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 діяльності сервісного центру МВС в частині прийому екзаменів, водійських посвідчень\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студента (ки) \_\_2\_\_ курсу \_ІП-31\_\_\_\_\_ групи

спеціальності 121 «Інженерія програмного забезпечення»

\_Віжуткіна Іллі Дмитровича\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник Марченко Олена Іванівна\_\_\_\_\_\_\_\_\_

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

(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-31\_\_ Семестр \_\_\_3\_\_

**З А В Д А Н Н Я**

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

\_\_\_\_\_\_\_Віжуткіну Іллі Дмитровичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи База даних для підтримки діяльності сервісного центру МВС в частині прийому екзаменів, водійських посвідчень

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

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

керівник роботи Марченко Олена Іванівна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

3. Вихідні дані до роботи створена база даних \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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\_\_\_\_06.11.2024**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсового проекту | Строк виконання етапів проекту | Примітка |
| 1 | Аналіз предметного середовища | 20.11.2024 |  |
| 2 | Побудова ER-моделі | 01.12.2024 |  |
| 3 | Побудова реляційної схеми з ER-моделі | 05.12.2024 |  |
| 4 | Створення бази даних, у форматі обраної системи управління базою даних | 08.12.2024 |  |
| 5 | Створення користувачів бази даних | 10.12.2024 |  |
| 6 | Імпорт даних з використанням засобів СУБД в створену базу даних | 12.12.2024 |  |
| 7 | Створення мовою SQL запитів | 14.12.2024 |  |
| 8 | Оптимізація роботи запитів | 16.12.2024 |  |
| 9 | Оформлення пояснювальної записки | 22.12.2024 |  |
| 10 | Захист курсової роботи | 25.12.2024 |  |

**Студент \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_** Віжуткін І.Д.**\_\_\_**

(підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Марченко О.І.**\_\_\_\_\_\_**

(підпис ) (прізвище та ініціали)

ЗМІСТ

[ВСТУП 3](#_Toc185793991)

[1 ОПИС ПРЕДМЕТНОГО СЕРЕДОВИЩА 6](#_Toc185793992)

[2 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ 8](#_Toc185793993)

[2.1 Система DMV (Department of Motor Vehicles) 8](#_Toc185793994)

[2.2 Система WORD (Wojewódzki Ośrodek Ruchu Drogowego) 9](#_Toc185793995)

[3 ПОСТАНОВКА ЗАДАЧІ 11](#_Toc185793996)

[4 ОПИС КОНЦЕПТУАЛЬНОЇ МОДЕЛІ 13](#_Toc185793997)

[4.1 Основні бізнес-правила 13](#_Toc185793998)

[4.2 Сутності 15](#_Toc185793999)

[4.3 Атрибути сутностей 17](#_Toc185794000)

[4.4 Структурні зв’язки 18](#_Toc185794001)

[4.5 ER-діаграма 20](#_Toc185794002)

[5 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ 22](#_Toc185794003)

[5.1 Вибір СУБД 22](#_Toc185794004)

[5.2 Характеристика таблиць 23](#_Toc185794005)

[5.3 Методи забезпечення цілісності даних 35](#_Toc185794006)

[5.4 Тексти таблиць 36](#_Toc185794007)

[5.5 Схема бази даних, реалізована засобами СУБД 43](#_Toc185794008)

[5.6 Користувачі та тексти їх створення 44](#_Toc185794009)

[6 РОБОТА З БАЗОЮ ДАНИХ 48](#_Toc185794010)

[6.1 Генератори 48](#_Toc185794011)

[6.2 Тексти представлень 48](#_Toc185794012)

[6.3 Тексти процедур і функцій 51](#_Toc185794013)

[6.4 Тексти тригерів 66](#_Toc185794014)

[6.5 SQL-запити 88](#_Toc185794015)

[6.6 Оптимізація 103](#_Toc185794016)

[ВИСНОВКИ 106](#_Toc185794017)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 108](#_Toc185794018)

[ДОДАТОК А ТЕКСТИ SQL-СКРИПТІВ 109](#_Toc185794019)

ВСТУП

У сучасному світі автоматизація процесів є ключовим фактором для забезпечення ефективної роботи всього навколо, включаючи і державні установи. Оскільки, одним із важливих напрямів діяльності сервісних центрів Міністерства внутрішніх справ є прийом екзаменів для отримання водійських посвідчень та подальша видача цих документів, то для виконання цих завдань необхідна надійна та ефективна система управління даними, яка забезпечить швидкість, зручність і безпеку обробки інформації.

Актуальність теми зумовлена зростанням кількості громадян, які звертаються до сервісних центрів, та необхідністю вдосконалення інформаційних систем для обслуговування цих звернень. Використання сучасних баз даних дозволяє оптимізувати роботу з великими обсягами інформації, забезпечити автоматизацію рутинних процесів, а також зменшити ризик помилок, пов’язаних із людським фактором.

Метою даної курсової роботи є створення бази даних, яка підтримуватиме діяльність сервісного центру МВС у частині прийому екзаменів та видачі водійських посвідчень. Основними завданнями є аналіз потреб такого сервісного центру, розробка структури бази даних та всіх необхідних компонентів, реалізація за допомогою системи управління базами даних PostgreSQL, а також перевірка працездатності створеної системи.

Призначення розроблюваної системи полягає в автоматизації ключових операцій, таких як реєстрація кандидатів, запис на іспити, проведення іспитів, видача водійських посвідчень та інших функцій, які здійснюються сервісними центрами МВС у межах обліку водіїв.

Вибір PostgreSQL як системи управління базами даних зумовлений її високою продуктивністю, зручністю використання, відкритим кодом та підтримкою активної спільноти розробників. Завдяки широкій доступності документації, навчальних матеріалів і практичних прикладів, PostgreSQL значно спрощує та прискорює процес розробки, при цьому надаючи весь необхідний функціонал для реалізації складних проєктів.

Таким чином, розробка бази даних для підтримки діяльності сервісного центру МВС сприятиме підвищенню якості та швидкості обслуговування громадян, а також забезпечить надійне зберігання даних та автоматизовану роботу з ними.

1 ОПИС ПРЕДМЕТНОГО СЕРЕДОВИЩА

Автомобілі є важливим елементом сучасного життя, і з кожним роком кількість нових водіїв зростає. Це створює високий попит на системи складання іспитів та отримання водійських посвідчень. Сервісні центри Міністерства внутрішніх справ (МВС) [1] виконують ключові функції, пов’язані з наданням цих послуг, забезпечуючи громадян зручними та прозорими процедурами.

У сервісних центрах громадяни проходять декілька важливих етапів. Спершу клієнти записуються на іспити, отримуючи відповідні талони. При зверненні до центру вони надають обов’язкові документи, перелік яких залежить від типу іспиту. Наприклад, для теоретичного іспиту необхідно подати паспорт та реєстрацію місця проживання. Для практичного іспиту додатково надаються сертифікати автошколи (або ця інформація стягується з інших ресурсів) та медична картка, а також при реєстрації талону вказується категорія транспортного засобу, тип коробки передач та власник автомобіля практичного іспиту (сервісний центр або автошкола).

Після реєстрації клієнти переходять до складання іспитів. Теоретичний іспит проводиться у спеціально обладнаних комп’ютерних залах. Клієнт проходить тестування разом із іншими учасниками. У разі успішного складання теоретичного іспиту клієнт отримує право на реєстрацію практичного іспиту. Практичний іспит починається на спеціально обладнаних майданчиках, де клієнт очікує на звільнення інспектора та транспортного засобу. Інспектор супроводжує процес іспиту, оцінює підготовленість учня та підстраховує для уникнення ДТП. Для уникнення неправомірних дій із боку інспектора чи клієнта обов’язково ведеться відеозапис. Всі дані про складені іспити зберігаються для подальшої обробки та видачі водійських посвідчень.

Реєстрація клієнтів є важливою складовою роботи сервісних центрів. На цьому етапі перевіряються подані документи, їхня чинність та повнота. Після перевірки клієнт отримує доступ до наступного етапу – складання іспиту. Організація теоретичних іспитів передбачає перевірку знань правил дорожнього руху у форматі тестування. Практичні іспити зосереджені на оцінці практичних навичок керування автомобілем. Для цього використовуються транспортні засоби, які закріплені за сервісним центром або надані автошколою клієнта.

Важливим аспектом роботи є управління транспортними засобами. Усі автомобілі, що використовуються під час іспитів, мають відповідати певним вимогам: бути справними, мати відповідну категорію та тип коробки передач. Інспектори, які проводять практичні іспити, координуються через сервісний центр, що дозволяє ефективно розподіляти навантаження та забезпечувати безперебійний процес.

Штрафи та контроль водійських посвідчень також є важливими складовими. У разі порушення правил дорожнього руху накладається штраф, інформація про який фіксується для подальшого контролю. При серйозних порушеннях посвідчення може бути тимчасово призупинене або вилучене, що дозволяє уникнути зловживань та небезпеки на дорозі.

Таким чином, сервісні центри МВС виконують важливу роль у забезпеченні прав громадян на водіння транспортних засобів. Вони підтримують комплексний підхід до обслуговування, який включає реєстрацію, іспити, видачу посвідчень, їх контроль та управління ресурсами. Такий підхід забезпечує високу ефективність роботи та зручність як для клієнтів, так і працівників.

2 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ

Для автоматизації процесів, пов’язаних із прийомом іспитів та видачою водійських посвідчень, державні установи в різних країнах використовують свої програмні рішення. У цьому розділі буде проаналізовано програмне забезпечення, що застосовується в США та Польщі.

2.1 Система DMV (Department of Motor Vehicles)

DMV [2] – це державна установа в США, яка займається проведенням іспитів для водіїв, видачею водійських посвідчень і реєстрацією транспортних засобів. Функціонал відрізняється залежно від штату, але основні принципи залишаються однаковими.

Переваги:

* Онлайн реєстрація на екзамени.
* Електронні теоретичні іспити. Як правило, існує можливість проходження тесту із дому.
* Можливість здавати практичні іспити на власному або спеціально обладненому арендованому автомобілі.
* Як правило, відсутність очікування перед наступною спробою при нескладанні теоретичного іспиту.
* Необмежена кількість спроб теоретичного і практичного іспитів.
* Можливість вивчення та здачі теорії окремо від практики.

Недоліки:

* Відсутність власних автомобілей сервісу DMV.
* Відсутність єдиної системи на національному рівні, що може ускладнювати процес обміну даними.
* Відсутність відеозапису практичного іспиту.

2.2 Система WORD (Wojewódzki Ośrodek Ruchu Drogowego)

WORD [3] – це воєводські центри в Польщі, які організовують іспити на водійські права. Дані про успішно складені іспити передаються до центральної бази CEPiK (Centralna Ewidencja Pojazdów i Kierowców), яка відповідає за видачу посвідчень.

Переваги:

* Онлайн реєстрація на екзамени.
* Електронні теоретичні іспити.
* Наявність власних автомобілей сервісу WORD.
* Можливість здавати практичні іспити на власному або спеціально обладненому арендованому автомобілі.
* Єдина національна система зберігання даних про водіїв CEPiK.
* Відсутність очікування між теоретичним і практичним іспитом.

Недоліки:

* Необхідність вивчення теорії та практики одночасно.
* Обмежена кількість спроб теоретичного та практичного іспитів.
* Відсутність відеозапису практичного іспиту.

Отже, системи DMV та WORD мають свої переваги та недоліки у процесі складання іспитів та видачі водійських посвідчень.

DMV пропонує зручні онлайн-опції для реєстрації та здачі теоретичних іспитів, а також дозволяє використовувати як власне, так і орендоване авто для практики. Відсутність обмежень на кількість спроб і можливість тестування вдома роблять систему дуже зручною. Однак, відсутність єдиної національної системи обміну даними між штатами та відсутність відеозапису практичних іспитів можуть ускладнювати деякі процеси.

WORD, у свою чергу, пропонує централізовану систему зберігання даних CEPiK, що забезпечує ефективний обмін інформацією між центрами. Також досить зручним є наявність власних автопарків для практичних іспитів. Проте з іншої строни, необхідність одночасного вивчення теорії та практики та обмежена кількість спроб для іспитів можуть бути досить незручними.

Загалом, обидва сервіси та їх системи мають основний необхідний функціонал та є досить ефективними, проте все одно мають моменти, які потребують покращення. Ідеальне рішення має поєднувати зручність, зрозумілість та прозорість реєстрації, складання іспитів, отримання посвідчень та інших процесів як для клієнтів, так і для працівників, враховуючи потреби обох сторін для забезпечення найкращого досвіду та ефективності.

