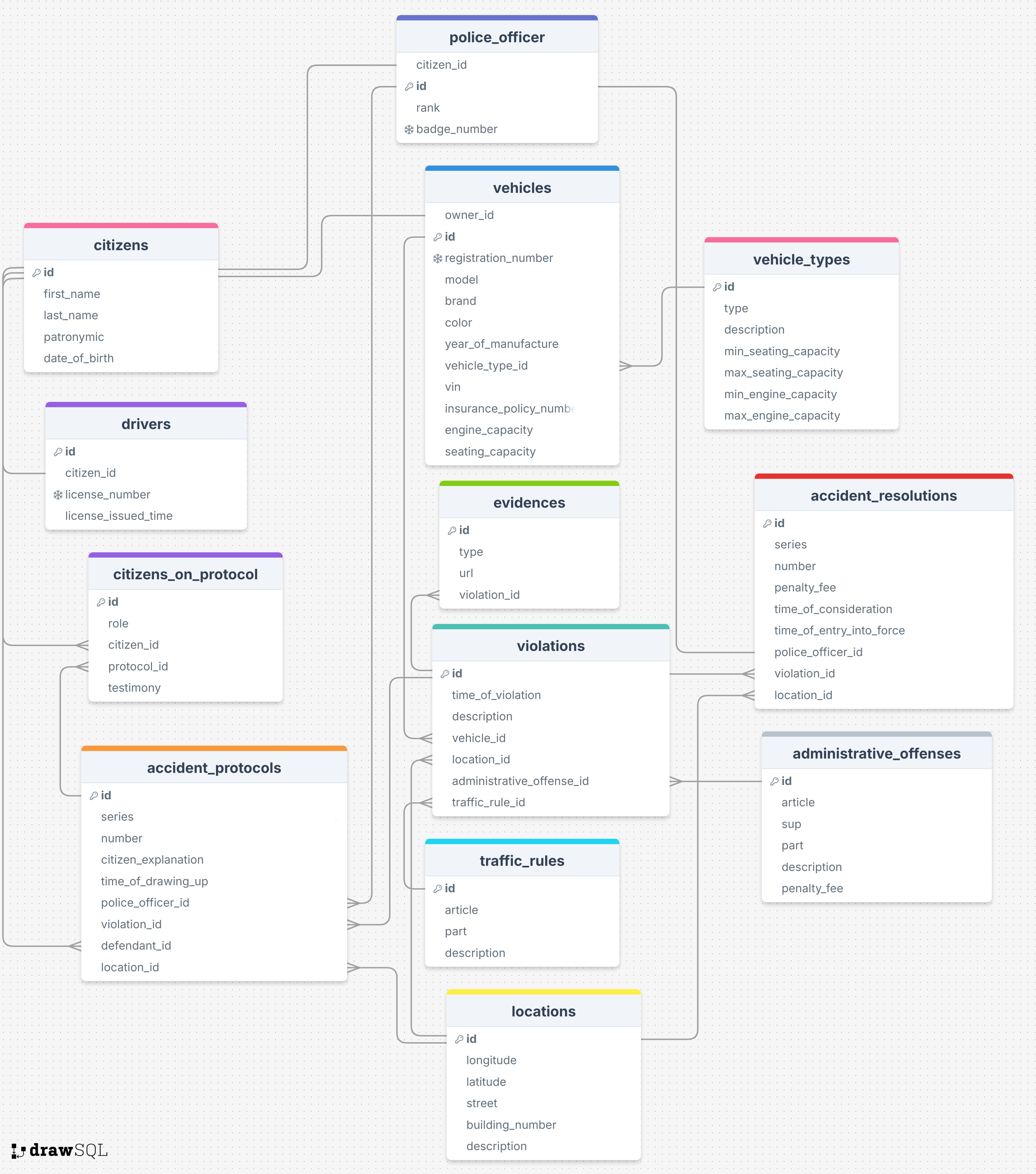


Рисунок 3.1 – ER-модель бази даних для підтримки системи фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху

3.6 Висновки

У цьому розділі було визначено основні бізнес-правила, які регулюють роботу системи фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху. На основі аналізу предметного середовища виділено основні сутності, що забезпечують структуру бази даних. Визначено зв’язки між сутностями та наведено ER-модель, яка є основою для подальшого проєктування реляційної схеми бази даних.

**4 РЕАЛЯЦІЙНА МОДЕЛЬ БАЗИ ДАНИХ**

Перехід від логічної моделі бази даних до її реалізації у конкретній системі управління базами даних (СУБД) є ключовим етапом проєктування інформаційної системи. На цьому етапі визначаються інструменти для зберігання, управління, обробки та забезпечення цілісності даних. Для створення бази даних системи підтримки фіксації адміністративних правопорушень у сфері забезпечення безпеки дорожнього руху було обрано СУБД PostgreSQL[6]. Це рішення ґрунтується на її технічних можливостях, функціональних перевагах і відповідності вимогам предметної області.

PostgreSQL[6] забезпечує повну відповідність принципам реляційної моделі, що робить її ідеальним вибором для роботи з базами даних, які включають складні зв’язки між сутностями. Система дозволяє ефективно створювати та управляти таблицями, забезпечуючи строгий контроль над відносинами між ними через механізми зовнішніх ключів. Це особливо важливо для проєкту, де велика кількість таблиць пов’язані складною структурою.

Однією з переваг PostgreSQL[6] є її розширена підтримка типів даних. Вона дозволяє працювати зі стандартними типами, такими як VARCHAR, INT, TEXT, а також із більш складними, включно з TIMESTAMPTZ і DECIMAL. Це сприяє точному моделюванню даних та відповідності специфічним вимогам. Наприклад, для сутності «locations» було використано типи DECIMAL для координат, що забезпечує високу точність географічних даних.

Важливою характеристикою PostgreSQL[6] є її потужний механізм перевірки даних. Система дозволяє створювати складні обмеження, включаючи перевірки унікальності, зовнішні ключі, користувацькі функції перевірки, та застосування тригерів. Завдяки цьому вдається реалізувати складні бізнес-правила, такі як контроль мінімального віку водія або перевірка валідності номерів посвідчень. Такі функції забезпечують високу цілісність даних і зменшують ризик помилок.

Ще однією перевагою є масштабованість і продуктивність PostgreSQL[6]. Система здатна обробляти значні обсяги даних та ефективно виконувати запити навіть за високого навантаження. Це робить її оптимальним вибором для системи, яка передбачає довготривале зберігання даних про адміністративні правопорушення, транспортні засоби, водіїв, а також їх подальшу обробку.

СУБД PostgreSQL є безкоштовною, з відкритим кодом, що усуває витрати на ліцензування та забезпечує прозорість роботи. Це особливо важливо для проєкту, орієнтованого на оптимізацію ресурсів. Крім того, велика спільнота користувачів і розробників PostgreSQL[6] забезпечує доступ до численних ресурсів для навчання та вирішення технічних питань.

Таким чином, вибір PostgreSQL[6] для реалізації бази даних зумовлений її функціональними перевагами, відповідністю технічним вимогам і здатністю забезпечувати надійну та ефективну роботу системи.

Після детального аналізу предметного середовища були побудовані необхідні відношення та визначені первинні та зовнішні ключі. Результати зображені в таблицях 4.1 – 4.15.

В таблиці 4.1 наведено опис таблиці «citizens», яка зберігає інформацію громадян.

Таблиця 4.1 – структура таблиці «citizens»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ім’я поля | Тип даних | Розмір | Ключ | Опис |
| id | integer | - | PK | унікальний ідентифікатор громадянина у базі даних |
| first\_name | varchar | 50 | - | імʼя громадянина |
| last\_name | varchar | 50 | - | прізвище громадянина |
| patronymic | varchar | 50 | - | по батькові громадянина |
| date\_of\_birth | date | - | - | дата народження громадянина |

В таблиці 4.2 наведено опис таблиці «drivers», яка зберігає інформацію про громадян, які є водіями.

Таблиця 4.2 – структура таблиці «drivers»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ім’я поля | Тип даних | Розмір | Ключ | Опис |
| id | integer | - | PK | унікальний ідентифікатор водія у базі даних |
| citizen\_id | int | - | FK | ідентифікатор громадянина |
| license\_number | char | 9 | - | номер посвідчення водія |
| license\_issued\_time | timestamptz | - | - | час та дата видачі посвідчення водія |

В таблиці 4.3 наведено опис таблиці «police\_officers», яка зберігає інформацію про громадян, які є поліцейськими.

Таблиця 4.3 – структура таблиці «police\_officers»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ім’я поля | Тип даних | Розмір | Ключ | Опис |
| id | integer | - | PK | унікальний ідентифікатор поліцейського у базі даних |
| citizen\_id | int | - | FK | ідентифікатор громадянина |
| badge\_number | char | 9 | - | номер поліцейського жетону |
| rank | ENUM('junior\_sergeant','sergeant','senior\_sergeant','junior\_lieutenant','lieutenant','senior\_lieutenant','captain','major','lieutenant\_colonel','colonel'); | - | - | звання поліцейського |

В таблиці 4.4 наведено опис таблиці «vehicle\_types», яка зберігає інформацію про типи автомобілів та їх характеристики.

Таблиця 4.4 – структура таблиці «vehicle\_types»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ім’я поля | Тип даних | Розмір | Ключ | Опис |
| id | integer | - | PK | унікальний ідентифікатор типу транспортного засобу у базі даних |
| name | varchar | 50 | - | назва типу транспортного засобу |
| description | text | - | - | опис типу транспортного засобу |
| min\_seating\_capacity | integer | - | - | мінімальна кількість сидячих місць типу транспортного засобу |
| max\_seating\_capacity | integer | - | - | максимальна кількість сидячих місць типу транспортного засобу |
| min\_engine\_capacity | decimal | 4, 2 | - | мінімальний об’єм двигуну типу транспортного засобу |
| max\_engine\_capacity | decimal | 4, 2 | - | максимальна об’єм двигуну типу транспортного засобу |

В таблиці 4.5 наведено опис таблиці «vehicles», яка зберігає інформацію про типи автомобілів та їх характеристики.

Таблиця 4.5 – структура таблиці «vehicles»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ім’я поля | Тип даних | Розмір | Ключ | Опис |
| id | integer | - | PK | унікальний ідентифікатор типу транспортного засобу у базі даних |
| owner\_id | integer | - | FK | ідентифікатор громадянина власника |
| vehicle\_type\_id | integer | - | FK | ідентифікатор типу транспортного засобу |
| vin | char | 17 | - | ідентифікаційний номер транспортного засобу |
| insurance\_policy\_number | char | 9 | - |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Таблиця 4.3 – Сутність "ProdCategory" створена для для репризентації категорій продуктів на складі. Структура таблиці наступна:

Таблиця 4.4 – Сутність "Order" створена для репрезентації замовлення покупця. Структура таблиці наступна:

Таблиця 4.5 – Сутність "OrderStatusLog" створена для репрезентації змін статусу замовлення. Структура таблиці наступна:

Таблиця 4.6 – Сутність "OrderItems" створена для репрезентації продуктів у замовленні. Структура таблиці наступна:

Таблиця 4.7 – Сутність "Supplier" створена для репрезентації постачальника. Структура таблиці наступна:

Таблиця 4.8 – Сутність "Payments" створена для репрезентації сплат покупця. Структура таблиці наступна:

Таблиця 4.9 – Сутність "Customer" створена для репрезентації замовників або покупців. Структура таблиці наступна:

Таблиця 4.10 – Сутність "Storage" створена для репрезентації складів. Структура таблиці наступна:

Таблиця 4.11 – Сутність "ProductMovement" створена для репрезентації переміщення продуктів. Структура таблиці наступна:

Таблиця 4.12 – Сутність "ProductArrival" створена для прибуття продуктів на склад. Структура таблиці наступна:

Таблиця 4.14 – Сутність "Shipments" створена для репрезентації доставки замовлень. Структура таблиці наступна:

Таблиця 4.15 – Сутність "ShipmentsStatusLog" створена для репрезентації змін статусу таблиці доставок. Структура таблиці наступна:

Відповідно до табличних описів було створено реляційну схему (рис. 4.1) в третій нормальній формі. Це досягнуто завдяки декомпозиції полів, відсутності зайвих даних, транзитивної незалежності атрибутів від первинного ключа та функціональної повної залежності атрибутів від первинного ключа.

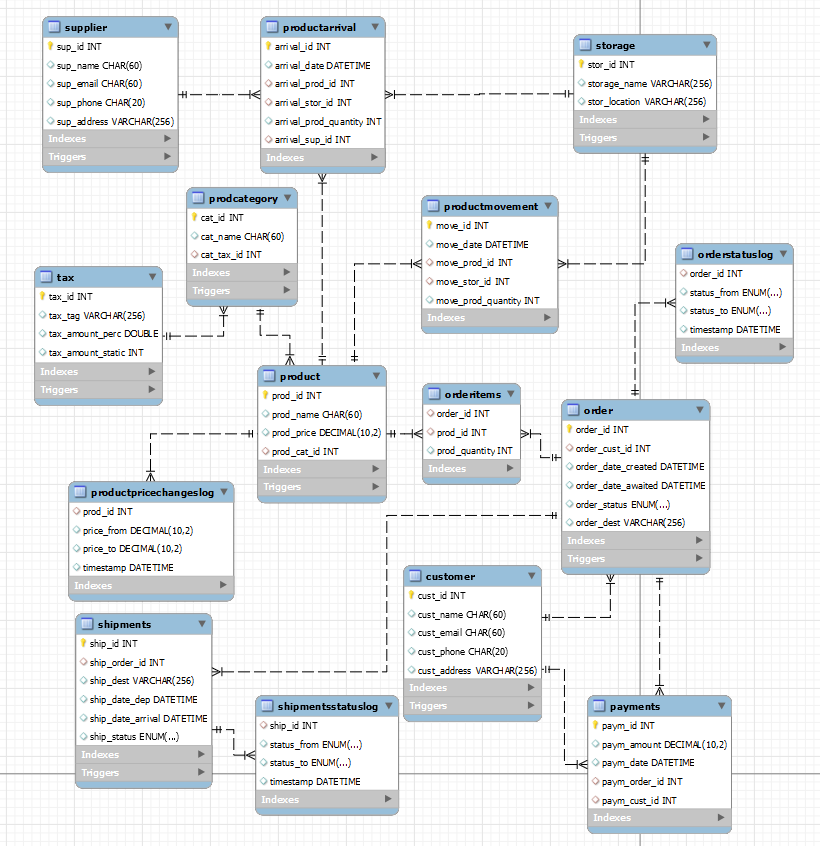


Рис. 4.1 – Реляційна схема бази даних реалізована засобами MySQL, використовуючи програмне забезпечення MySQL Worckbench[4]

У цьому розділі було наведено обґрунтування вибору MySQL в якості СУБД, були побудовані необхідні відношення та обмеження, створена реляційна схема засобами MySQL (рис. 4.1).

**5 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ**

5.1 Створення бази даних

Результатом проектування бази даних є сформований SQL-скрипт, який використовується для створення об’єктів, які були наведені в ER-моделі та табличних описах:

CREATE DATABASE STORAGE\_MANAGER;

use STORAGE\_MANAGER;

-- DB creation

CREATE TABLE `Product` (

`prod\_id` int AUTO\_INCREMENT PRIMARY KEY,

`prod\_name` char(60) NOT NULL,

`prod\_price` decimal(10,2) NULL,

`prod\_cat\_id` int NULL

);

CREATE TABLE `ProductPriceChangesLog` (

`prod\_id` int NOT NULL,

`price\_from` decimal(10,2) NULL,

`price\_to` decimal(10,2) NULL,

`timestamp` DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (prod\_id) REFERENCES Product(prod\_id)

);

CREATE TABLE `ProdCategory` (

`cat\_id` int NOT NULL AUTO\_INCREMENT PRIMARY KEY,

`cat\_name` char(60) NULL,

`cat\_tax\_id` int NULL

);

CREATE TABLE `Order` (

`order\_id` int AUTO\_INCREMENT PRIMARY KEY,

`order\_cust\_id` int NOT NULL,

`order\_date\_created` DATETIME DEFAULT CURRENT\_TIMESTAMP NOT NULL,

`order\_date\_awaited` DATETIME NULL,

`order\_status` ENUM("processing", "awaiting payment", "collecting", "ready to send", "inroute", "completed", "failed") DEFAULT "processing",

`order\_dest` varchar(256) NULL

);

CREATE TABLE `OrderStatusLog` (

`order\_id` int NOT NULL,

`status\_from` ENUM("processing", "awaiting payment", "collecting", "ready to send", "inroute", "completed", "failed") NULL,

`status\_to` ENUM("processing", "awaiting payment", "collecting", "ready to send", "inroute", "completed", "failed"),

`timestamp` DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (order\_id) REFERENCES `Order`(order\_id)

);

CREATE TABLE `OrderItems` (

`order\_id` int NOT NULL,

`prod\_id` int NOT NULL,

`prod\_quantity` int NOT NULL

);

CREATE TABLE `Supplier` (

`sup\_id` int AUTO\_INCREMENT PRIMARY KEY,

`sup\_name` char(60) NOT NULL,

`sup\_email` char(60) NULL,

`sup\_phone` char(20) NULL,

`sup\_address` varchar(256) NULL

);

CREATE TABLE `Payments` (

`paym\_id` int AUTO\_INCREMENT PRIMARY KEY,

`paym\_amount` decimal(10,2) NOT NULL,

`paym\_date` DATETIME DEFAULT CURRENT\_TIMESTAMP NOT NULL,

`paym\_order\_id` int NOT NULL,

`paym\_cust\_id` int NOT NULL

);