3 ПОСТАНОВКА ЗАДАЧІ

Розробка бази даних для підтримки діяльності сервісного центру Міністерства внутрішніх справ (МВС) спрямована на автоматизацію основних процесів, таких як реєстрація клієнтів і їх документів, запис на теоретичні та практичні іспити, обробка результатів іспитів, включаючи теоретичні відповіді та відеозаписи практичних випробувань, видача водійських посвідчень, оновлення їх статусів та фіксація штрафів. База даних повинна також містити інформацію про інспекторів, сервісні центри, автомобілі та теоретичні матеріали. Метою є створення інтегрованої системи, що дозволить ефективно управляти процесами прийому іспитів, видачі посвідчень та систематизації облікової документації в сервісних центрах МВС. Така система повинна сприяти централізованому зберіганню даних, ефективній обробці інформації та зручному доступу для всіх категорій користувачів.

Для досягнення цієї мети необхідно вирішити кілька основних завдань. Першим етапом є формулювання вимог до структури бази даних, з урахуванням детального аналізу предметного середовища і існуючих програмних рішень. Далі слід розробити концептуальну ER-модель, яка відобразить основні сутності та їх взаємозв'язки. Наступний крок – побудова реляційної моделі бази даних, що відповідатиме вимогам до цілісності та нормалізації даних. Реалізація бази даних буде здійснюватися в СУБД PostgreSQL, із використанням SQL-скриптів для створення таблиць, тригерів, процедур, функцій та інших необхідних об’єктів.

Розроблювана база даних повинна відповідати кільком важливим вимогам. До функціональних вимог належать зберігання і обробка даних про клієнтів, інспекторів, автомобілі, сервісні центри та іспити. Система повинна забезпечувати реалізацію функцій запису на іспити, обробки результатів, видачі посвідчень, їх підтримки, генерації звітів та статистики. Важливими є також вимоги до цілісності даних, включаючи реалізацію бізнес-правил, таких як, наприклад, перевірка документів перед записом на іспити, уникнення дублювання записів або посвідчень, обмеження на атрибути та інші правила, які забезпечують коректну роботу системи.

Крім того, база даних повинна гарантувати надійність і доступність у багатокористувацькому режимі, а також забезпечити високу швидкість обробки запитів і надійний захист даних. Для цього передбачається використання СУБД PostgreSQL, що забезпечує високу ефективність та швидкість розробки. Проєктування бази даних передбачає також нормалізацію до третьої нормальної форми (3НФ), що дозволить оптимізувати структуру даних, забезпечити їх цілісність і ефективне управління.

Таким чином, створення бази даних для сервісного центру МВС у частині прийому іспитів та видачі посвідчень має на меті автоматизацію важливих процесів, підвищення ефективності управління даними та забезпечення зручності для всіх користувачів системи. Врахування всіх зазначених вимог дозволить створити надійний інструмент для оптимізації діяльності та обслуговування громадян.

1. ОПИС КОНЦЕПТУАЛЬНОЇ МОДЕЛІ
   1. Основні бізнес-правила

Для забезпечення коректної роботи інформаційної системи сервісного центру МВС в частині прийомі іспитів та видачі посвідчень розроблено набір бізнес-правил, які випливають з аналізу предметного середовища та існуючих програмних продуктів. Вони визначають ключові обмеження та правила, які впливають на функціонування бази даних.

* Обмеження віку клієнта

Перед записом на практичний іспит необхідно перевірити вік клієнта. В залежності від цього та обраної категорії, робимо рішення, щодо запису.

* Необхідність документів клієнта

Перед записом на теоретичний іспит необхідно перевірити, що клієнт має активний паспорт та реєстрацію місця проживання. Для практичного іспиту необхідно також перевірити наявність медичної довідки та сертифікату автошколи.

* Обмеження здачі однакових категорій

Перед записом на практичний іспит необхідно перевірити, що клієнт немає вже відкритої категорії із заданої коробкою передач.

* Обмеження одного активного квитка

Перед записом на теоретичний або практичний іспит необхідно перевірити, що клієнт немає вже активного квитка.

* Проміжок між квитками

Щоб клієнт спочатку довчився після невдалої спроби, необхідно зробити паузу, тому вирішено зробити перевірку перед записом на іспит, що клієнт не мав спроб за минулі 10 днів.

* Перевірка одного іспиту

Необхідно перевіряти, що за кожним теоретичним або практичним талоном закріплено тільки один відповідний іспит.

* Перевірка теоретичного іспиту

Перед записом на практичний іспит необхідно перевірити, що клієнт має активний теоретичний іспит.

* Перевірка наявності необхідного сервісного автомобіля

Оскільки практичний іспит вимагає вибору автомобіля, перед записом на практичний іспит необхідно впевнитися, що автомобіль для обраної категорії і коробки передач існує на локації сервісного центру.

* Перевірка обраного автомобіля

Оскільки обрана категорія і коробка передач на практичний іспит має відповідати автомобілю, на якому проходить іспит, то маємо впевнитися, що обраний автомобіль відповідає зазначеній інформації в талоні.

* Перевірка посвідчення інспектора

Оскільки практичний іспит вимагає інспектора із відповідними категоріями, враховуючи коробку передач, то маємо перевірити, що обраний інспектор має необхідне посвідчення.

* Перевірка документів інспектора

Оскільки практичний іспит має проводити інспектор із активним сертифікатом, то необхідно перевірити його наявність.

* Автоматична зміна статусу талону

Після проходження реєстрації по талону необхідно автоматично змінити його статус на «використаний».

* Видача посвідчення після іспиту

Після успішного складеного практичного іспиту необхідно автоматично видавати посвідчення водія або додавати нову категорію до вже існуючого.

* Обмеження дати та часу

Перед оформленням будь-яких записів в системі, що містять час, необхідно впевнитися, що дата початку відповідає поточному дню, а дата кінця більша за дату оформлення.

* 1. Сутності

Сутності є ключовими елементами бази даних, що відображають об`єкти реального світу в системі. Для побудови бази даних в рамках цієї курсової роботи було виділено наступні сутності:

* Клієнти (client)

Представляє клієнтів.

* Інспектори (inspector)

Представляє інспекторів.

* Документи (document)

Представляє документи клієнтів та інспекторів.

* Теоретичні талони (theory\_ticket)

Представляє записи на теоретичний іспит.

* Практичні талони (driving\_ticket)

Представляє записи на практичний іспит.

* Теоретичні іспити (theory\_exam)

Представляє теоретичні іспити.

* Практичні іспити (driving\_exam)

Представляє практичні іспити.

* Питання (question)

Представляє питання на теоретичний іспит.

* Відповіді (answer)

Представляє відповіді на теоретичні питання.

* Результат теоретичного іспиту (theory\_exam\_result)

Представляє питання і відповіді конкретного іспиту конкретного клієнта.

* Сервісний центр (service\_center)

Представляє конкретний фізичний сервісний центр.

* Автомобіль (car)

Представляє автомобілі сервісу або автошкіл.

* Посвідчення водія (driving\_license)

Представляє посвідчення клієнтів та інспекторів.

* Категорії посвідчення водія (driving\_license\_category)

Представляє відкриті категорії для конкретного посвідчення.

* Статуси посвідчення (driving\_license\_status)

Представляє додаткові статуси посвідчення.

* Штрафи (fine)

Представляє штрафи для конкретного посвідчення.

* 1. Атрибути сутностей

Кожна сутність у базі даних має чітко визначені атрибути, що відображають характеристики. Детальний перелік наведено в таблиці 4.3.1.

Таблиця 4.3.1 – Сутності та їх атрибути.

|  |  |
| --- | --- |
| **Сутність** | **Атрибути** |
| client | first\_name, last\_name, middle\_name, date\_of\_birth, address, phone\_number, email |
| inspector | first\_name, last\_name, middle\_name |
| document | type, end\_date, info\_file\_path |
| theory\_ticket | datetime, status |
| driving\_ticket | datetime, status, gearbox, car\_owner, category |
| theory\_exam | - |
| driving\_exam | exam\_result, video\_file\_path |
| question | text, photo\_file\_path |
| answer | text, photo\_file\_path, is\_correct |
| theory\_exam\_result | - |
| service\_center | address, number |
| car | car\_owner, gearbox, license\_plate, category |
| driving\_license | start\_date, end\_date |
| driving\_license\_category | category, gearbox |
| driving\_license\_status | status, start\_date, end\_date, reason |
| fine | status, issued\_date, reason, sum |

* 1. Структурні зв’язки

Для забезпечення повноти бази даних було визначено зв’язки між

сутностями.

* client – document

Сутність client пов’язана із document зв’язком "один до багатьох", бо клієнт може мати декілька документів.

* client – theory\_ticket

Сутність client пов’язана із theory\_ticket зв’язком "один до багатьох", бо клієнт може зробити декілька спроб, тобто мати декілька теоретичних талонів.

* сlient – driving\_ticket

Сутність client пов’язана із driving\_ticket зв’язком "один до багатьох", бо клієнт може зробити декілька спроб, тобто мати декілька практичних талонів.

* client – driving\_license

Сутність client пов’язана із driving\_license зв’язком "один до багатьох", бо минулі посвідчення можуть закінчитися або бути анульовані.

* driving\_license – driving\_license\_status

Сутність driving\_license пов’язана із driving\_license\_status зв’язком "один до багатьох", бо посвідчення може вилучатися багато разів.

* driving\_license – fine

Сутність driving\_license пов’язана із fine зв’язком "один до багатьох", бо на одне посвідчення може бути виписано багато штрафів.

* driving\_license – driving\_license\_category

Сутність driving\_license пов’язана із driving\_license\_category зв’язком "один до багатьох", бо одне посвідчення може мати багато категорій.

* service\_center – theory\_ticket

Сутність service\_center пов’язана із theory\_ticket зв’язком "один до багатьох", бо на одному сервісному центрі можуть бути проведено багато теоретичних іспитів.

* service\_center – driving\_ticket

Сутність service\_center пов’язана із theory\_ticket зв’язком "один до багатьох", бо на одному сервісному центрі можуть бути проведено багато практичних іспитів.

* service\_center – car

Сутність service\_center пов’язана із car зв’язком "один до багатьох", бо до одного сервісного центру можуть бути прикріплені багато автомобілів.

* service\_center – inspector

Сутність service\_center пов’язана із inspector зв’язком "один до багатьох", бо в одному сервісному центрі можуть працювати багато інспекторів.

* car – driving\_exam

Сутність car пов’язана із driving\_exam зв’язком "один до багатьох", бо одна машина може бути використана на різних практичних іспитах.

* inspector – document

Сутність inspector пов’язана із document зв’язком "один до багатьох", бо інспектор може мати декілька документів.

* inspector – driving\_license

Сутність inspector пов’язана із driving\_license зв’язком "один до багатьох", бо минулі посвідчення можуть закінчитися або бути анульовані.

* inspector – driving\_exam

Сутність inspector пов’язана із driving\_exam зв’язком "один до багатьох", бо інспектор може провести багато іспитів.

* theory\_ticket – theory\_exam

Сутність theory\_ticket пов’язана із theory\_exam зв’язком "один до одного", бо для кожного теоретичного талону можливе тільки одне використання для створення теоретичного іспиту.

* driving\_ticket – driving\_exam

Сутність driving\_ticket пов’язана із driving\_exam зв’язком "один до одного", бо для кожного практичного талону можливе тільки одне використання для створення практичного іспиту.

* theory\_exam – theory\_exam\_result

Сутність theory\_exam пов’язана із theory\_exam\_result зв’язком "один до багатьох", бо в теоретичному екзамені багато тестових питань із зазначеними відповідями.

* question – theory\_exam\_result

Сутність question пов’язана із theory\_exam\_result зв’язком "один до багатьох", бо в теоретичному екзамені багато тестових питань.

* answer – theory\_exam\_result

Сутність answer пов’язана із theory\_exam\_result зв’язком "один до багатьох", бо в теоретичному екзамені багато тестових питань, а значить і багато зазначених відповідей.

* 1. ER-діаграма

На рисунку 4.5.1 зображено ER-діаграму, яка ілюструє структуру майбутньої бази даних із зображеними сутностями та зв’язками між ними.