CREATE TABLE `Customer` (

`cust\_id` int AUTO\_INCREMENT PRIMARY KEY,

`cust\_name` char(60) NOT NULL,

`cust\_email` char(60) NULL,

`cust\_phone` char(20) NULL,

`cust\_address` varchar(256) NULL

);

CREATE TABLE `Storage` (

`stor\_id` int AUTO\_INCREMENT PRIMARY KEY,

`storage\_name` varchar(256) NULL,

`stor\_location` varchar(256) NOT NULL

);

CREATE TABLE `ProductMovement` (

`move\_id` int AUTO\_INCREMENT PRIMARY KEY,

`move\_date` DATETIME DEFAULT CURRENT\_TIMESTAMP,

`move\_prod\_id` int NOT NULL,

`move\_stor\_id` int NOT NULL,

`move\_prod\_quantity` int NOT NULL

);

CREATE TABLE `ProductArrival` (

`arrival\_id` int AUTO\_INCREMENT PRIMARY KEY,

`arrival\_date` DATETIME DEFAULT CURRENT\_TIMESTAMP,

`arrival\_prod\_id` int NOT NULL,

`arrival\_stor\_id` int NOT NULL,

`arrival\_prod\_quantity` int NOT NULL,

`arrival\_sup\_id` int NOT NULL

);

CREATE TABLE `Tax` (

`tax\_id` int AUTO\_INCREMENT PRIMARY KEY,

`tax\_tag` varchar(256) NULL,

`tax\_amount\_perc` double DEFAULT 0 NOT NULL,

`tax\_amount\_static` int DEFAULT 0 NOT NULL

);

CREATE TABLE `Shipments` (

`ship\_id` int AUTO\_INCREMENT PRIMARY KEY,

`ship\_order\_id` int UNIQUE NOT NULL,

`ship\_dest` varchar(256) NOT NULL,

`ship\_date\_dep` DATETIME DEFAULT CURRENT\_TIMESTAMP,

`ship\_date\_arrival` DATETIME NULL,

`ship\_status` ENUM("inroute", "arrived", "lost") DEFAULT "inroute"

);

CREATE TABLE `ShipmentsStatusLog` (

`ship\_id` int NOT NULL,

`status\_from` ENUM("inroute", "arrived", "lost") NULL,

`status\_to` ENUM("inroute", "arrived", "lost"),

`timestamp` DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (ship\_id) REFERENCES `Shipments`(ship\_id)

);

-- Adding foreign keys

ALTER TABLE `Product` ADD FOREIGN KEY (`prod\_cat\_id`) REFERENCES `ProdCategory` (`cat\_id`);

ALTER TABLE `ProdCategory` ADD FOREIGN KEY (`cat\_tax\_id`) REFERENCES `Tax` (`tax\_id`);

ALTER TABLE `Order` ADD FOREIGN KEY (`order\_cust\_id`) REFERENCES `Customer` (`cust\_id`);

ALTER TABLE `OrderItems` ADD FOREIGN KEY (`order\_id`) REFERENCES `Order` (`order\_id`);

ALTER TABLE `OrderItems` ADD FOREIGN KEY (`prod\_id`) REFERENCES `Product` (`prod\_id`);

ALTER TABLE `Payments` ADD FOREIGN KEY (`paym\_order\_id`) REFERENCES `Order` (`order\_id`);

ALTER TABLE `Payments` ADD FOREIGN KEY (`paym\_cust\_id`) REFERENCES `Customer` (`cust\_id`);

ALTER TABLE `ProductMovement` ADD FOREIGN KEY (`move\_prod\_id`) REFERENCES `Product` (`prod\_id`);

ALTER TABLE `ProductMovement` ADD FOREIGN KEY (`move\_stor\_id`) REFERENCES `Storage` (`stor\_id`);

ALTER TABLE `ProductArrival` ADD FOREIGN KEY (`arrival\_prod\_id`) REFERENCES `Product` (`prod\_id`);

ALTER TABLE `ProductArrival` ADD FOREIGN KEY (`arrival\_stor\_id`) REFERENCES `Storage` (`stor\_id`);

ALTER TABLE `ProductArrival` ADD FOREIGN KEY (`arrival\_sup\_id`) REFERENCES `Supplier` (`sup\_id`);

ALTER TABLE `Shipments` ADD FOREIGN KEY (`ship\_order\_id`) REFERENCES `Order` (`order\_id`);

-- Constraints

-- CHECK constraints

ALTER TABLE `Supplier` ADD CONSTRAINT SUPP\_EMAIL\_VERIF CHECK (sup\_email like "%@%");

ALTER TABLE `Customer` ADD CONSTRAINT CUST\_EMAIL\_VERIF CHECK (cust\_email like "%@%");

ALTER TABLE `Supplier` ADD CONSTRAINT SUPP\_PHONE\_VERIF CHECK (sup\_phone like "\_\_\_-\_\_\_-\_\_\_\_");

ALTER TABLE `Customer` ADD CONSTRAINT CUST\_PHONE\_VERIF CHECK (cust\_phone like "\_\_\_-\_\_\_-\_\_\_\_");

ALTER TABLE `OrderItems` ADD CONSTRAINT MIN\_PROD\_AMOUNT\_ORDER\_CHECK CHECK (prod\_quantity>0);

ALTER TABLE `ProductMovement` ADD CONSTRAINT MIN\_MOVE\_PROD\_CHECK CHECK (move\_prod\_quantity!=0);

ALTER TABLE `Payments` ADD CONSTRAINT MIN\_PAYM\_AMOUNT\_CHECK CHECK (paym\_amount>0);

ALTER TABLE `ProductArrival` ADD CONSTRAINT MIN\_ARR\_PROD\_CHECK CHECK (arrival\_prod\_quantity>0);

ALTER TABLE `Product` ADD CONSTRAINT MIN\_PROD\_PRICE\_CHECK CHECK (prod\_price>=0);

ALTER TABLE `Shipments` ADD CONSTRAINT SHIP\_DEP\_LESS\_AWAITED\_CHECK CHECK (ship\_date\_arrival is NULL or ship\_date\_dep < ship\_date\_arrival);

ALTER TABLE `Order` ADD CONSTRAINT ORDER\_CREATED\_LESS\_AWAITED\_CHECK CHECK (order\_date\_awaited is NULL or order\_date\_created < order\_date\_awaited);

-- UNIQUE constraints

ALTER TABLE `OrderItems` ADD CONSTRAINT UNIQUE\_ORDER\_PROD\_COMB UNIQUE (order\_id, prod\_id);

5.2 Імпортування даних в таблиці

Для імпортування згенерованих даних можна використати команду LOAD DATA в MySQL вказавши шлях до файлу з даними та додаткову інформацію щодо файлу:

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Tax.csv"

INTO TABLE `Tax`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (tax\_id,tax\_tag,tax\_amount\_perc,tax\_amount\_static);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_ProdCategory.csv"

INTO TABLE `ProdCategory`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (cat\_id,cat\_name,cat\_tax\_id);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Product.csv"

INTO TABLE `Product`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (prod\_id,prod\_name,prod\_price,prod\_cat\_id);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Customer.csv"

INTO TABLE `Customer`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (cust\_id,cust\_name,cust\_email,cust\_phone,cust\_address);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Supplier.csv"

INTO TABLE `Supplier`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (sup\_id,sup\_name,sup\_email,sup\_phone,sup\_address);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Order.csv"

INTO TABLE `Order`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (order\_id,order\_cust\_id,order\_date\_created,order\_date\_awaited,order\_status,order\_dest);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Storage.csv"

INTO TABLE `Storage`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (stor\_id,stor\_location);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_ProductArrival.csv"

INTO TABLE `ProductArrival`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (arrival\_id,arrival\_date,arrival\_prod\_id,arrival\_stor\_id,arrival\_prod\_quantity,arrival\_sup\_id);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_ProductMovement.csv"

INTO TABLE `ProductMovement`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (move\_id,move\_date,move\_prod\_id,move\_stor\_id,move\_prod\_quantity);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Payments.csv"

INTO TABLE `Payments`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (paym\_id,paym\_amount,paym\_date,paym\_order\_id,paym\_cust\_id);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_OrderItems.csv"

INTO TABLE `OrderItems`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (order\_id,prod\_id,prod\_quantity);

LOAD DATA LOCAL INFILE "E:/BD/DB\_automation/data\_Shipments.csv"

INTO TABLE `Shipments`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS (ship\_id,ship\_order\_id,ship\_dest,ship\_date\_dep,ship\_date\_arrival,ship\_status);

**6 СТВОРЕННЯ КОРИСТУВАЧІВ БАЗИ ДАНИХ**

6.1 Адміністратор

-- head admin

CREATE ROLE head\_administrator;

GRANT ALL ON storage\_manager.\* TO head\_administrator;

CREATE USER 'admin\_user'@'localhost' IDENTIFIED BY 'admn123';

GRANT head\_administrator TO 'admin\_user'@'localhost';

SET DEFAULT ROLE ALL TO 'admin\_user'@'localhost';

6.2 Адміністратор складу

-- storage admin

CREATE ROLE storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.tax TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.prodcategory TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.product TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.`Storage` TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.ProductArrival TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.ProductMovement TO storage\_administrator;

GRANT SELECT ON storage\_manager.`Order` TO storage\_administrator;

GRANT SELECT ON storage\_manager.OrderItems TO storage\_administrator;

GRANT SELECT ON storage\_manager.Payments TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.Shipments TO storage\_administrator;

GRANT SELECT ON storage\_manager.Customer TO storage\_administrator;

GRANT SELECT ON storage\_manager.Supplier TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON category\_tax\_view TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON product\_with\_tax TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON overal\_stor\_move TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_prod\_stats TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON product\_quantities TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON not\_send\_but\_ready\_orders TO storage\_administrator;

GRANT UPDATE, DELETE, INSERT, SELECT ON order\_sums\_payments TO storage\_administrator;

GRANT SELECT ON storage\_manager.OrderStatusLog TO storage\_administrator;

GRANT SELECT ON storage\_manager.ShipmentsStatusLog TO storage\_administrator;

GRANT SELECT ON storage\_manager.ProductPriceChangesLog TO storage\_administrator;

CREATE USER 'stor\_user'@'localhost' IDENTIFIED BY 'stor123';

GRANT storage\_administrator TO 'stor\_user'@'localhost';

6.3 Замовник

-- customer

CREATE ROLE customer\_role;

GRANT SELECT ON storage\_manager.tax TO customer\_role;

GRANT SELECT ON storage\_manager.prodcategory TO customer\_role;

GRANT SELECT ON storage\_manager.product TO customer\_role;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.`Order` TO customer\_role;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.OrderItems TO customer\_role;

GRANT INSERT ON storage\_manager.Payments TO customer\_role;

GRANT SELECT ON storage\_manager.Shipments TO customer\_role;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.Customer TO customer\_role;

GRANT SELECT ON category\_tax\_view TO customer\_role;

GRANT SELECT ON product\_with\_tax TO customer\_role;

GRANT SELECT ON not\_send\_but\_ready\_orders TO customer\_role;

GRANT SELECT ON order\_sums\_payments TO customer\_role;

GRANT SELECT ON storage\_manager.OrderStatusLog TO customer\_role;

GRANT SELECT ON storage\_manager.ShipmentsStatusLog TO customer\_role;

GRANT SELECT ON storage\_manager.ProductPriceChangesLog TO customer\_role;

CREATE USER 'cust\_user'@'localhost' IDENTIFIED BY 'cust123';

GRANT customer\_role TO 'cust\_user'@'localhost';

6.4 Менеджер доставки

-- deliverer

CREATE ROLE delivery\_role;

GRANT SELECT ON storage\_manager.prodcategory TO delivery\_role;

GRANT SELECT ON storage\_manager.product TO delivery\_role;

GRANT SELECT ON storage\_manager.`Storage` TO delivery\_role;

GRANT SELECT ON storage\_manager.`Order` TO delivery\_role;

GRANT SELECT ON storage\_manager.OrderItems TO delivery\_role;

GRANT UPDATE, DELETE, INSERT, SELECT ON storage\_manager.Shipments TO delivery\_role;

GRANT SELECT ON storage\_manager.Customer TO delivery\_role;

GRANT SELECT ON overal\_stor\_move TO delivery\_role;

GRANT SELECT ON storage\_prod\_stats TO delivery\_role;

GRANT SELECT ON product\_quantities TO delivery\_role;

GRANT SELECT ON not\_send\_but\_ready\_orders TO delivery\_role;

GRANT SELECT ON order\_sums\_payments TO delivery\_role;

GRANT SELECT ON storage\_manager.OrderStatusLog TO delivery\_role;

GRANT SELECT ON storage\_manager.ShipmentsStatusLog TO delivery\_role;

GRANT SELECT ON storage\_manager.ProductPriceChangesLog TO delivery\_role;

CREATE USER 'deliv\_user'@'localhost' IDENTIFIED BY 'deli123';

GRANT delivery\_role TO 'deliv\_user'@'localhost';

Результат виконання команд для створення ролей та користувачів командами SQL на рис. 6.1 та рис. 6.2.

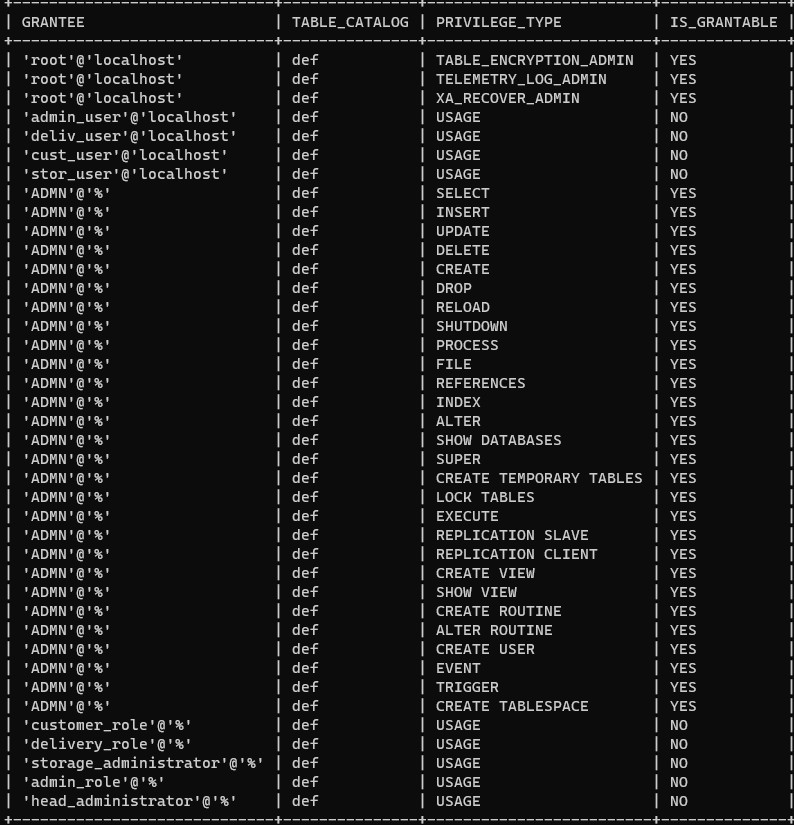


Рис. 6.1 – Створені користувачі в базі даних

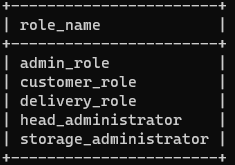


Рис. 6.2 – Створені ролі в базі даних

**7 РОБОТА З БАЗОЮ ДАНИХ**

7.1 Тригери

Відповідно до моїх бізнес правил були розроблені наступні тригери на таблицях.

7.1.1 Триггер ClearTax

Тригер ClearTax ставить нулові значення на всіх категоріях які мали видалений податок. Приклад роботи показано на рис. 7.1.1.1 та 7.1.1.2.

DELIMITER //

CREATE TRIGGER ClearTax

BEFORE DELETE ON TAX

FOR EACH ROW

BEGIN

UPDATE ProdCategory SET `cat\_tax\_id` = Null WHERE `cat\_tax\_id` = OLD.tax\_id;

END //

DELIMITER ;

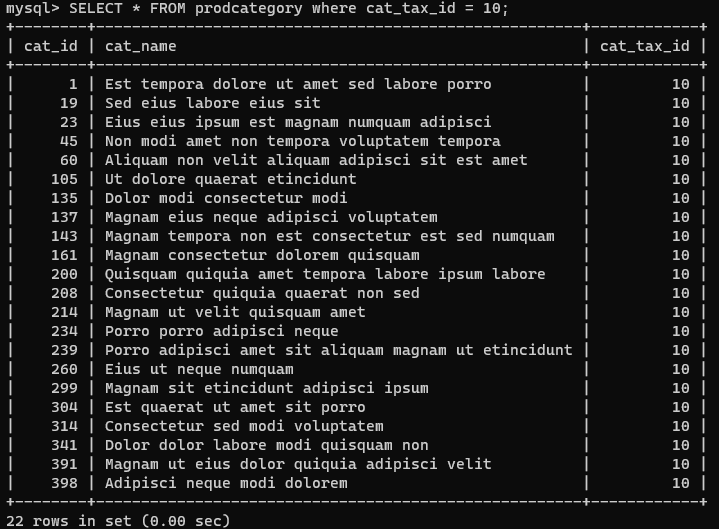


Рис. 7.1.1.1 – Таблиця категорій до видалення податку

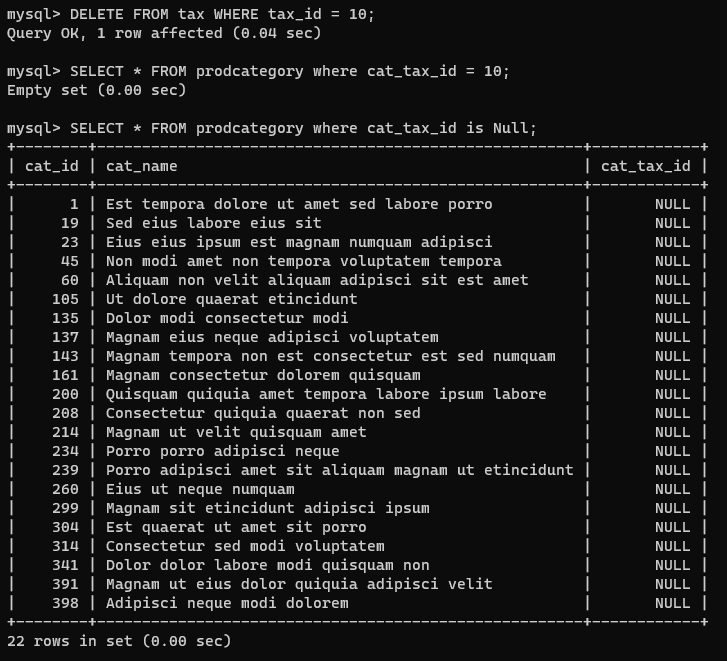


Рис. 7.1.1.2 – Таблиця категорій після видалення податку

7.1.2 Триггер ClearCat

Тригер ClearCat ставить нулові значення на категорію всіх продуктів після видалення категорії. Приклад роботи показано на рис. 7.1.2.1 та рис. 7.1.2.2.

DELIMITER //

CREATE TRIGGER ClearCat

BEFORE DELETE ON PRODCATEGORY

FOR EACH ROW

BEGIN

UPDATE product SET `prod\_cat\_id` = Null WHERE `prod\_cat\_id` = OLD.cat\_id;

END //

DELIMITER ;

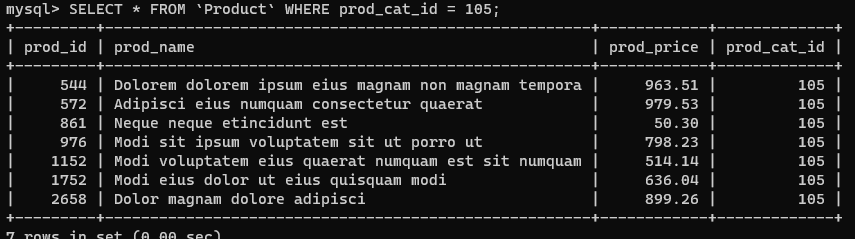


Рис. 7.1.2.1 – Таблиця продуктів до видалення категорії

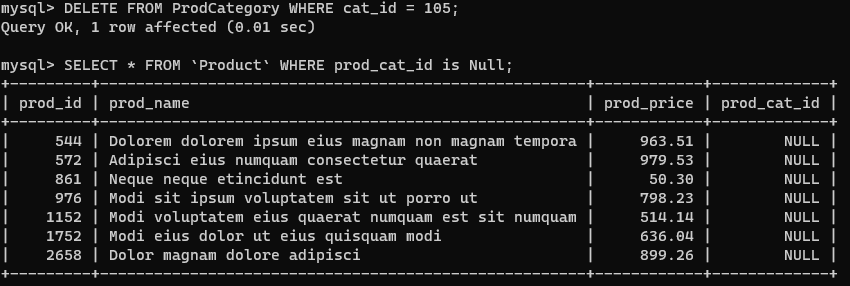


Рис. 7.1.2.2 – Таблиця продуктів після видалення категорії

7.1.3 Триггер DeleteProduct

Триггер DeleteProduct створений для видалення всіх пересувань та привезень видаленого продукту а також видалення всіх додавань до замовлення. Приклад роботи показано на рис. 7.1.3.1 та рис. 7.1.3.2.

DELIMITER //

CREATE TRIGGER DeleteProduct

BEFORE DELETE ON PRODUCT

FOR EACH ROW

BEGIN

DELETE FROM PRODUCTMOVEMENT WHERE `move\_prod\_id` = OLD.prod\_id;

DELETE FROM PRODUCTARRIVAL WHERE `arrival\_prod\_id` = OLD.prod\_id;

DELETE FROM ORDERITEMS WHERE `prod\_id` = OLD.prod\_id;

END //

DELIMITER ;

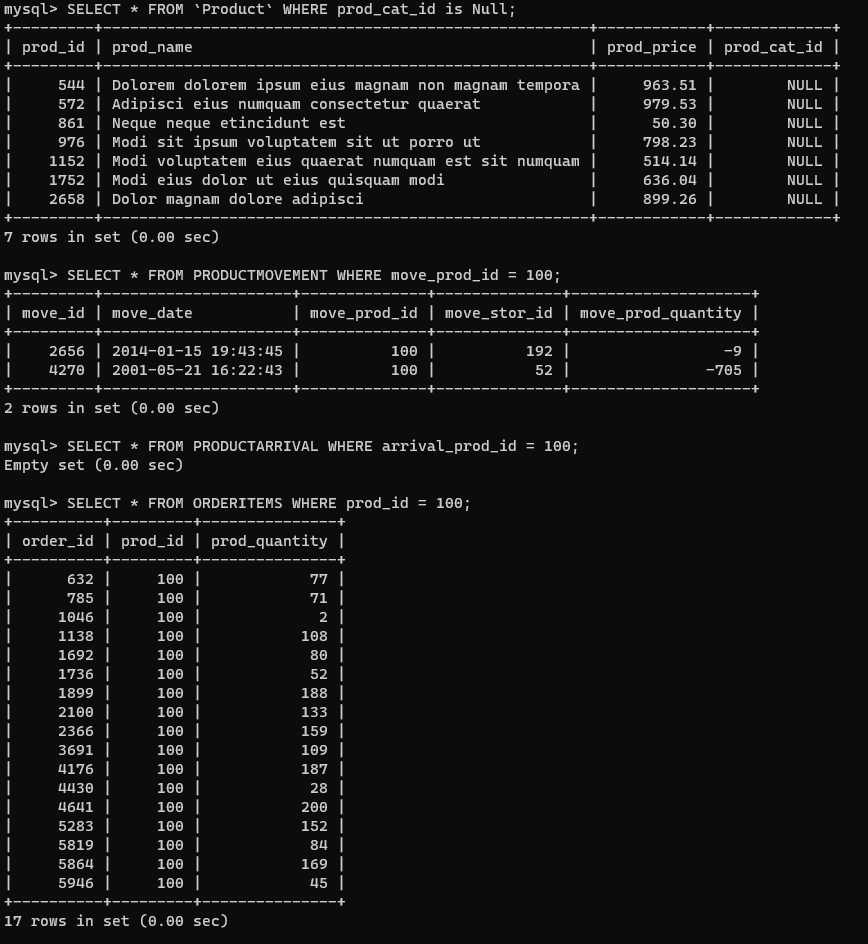


Рис. 7.1.3.1 – Таблиці надходжень, пересувань та продуктів замовлення до видалення продукта

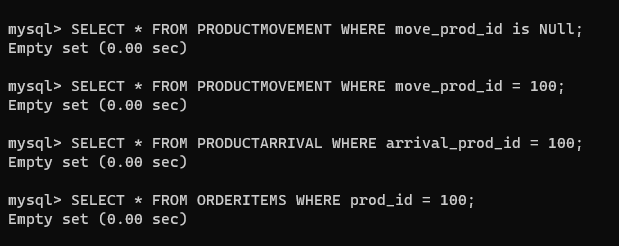


Рис. 7.1.3.2 – Таблиці надходжень, пересувань та продуктів замовлення після видалення продукта

7.1.4 Триггер DeleteStorage

Триггер DeleteStorage видаляє пересування та надходження продуктів зі та в склад що було видалено. Приклад роботи показано на рис. 7.1.4.1 та рис. 7.1.4.2.

DELIMITER //

CREATE TRIGGER DeleteStorage

BEFORE DELETE ON `STORAGE`

FOR EACH ROW

BEGIN

DELETE FROM PRODUCTMOVEMENT WHERE `move\_stor\_id` = OLD.stor\_id;

DELETE FROM PRODUCTARRIVAL WHERE `arrival\_stor\_id` = OLD.stor\_id;

END //

DELIMITER ;

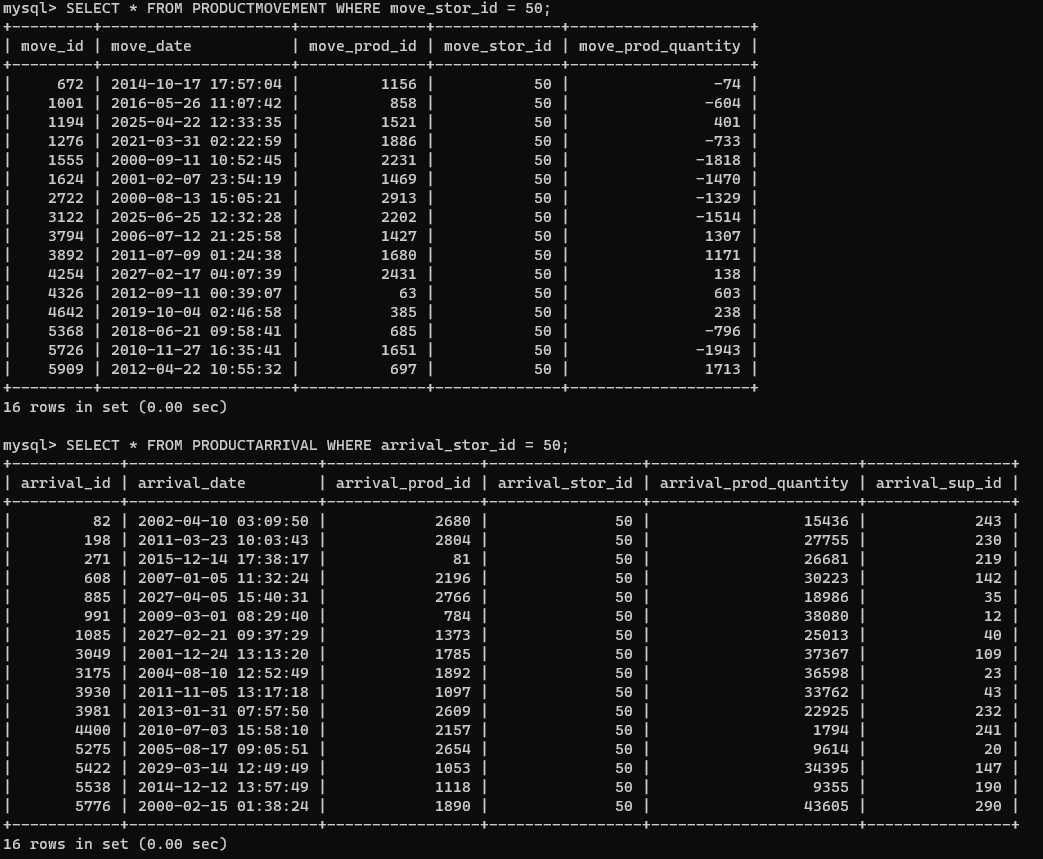


Рис. 7.1.4.1 – Таблиці надходжень та пересувань продуктів до видалення склада

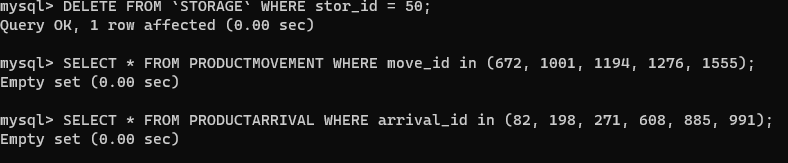


Рис. 7.1.4.2 – Таблиці надходжень та пересувань продуктів після видалення склада

7.1.5 Триггер ClearSupp

Тригер ClearSupp ставить нулові значення на всіх доставках видаленого постачальника. Приклад роботи показано на рис. 7.1.5.1 та рис. 7.1.5.2.

DELIMITER //

CREATE TRIGGER ClearSupp

BEFORE DELETE ON SUPPLIER

FOR EACH ROW

BEGIN

UPDATE PRODUCTARRIVAL SET `arrival\_sup\_id` = Null WHERE `arrival\_sup\_id` = OLD.sup\_id;

END //

DELIMITER ;

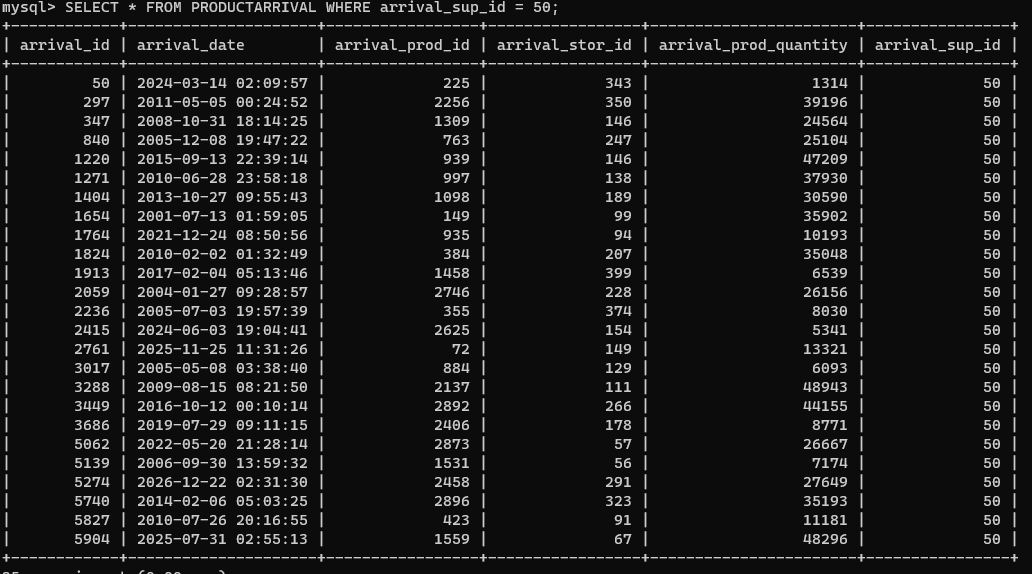


Рис. 7.1.5.1 – Таблиця надходжень продуктів до видалення постачальника

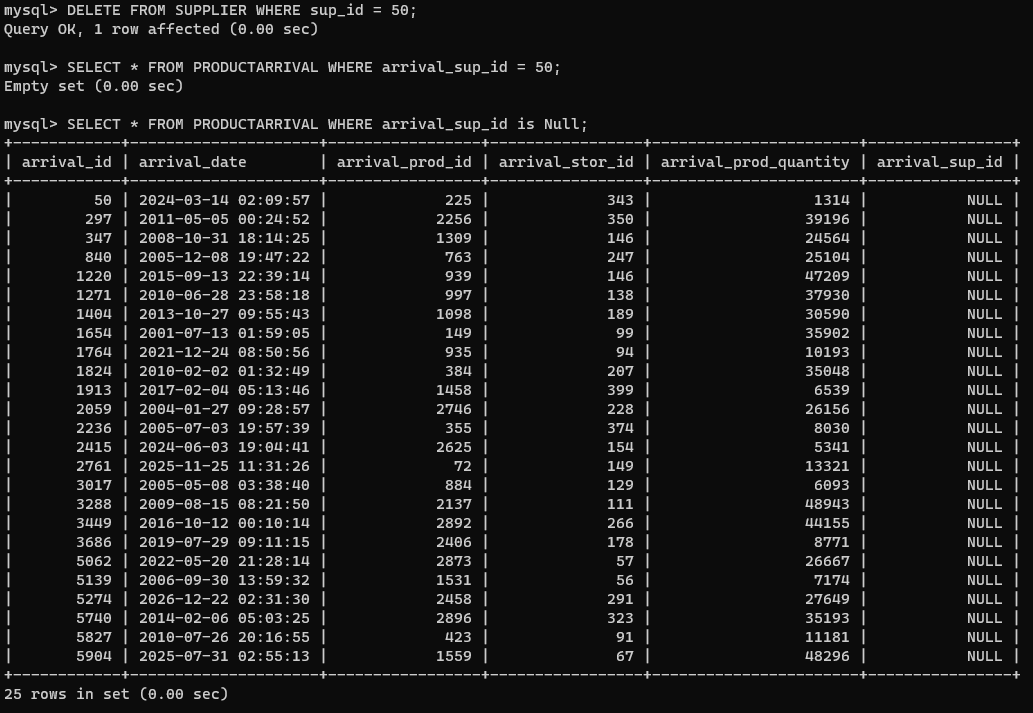


Рис. 7.1.5.2 – Таблиця надходжень продуктів після видалення постачальника

7.1.6 Триггер DeleteOrder

Триггер DeleteOrder виконується перед видаленням замовлення та видаляє пов’язані з замовленням доставки, оплати та додані товари. Приклад роботи показано на рис. 7.1.6.1 та рис. 7.1.6.2.