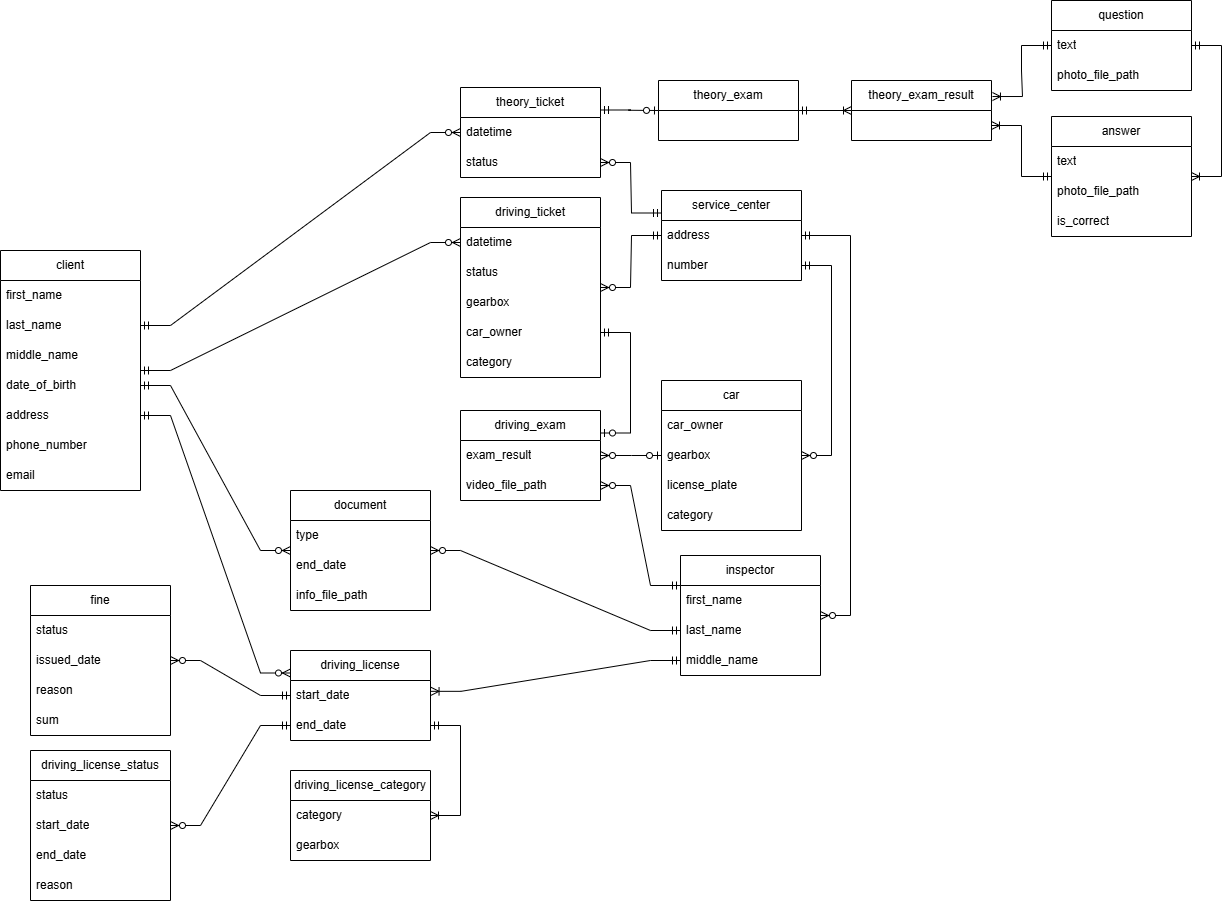


Рисунок 4.5.1 – ER-діаграма

1. РЕАЛІЗАЦІЯ БАЗИ ДАНИХ
   1. Вибір СУБД

Для реалізації бази даних у цій курсовій роботі було обрано PostgreSQL, оскільки ця система управління базами даних надає всі необхідні функціональні можливості для ефективної роботи з великими обсягами даних [5], що є критичним для завдань сервісного центру МВС.

Однією з основних причин вибору PostgreSQL є її стабільність та масштабованість. Система добре справляється з обробкою складних запитів, що є важливим для організації ефективної роботи сервісного центру. Крім того, PostgreSQL має механізми для підтримки цілісності даних та забезпечення високої продуктивності навіть при високому навантаженні.

Також PostgreSQL підтримує широкий набір можливостей для роботи з різними типами даних, що забезпечує високу гнучкість при зберіганні та обробці інформації. Вибір цієї СУБД дозволяє ефективно реалізувати як прості, так і складні операції з даними, що дає змогу без додаткових налаштувань або обмежень працювати з різноманітними типами та запитами.

Ще одним важливим фактором є відкрите програмне забезпечення. PostgreSQL є безкоштовною СУБД, що дозволяє використовувати її з обмеженими коштами. Крім того, наявність великої спільноти користувачів та розробників значно полегшує процес вирішення технічних питань, завдяки широкій базі документації, навчальних матеріалів та ресурсів.

Нарешті, PostgreSQL забезпечує високий рівень безпеки завдяки вбудованим механізмам шифрування даних та контролю доступу, що критично важливо для обробки персональних даних, особливо у випадку державних організацій. Це дозволяє гарантувати конфіденційність і захист інформації про клієнтів та інші чутливі дані.

Загалом, завдяки своїй надійності, масштабованості, гнучкості та безпеці, PostgreSQL є оптимальним вибором для створення бази даних для сервісного центру МВС.

* 1. Характеристика таблиць

У даній базі даних було створено наступні основні таблиці, які зберігають інформацію, необхідну для автоматизації роботи сервісного центру МВС. Кожна таблиця має чітко визначене призначення та відповідні атрибути для зберігання даних.

* client

Таблиця 5.2.1 «client» призначена для зберігання особистих даних клієнтів, які здають іспити на водійські посвідчення.

Таблиця 5.2.1 – Структура таблиці «client»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| first\_name | VARCHAR | 50 |  | Ім’я |
| last\_name | VARCHAR | 50 |  | Прізвище |
| middle\_name | VARCHAR | 50 |  | По батькові |
| date\_of\_birth | DATE |  |  | Дата народження |
| address | VARCHAR | 100 |  | Адреса проживання |
| phone\_number | VARCHAR | 30 |  | Номер телефона |
| email | VARCHAR | 50 |  | Пошта |

* inspector

Таблиця 5.2.2 «inspector» призначена для зберігання особистих даних інспекторів, які приймають практичні іспити.

Таблиця 5.2.2 – Структура таблиці «inspector»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| first\_name | VARCHAR | 50 |  | Ім’я |
| last\_name | VARCHAR | 50 |  | Прізвище |
| middle\_name | VARCHAR | 50 |  | По батькові |
| service\_center\_id | INT |  | FK | Унікальний ідентифікатор сервісного центру, де працює інспктор |

* document

Таблиця 5.2.3 «document» призначена для зберігання документів клієнтів та інспекторів.

Таблиця 5.2.3 – Структура таблиці «document»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| type | document\_type ENUM |  |  | Тип документу |

Продовження таблиці 5.2.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| end\_date | DATE |  |  | Дата закінчення дії документу |
| info\_file\_path | VARCHAR | 255 |  | Шлях до файлу документа (сканкопія, витяг тощо) |
| client\_id | INT |  | FK | Унікальний ідентифікатор клієнта (якщо документ клієнта) |
| inspector\_id | INT |  | FK | Унікальний ідентифікатор інспектора (якщо документ інспектора) |

* theory\_ticket

Таблиця 5.2.4 «theory\_ticket» призначена для зберігання талонів (записів) клієнтів на теоретичні іспити.

Таблиця 5.2.4 – Структура таблиці «theory\_ticket»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| datetime | TIMESTAMP |  |  | Час початку реєстрації |
| status | ticket\_status\_type |  |  | Статус квитку |
| service\_center\_id | INT |  | FK | Унікальний ідентифікатор сервісного центру |
| client\_id | INT |  | FK | Унікальний ідентифікатор клієнта |

* driving\_ticket

Таблиця 5.2.5 «driving\_ticket» призначена для зберігання талонів (записів) клієнтів на практичні іспити.

Таблиця 5.2.5 – Структура таблиці «driving\_ticket»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| datetime | TIMESTAMP |  |  | Час початку реєстрації |
| status | ticket\_status\_type |  |  | Статус квитку |

Продовження таблиці 5.2.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| gearbox | gearbox\_type |  |  | Тип коробки передач |
| car\_owner | car\_owner\_type |  |  | Власник автомобіля |
| category | category\_type |  |  | Категорія автомобіля |
| service\_center\_id | INT |  | FK | Унікальний ідентифікатор сервісного центру |
| client\_id | INT |  | FK | Унікальний ідентифікатор клієнта |

* theory\_exam

Таблиця 5.2.6 «theory\_exam» призначена для зберігання інформації про проведені теоретичні іспити для кожного клієнта.

Таблиця 5.2.6 – Структура таблиці «theory\_exam»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| theory\_ticket\_id | INT |  | FK | Унікальний ідентифікатор квитка на теоретичний іспит |

* driving\_exam

Таблиця 5.2.7 «driving \_exam» призначена для зберігання інформації про проведені практичні іспити для кожного клієнта.

Таблиця 5.2.7 – Структура таблиці «driving\_exam»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| exam\_result | exam\_result\_type |  |  | Результат іспиту |
| video\_file\_path | VARCHAR | 255 |  | Шлях до відеофайлу з іспитом |
| driving\_ticket\_id | INT |  | FK | Унікальний ідентифікатор квитка на практичний іспит |
| inspector\_id | INT |  | FK | Унікальний ідентифікатор інспектора |
| car\_id | INT |  | FK | Унікальний ідентифікатор автомобіля |

* question

Таблиця 5.2.8 «question» призначена для зберігання теоретичних питань для відповідних іспитів.

Таблиця 5.2.8 – Структура таблиці «question\_exam»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| text | VARCHAR | 500 |  | Текст запитання |
| photo\_file\_path | VARCHAR | 255 |  | Шлях до файлу додаткового зображення |

* answer

Таблиця 5.2.9 «answer» призначена для зберігання можливих відповідей на теоретичні питання.

Таблиця 5.2.9 – Структура таблиці «answer»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| text | VARCHAR | 500 |  | Текст відповіді |
| is\_correct | BOOLEAN |  |  | Позначення правильної відповіді |
| photo\_file\_path | VARCHAR | 255 |  | Шлях до файлу додаткового зображення |

Продовження таблиці 5.2.9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| question\_id | INT |  | FK | Унікальний ідентифікатор питання |

* theory\_exam\_result

Таблиця 5.2.10 «theory\_exam\_result» призначена для зберігання питань та відповідей клієнтів до конкретного теоретичного іспиту.

Таблиця 5.2.10 – Структура таблиці «theory\_exam\_result»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| thoery\_exam\_id | INT |  | FK | Унікальний ідентифікатор теоретичного іспиту |
| question\_id | INT |  | FK | Унікальний ідентифікатор запитання |
| selected\_answer\_id | INT |  | FK | Унікальний ідентифікатор обраної відповіді |

* service\_center

Таблиця 5.2.11 «service\_center» призначена для зберігання інформації про сервісні центри.

Таблиця 5.2.11 – Структура таблиці «service\_center»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| address | VARCHAR | 100 |  | Адреса сервісного центру |
| number | INT |  |  | Номер сервісного центру |

* car

Таблиця 5.2.12 «car» призначена для зберігання автомобілей, які використовуються під час практичних іспитів.

Таблиця 5.2.12 – Структура таблиці «car»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| car\_owner | car\_owner\_type |  |  |  |
| gearbox | gearbox\_type |  |  |  |
| license\_plate | VARCHAR | 10 |  |  |
| category | category\_type |  |  |  |

Продовження таблиці 5.2.12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| service\_center\_id | INT |  | FK | Унікальний ідентифікатор сервісного центру |

* driving\_license

Таблиця 5.2.13 «driving\_license» призначена для зберігання посвідчень клієнтів та інспекторів.

Таблиця 5.2.13 – Структура таблиці «driving\_license»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| start\_date | DATE |  |  | Дата видачі |
| end\_date | DATE |  |  | Дата закінчення |
| client\_id | INT |  | FK | Унікальний ідентифікатор клієнта (якщо посвідчення клієнта) |
| inspector\_id | INT |  | FK | Унікальний ідентифікатор інспектора (якщо посвідчення інспектора) |

* driving\_license\_category

Таблиця 5.2.14 «driving\_license\_category» призначена для зберігання категорій кожного посвідчення.

Таблиця 5.2.14 – Структура таблиці «driving\_license\_category»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| category | category\_type |  |  | Категорія транспортного засобу |
| gearbox | gearbox\_type |  |  | Тип коробки передач |
| driving\_license\_id | INT |  | FK | Унікальний ідентифікатор посвідчення |

* driving\_license\_status

Таблиця 5.2.15 «driving\_license\_status» призначена для зберігання статусів для кожного посвідчення.

Таблиця 5.2.15 – Структура таблиці «driving\_license\_status»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| status | driving\_license\_status\_type |  |  | Статус посвідчення |
| start\_date | DATE |  |  | Дата початку статусу |

Продовження таблиці 5.2.15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| end\_date | DATE |  |  | Дата кінця статусу |
| reason | VARCHAR | 500 |  | Причина |
| driving\_license\_id | INT |  | FK | Унікальний ідентифікатор посвідчення |

* fine

Таблиця 5.2.16 «fine» призначена для зберігання штрафів для кожного посвідчення.

Таблиця 5.2.16 – Структура таблиці «fine»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| id | INT |  | PK | Унікальний ідентифікатор |
| status | fine\_status\_type |  |  | Статус |
| issued\_date | DATE |  |  | Дата накладання |
| reason | VARCHAR |  |  | Причина |
| sum | INT |  |  | Сума |
| driving\_license\_id | INT |  | FK | Унікальний ідентифікатор посвідчення |

Під час створення таблиць були створені користувацькі типи даних, які використовувалися для зберігання специфічної повторюваної інформації. Їхні можливі значення наведені в таблиці 5.2.17.