DELIMITER //

CREATE TRIGGER DeleteOrder

BEFORE DELETE ON `ORDER`

FOR EACH ROW

BEGIN

DELETE FROM OrderItems WHERE `order\_id` = OLD.order\_id;

DELETE FROM Shipments WHERE `ship\_order\_id` = OLD.order\_id;

DELETE FROM Payments WHERE `paym\_order\_id` = OLD.order\_id;

END //

DELIMITER ;

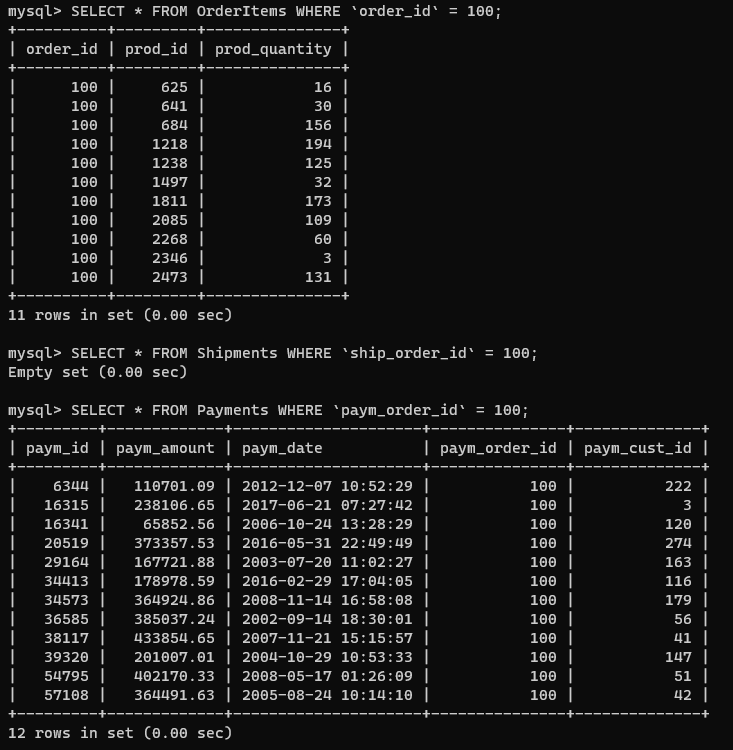


Рис. 7.1.6.1 – Таблиці доданих продуктів, доставок та сплат до видалення замовлення

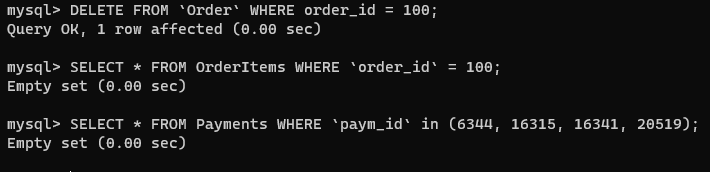


Рис. 7.1.6.2 – Таблиці доданих продуктів, доставок та сплат після видалення замовлення

7.1.7 Триггер DeleteCustomer

Триггер DeleteCustomer виконується видалення замовника та видаляє всі замовлення та сплати, що були створені ним. Приклад роботи показано на рис. 7.1.7.1 та рис. 7.1.7.3.

DELIMITER //

CREATE TRIGGER DeleteCustomer

BEFORE DELETE ON Customer

FOR EACH ROW

BEGIN

DELETE FROM `Order` WHERE `order\_cust\_id` = OLD.cust\_id;

DELETE FROM Payments WHERE `paym\_cust\_id` = OLD.cust\_id;

END //

DELIMITER ;

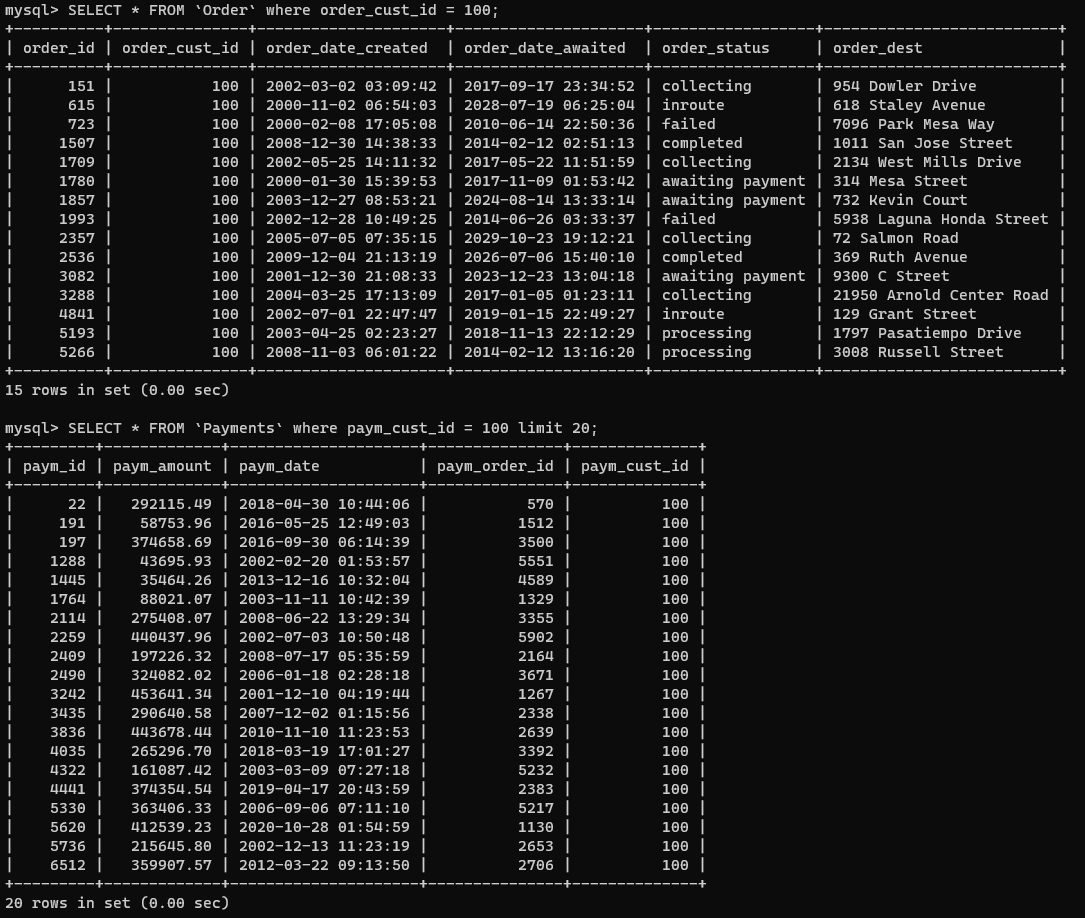


Рис. 7.1.7.1 – Таблиці сплат та замовлень до видалення замовника

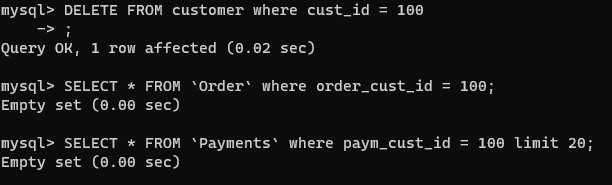


Рис. 7.1.7.3 – Таблиці сплат та замовлень після видалення замовника

7.1.8 Триггер LogPriceChangeInsert

Тригер LogPriceChangeInsert спрацьовує при створенні нового продукту записуючи час створення та ціну продукту. Приклад роботи показано на рис. 7.1.8.1.

DELIMITER //

CREATE TRIGGER LogPriceChangeInsert

AFTER INSERT ON Product

FOR EACH ROW

BEGIN

INSERT INTO ProductPriceChangesLog (`prod\_id`, `price\_from`, `price\_to`) VALUES (new.prod\_id, null, new.prod\_price);

END //

DELIMITER ;

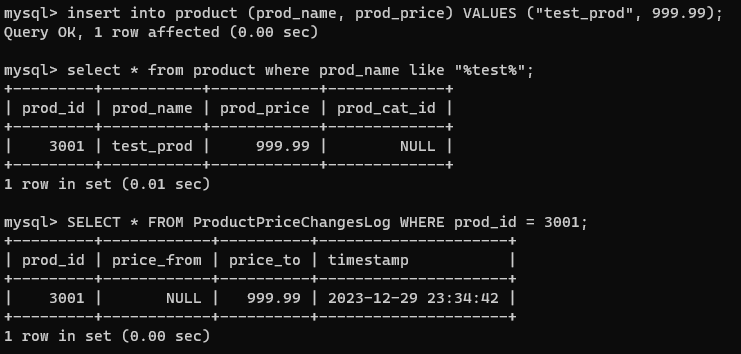


Рис. 7.1.8.1 – Приклад роботи тригера LogPriceChangeInsert

7.1.9 Триггер LogOrderStatusInsert

Тригер LogOrderStatusInsert спрацьовує при додаванні нового замовлення та записує його статус та час. Приклад роботи показано на рис. 7.1.9.1.

DELIMITER //

CREATE TRIGGER LogOrderStatusInsert

AFTER INSERT ON `Order`

FOR EACH ROW

BEGIN

INSERT INTO OrderStatusLog (`order\_id`, `status\_from`, `status\_to`) VALUES (new.order\_id, null, new.order\_status);

END //

DELIMITER ;

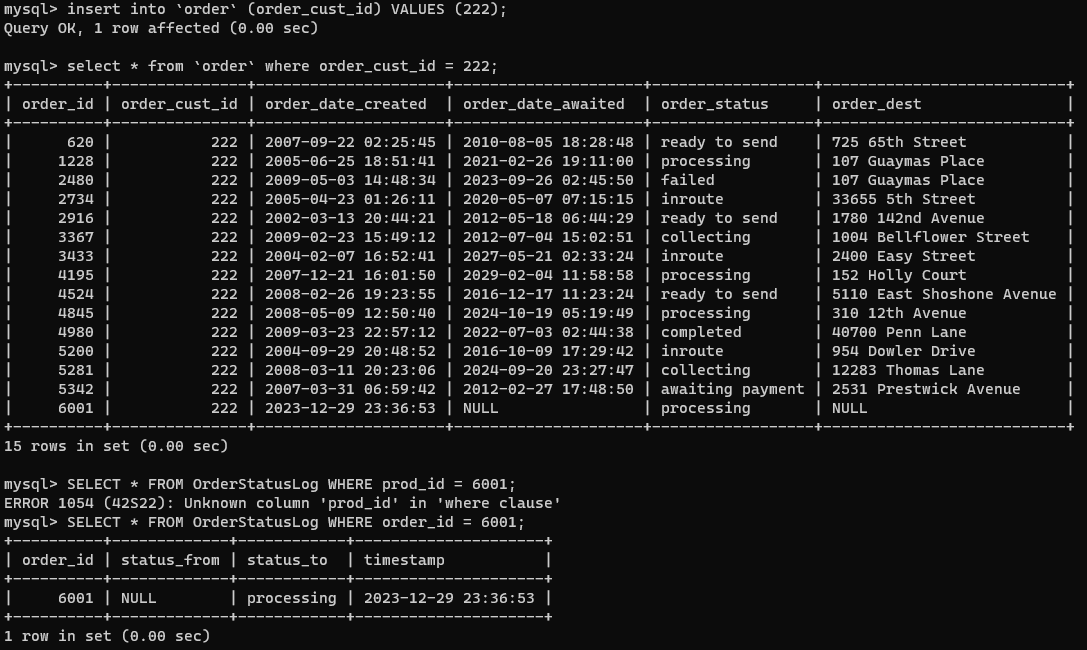


Рис. 7.1.9.1 – Приклад роботи тригера LogPriceChangeInsert

7.1.10 Триггер LogShipmentsStatusInsert

Тригер LogShipmentsStatusInsert спрацьовує схоже з минулим та записує при створенні нової її час та статус. Приклад роботи показано на рис. 7.1.10.1.

DELIMITER //

CREATE TRIGGER LogShipmentsStatusInsert

AFTER INSERT ON `Shipments`

FOR EACH ROW

BEGIN

INSERT INTO ShipmentsStatusLog (`ship\_id`, `status\_from`, `status\_to`) VALUES (new.ship\_id, null, new.ship\_status);

END //

DELIMITER ;

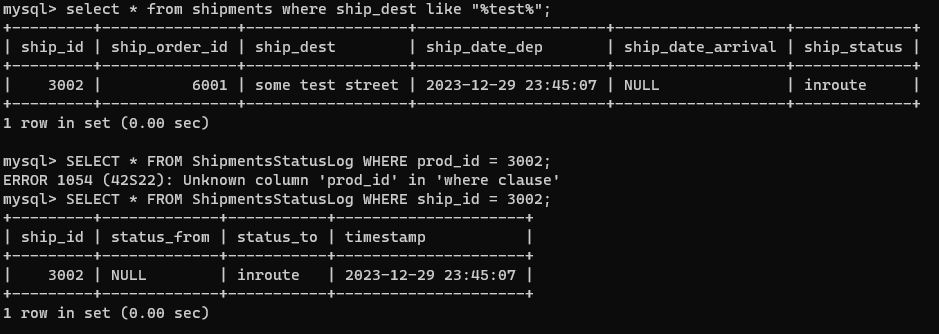


Рис. 7.1.10.1 – Приклад роботи тригера LogPriceChangeInsert

7.1.11 Триггер LogPriceChange

Тригер LogPriceChange спрацьовує при оновленні ціни на продукт та записує її старе та нове значення разом з датою. Приклад роботи показано на рис. 7.1.11.

DELIMITER //

CREATE TRIGGER LogPriceChange

AFTER UPDATE ON Product

FOR EACH ROW

BEGIN

IF old.prod\_price != new.prod\_price THEN

INSERT INTO ProductPriceChangesLog (`prod\_id`, `price\_from`, `price\_to`) VALUES (new.prod\_id, old.prod\_price, new.prod\_price);

END IF;

END //

DELIMITER ;

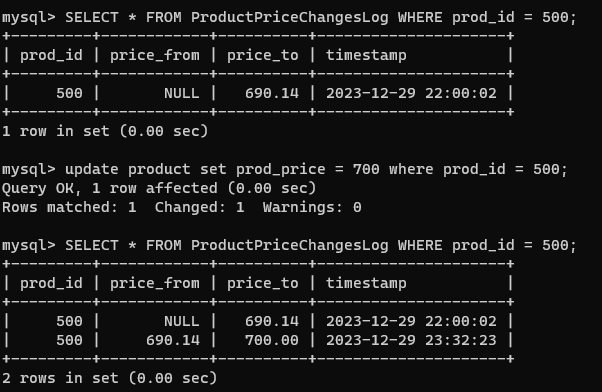


Рис. 7.1.11.1 – Приклад роботи тригера LogPriceChange

7.1.12 Триггер LogOrderStatus

Тригер LogOrderStatus спрацьовує при оновленні статуса замовлення записуючи старий та новий статуси разом з часом зміни. Приклад роботи показано на рис. 7.1.12.1.

DELIMITER //

CREATE TRIGGER LogOrderStatus

AFTER UPDATE ON `Order`

FOR EACH ROW

BEGIN

IF old.order\_status != new.order\_status THEN

INSERT INTO OrderStatusLog (`order\_id`, `status\_from`, `status\_to`) VALUES (new.order\_id, old.order\_status, new.order\_status);

END IF;

END //

DELIMITER ;

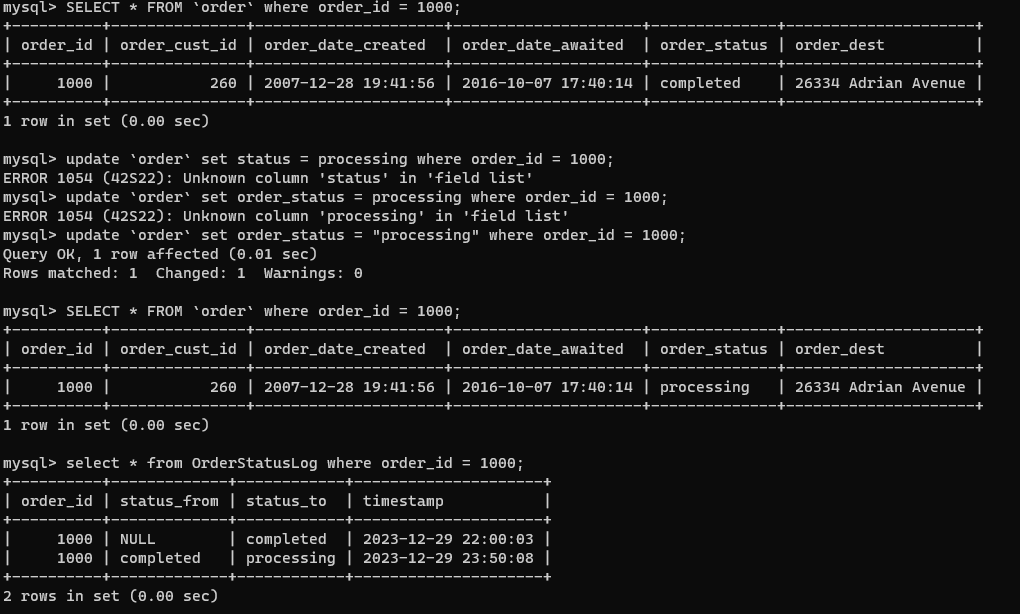


Рис. 7.1.12.1 – Приклад роботи тригера LogOrderStatus

7.1.13 Триггер LogShipmentsStatus

Тригер LogShipmentsStatus спрацьовує при зміни статуса доставки записуючи нове та минуле значення разом з часом. Приклад роботи показано на рис. 7.1.13.1.

DELIMITER //

CREATE TRIGGER LogShipmentsStatus

AFTER UPDATE ON `Shipments`

FOR EACH ROW

BEGIN

INSERT INTO ShipmentsStatusLog (`ship\_id`, `status\_from`, `status\_to`) VALUES (new.ship\_id, old.ship\_status, new.ship\_status);

END //

DELIMITER ;

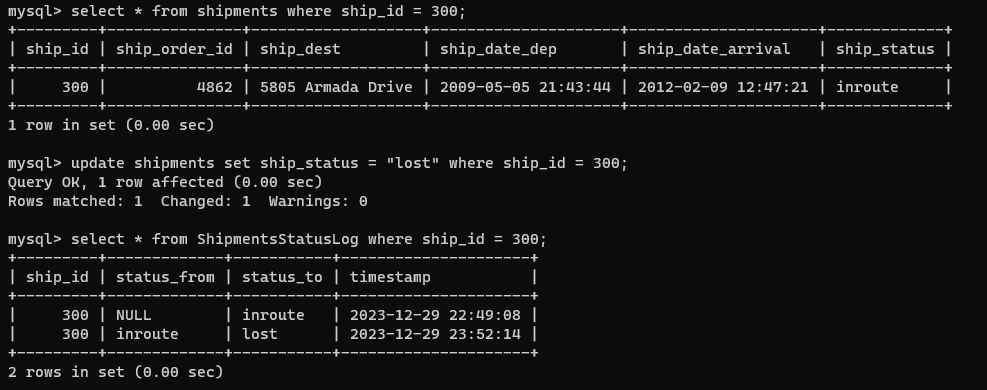


Рис. 7.1.13.1 – Приклад роботи тригера LogShipmentsStatus

7.2 Тексти представлень

7.2.1 Представлення category\_tax\_view

Представлення category\_tax\_view створено для швидкого доступу до податків кожної категорії. Приклад роботи на рис. 7.2.1.

CREATE OR REPLACE VIEW category\_tax\_view AS

SELECT cat\_id, cat\_name, tax\_tag, tax\_amount\_perc, tax\_amount\_static FROM ProdCategory

LEFT JOIN Tax ON ProdCategory.cat\_tax\_id = Tax.tax\_id;

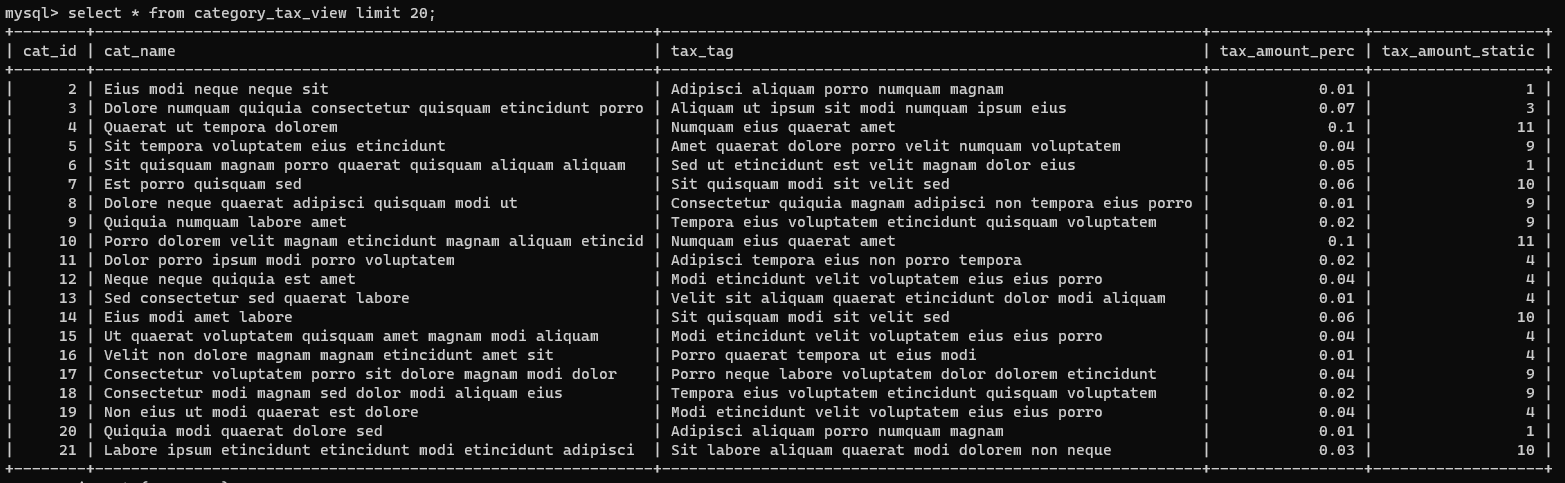


Рис. 7.2.1 – Приклад роботи представлення category\_tax\_view

7.2.2 Представлення product\_with\_tax

Представлення product\_with\_tax створено для швидкого доступу до продуктів з вже обрахованими цінами з податками. Приклад роботи показано на рис. 7.2.2.

CREATE OR REPLACE VIEW product\_with\_tax AS

SELECT prod\_id, prod\_name, Product.prod\_price, ROUND(Product.prod\_price + Product.prod\_price \* IFNULL(category\_tax\_view.tax\_amount\_perc,0) + IFNULL(category\_tax\_view.tax\_amount\_static,0), 2) AS prod\_taxed\_price, prod\_cat\_id FROM Product

LEFT JOIN category\_tax\_view ON Product.prod\_cat\_id = category\_tax\_view.cat\_id;

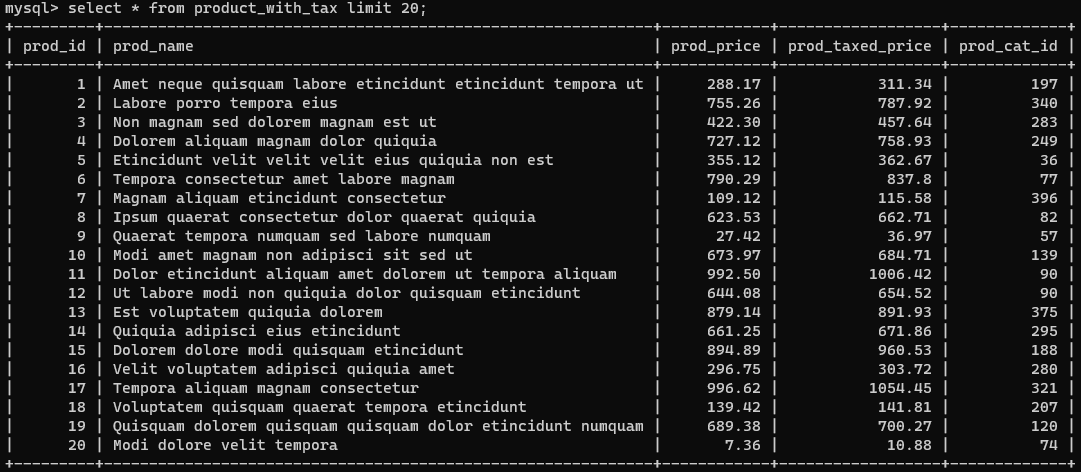


Рис. 7.2.2 – Приклад роботи представлення product\_with\_tax

7.2.3 Представлення overal\_stor\_move

Представлення overal\_stor\_move для переведення двох таблиць пересувань продуктів в одне представлення для обрахунку об’ємів кожного складу.

CREATE OR REPLACE VIEW overal\_stor\_move AS

SELECT ProductMovement.move\_stor\_id as stor\_id, ProductMovement.move\_prod\_id as prod\_id, ProductMovement.move\_prod\_quantity as prod\_move FROM ProductMovement

UNION

SELECT ProductArrival.arrival\_stor\_id, ProductArrival.arrival\_prod\_id, ProductArrival.arrival\_prod\_quantity FROM ProductArrival ORDER BY stor\_id ASC;

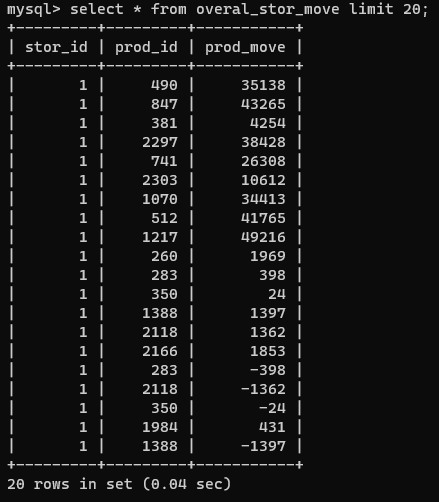


Рис. 7.2.3 – Приклад роботи представлення overal\_stor\_move

7.2.4 Представлення storage\_prod\_stats

Представлення storage\_prod\_stats створено для обрахунку об’ємів продуктів на кожному складі. Приклад роботи на рис. 7.2.4.

CREATE OR REPLACE VIEW storage\_prod\_stats AS

SELECT stor\_id, prod\_id, SUM(prod\_move) as prod\_quantity FROM overal\_stor\_move GROUP BY stor\_id, prod\_id ORDER BY stor\_id ASC, prod\_id ASC;

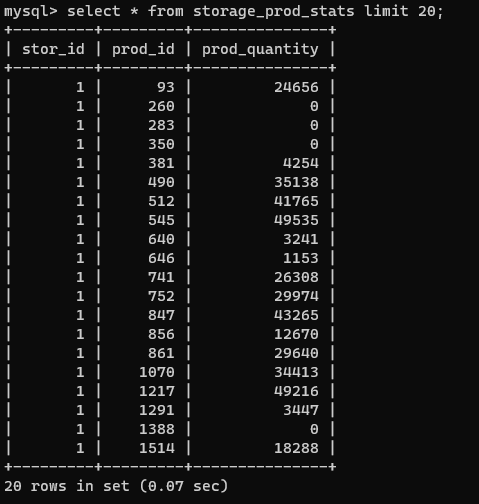


Рис. 7.2.4 – Приклад роботи представлення storage\_prod\_stats

7.2.5 Представлення product\_quantities

Представлення product\_quantities створено для підрахунку об’ємів всіх продуктів на всіх складах. Приклад роботи показано на рис. 7.2.5.

CREATE OR REPLACE VIEW product\_quantities AS

SELECT Product.prod\_id,prod\_name,prod\_price,prod\_cat\_id, IFNULL(SUM(prod\_quantity),0) as prod\_quantity FROM Product LEFT JOIN storage\_prod\_stats on Product.prod\_id = storage\_prod\_stats.prod\_id GROUP BY Product.prod\_id;

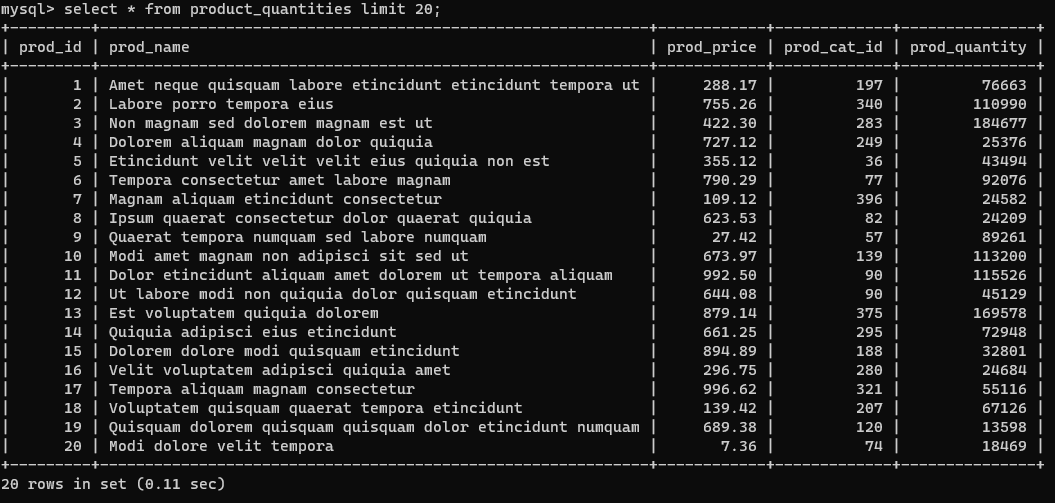


Рис. 7.2.5 – Приклад роботи представлення product\_quantities

7.2.6 Представлення not\_send\_but\_ready\_orders

Представлення not\_send\_but\_ready\_orders повертає замовлення що готові до відправки але ще не відправлені. Приклад роботи на рис. 7.2.6.

CREATE OR REPLACE VIEW not\_send\_but\_ready\_orders AS

SELECT \* FROM `Order` WHERE order\_status = "ready to send" AND order\_id not in (SELECT ship\_order\_id FROM Shipments);

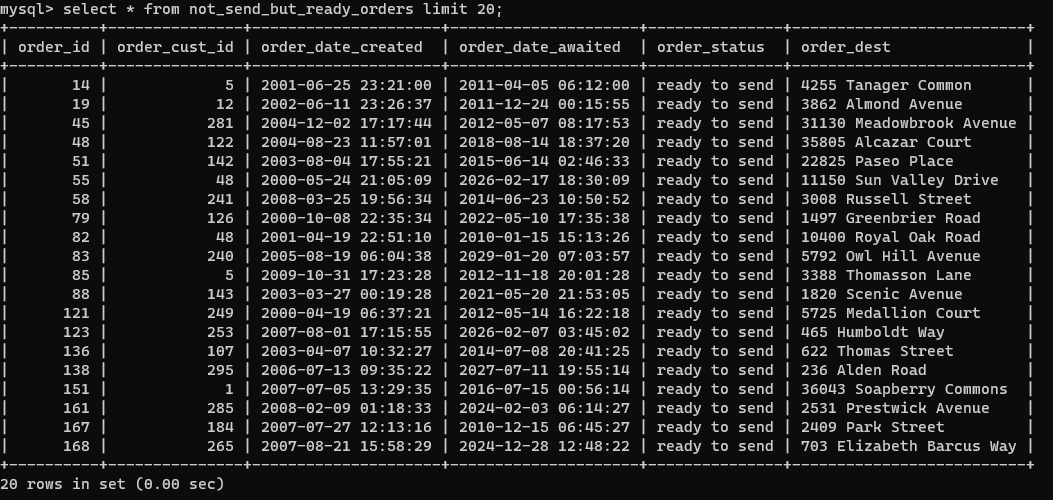


Рис. 7.2.6 – Приклад роботи представлення not\_send\_but\_ready\_orders

7.2.7 Представлення order\_sums\_payments

Представлення order\_sums\_payments повертає замовлення з їх сплатами. Приклад роботи на рис. 7.2.7.

CREATE OR REPLACE VIEW order\_sums\_payments AS

SELECT

o.\*,

SUM(order\_sum) as order\_sum,

SUM(sum\_payed) as sum\_payed

FROM `Order` o

LEFT JOIN (

SELECT p.paym\_order\_id as order\_id, p.paym\_amount as sum\_payed, 0 as order\_sum FROM `Payments` p

UNION

SELECT oi.order\_id, 0, pwt.prod\_taxed\_price \* oi.prod\_quantity FROM `OrderItems` oi

INNER JOIN `product\_with\_tax` pwt ON oi.`prod\_id` = pwt.`prod\_id`

) c ON c.order\_id = o.order\_id

group by o.order\_id;

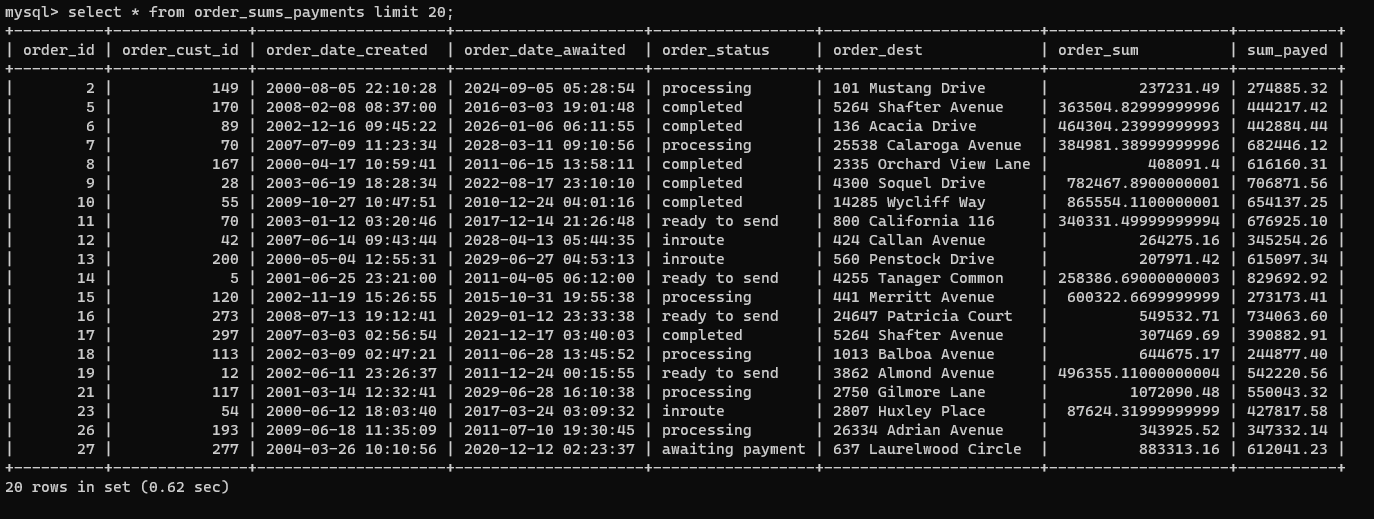


Рис. 7.2.7 – Приклад роботи представлення order\_sums\_payments

7.3 Тексти збережених процедур та функцій.

Відповідно до моїх бізнес правил були розроблені наступні функції та процедури.

7.3.1 Функція CalculateOrderSum

Функція CalculateOrderSum обраховує суму до сплати за окреме замовлення та повертає його як округлене значення. Приклад роботи показано на рис. 7.3.1.1.

DELIMITER //

CREATE FUNCTION CalculateOrderSum(calc\_order\_id int) RETURNS DOUBLE

READS SQL DATA

NOT DETERMINISTIC

BEGIN

DECLARE result DOUBLE;

SELECT SUM(OrderItems.prod\_quantity \* product\_with\_tax.prod\_taxed\_price) INTO result FROM OrderItems INNER JOIN product\_with\_tax ON OrderItems.prod\_id = product\_with\_tax.prod\_id AND OrderItems.order\_id = calc\_order\_id;

RETURN ROUND(result,2);

END //

DELIMITER ;



Рис. 7.3.1.1 – Приклад результату від функції CalculateOrderSum

7.3.2 Функція CalculateOrderAmountPayed

Функція CalculateOrderAmountPayed обраховує вже сплачену за замовлення суму та повертає її як округлене значення. Приклад роботи показано на рис. 7.3.2.1.