Таблиця 5.2.17 – Користувацькі типи даних

|  |  |
| --- | --- |
| **Тип** | **Можливі значення** |
| document\_type | 'passport', 'medical\_certificate', 'registration\_of\_residence', 'autoschool\_certificate', 'inspector\_certificate' |
| gearbox\_type | 'manual', 'automatic' |
| car\_owner\_type | 'service', 'autoschool' |
| exam\_result\_type | 'pending', 'passed', 'failed' |
| fine\_status\_type | 'pending', 'paid' |
| driving\_license\_status\_type | 'suspended', 'revoked' |
| ticket\_status\_type | 'pending', 'used', 'cancelled', 'expired' |
| category\_type | 'A', 'A1', 'B', 'B1', 'C', 'C1', 'D', 'D1', 'T', 'BE', 'C1E', 'CE', 'D1E', 'DE' |

* 1. Методи забезпечення цілісності даних

У цій базі даних застосовуються різні методи забезпечення цілісності даних, що гарантують правильність, точність і надійність збереження інформації.

* PRIMARY KEY та SERIAL

У кожній табилиці визначено поле як первинний ключ (PRIMARY KEY), що гарантує унікальність записів та використовуються для зв’язків з іншими таблицями. Для автоматичного його генерування використовуються генератор SERIAL, який при кожному новому записі збільшує його ідентифікатор на 1.

* FOREIGN KEY

Зовнішні ключи (FOREIGN KEY) використовуються для зв’язків між таблицями. Це забезпечує цілісність даних та запобігає створення посилань на неіснуючи записи. Наприклад, інспектор може працювати тільки в існуючому сервісному центрі.

* CHECK

За допомогою обмеження CHECK здійснюються перевірки значень при вставці чи оновленні даних. Наприклад, можемо перевіряти, що сума штрафу більше за 0.

* UNIQUE

Обмеження UNIQUE гарантує, що значення в певному полі є унікальним серед інших записів таблиці. Це запобігає внесення однакових даних, таких як, наприклад, номер телефона.

* Тригери

Тригери використовуються для автоматичного виконання певних дій при маніпуляціях з даними у таблиці. Наприклад, зміна статусу талона після його використання. Більш детально тригери будуть наведені в наступному розділі «Робота з базою даних».

5.4 Тексти таблиць

Після табличного опису бази даних та аналізу методів забезпечення цілісності, створимо усі необхідні таблиці використовуючи СУБД PostgreSQL.

* client

CREATE TABLE client (

id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

middle\_name VARCHAR(50) NOT NULL,

date\_of\_birth DATE NOT NULL CHECK (date\_of\_birth < CURRENT\_DATE),

address VARCHAR(100) NOT NULL,

phone\_number VARCHAR(30) NOT NULL UNIQUE,

email VARCHAR(50) NOT NULL UNIQUE

);

* service\_center

CREATE TABLE service\_center (

id SERIAL PRIMARY KEY,

address VARCHAR(100) NOT NULL UNIQUE,

number INT NOT NULL UNIQUE

);

* inpsector

CREATE TABLE inspector (

id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

middle\_name VARCHAR(50) NOT NULL,

service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE

);

* document

CREATE TABLE document (

id SERIAL PRIMARY KEY,

type document\_type NOT NULL,

end\_date DATE NOT NULL CHECK (end\_date > CURRENT\_DATE),

info\_file\_path VARCHAR(255) NOT NULL UNIQUE,

client\_id INT REFERENCES client(id) ON DELETE CASCADE,

inspector\_id INT REFERENCES inspector(id) ON DELETE CASCADE,

CHECK (client\_id IS NOT NULL OR inspector\_id IS NOT NULL)

);

* car

CREATE TABLE car (

id SERIAL PRIMARY KEY,

car\_owner car\_owner\_type NOT NULL,

gearbox gearbox\_type NOT NULL,

license\_plate VARCHAR(10) NOT NULL UNIQUE,

category category\_type NOT NULL,

service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE

);

* theory\_ticket

CREATE TABLE theory\_ticket (

id SERIAL PRIMARY KEY,

datetime TIMESTAMP NOT NULL CHECK(datetime > CURRENT\_TIMESTAMP),

status ticket\_status\_type NOT NULL DEFAULT 'pending',

service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE,

client\_id INT NOT NULL REFERENCES client(id) ON DELETE CASCADE

);

* driving\_ticket

CREATE TABLE driving\_ticket (

id SERIAL PRIMARY KEY,

datetime TIMESTAMP NOT NULL CHECK(datetime > CURRENT\_TIMESTAMP),

status ticket\_status\_type NOT NULL DEFAULT 'pending',

gearbox gearbox\_type NOT NULL,

category category\_type NOT NULL,

car\_owner car\_owner\_type NOT NULL,

service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE,

client\_id INT NOT NULL REFERENCES client(id) ON DELETE CASCADE

);

* theory\_exam

CREATE TABLE theory\_exam (

id SERIAL PRIMARY KEY,

theory\_ticket\_id INT NOT NULL UNIQUE REFERENCES theory\_ticket(id) ON DELETE CASCADE

);

* question

CREATE TABLE question (

id SERIAL PRIMARY KEY,

text VARCHAR(500) NOT NULL UNIQUE,

photo\_file\_path VARCHAR(255) UNIQUE

);

* answer

CREATE TABLE answer (

id SERIAL PRIMARY KEY,

text VARCHAR(500) NOT NULL,

photo\_file\_path VARCHAR(255) UNIQUE,

is\_correct BOOLEAN NOT NULL,

question\_id INT NOT NULL REFERENCES question(id) ON DELETE CASCADE,

UNIQUE (question\_id, text)

);

* theory\_exam\_result

CREATE TABLE theory\_exam\_result (

id SERIAL PRIMARY KEY,

theory\_exam\_id INT NOT NULL REFERENCES theory\_exam(id) ON DELETE CASCADE,

question\_id INT NOT NULL REFERENCES question(id) ON DELETE CASCADE,

selected\_answer\_id INT NOT NULL REFERENCES answer(id) ON DELETE CASCADE,

UNIQUE (theory\_exam\_id, question\_id)

);

* driving\_exam

CREATE TABLE driving\_exam (

id SERIAL PRIMARY KEY,

exam\_result exam\_result\_type NOT NULL DEFAULT 'pending',

video\_file\_path VARCHAR(255) UNIQUE,

driving\_ticket\_id INT NOT NULL UNIQUE REFERENCES driving\_ticket(id) ON DELETE CASCADE,

inspector\_id INT NOT NULL REFERENCES inspector(id) ON DELETE CASCADE,

car\_id INT REFERENCES car(id) ON DELETE CASCADE

);

* driving\_license

CREATE TABLE driving\_license (

id SERIAL PRIMARY KEY,

start\_date DATE DEFAULT CURRENT\_DATE CHECK (start\_date = CURRENT\_DATE),

end\_date DATE NOT NULL,

client\_id INT REFERENCES client(id) ON DELETE CASCADE,

inspector\_id INT REFERENCES inspector(id) ON DELETE CASCADE,

CHECK (inspector\_id IS NOT NULL OR client\_id IS NOT NULL),

CHECK (end\_date > start\_date)

);

* driving\_license\_category

CREATE TABLE driving\_license\_category (

id SERIAL PRIMARY KEY,

category category\_type NOT NULL,

gearbox gearbox\_type NOT NULL,

driving\_license\_id INT NOT NULL REFERENCES driving\_license(id) ON DELETE CASCADE,

UNIQUE (category, gearbox, driving\_license\_id)

);

* fine

CREATE TABLE fine (

id SERIAL PRIMARY KEY,

status fine\_status\_type NOT NULL DEFAULT 'pending',

issued\_date DATE DEFAULT CURRENT\_DATE CHECK (issued\_date = CURRENT\_DATE),

reason VARCHAR(500) NOT NULL,

sum INT NOT NULL CHECK (sum > 0),

driving\_license\_id INT NOT NULL REFERENCES driving\_license(id) ON DELETE CASCADE

);

* driving\_license\_status

CREATE TABLE driving\_license\_status (

id SERIAL PRIMARY KEY,

status driving\_license\_status\_type NOT NULL,

start\_date DATE DEFAULT CURRENT\_DATE CHECK (start\_date = CURRENT\_DATE),

end\_date DATE,

reason VARCHAR(500) NOT NULL,

driving\_license\_id INT NOT NULL REFERENCES driving\_license(id) ON DELETE CASCADE,

CHECK (end\_date IS NULL OR end\_date > start\_date)

);

5.5 Схема бази даних, реалізована засобами СУБД

На рисунку 5.5.1 зображена схема таблиць, реалізована засобами СУБД PostgreSQL, що відображає узагальнений вигляд бази даних, включаючи взаємозв’язки між таблицями та їх атрибутами.

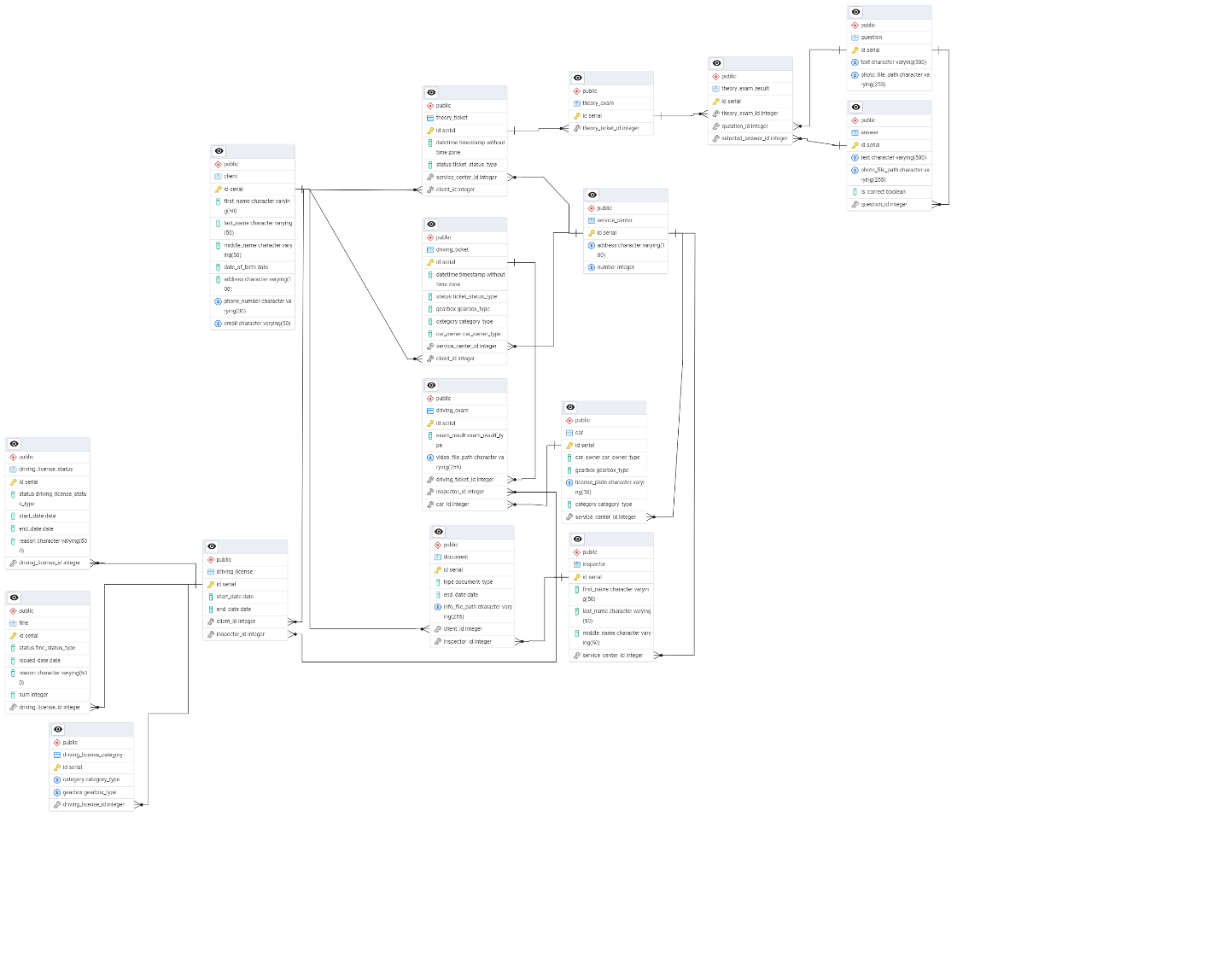


Рисунок 5.5.1 – Схема бази даних, згенерована засобами СУБД

5.6 Користувачі та тексти їх створення

У даній системі розрізняються кілька категорій користувачів, кожна з яких має свої права та можливості, тобто кожен має доступ до конкретної частини системи в залежності від своєї ролі.

* theory\_manager

Роль theory\_manager призначена для працівників, які відповідають за теоретичний матеріал для іспитів, а саме за тестові питання та відповіді на них.

CREATE USER theory\_manager WITH PASSWORD '1';

GRANT SELECT, INSERT, UPDATE, DELETE ON question TO theory\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON answer TO theory\_manager;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO theory\_manager;

* service\_center\_manager

Роль service\_center\_manager призначена для працівників, які відповідають за сервісні центри, включаючи їх працівників та автомобілі.

CREATE USER service\_center\_manager WITH PASSWORD '1';

GRANT SELECT, INSERT, UPDATE, DELETE ON service\_center TO service\_center\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON car TO service\_center\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON inspector TO service\_center\_manager;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO service\_center\_manager;

* main\_manager

Роль main\_manager призначена для працівників, які відповідають за реєстрацію клієнтів, видачу талонів та організацію іспитів.

CREATE USER main\_manager WITH PASSWORD '1';

GRANT SELECT, INSERT, UPDATE, DELETE ON client TO main\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON document TO main\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON driving\_ticket TO main\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON theory\_ticket TO main\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON driving\_exam TO main\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON theory\_exam TO main\_manager;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO main\_manager;

* theory\_exam\_result\_manager

Роль theory\_exam\_result\_manager призначена для працівників, які відповідають за проведення теоретичного іспиту, фіксують питання та відповіді клієнта.

CREATE USER theory\_exam\_result\_manager WITH PASSWORD '1';

GRANT SELECT, INSERT, UPDATE, DELETE ON theory\_exam\_result TO theory\_exam\_result\_manager;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO theory\_exam\_result\_manager;

* driving\_license\_manager

Роль driving\_license\_manager призначена для працівників, які відповідають за управління водійськими посвідченнями, їх категоріями та штрафами.

CREATE USER driving\_license\_manager WITH PASSWORD '1';

GRANT SELECT, INSERT, UPDATE, DELETE ON driving\_license TO driving\_license\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON driving\_license\_category TO driving\_license\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON driving\_license\_status TO driving\_license\_manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON fine TO driving\_license\_manager;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO driving\_license\_manager;

* analysis\_manager

Роль analysis\_manager призначена для працівників, які відповідають за аналіз діяльності всіх сервісних центрів.

CREATE USER analysis\_manager WITH PASSWORD '1';

GRANT SELECT ON ALL TABLES IN SCHEMA public TO analysis\_manager;

6 РОБОТА З БАЗОЮ ДАНИХ

6.1 Генератори

У кожній таблиці, вказаній в розділі «Реалізація бази даних», для автоматичного створення унікальних значень для полів з ідентифікаторами (id) було використано механізм PostgreSQL SERIAL. Він гарантує, що при додаванні нового запису, полю присвоюється наступне значення, взяте з генератора.

6.2 Тексти представлень

Представлення в базі даних дозволяють об’єднати кілька таблиць для спрощеного та централізованого доступу до різних даних без необхідності кожного разу писати складні SQL-запити.

* all\_tickets

Представлення all\_tickets об'єднує інформацію про теоретичні та практичні квитки, забезпечуючи зручний доступ до всіх квитків клієнтів. Це представлення необхідне для того, щоб мати змогу одночасно працювати з усіма видами квитків в одному запиті, що важливо для аналізу всіх записів на іспити, незалежно від їх типу.

CREATE OR REPLACE VIEW all\_tickets AS

SELECT dt.id, dt.client\_id, dt.datetime, dt.status, dt.service\_center\_id, 'driving' AS ticket\_type

FROM driving\_ticket dt

UNION ALL

SELECT tt.id, tt.client\_id, tt.datetime, tt.status, tt.service\_center\_id, 'theory' AS ticket\_type

FROM theory\_ticket tt;

На рисунку 6.2.1 наведено вигляд цього представлення.

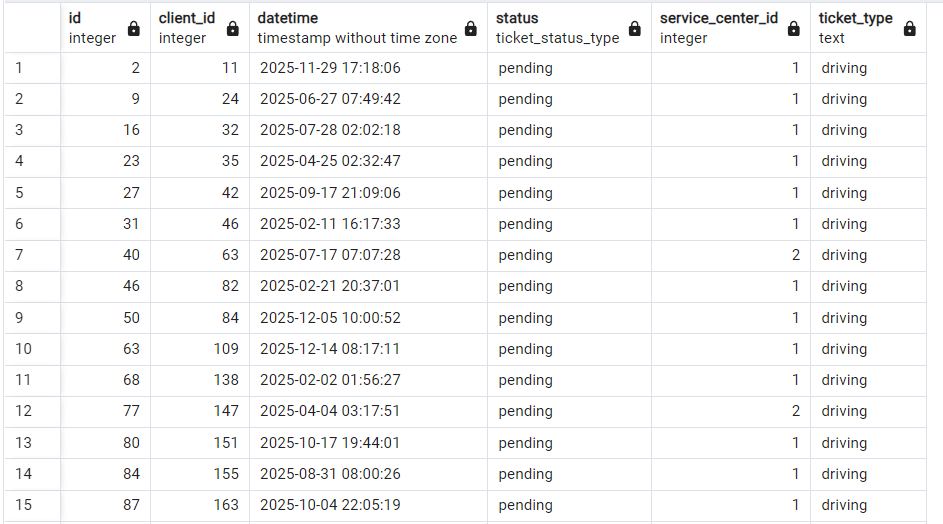


Рисунок 6.2.1 – Вигляд представлення all\_tickets

* theory\_exam\_result\_info

Представлення theory\_exam\_result\_info збирає всі результати теоретичних іспитів, надаючи детальну інформацію про запитання, вибрані відповіді та їх правильність. Воно необхідне для оперативного збору даних про успішність клієнтів на теоретичних іспитах, що дозволяє зручніше аналізувати результати, визначати рівень клієнтів та складності тестів.

CREATE OR REPLACE VIEW theory\_exam\_result\_info AS

SELECT tt.datetime, c.id AS client\_id, ter.theory\_exam\_id, q.text AS question, a.text AS answer, a.is\_correct

FROM theory\_exam\_result ter

JOIN question q ON ter.question\_id = q.id

JOIN answer a ON ter.selected\_answer\_id = a.id

JOIN theory\_exam te ON ter.theory\_exam\_id = te.id

JOIN theory\_ticket tt ON te.theory\_ticket\_id = tt.id

JOIN client c ON tt.client\_id = c.id;

На рисунку 6.2.2 наведено вигляд цього представлення.

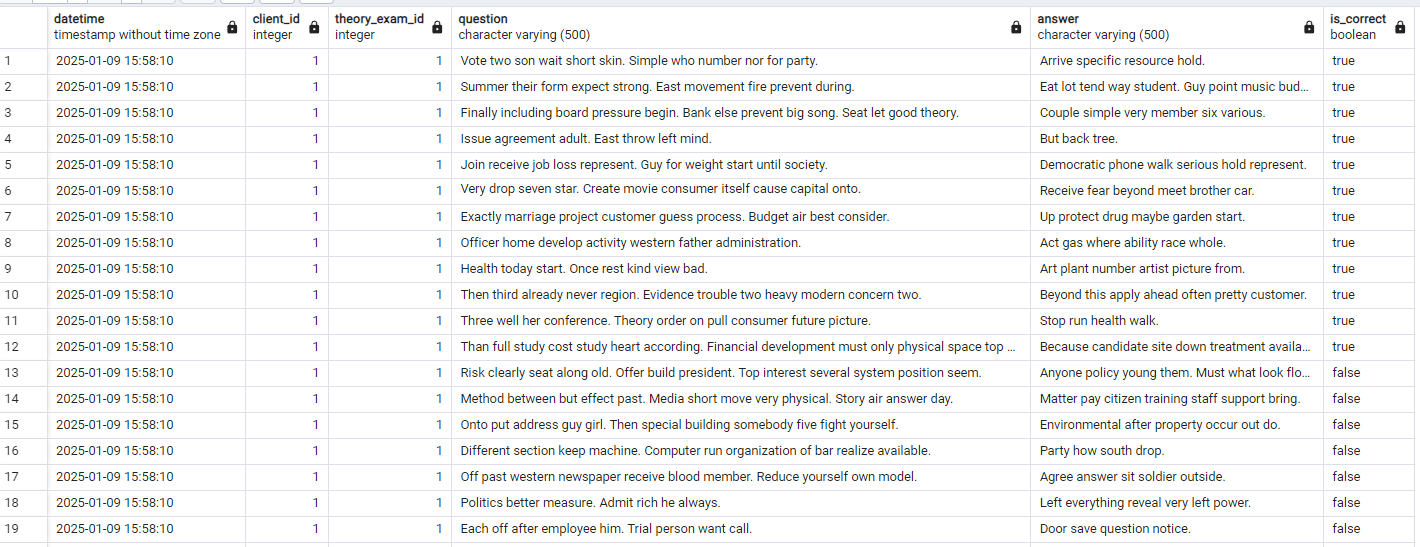


Рисунок 6.2.2 – Вигляд представлення theory\_exam\_result\_info

* person\_opened\_categories

Представлення person\_opened\_categories містить інформацію про всі категорії водійських посвідчень для клієнтів і інспекторів. Його використання необхідне для централізованого отримання актуальної інформації про категорії посвідчень, без необхідності об'єднувати дані з кількох таблиць. Це представлення є важливим для перевірок та моніторингу водійських категорій, що мають діючу ліцензію, а також для подальшого аналізу чи звітності.

CREATE OR REPLACE VIEW person\_opened\_categories AS

SELECT c.id, 'client' AS person\_type, dlc.category, dlc.gearbox

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

JOIN driving\_license\_category dlc ON dl.id = dlc.driving\_license\_id

WHERE dl.end\_date > CURRENT\_DATE

UNION

SELECT i.id, 'inspector' AS person\_type, dlc.category, dlc.gearbox

FROM inspector i

JOIN driving\_license dl ON i.id = dl.inspector\_id

JOIN driving\_license\_category dlc ON dl.id = dlc.driving\_license\_id

WHERE dl.end\_date > CURRENT\_DATE;

На рисунку 6.2.3 наведено вигляд цього представлення.

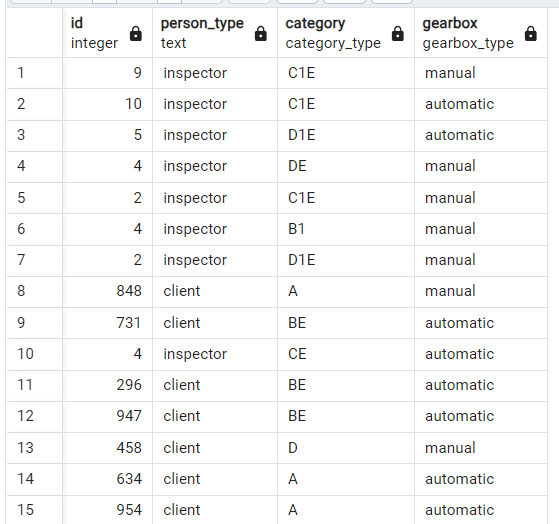


Рисунок 6.2.3 – Вигляд представлення person\_opened\_categories

6.3 Тексти процедур і функцій

Функції та процедури в базі даних створені для автоматизації повторюваних операцій. Вони дозволяють ефективно виконувати схожі дії, адаптуючись до переданих даних, якщо такі є, забезпечуючи гнучкість і простоту використання.

* get\_client\_age

Функція get\_client\_age обчислює вік клієнта на основі його дати народження. Це корисно для перевірки вікових обмежень, наприклад, при записі на практичний іспит.

CREATE OR REPLACE FUNCTION get\_client\_age(client\_id INT)

RETURNS INT AS $$

BEGIN

RETURN (

SELECT EXTRACT(YEAR FROM AGE(CURRENT\_DATE, date\_of\_birth))

FROM client

WHERE id = $1

);

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.1 наведено результат виконання цієї функції на клієнті з id = 5, який народився 2004-02-19.

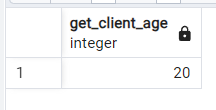


Рисунок 6.3.1 – Приклад виклику функції get\_client\_age

* client\_has\_active\_ticket

Функція client\_has\_active\_ticket повертає булеве значення в залежності від того, чи клієнт вже має активний талон на будь-який іспит. Це допомагає запобігти реєстрації на декілька іспитів одночасно.

CREATE OR REPLACE FUNCTION client\_has\_active\_ticket(client\_id INT)

RETURNS BOOLEAN AS $$

BEGIN

RETURN EXISTS (

SELECT 1

FROM all\_tickets at

WHERE at.client\_id = $1

AND status = 'pending'

);

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.2 наведено результат виконання цієї фунції на клієнті, який немає активних талонів.



Рисунок 6.3.2 – Приклад виклику функції client\_has\_active\_ticket

* get\_theory\_exam\_result

Функція get\_theory\_exam\_result підраховує кількість правильних відповідей на теоретичному іспиті, що допомагає оцінити успішність.