DELIMITER //

CREATE FUNCTION CalculateOrderAmountPayed(calc\_order\_id int) RETURNS DOUBLE

READS SQL DATA

NOT DETERMINISTIC

BEGIN

DECLARE result DOUBLE;

SELECT SUM(Payments.paym\_amount) INTO result FROM Payments WHERE Payments.paym\_order\_id = calc\_order\_id;

RETURN ROUND(result,2);

END //

DELIMITER ;



Рис. 7.3.2.1 – Приклад результату функції CalculateOrderAmountPayed

7.3.3 Процедура UpdateAllPayedOrders

Процедура UpdateAllPayedOrders оновлює всі замовлення що були сплачені замовниками, пересуваючи їх на подальший етап статуса. Приклад роботи показано на рис. 7.3.3.1 та рис. 7.3.3.2.

DELIMITER //

CREATE PROCEDURE UpdateAllPayedOrders()

BEGIN

UPDATE `Order` o INNER JOIN order\_sums\_payments osp ON o.order\_id = osp.order\_id AND o.order\_status = "awaiting payment" AND order\_sum <= sum\_payed

SET o.order\_status = 'collecting';

END //

DELIMITER ;

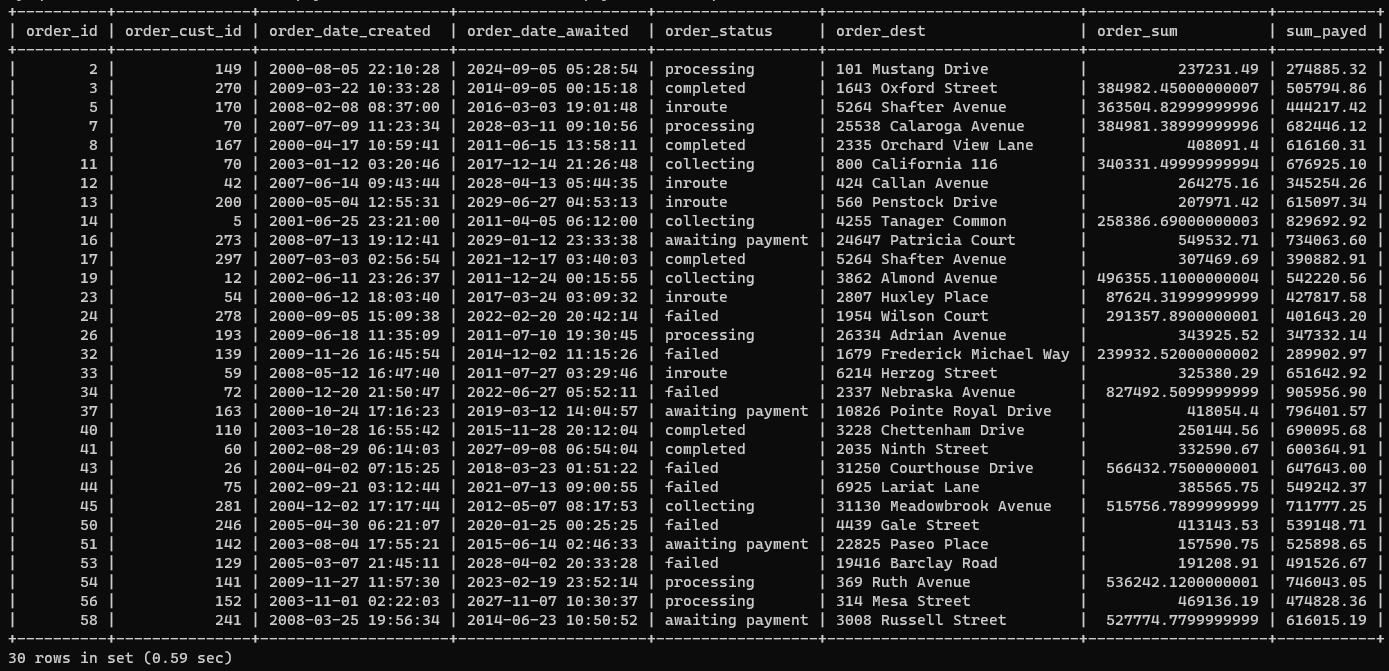


Рис. 7.3.3.1 – Приклад статусів замовлень до виклику UpdateAllPayedOrders

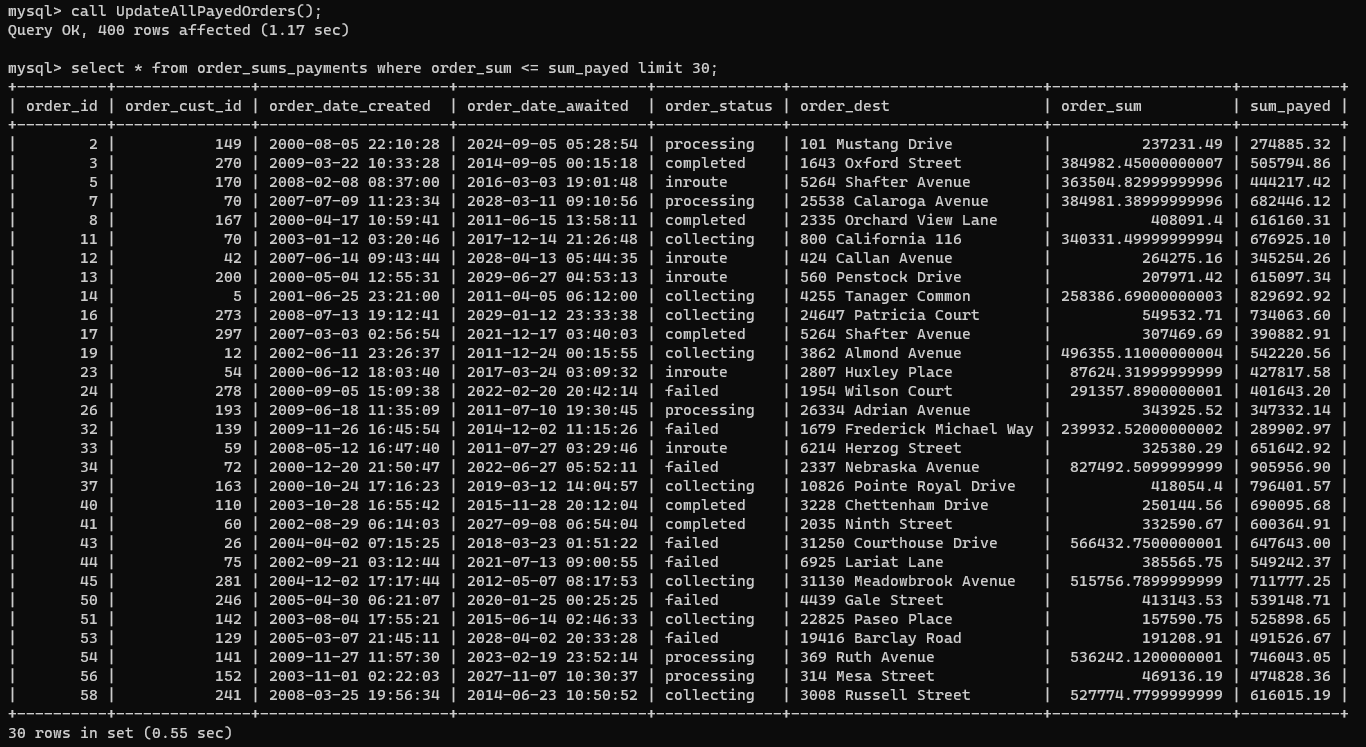


Рис. 7.3.3.2 – Приклад статусів замовлень після виклику UpdateAllPayedOrders

7.3.4 Процедура SendAllReadyOrders

Процедура SendAllReadyOrders створює доставки для всіх готовий до відправки замовлень. Приклад роботи показано на рис. 7.3.4.1.

DELIMITER //

CREATE PROCEDURE SendAllReadyOrders()

BEGIN

INSERT INTO Shipments (ship\_order\_id, ship\_dest) SELECT order\_id, order\_dest FROM not\_send\_but\_ready\_orders WHERE order\_dest IS NOT NULL;

UPDATE `Order` SET order\_status = "inroute" WHERE order\_status = "ready to send" AND order\_id in (SELECT ship\_order\_id FROM Shipments);

END //

DELIMITER ;

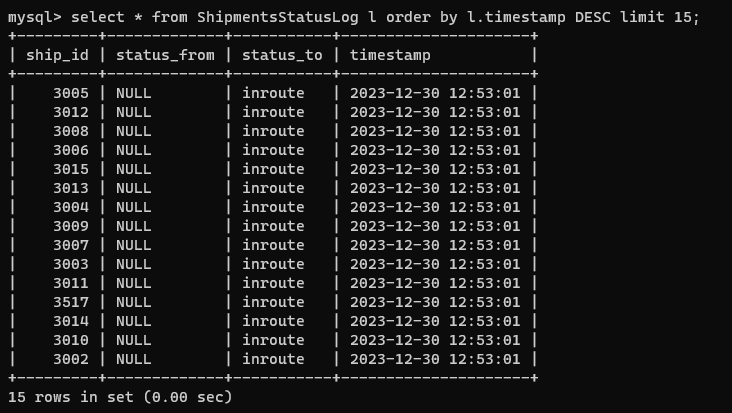


Рис. 7.3.4.1 – Новостворені доставки після запуску SendAllReadyOrders

7.3.5 Процедура CompleteAllArrivedOrders

Процедура CompleteAllArrivedOrders завершує всі замовлення, що були доставлені. Приклад роботи показано на рис. 7.3.5.1.

DELIMITER //

CREATE PROCEDURE CompleteAllArrivedOrders()

BEGIN

UPDATE `Order` SET order\_status = "completed" WHERE order\_status = "inroute" AND order\_id in (SELECT ship\_order\_id FROM Shipments WHERE ship\_status = "arrived");

END //

DELIMITER ;

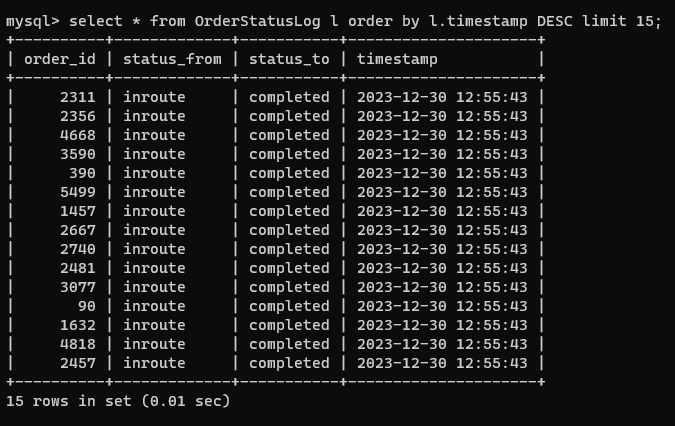


Рис. 7.3.5.1 – Оновленні статуси замовлень після запуску CompleteAllArrivedOrders

7.3.6 Процедура AssembleProduct

Процедура AssembleProduct надає інформацію де можна знайти цей продукт на складах. Приклад роботи показано на рис. 7.3.6.1.

DELIMITER //

CREATE PROCEDURE AssembleProduct(IN asmbl\_prod\_id int, IN asmbl\_prod\_quantity INT)

BEGIN

DECLARE cur\_stor\_id INT DEFAULT 0;

DECLARE cur\_stor\_quatity INT DEFAULT 0;

DECLARE done INT DEFAULT 0;

DECLARE a\_prod\_curs CURSOR FOR SELECT stor\_id, prod\_quantity FROM storage\_prod\_stats WHERE prod\_id = asmbl\_prod\_id ORDER BY prod\_quantity DESC;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

CREATE TEMPORARY TABLE TAKE\_PRODUCTS(

`stor\_id` int,

`prod\_quantity` int

);

OPEN a\_prod\_curs;

read\_loop: LOOP

FETCH a\_prod\_curs INTO cur\_stor\_id, cur\_stor\_quatity;

IF done THEN

LEAVE read\_loop;

END IF;

IF cur\_stor\_quatity <= 0 THEN

LEAVE read\_loop;

END IF;

IF asmbl\_prod\_quantity - cur\_stor\_quatity <= 0 THEN

INSERT INTO TAKE\_PRODUCTS (stor\_id, prod\_quantity) VALUES (cur\_stor\_id, asmbl\_prod\_quantity);

SET asmbl\_prod\_quantity = 0;

LEAVE read\_loop;

ELSE

INSERT INTO TAKE\_PRODUCTS (stor\_id, prod\_quantity) VALUES (cur\_stor\_id, cur\_stor\_quatity);

SET asmbl\_prod\_quantity = asmbl\_prod\_quantity - cur\_stor\_quatity;

END IF;

END LOOP;

SELECT \* FROM TAKE\_PRODUCTS;

IF asmbl\_prod\_quantity > 0 THEN

SELECT CONCAT("!!!Failed to find ", asmbl\_prod\_quantity, " more products of product\_", asmbl\_prod\_id, "!!!") AS ERROR\_NOTICE;

END IF;

DROP TABLE TAKE\_PRODUCTS;

CLOSE a\_prod\_curs;

END //

DELIMITER ;

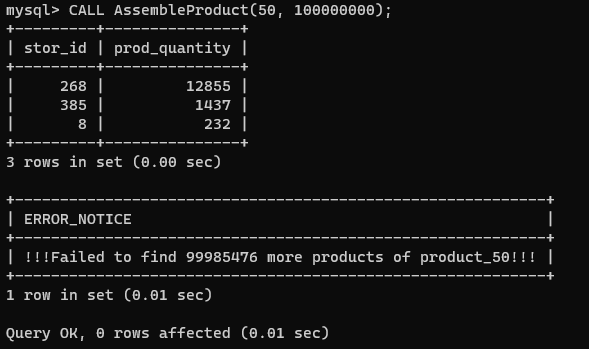


Рис. 7.3.6.1 – Результат коли ми намагаємось зібрати 100000000 одиниць продукту з ідентифікатором 50

7.3.7 Процедура AssembleOrder

Процедура AssembleOrder виписує на яких складах взяти це замовлення. Приклад роботи показано на рис. 7.3.7.1.

DELIMITER //

CREATE PROCEDURE AssembleOrder(IN asmbl\_order\_id int)

BEGIN

DECLARE cur\_prod\_id INT DEFAULT 0;

DECLARE cur\_prod\_quatity INT DEFAULT 0;

DECLARE done INT DEFAULT 0;

DECLARE curs CURSOR FOR SELECT prod\_id, prod\_quantity FROM OrderItems WHERE order\_id = asmbl\_order\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN curs;

SELECT "=============================================================================================================" AS `SEPARATOR`;

read\_loop: LOOP

FETCH curs INTO cur\_prod\_id, cur\_prod\_quatity;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT("Collect ", cur\_prod\_quatity, " of product\_", cur\_prod\_id, " from these storages:") as SYSTEM\_MESSAGE;

CALL AssembleProduct(cur\_prod\_id, cur\_prod\_quatity);

END LOOP;

CLOSE curs;

SELECT "=============================================================================================================" AS `SEPARATOR`;

END //

DELIMITER ;

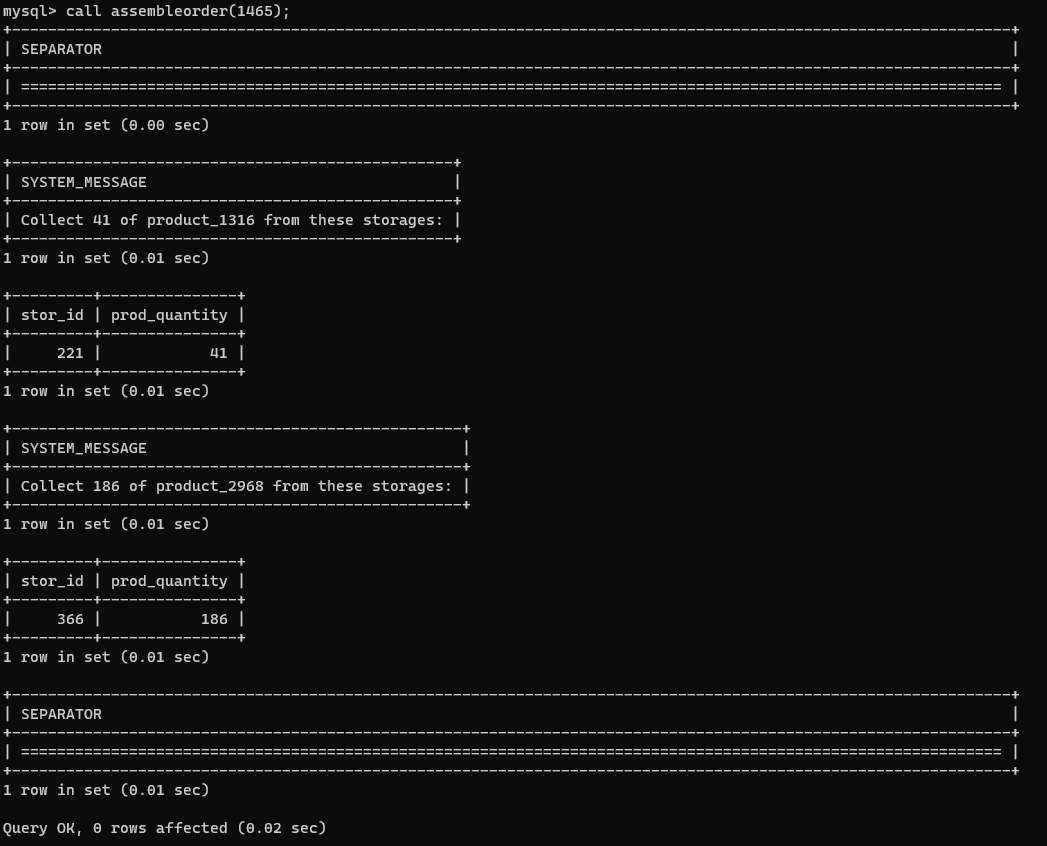


Рис. 7.3.7.1 – Результат збору замовлення

7.3.8 Процедура ArriveAbsentProducts

Процедура ArriveAbsentProducts додає всі невистачаючі продукти до складів від вказаного постачальника. Приклад роботи показано на рис. 7.3.8.1.