CREATE OR REPLACE FUNCTION get\_theory\_exam\_result(theory\_exam\_id INT)

RETURNS INT AS $$

BEGIN

RETURN (

SELECT COUNT(\*)

FROM theory\_exam\_result\_info teri

WHERE teri.theory\_exam\_id = $1

AND teri.is\_correct = TRUE

);

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.3 наведено результат виконання цієї функції на конкретному іспиті клієнта, який отримав 15 балів.

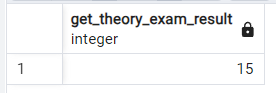


Рисунок 6.3.3 – Приклад виклику функції get\_theory\_exam\_result

* client\_has\_active\_passed\_theory\_exam

Функція client\_has\_active\_passed\_theory\_exam повертає булеве значення в залежності від того, чи має клієнт активний складений теоретичний іспит. Це допомагає визначити, чи відповідає клієнт умовам доступу до наступного етапу отримання посвідчення.

CREATE OR REPLACE FUNCTION client\_has\_active\_passed\_theory\_exam(client\_id INT)

RETURNS BOOLEAN AS $$

BEGIN

RETURN EXISTS (

SELECT 1

FROM theory\_exam te

JOIN theory\_ticket tt ON te.theory\_ticket\_id = tt.id

WHERE tt.client\_id = $1

AND get\_theory\_exam\_result(te.id) >= 18

AND CURRENT\_DATE - INTERVAL '2 years' < tt.datetime

);

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.4 наведено результат виконання цієї функції на конкретному клієнті, який не має активного теоретичного іспиту.

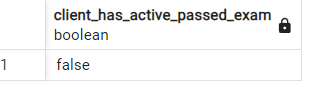


Рисунок 6.3.4 – Приклад виклику функції client\_has\_active\_passed\_theory\_exam

* person\_has\_active\_document

Функція person\_has\_active\_document повертає булеве значення в залежності від того, чи має особа (клієнт або інспектор) активний документ, що необхідно для визначення прав на участь в іспитах.

CREATE OR REPLACE FUNCTION person\_has\_active\_document(person\_type VARCHAR(255), person\_id INT, doc\_type document\_type)

RETURNS BOOLEAN AS $$

BEGIN

IF person\_type = 'client' THEN

RETURN EXISTS (

SELECT 1

FROM document d

WHERE d.client\_id = $2

AND d.type = $3

AND end\_date > CURRENT\_DATE

);

ELSIF person\_type = 'inspector' THEN

RETURN EXISTS (

SELECT 1

FROM document d

WHERE d.inspector\_id = $2

AND d.type = $3

AND end\_date > CURRENT\_DATE

);

END IF;

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.5 наведено результат виконання цієї функції для медичної картки на конкретному клієнті, який її має.

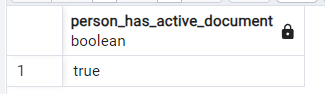


Рисунок 6.3.5 – Приклад виклику функції person\_has\_active\_document

* get\_person\_last\_active\_driving\_license

Функція get\_person\_last\_active\_driving\_license повертає унікальний ідентифікатор останнього активного водійського посвідчення особи (клієнта чи інспектора), що допомагає легше отримувати посвідчення та в одночас перевіряти, чи таке взагалі існує.

CREATE OR REPLACE FUNCTION get\_person\_last\_active\_driving\_license(person\_type VARCHAR(255), person\_id INT)

RETURNS INT AS $$

DECLARE

last\_driving\_license\_id INT;

BEGIN

IF $1 = 'client' THEN

SELECT MAX(dl.id) INTO last\_driving\_license\_id

FROM driving\_license dl

WHERE dl.client\_id = $2

AND dl.end\_date > CURRENT\_DATE

AND NOT EXISTS (

SELECT 1

FROM driving\_license\_status dls

WHERE dls.driving\_license\_id = dl.id

AND dls.status = 'revoked'

AND (dls.end\_date IS NULL OR dls.end\_date > CURRENT\_DATE)

);

ELSE

SELECT MAX(dl.id) INTO last\_driving\_license\_id

FROM driving\_license dl

WHERE dl.inspector\_id = $2

AND dl.end\_date > CURRENT\_DATE

AND NOT EXISTS (

SELECT 1

FROM driving\_license\_status dls

WHERE dls.driving\_license\_id = dl.id

AND dls.status = 'revoked'

AND (dls.end\_date IS NULL OR dls.end\_date > CURRENT\_DATE)

);

END IF;

RETURN last\_driving\_license\_id;

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.6 наведено приклад виконання цієї функції на конкретному інспекторі, який має активне посвідчення.

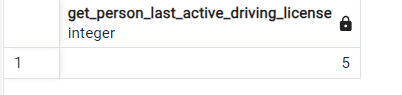


Рисунок 6.3.6 – Приклад виклику функції get\_person\_last\_active\_driving\_license

* get\_tickets\_count\_for\_day

Функція get\_tickets\_count\_for\_day підраховує кількість теоретичних і практичних квитків, виданих на вказану дату і центрі. Це допомагає планувати та управляти роботою сервісних центрів.

CREATE OR REPLACE FUNCTION get\_tickets\_count\_for\_day(day DATE, service\_center\_id INT)

RETURNS TABLE(theory\_tickets\_count BIGINT, driving\_tickets\_count BIGINT) AS $$

BEGIN

RETURN QUERY SELECT

COUNT(CASE WHEN at.ticket\_type = 'theory' THEN 1 END) AS theory\_tickets\_count, COUNT(CASE WHEN at.ticket\_type = 'driving' THEN 1 END) AS driving\_tickets\_count

FROM all\_tickets at

WHERE DATE(at.datetime) = $1

AND at.service\_center\_id = $2;

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.7 наведено приклад виконання цієї функції для дати 2024-12-15 та сервісного центру з ідентифікатором 1.

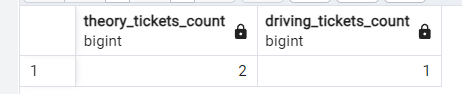


Рисунок 6.3.7 – Приклад виклику функції get\_tickets\_count\_for\_day

* get\_appropriate\_inspectors\_for\_driving\_exam

Функція get\_appropriate\_inspectors\_for\_driving\_exam знаходить іспекторів, які можуть провести практичний іспит, враховуючи наявність відповідної категорії, типу коробки передач, документів та місця роботи.

CREATE OR REPLACE FUNCTION get\_appropriate\_inspectors\_for\_driving\_exam(driving\_ticket\_id INT)

RETURNS TABLE(inspector\_id INT, inspector\_name TEXT) AS $$

BEGIN

RETURN QUERY SELECT

i.id, i.first\_name || ' ' || i.last\_name || ' ' || i.middle\_name

FROM inspector i

JOIN service\_center sc ON i.service\_center\_id = sc.id

JOIN driving\_ticket dt ON sc.id = dt.service\_center\_id

JOIN person\_opened\_categories poc ON i.id = poc.id

WHERE dt.id = $1

AND poc.person\_type = 'inspector'

AND poc.category = dt.category

AND poc.gearbox = dt.gearbox

AND person\_has\_active\_document('inspector', i.id, 'inspector\_certificate');

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.8 наведено приклад виконання цієї функції для практичного талону, із зазначеною категорію А та автоматичною коробкою передач.

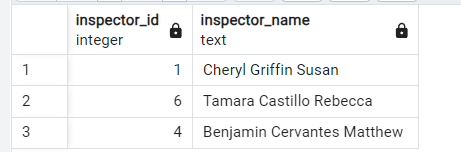


Рисунок 6.3.8 – Приклад виклику функції get\_appropriate\_inspectors\_for\_driving\_exam

* get\_new\_theory\_exam\_questions

Функція get\_new\_theory\_exam\_questions вибирає нові питання для теоретичного іспиту клієнта, враховуючи його минулі спроби, чим забезпечує уникнення повторюваних питань.

CREATE OR REPLACE FUNCTION get\_new\_theory\_exam\_questions(client\_id INT)

RETURNS TABLE(question\_id INT, question\_text VARCHAR(500), photo\_file\_path VARCHAR(255)) AS $$

BEGIN

DROP TABLE IF EXISTS left\_questions;

CREATE TEMP TABLE left\_questions AS

SELECT q.id, q.text, q.photo\_file\_path

FROM question q

WHERE q.text NOT IN (

SELECT question

FROM theory\_exam\_result\_info teri

WHERE teri.client\_id = $1

)

ORDER BY RANDOM()

LIMIT 20;

IF (SELECT COUNT(\*) FROM left\_questions) = 20 THEN

RETURN QUERY SELECT \* FROM left\_questions;

ELSE

RETURN QUERY SELECT q.id, q.text, q.photo\_file\_path

FROM question q

ORDER BY RANDOM()

LIMIT 20;

END IF;

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.9 наведено приклад виконання цієї функції для клієнта, який вже складав іспит. Повернені тестові запитання не повторюються із минулими.

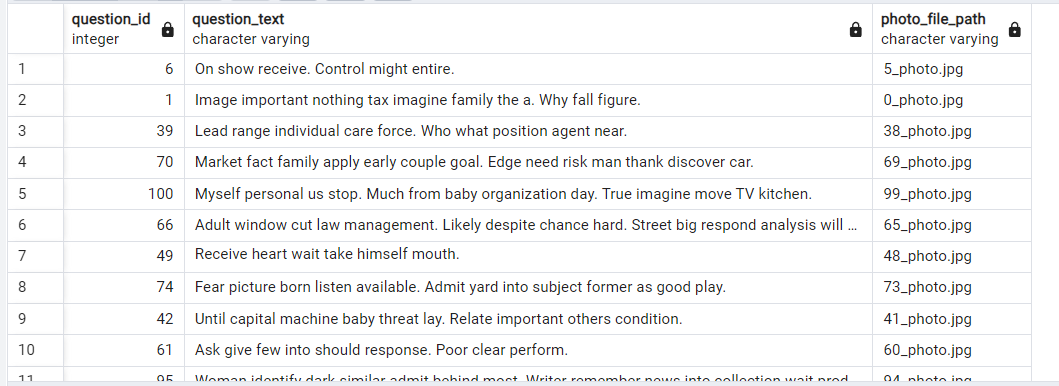


Рисунок 6.3.9 – Приклад виклику функції get\_new\_theory\_exam\_questions

* update\_expired\_tickets\_status

Процедура update\_expired\_tickets\_status оновлює статус талонів на «прострочений», якщо вони не були вчасно використані. Це надає змогу виявляти клієнтів, які заважають ефективно працювати сервісним центрам.

CREATE OR REPLACE PROCEDURE update\_expired\_tickets\_status()

AS $$

BEGIN

UPDATE theory\_ticket

SET status = 'expired'

WHERE datetime < CURRENT\_TIMESTAMP - INTERVAL '1 day'

AND status = 'pending';

UPDATE driving\_ticket

SET status = 'expired'

WHERE datetime < CURRENT\_TIMESTAMP

AND status = 'pending';

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.10 зображено талони, один з яких вже прострочений (id = 2553). Після виклику процедури, цей талон отримує відповідний статус, що зображено на рисунку 6.3.11.

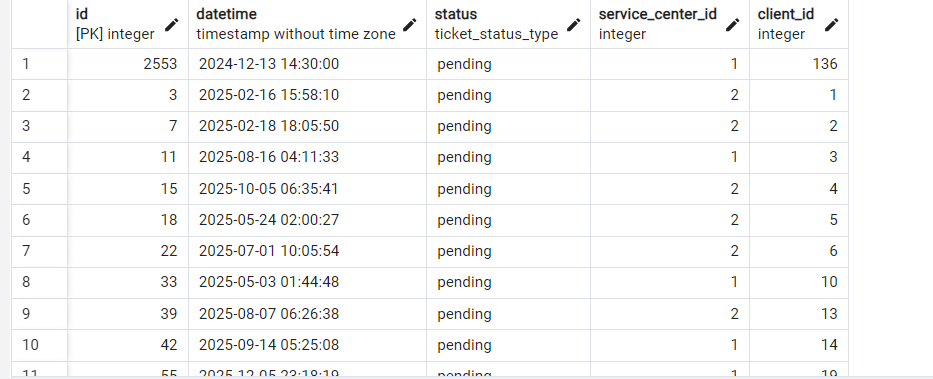


Рисунок 6.3.10 – Приклад простроченого талону перед застосуванням процедури update\_expired\_tickets\_status

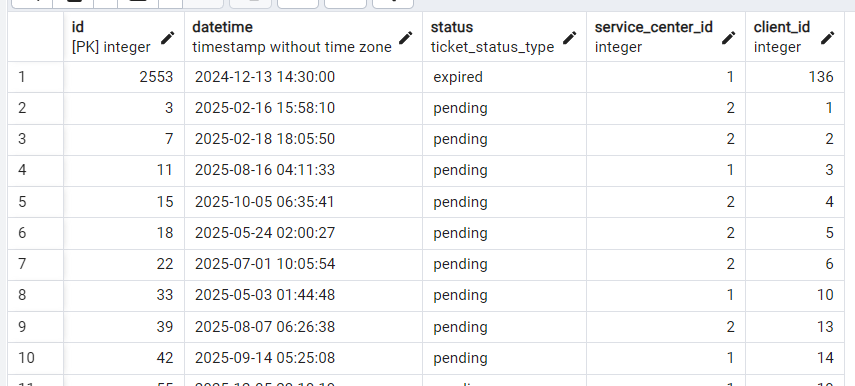


Рисунок 6.3.11 – Приклад простроченого талону після застосування процедури update\_expired\_tickets\_status

* double\_pending\_fines

Процедура double\_pending\_fines подвоює суму штрафів для тих, що ще не оплачені і перевищили двотижневий термін з моменту видачі. Це спонукає клієнтів вчасно оплачувати будь-які порушення.