DELIMITER //

CREATE PROCEDURE ArriveAbsentProducts(IN arr\_sup\_id INT)

BEGIN

INSERT INTO ProductArrival (

arrival\_prod\_id,

arrival\_stor\_id,

arrival\_prod\_quantity,

arrival\_sup\_id

)

SELECT

prod\_id,

stor\_id,

(-prod\_quantity),

arr\_sup\_id

FROM storage\_prod\_stats

WHERE prod\_quantity < 0;

END //

DELIMITER ;

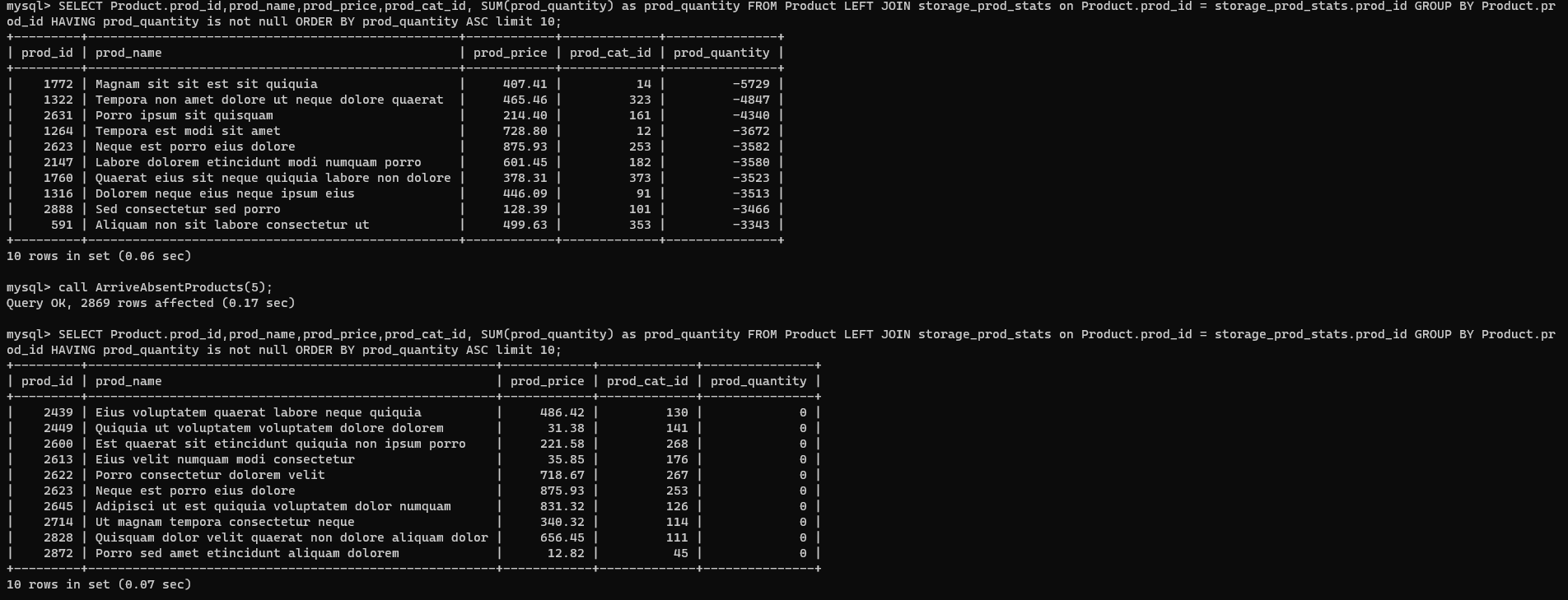


Рис. 7.3.8.1 – Закриття різниці за рахунок вказаного постачальника

7.3.9 Процедура MarkAsLostShips

Процедура MarkAsLostShips маркує всі затримані на вказану кількість днів доставки як втрачені та прив’язані до них замовлення. Приклад роботи показано на рис. 7.3.9.1.

DELIMITER //

CREATE PROCEDURE MarkAsLostShips(IN MAX\_DAYS\_DELAY INT)

BEGIN

UPDATE Shipments

INNER JOIN `Order` ON

ship\_order\_id = order\_id

AND

ship\_status = "inroute"

AND

DATEDIFF(CURRENT\_TIMESTAMP, order\_date\_awaited) > MAX\_DAYS\_DELAY

SET

ship\_status = "lost",

order\_status = "failed";

END //

DELIMITER ;

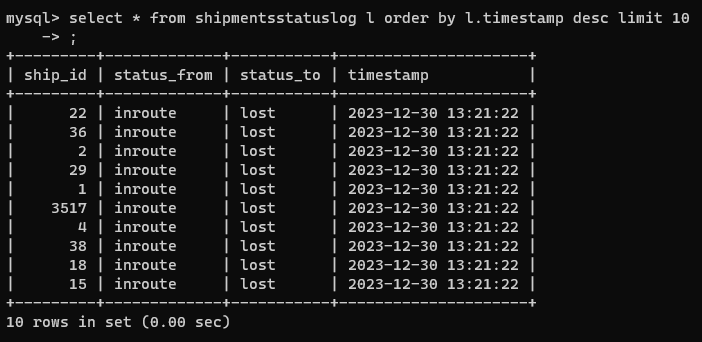


Рис. 7.3.9.1 – Втрата всіх затриманих доставок

7.3.10 Процедура DeleteOldFailedOrders

Процедура DeleteOldFailedOrders видаляє всі втрачені замовлення до вказаної дати. Приклад роботи показано на рис. 7.3.10.1.

DELIMITER //

CREATE PROCEDURE DeleteOldFailedOrders(IN date\_limit DATETIME)

BEGIN

DELETE FROM `Order`

WHERE order\_status = "failed" AND LastTimeOrderUpdated(order\_id) < date\_limit;

END //

DELIMITER ;

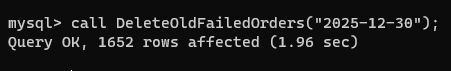


Рис. 7.3.10.1 – Результат запуску DeleteOldFailedOrders

7.3.11 Функція LastTimeOrderUpdated

Функція LastTimeOrderUpdated повертає дату коли це статус цього замовлення було оновлено. Приклад роботи показано на рис. 7.3.11.1.

DELIMITER //

CREATE FUNCTION LastTimeOrderUpdated(up\_order\_id int) RETURNS DATETIME

READS SQL DATA

NOT DETERMINISTIC

BEGIN

DECLARE result DATETIME;

SELECT MAX(`timestamp`) INTO result FROM OrderStatusLog WHERE order\_id = up\_order\_id;

RETURN result;

END //

DELIMITER ;

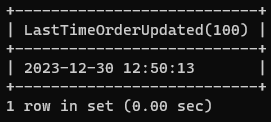


Рис. 7.3.11.1 – Результат запуску LastTimeOrderUpdated

7.4 SQL-запити

1) Запит на отримання історії сплат від кожного користувача. Результат показано на рис. 7.4.1.

SELECT cust\_id, cust\_name, paym\_id, paym\_amount, paym\_date FROM customer LEFT JOIN payments ON cust\_id = paym\_cust\_id ORDER BY cust\_id ASC, paym\_date DESC, paym\_id DESC;

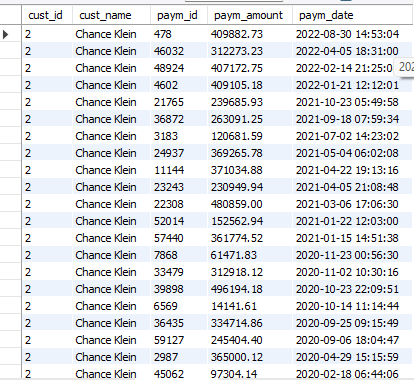


Рис. 7.4.1 – Результат запиту на історію сплат користувача

2) Запит для отримання історії надходжень від всіх постачальників. Результат можна побачити на рис. 7.4.2.

SELECT sup\_id, sup\_name, prod\_id, prod\_name, SUM(arrival\_prod\_quantity) AS SUPPLIED\_AMOUNT FROM supplier LEFT JOIN productarrival ON sup\_id = arrival\_sup\_id LEFT JOIN product ON prod\_id = arrival\_prod\_id group by sup\_id, prod\_id ORDER BY sup\_id ASC, prod\_cat\_id ASC;

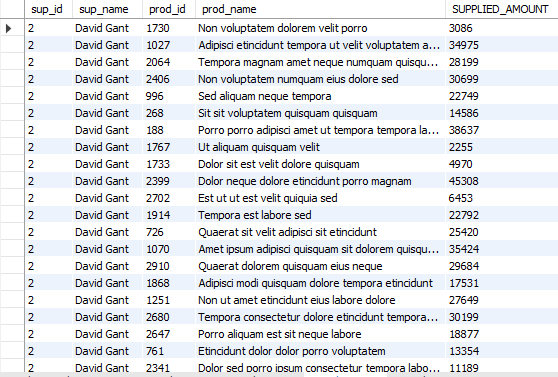


Рис. 7.4.2 – Результат запиту на об’єми доставки кожного постачальника

3) Запит на статистику всіх продуктів їх кількість, ціну та ціну з ПДВ. Результат можна побачити на рис. 7.4.3.

SELECT overal\_stor\_move.prod\_id, prod\_name, prod\_price, prod\_taxed\_price, SUM(prod\_move) as overal\_amount FROM overal\_stor\_move LEFT JOIN product\_with\_tax ON overal\_stor\_move.prod\_id = product\_with\_tax.prod\_id GROUP BY overal\_stor\_move.prod\_id ORDER BY prod\_id ASC;

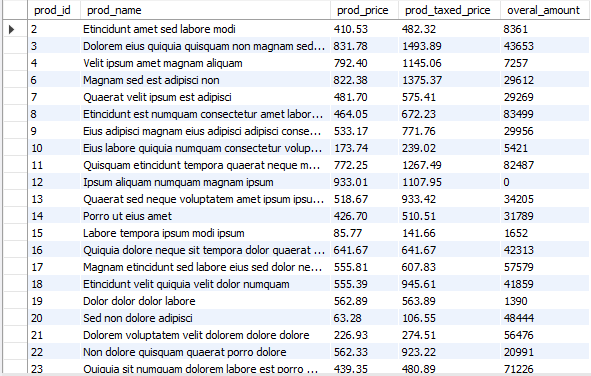


Рис. 7.4.3 – Результат запиту на статистику всіх продуктів

4) Запит на детальну інформацію про замовлення їх статус оплати та статус доставки. Результат можна побачити на рис. 7.4.4.

SELECT

order\_id,

order\_cust\_id,

order\_date\_created,

order\_date\_awaited,

order\_status,

(IF(order\_sum <= sum\_payed, "PAYED", "UNPAYED")) as PAYM\_STATUS,

ship\_dest,

ship\_date\_dep,

ship\_date\_arrival,

ship\_status

FROM order\_sums\_payments

LEFT JOIN shipments ON order\_id = ship\_order\_id

ORDER BY

order\_status DESC,

PAYM\_STATUS ASC,

ship\_status DESC,

order\_date\_created DESC,

ship\_date\_dep DESC,

ship\_date\_arrival DESC;

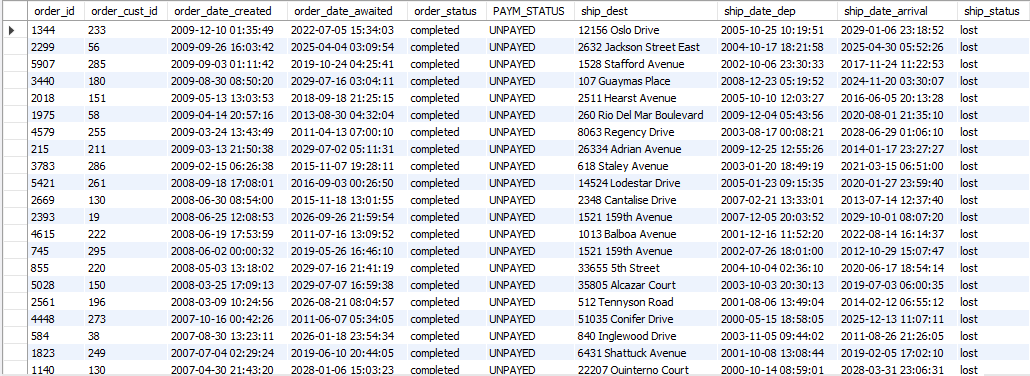


Рис. 7.4.4 – Результат запиту на детальну інформацію про замовлення

5) Запит на замовлення та їх час переходу на інші статуси. Результат можна побачити на рис. 7.4.5.

SELECT

o.order\_id,

o.order\_cust\_id,

o.order\_date\_created,

o.order\_date\_awaited,

o.order\_dest,

o.order\_status AS CURRENT\_STATUS,

MAX(p.timestamp) as PROCESS\_TIMESTAMP,

MAX(ap.timestamp) as AWAITING\_PAYM\_TIMESTAMP,

MAX(c.timestamp) as COLLECTION\_TIMESTAMP,

MAX(rts.timestamp) as READY\_TO\_SEND\_TIMESTAMP,

MAX(i.timestamp) as INROUNTE\_TIMESTAMP,

MAX(comp.timestamp) as COMPLETED\_TIMESTAMP

FROM

`Order` o

LEFT JOIN

orderstatuslog p ON o.order\_id = p.order\_id AND p.status\_to = "processing"

LEFT JOIN

orderstatuslog ap ON o.order\_id = ap.order\_id AND ap.status\_to = "awaiting payment"

LEFT JOIN

orderstatuslog c ON o.order\_id = c.order\_id AND c.status\_to = "collecting"

LEFT JOIN

orderstatuslog rts ON o.order\_id = rts.order\_id AND rts.status\_to = "ready to send"

LEFT JOIN

orderstatuslog i ON o.order\_id = i.order\_id AND i.status\_to = "inroute"

LEFT JOIN

orderstatuslog comp ON o.order\_id = comp.order\_id AND comp.status\_to = "completed"

GROUP BY

o.order\_id

ORDER BY

PROCESS\_TIMESTAMP DESC,

AWAITING\_PAYM\_TIMESTAMP DESC,

COLLECTION\_TIMESTAMP DESC,

READY\_TO\_SEND\_TIMESTAMP DESC,

INROUNTE\_TIMESTAMP DESC,

COMPLETED\_TIMESTAMP DESC;

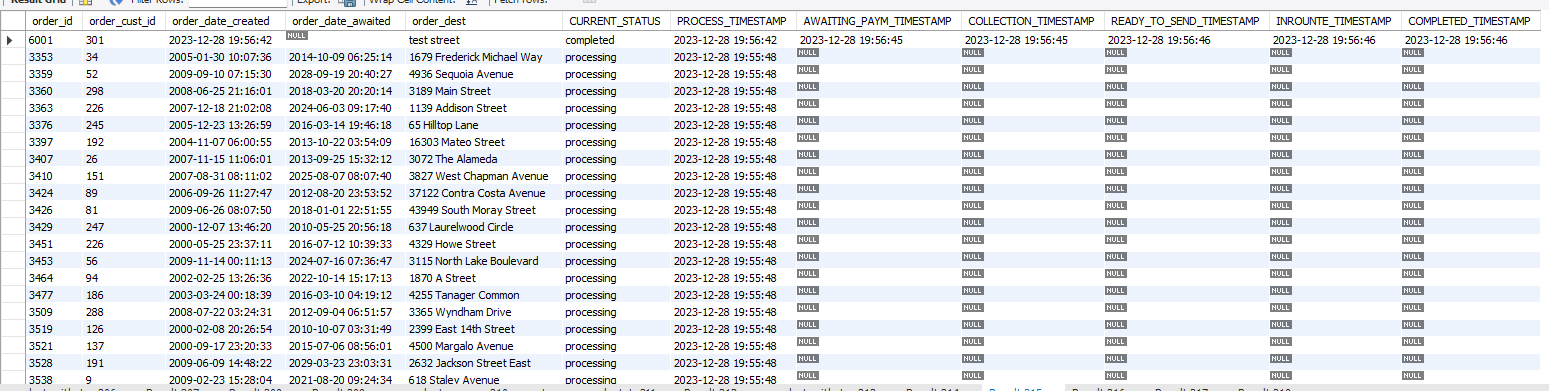


Рис. 7.4.5 – Результат запиту на замовлення та їх час переходу на інші статуси

6) Запит на отримання даних про час переходу на інший статус кожної доставки. Результат можна побачити на рис. 7.4.6.

SELECT

s.ship\_id,

s.ship\_order\_id,

s.ship\_date\_dep,

s.ship\_date\_arrival,

s.ship\_status AS CURRENT\_STATUS,

MAX(i.timestamp) as INROUTE\_TIMESTAMP,

MAX(a.timestamp) as ARRIVED\_TIMESTAMP,

MAX(l.timestamp) as LOST\_TIMESTAMP

FROM

`Shipments` s

LEFT JOIN

shipmentsstatuslog i ON s.ship\_id = i.ship\_id AND i.status\_to = "inroute"

LEFT JOIN

shipmentsstatuslog a ON s.ship\_id = a.ship\_id AND a.status\_to = "arrived"

LEFT JOIN

shipmentsstatuslog l ON s.ship\_id = l.ship\_id AND l.status\_to = "lost"

GROUP BY

s.ship\_id

ORDER BY

INROUTE\_TIMESTAMP DESC,

ARRIVED\_TIMESTAMP DESC,

LOST\_TIMESTAMP DESC;

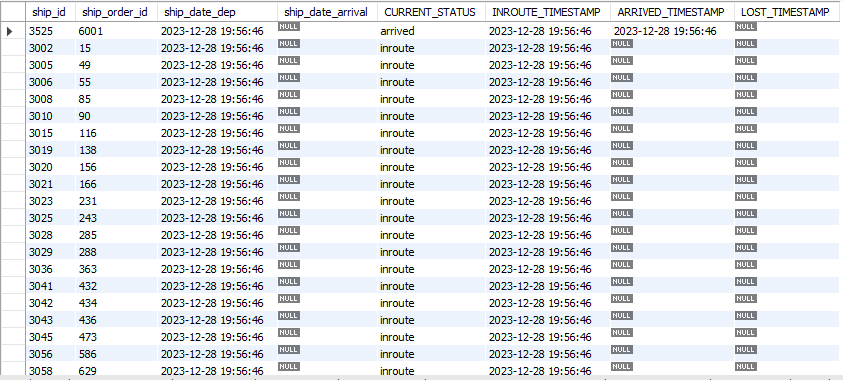


Рис. 7.4.6 – Результат запиту на замовлення та їх час переходу на інші статуси

7) Запит на отримання кількості замовлень кожного користувача кількість сплачених, кількість завершених замовлень. Результат можна побачити на рис. 7.4.7.

SELECT

c.cust\_id,

c.cust\_name,

COUNT(o.order\_id) AS CREATED,

COUNT(osp.order\_id) AS PAYED,

COUNT(CASE WHEN o.order\_status = 'completed' THEN o.order\_id END) AS COMPLETED

FROM

customer c

LEFT JOIN

`Order` o ON c.cust\_id = o.order\_cust\_id

LEFT JOIN

order\_sums\_payments osp ON o.order\_id = osp.order\_id AND osp.order\_sum <= osp.sum\_payed

GROUP BY

c.cust\_id, c.cust\_name;



Рис. 7.4.7 – Результат запиту на отримання кількості замовлень кожного користувача

8) Запит на статистику зміни цін на продукти протягом їх існування. Мінімальна, середня, максимальна ціни та ціна що стоїть зараз. Результат запуску можна побачити на рис. 7.4.8.

SELECT

Product.prod\_id,

prod\_name,

min(price\_to) AS MIN\_PRICE,

avg(price\_to) AS AVG\_PRICE,

max(price\_to) AS MAX\_PRICE,

prod\_price AS CUR\_PRICE

FROM Product

LEFT JOIN productpricechangeslog ON Product.prod\_id = productpricechangeslog.prod\_id group by Product.prod\_id;

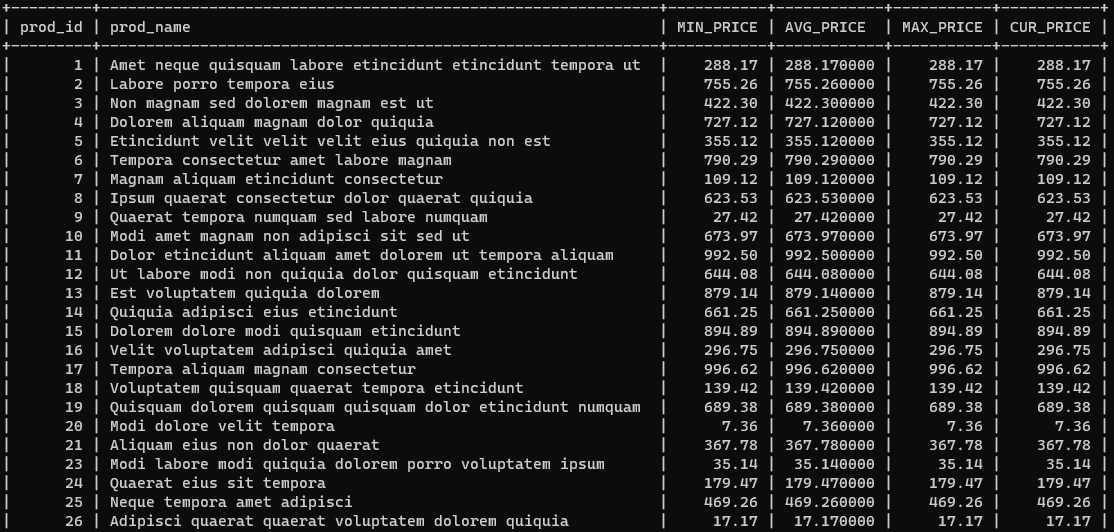


Рис. 7.4.8 – Результат запиту на статистику зміни цін на продукти протягом їх існування

9) Запит на обрахунки податків що потрібно сплатити за минулий місяць. Результат виконання запиту можна побачити на рис. 7.4.9.

SELECT

tax\_id,

tax\_tag,

ROUND(SUM(TAXED\_AMOUNT),2) as AMOUNT\_OWED

FROM (

SELECT

t.tax\_id,

t.tax\_tag,

pc.cat\_id,

p.prod\_id,

sum(oi.prod\_quantity \* (p.prod\_price \* t.tax\_amount\_perc + t.tax\_amount\_static)) AS TAXED\_AMOUNT

FROM Tax t

INNER JOIN prodcategory pc ON pc.cat\_tax\_id = t.tax\_id

INNER JOIN product\_with\_tax p on p.prod\_cat\_id = pc.cat\_id

INNER JOIN orderitems oi on oi.prod\_id = p.prod\_id

INNER JOIN `order` o on oi.order\_id = o.order\_id and YEAR(o.order\_date\_created) = YEAR(CURDATE() - INTERVAL 1 MONTH)

AND MONTH(o.order\_date\_created) = MONTH(CURDATE() - INTERVAL 1 MONTH)

GROUP BY p.prod\_id

) t

GROUP BY tax\_id

;

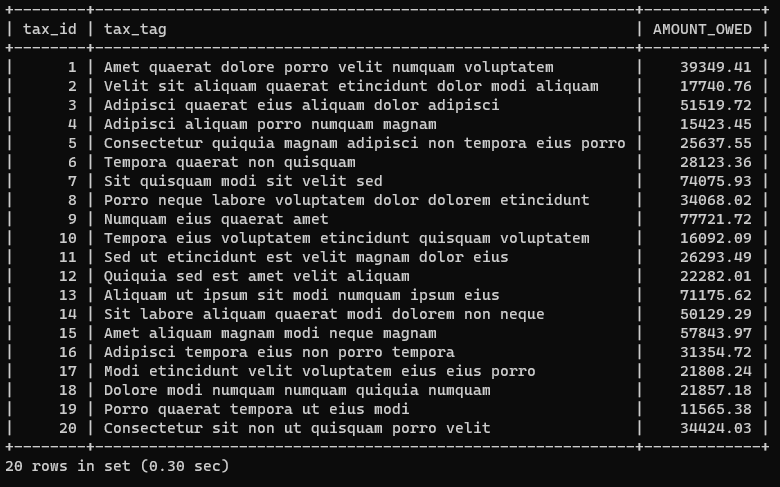


Рис. 7.4.9 – Результат запиту на обрахунки податків що потрібно сплатити за минулий місяць

10) Запит на об’єми продаж кожного продукта кожного місяцю за минулий рік, для прогнозування цьогорічних продаж. Результат можна побачити на рис. 7.4.10.

SELECT

p.prod\_id,

IFNULL(SUM(jan.prod\_quantity), 0) AS \_JAN\_,

IFNULL(SUM(feb.prod\_quantity), 0) AS \_FEB\_,

IFNULL(SUM(mar.prod\_quantity), 0) AS \_MAR\_,

IFNULL(SUM(apr.prod\_quantity), 0) AS \_APR\_,

IFNULL(SUM(may.prod\_quantity), 0) AS \_MAY\_,

IFNULL(SUM(jun.prod\_quantity), 0) AS \_JUN\_,

IFNULL(SUM(jul.prod\_quantity), 0) AS \_JUL\_,

IFNULL(SUM(aug.prod\_quantity), 0) AS \_AUG\_,

IFNULL(SUM(sep.prod\_quantity), 0) AS \_SEP\_,

IFNULL(SUM(oct.prod\_quantity), 0) AS \_OCT\_,

IFNULL(SUM(nov.prod\_quantity), 0) AS \_NOV\_,

IFNULL(SUM(`dec`.prod\_quantity), 0) AS \_DEC\_

FROM product p

INNER JOIN orderitems oi ON p.prod\_id = oi.prod\_id

INNER JOIN `order` o on oi.order\_id = o.order\_id and YEAR(o.order\_date\_created) = YEAR(CURDATE()) - 1

LEFT JOIN orderitems jan on jan.order\_id = o.order\_id and month(o.order\_date\_created) = 1

LEFT JOIN orderitems feb on feb.order\_id = o.order\_id and month(o.order\_date\_created) = 2

LEFT JOIN orderitems mar on mar.order\_id = o.order\_id and month(o.order\_date\_created) = 3

LEFT JOIN orderitems apr on apr.order\_id = o.order\_id and month(o.order\_date\_created) = 4

LEFT JOIN orderitems may on may.order\_id = o.order\_id and month(o.order\_date\_created) = 5

LEFT JOIN orderitems jun on jun.order\_id = o.order\_id and month(o.order\_date\_created) = 6

LEFT JOIN orderitems jul on jul.order\_id = o.order\_id and month(o.order\_date\_created) = 7

LEFT JOIN orderitems aug on aug.order\_id = o.order\_id and month(o.order\_date\_created) = 8

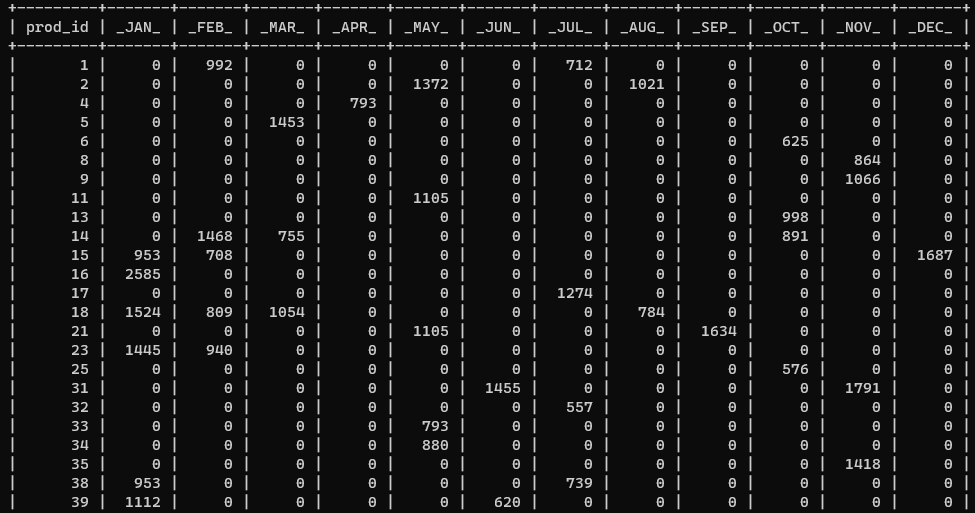
LEFT JOIN orderitems sep on sep.order\_id = o.order\_id and month(o.order\_date\_created) = 9

LEFT JOIN orderitems oct on oct.order\_id = o.order\_id and month(o.order\_date\_created) = 10

LEFT JOIN orderitems nov on nov.order\_id = o.order\_id and month(o.order\_date\_created) = 11

LEFT JOIN orderitems `dec` on `dec`.order\_id = o.order\_id and month(o.order\_date\_created) = 12

GROUP BY p.prod\_id;

Рис. 7.4.10 – Результат запиту на об’єми продаж кожного продукта кожного місяця

7.5 Індекси та результати оптимізації

В базі даних мало застосовуються запити які могли б бути покращені індексацію, основна задача цієї оптимізації полягає у правильному формулюванні запитів, уникненню зайвих формолювань. Але задля уникнення майбутніх затримок можна додати індексацію на всі текстові значення замовників, постачальників та замовлень для швидкого пошуку поміж цих значень. Запит для перевірки дії оптимізації буде робити з’єднання з таблиць замовників та постачальників та шукати які адреса співпадають з адресами в замовленнях. Результат запуску без оптимізації можна побачити на рис. 7.5.1.

SELECT 0 FROM

(

SELECT \* FROM customer

union

SELECT \* FROM supplier

) u

INNER JOIN `Order` o ON o.order\_dest = u.cust\_address ORDER BY cust\_address LIMIT 1;

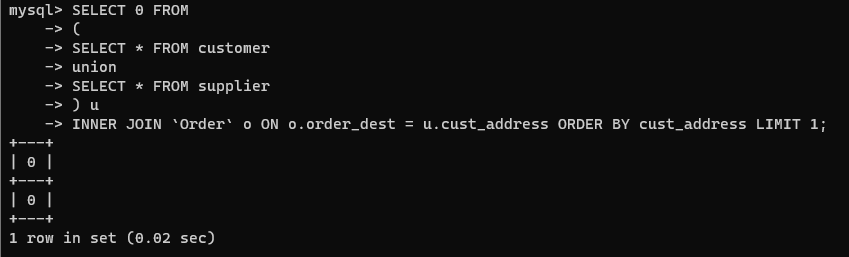


Рис. 7.5.1 – Результат запиту без оптимізації

Для оптимізації додамо індекси до імен, адрес, номерів телефонів та електронних адрес замовників та постачальників.

CREATE INDEX idxs\_customer ON Customer(cust\_name, cust\_address, cust\_phone, cust\_email);

CREATE INDEX idxs\_supplier ON supplier(sup\_name, sup\_address, sup\_phone, sup\_email);

CREATE INDEX idxs\_order ON `order`(order\_dest);

Виконаємо минулий запит по новій, результат можна побачити на рис. 7.5.2.

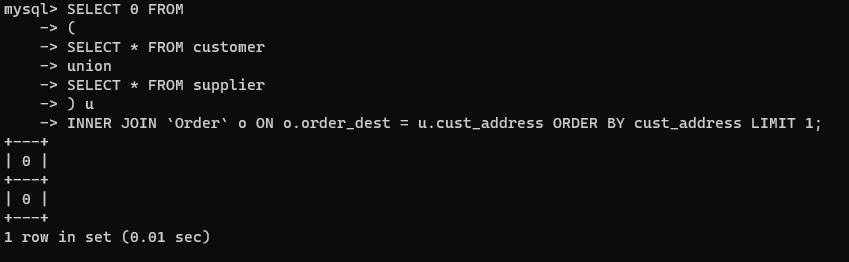


Рис. 7.5.2 – Результат запиту після оптимізації

Як можем побачити прискорення запиту становить 0.01 секунду, що відносно минулого результату є вдвічі краще, але не несе відчутної різниці на цьому розмірі бази даних.

**ВИСНОВКИ**

У ході виконання даної курсової роботи було проведено аналіз предметного середовища, визначено основні завдання та вимоги до розроблюваної бази даних для управління складським обліком на підприємстві. Орієнтуючись на виявлені потреби підприємства, робота націлено на розробку структури бази даних, спроможної ефективно вирішувати ключові завдання автоматизації та оптимізації складського обліку.

В результаті дослідження бізнес-правил та огляду існуючих програмних продуктів було з'ясовано, що створення бази даних є необхідним для об'єктивного складського обліку, що в свою чергу сприятиме підвищенню ефективності та прозорості управління даними.

ER-модель розробленої бази даних наочно відображає ключові сутності та їх взаємодії в контексті складського обліку. Реалізація бази даних включає створення таблиць, імпортування даних, створення користувачів та розробку функціоналу для ефективної взаємодії з нею. Забезпечено багатокористувацький доступ та враховано особливості складського обліку через впровадження різноманітних компонентів, таких як тексти генераторів, збережені процедури, тригери, представлення та SQL-запити.

Мета роботи – підвищення ефективності та прозорості складського обліку на підприємстві – була досягнута успішно. Розроблена база даних володіє потенціалом стати ефективним інструментом для управління складськими процесами, не лише автоматизуючи ключові операції, але й забезпечуючи необхідний моніторинг та аналіз даних для обґрунтованого прийняття рішень. Надзвичайно важливою є також підтримка безпеки даних та забезпечення конфіденційності.

Розроблена база даних відповідає вимогам ефективного управління складським обліком на підприємстві, проявляючи стійкість до навантажень, легку масштабованість та забезпечуючи швидкий доступ до інформації. Зокрема, вона дозволить підвищити ефективність та прозорість складського обліку, роблячи об'єктивними рішення на кожному етапі цього процесу.

Отже, впровадження даної бази даних стане значущим кроком у напрямку оптимізації та автоматизації управління складським обліком на підприємстві, сприяючи підвищенню ефективності обробки даних у підприємницькому середовищі.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. <https://e-driver.mvs.gov.ua/>
2. <https://bdr.mvs.gov.ua/>
3. <https://apps.apple.com/us/app/%D1%88%D1%82%D1%80%D0%B0%D1%84%D0%B8-%D0%BF%D0%B4%D1%80/id1541875502>
4. <https://splata.shtrafy.ua/>
5. <https://en.wikipedia.org/wiki/Relational_database>
6. <https://www.postgresql.org/>