CREATE OR REPLACE PROCEDURE double\_pending\_fines()

AS $$

BEGIN

UPDATE fine

SET sum = sum \* 2

WHERE status = 'pending'

AND issued\_date < CURRENT\_DATE - INTERVAL '2 week';

END;

$$ LANGUAGE plpgsql;

На рисунку 6.3.12 зображено штрафи, два з яких вже прострочені (id = 3 та id = 4) більше ніж на 14 днів. Після виклику процедури, їх сума збільшиться вдвічі, що зображено на рисунку 6.3.13.

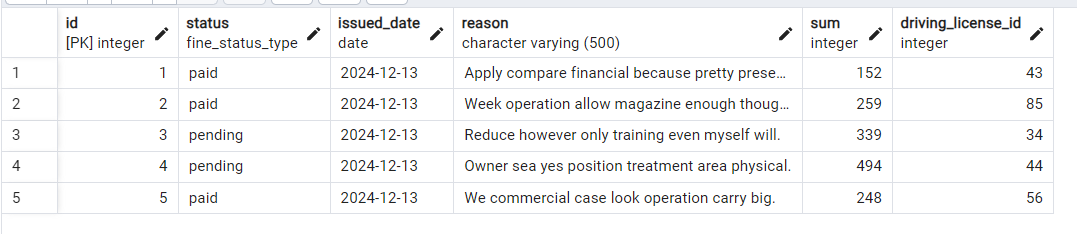


Рисунок 6.3.12 – Приклад прострочених штрафів перед застосуванням процедури double\_pending\_fines

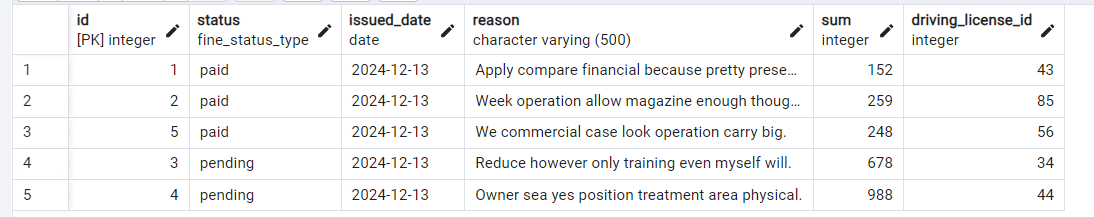


Рисунок 6.3.13 – Приклад прострочених штрафів після застосування процедури double\_pending\_fines

6.4 Тексти тригерів

Тригери в базі даних використовуються для автоматизації та забезпечення цілісності даних під час виконання операцій з таблицями згідно з необхідними бізнес-правилами. Вони дозволяють виконувати певні дії до або після вставки, оновлення та видалення даних.

* check\_ticket\_datetime

Тригери, що використовують тригерну функцію check\_ticket\_datetime при додаванні талону, перевіряють, чи не потрапляє дата та час створення квитка в минуле. Може здатися, що даний тригер можна змінити обмеженням CHECK, але тоді ми не зможемо в деяких випадках змінювати, наприклад, статус квитка без зміни його дати та часу.

CREATE OR REPLACE FUNCTION check\_ticket\_datetime()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_TABLE\_NAME = 'theory\_ticket' AND NEW.datetime < CURRENT\_TIMESTAMP THEN

RAISE EXCEPTION 'Theory ticket datetime cannot be in the past';

ELSIF TG\_TABLE\_NAME = 'driving\_ticket' AND NEW.datetime < CURRENT\_TIMESTAMP THEN

RAISE EXCEPTION 'Driving ticket datetime cannot be in the past';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_ticket\_datetime\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ticket\_datetime();

CREATE TRIGGER check\_ticket\_datetime\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ticket\_datetime();

На рисунку 6.4.1 зображено помилку при спробу додавання талону на минулу дату.

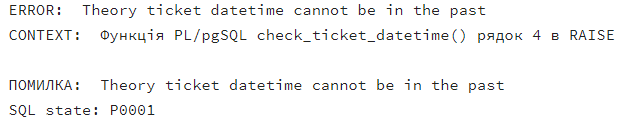


Рисунок 6.4.1 – Помилка при додаванні талону на минулу дату

* check\_client\_age

Тригер, що використовує тригерну функцію check\_client\_age при додаванні практичного талону, перевіряє, що вік клієнта відповідає необхідному для отримання заданої категорії посвідчення.

CREATE OR REPLACE FUNCTION check\_client\_age()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.category IN ('A', 'A1') AND get\_client\_age(NEW.client\_id) < 16 THEN

RAISE EXCEPTION 'Client must be at least 16 years old to get category A driving ticket';

ELSIF NEW.category IN ('B', 'B1') AND get\_client\_age(NEW.client\_id) < 18 THEN

RAISE EXCEPTION 'Client must be at least 18 years old to get a driving ticket';

ELSIF NEW.category IN ('BE', 'CE', 'C1E') AND get\_client\_age(NEW.client\_id) < 19 THEN

RAISE EXCEPTION 'Client must be at least 19 years old to get category BE, CE or C1E driving ticket';

ELSIF NEW.category IN ('D', 'D1', 'DE', 'D1E', 'T') AND get\_client\_age(NEW.client\_id) < 21 THEN

RAISE EXCEPTION 'Client must be at least 21 years old to get category D1, D or T driving ticket';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_age\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_age();

На рисунку 6.4.2 зображено помилку при спробі записатися на практичний іспит, категорія якого не відповідає віку клієнта.

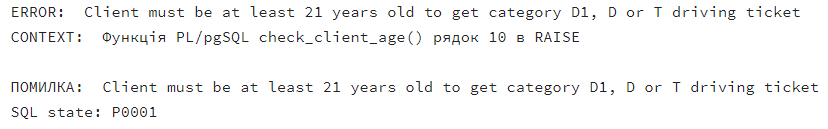


Рисунок 6.4.2 – Помилка при записі на практичний іспит з категорію, яка не відповідає віку

* check\_client\_gets\_new\_category

Тригер, що використовує тригерну функцію check\_client\_gets\_new\_category при додаванні практичного талону, перевіряє, що клієнт отримує категорію, якої в нього ще немає.

CREATE OR REPLACE FUNCTION check\_client\_gets\_new\_category()

RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (

SELECT 1

FROM person\_opened\_categories

WHERE person\_type = 'client'

AND id = NEW.client\_id

AND category = NEW.category

AND gearbox = NEW.gearbox

) THEN

RAISE EXCEPTION 'Client already has the specified category and gearbox';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_gets\_new\_category\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_gets\_new\_category();

На рисунку 6.4.3 зображено помилку при спробі записатися на практичний іспит з категорією, яка вже відкрита у клієнта.

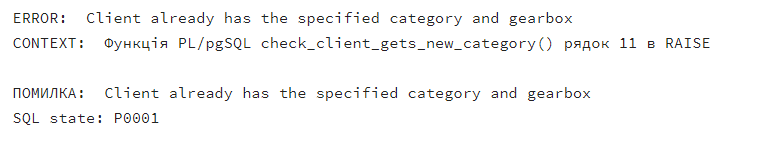


Рисунок 6.4.3 – Помилка при записі на практичний іспит з категорію, яка вже є у клієнта

* check\_client\_has\_nessesary\_documents

Тригери, що використовують тригерну функцію check\_client\_has\_nessasary\_documents при додаванні талону, перевіряють, що клієнт має необхідні документи для заданого типу іспиту.

CREATE OR REPLACE FUNCTION check\_client\_has\_nessesary\_documents()

RETURNS TRIGGER AS $$

BEGIN

IF NOT person\_has\_active\_document('client', NEW.client\_id, 'passport') OR NOT person\_has\_active\_document('client', NEW.client\_id, 'registration\_of\_residence') THEN

RAISE EXCEPTION 'Client must have a passport and a registration of residence';

ELSIF TG\_TABLE\_NAME = 'driving\_ticket' AND (NOT person\_has\_active\_document('client', NEW.client\_id, 'medical\_certificate') OR NOT person\_has\_active\_document('client', NEW.client\_id, 'autoschool\_certificate')) THEN

RAISE EXCEPTION 'Client must have a medical certificate and an autoschool certificate';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_has\_nessesary\_documents\_theory\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_has\_nessesary\_documents();

CREATE TRIGGER check\_client\_has\_nessesary\_documents\_driving\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_has\_nessesary\_documents();

На рисунку 6.4.4 зображено помилку при спробі клієнта записатися на практичний іспит, не маючи при цьому активного паспорту.

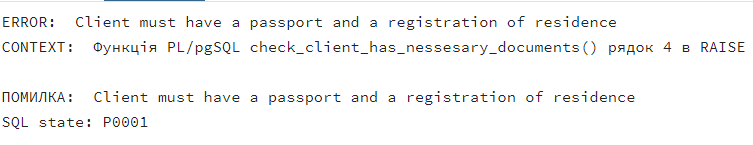


Рисунок 6.4.4 – Помилка при записі на практичний іспит клієнта, який немає паспорту

* check\_only\_one\_active\_ticket

Тригери, що використовують тригерну функцію check\_only\_one\_active\_ticket при додаванні талону, перевіряють, що клієнт має тільки один активний запис. Це зроблено так, бо клієнт не може знати заздалегідь, чи він складе іспит, чи ні.

CREATE OR REPLACE FUNCTION check\_only\_one\_active\_ticket()

RETURNS TRIGGER AS $$

BEGIN

IF client\_has\_active\_ticket(NEW.client\_id) THEN

RAISE EXCEPTION 'Client can have only one active ticket';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_only\_one\_active\_theory\_ticket\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_only\_one\_active\_ticket();

CREATE TRIGGER check\_only\_one\_active\_driving\_ticket\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_only\_one\_active\_ticket();

На рисунку 6.4.5 зображено помилку при спробі клієнта записатися на практичний іспит, маючи при цьому вже активний талон.

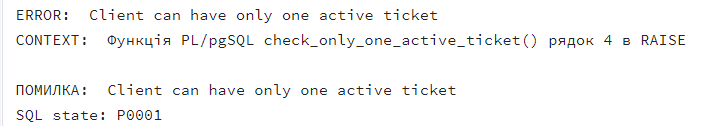


Рисунок 6.4.5 – Помилка при записі на практичний іспит клієнта, який вже має активний талон

* check\_ten\_days\_between\_tickets

Тригери, що використовують тригерну функцію check\_ten\_days\_between\_tickets при додаванні талону, перевіряють, щоб між спробами складання одного типу іспиту було хоча б 10 днів. Це зроблено, щоб клієнт спочатку довчився перед перескладанням.

CREATE OR REPLACE FUNCTION check\_ten\_days\_between\_tickets()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_TABLE\_NAME = 'theory\_ticket' AND EXISTS (

SELECT 1

FROM theory\_ticket

WHERE client\_id = NEW.client\_id

AND datetime > NEW.datetime - INTERVAL '10 days'

AND status != 'cancelled'

) THEN

RAISE EXCEPTION 'Client must wait at least 10 days between theory tickets';

ELSIF TG\_TABLE\_NAME = 'driving\_ticket' AND EXISTS (

SELECT 1

FROM driving\_ticket

WHERE client\_id = NEW.client\_id

AND datetime > NEW.datetime - INTERVAL '10 days'

AND status != 'cancelled'

) THEN

RAISE EXCEPTION 'Client must wait at least 10 days between driving tickets';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_ten\_days\_between\_theory\_tickets\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ten\_days\_between\_tickets();

CREATE TRIGGER check\_ten\_days\_between\_driving\_tickets\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ten\_days\_between\_tickets();

На рисунку 6.4.6 зображено помилку при спробі клієнта записатися на теоретичний іспит після невдалої спроби, яка була менше 10 днів тому.

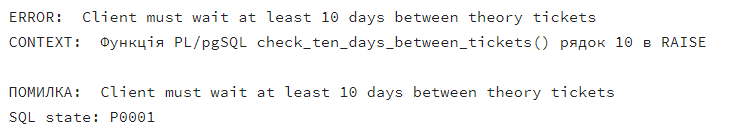


Рисунок 6.4.6 – Помилка при записі на теоретичний іспит клієнта, який мав іншу спробу менше 10 днів тому

* check\_client\_has\_active\_passed\_theory\_exam

Тригер, що використовує тригерну функцію check\_client\_has\_active\_passed\_theory\_exam при додаванні практичного талону, перевіряє, що клієнт має активний теоретичний іспит перед записом на практичну частину. Це зроблено для того, щоб спочатку клієнт здавав теорію, а тільки потім допускався до керування транспортним засобом.

CREATE OR REPLACE FUNCTION check\_client\_has\_active\_passed\_theory\_exam()

RETURNS TRIGGER AS $$

BEGIN

IF NOT client\_has\_active\_passed\_theory\_exam(NEW.client\_id) THEN

RAISE EXCEPTION 'Client must have passed the theory exam';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_has\_active\_passed\_theory\_exam\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_has\_active\_passed\_theory\_exam();

На рисунку 6.4.7 зображено помилку при спробі клієнта записатися на практичний іспит, не здававши при цьому теоретичну частину.

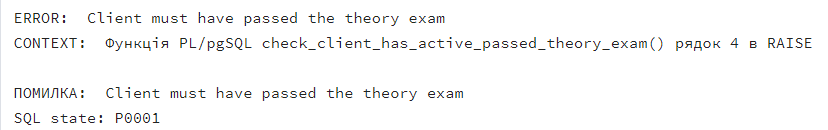


Рисунок 6.4.7 – Помилка при записі на практичний іспит без активного теоретичного.

* update\_ticket\_status\_after\_exam\_creation

Тригери, що використовують тригерну функцію update\_ticket\_status\_after\_exam\_creation при додаванні іспиту, автоматично змінюють статус відповідного талону на «використано», щоб це не було необхідно робити вручну.

CREATE OR REPLACE FUNCTION update\_ticket\_status\_after\_exam\_creation()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_TABLE\_NAME = 'theory\_exam' THEN

UPDATE theory\_ticket

SET status = 'used'

WHERE id = NEW.theory\_ticket\_id;

ELSIF TG\_TABLE\_NAME = 'driving\_exam' THEN

UPDATE driving\_ticket

SET status = 'used'

WHERE id = NEW.driving\_ticket\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_theory\_ticket\_status\_trigger

AFTER INSERT ON theory\_exam

FOR EACH ROW

EXECUTE FUNCTION update\_ticket\_status\_after\_exam\_creation();

CREATE TRIGGER update\_driving\_ticket\_status\_trigger

AFTER INSERT ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION update\_ticket\_status\_after\_exam\_creation();

На рисунку 6.4.8 зображено теоретичний талон (id = 5) перед його використанням. Після створення відповідного теоретичного іспиту, використаний талон автоматично змінює статус на «використано», що зображено на рисунку 6.4.9.

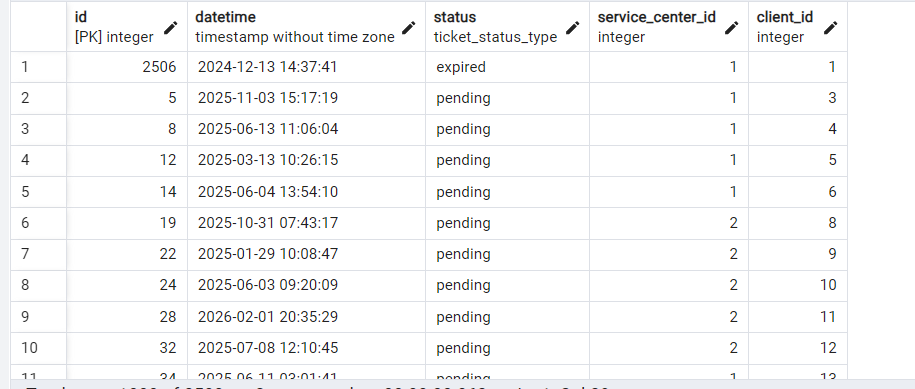


Рисунок 6.4.8 – Статус талону (id = 5) перед використанням

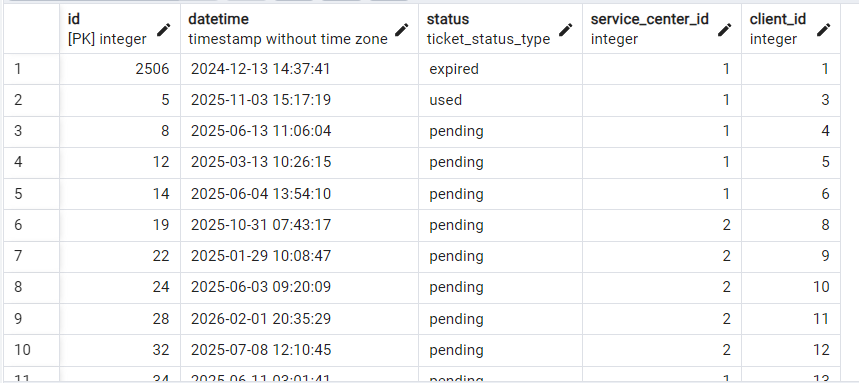


Рисунок 6.4.9 – Автоматично змінений статус талону (id = 5) після створення теоретичного іспиту по ньому

* check\_car\_availablity

Тригер, що використовує тригерну функцію check\_car\_availablity при додаванні практичного талону, за умови, що в записі вказаний автомобіль сервісу, перевіряє, що автомобіль із заданої категорією та коробкою передач існує в вказаному сервісному центрі. Це зроблено, щоб впевнитися, що практичний іспит можна буде провести на заданій локації.

CREATE OR REPLACE FUNCTION check\_car\_availability()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.car\_owner = 'service' AND NOT EXISTS (

SELECT 1

FROM car

WHERE service\_center\_id = NEW.service\_center\_id

AND gearbox = NEW.gearbox

AND category = NEW.category

AND car\_owner = 'service'

) THEN

RAISE EXCEPTION 'No available car with the specified gearbox and category at the service center';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_car\_availability\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_car\_availability();

На рисунку 6.4.10 зображено помилку при спробі клієнта записатися на практичний іспит в сервісний центр, що немає відповідного автомобіля.

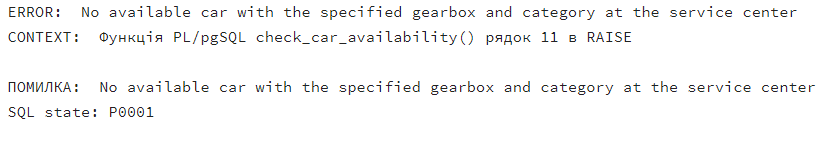


Рисунок 6.4.10 – Помилка при записі на практичний іспит в сервісний центр, який немає відповідного автомобіля

* check\_car\_is\_appropriate

Тригер, що використовує тригерну функцію check\_car\_is\_appropriate при додаванні чи зміні практичного іспиту, перевіряє, чи машина відповідає характеристикам, зазначених у талоні. Це зроблено, щоб інспектор не помилився при виборі автомобіля для екзамену, а якщо і помилився, то щоб клієнт не отримав неправильну категорію.

CREATE OR REPLACE FUNCTION check\_car\_is\_appropriate()

RETURNS TRIGGER AS $$

DECLARE

driving\_ticket\_category VARCHAR(255);

driving\_ticket\_gearbox VARCHAR(255);

car\_category VARCHAR(255);

car\_gearbox VARCHAR(255);

BEGIN

IF NEW.car\_id IS NOT NULL THEN

SELECT category, gearbox INTO driving\_ticket\_category, driving\_ticket\_gearbox

FROM driving\_ticket

WHERE id = NEW.driving\_ticket\_id;

SELECT category, gearbox INTO car\_category, car\_gearbox

FROM car

WHERE id = NEW.car\_id;

IF driving\_ticket\_category != car\_category OR driving\_ticket\_gearbox != car\_gearbox THEN

RAISE EXCEPTION 'Car is not appropriate for the driving ticket';

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_car\_is\_appropriate\_trigger

BEFORE INSERT ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION check\_car\_is\_appropriate();

На рисунку 6.4.11 зображено помилку при спробі встановити автомобіль для практичного іспиту, який не відповідає зазначеному у талоні клієнта.

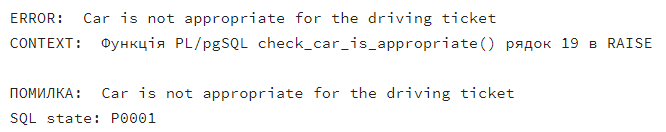


Рисунок 6.4.11 – Помилка при встановленні автомобіля для практичного іспиту, що не відповідає зазначеному у талоні

* check\_inspector\_capability

Тригер, що використовує тригерну функцію check\_inspector\_capability при створенні практичного іспиту, перевіряє, що обраний інспектор має необхідну категорію, враховуючи коробку передач.

CREATE OR REPLACE FUNCTION check\_inspector\_capability()

RETURNS TRIGGER AS $$

BEGIN

IF NOT EXISTS (

SELECT 1

FROM person\_opened\_categories

WHERE person\_type = 'inspector'

AND id = NEW.inspector\_id

AND category = (SELECT category FROM driving\_ticket dt WHERE dt.id = NEW.driving\_ticket\_id)

AND gearbox = (SELECT gearbox FROM driving\_ticket dt WHERE dt.id = NEW.driving\_ticket\_id)

) THEN

RAISE EXCEPTION 'Inspector does not have the necessary category';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_inspector\_capability\_trigger

BEFORE INSERT ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION check\_inspector\_capability();

На рисунку 6.4.12 зображено помилку при спробі створити практичний іспит із інспектором, який немає відповідної категорії.

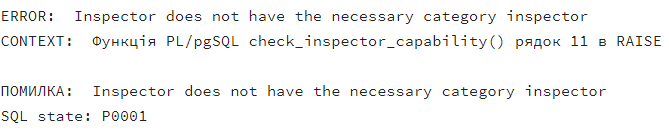


Рисунок 6.4.12 – Помилка при створенні іспиту із інспектором, який немає необхідної категорії

* check\_inspector\_has\_necessary\_documents

Тригер, що використовує тригерну функцію check\_inspector\_has\_necessary\_documents при створенні практичного іспиту, перевіряє, що інспектор має найважливіший документ – сертифікат, що підтверджує змогу проводити іспитування.

CREATE OR REPLACE FUNCTION check\_inspector\_has\_necessary\_documents()

RETURNS TRIGGER AS $$

BEGIN

IF NOT person\_has\_active\_document('inspector', NEW.inspector\_id, 'inspector\_certificate') THEN

RAISE EXCEPTION 'Inspector must have an inspector certificate';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_inspector\_has\_necessary\_documents\_trigger

BEFORE INSERT ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION check\_inspector\_has\_necessary\_documents();

На рисунку 6.4.13 зображено помилку при спробі створити практичний іспит із інспектором, який немає відповідного сертифікату або має прострочений.

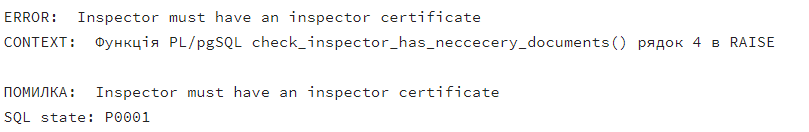


Рисунок 6.4.13 – Помилка при створенні іспиту із інспектором, який немає активного сертифікату

* create\_driving\_license\_or\_add\_category\_after\_driving\_exam

Тригер, що використовує тригерну функцію create\_driving\_license\_or\_add\_category\_after\_driving\_exam при створенні або оновлені практичного іспиту, необхідний для автоматичної видачі посвідчення або додавання категорії після успішної здачі.

CREATE OR REPLACE FUNCTION create\_driving\_license\_or\_add\_category\_after\_driving\_exam()

RETURNS TRIGGER AS $$

DECLARE

client\_id INT;

last\_active\_driving\_license\_id INT;

BEGIN

IF NEW.exam\_result != 'passed' THEN

RETURN NEW;

END IF;

SELECT dt.client\_id INTO client\_id

FROM driving\_ticket dt

WHERE id = NEW.driving\_ticket\_id;

last\_active\_driving\_license\_id = get\_person\_last\_active\_driving\_license('client', client\_id);

IF last\_active\_driving\_license\_id IS NULL THEN

INSERT INTO driving\_license (end\_date, client\_id)

VALUES (CURRENT\_DATE + INTERVAL '2 year', client\_id);

last\_active\_driving\_license\_id = get\_person\_last\_active\_driving\_license('client', client\_id);

END IF;

INSERT INTO driving\_license\_category (category, gearbox, driving\_license\_id)

VALUES ((SELECT category FROM driving\_ticket WHERE id = NEW.driving\_ticket\_id), (SELECT gearbox FROM driving\_ticket WHERE id = NEW.driving\_ticket\_id), last\_active\_driving\_license\_id);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER create\_driving\_license\_or\_add\_category\_after\_driving\_exam\_trig

AFTER INSERT OR UPDATE ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION create\_driving\_license\_or\_add\_category\_after\_driving\_exam();

На рисунку 6.4.14 зображено посвідчення клієнта (id = 30). Спочатку їх немає, але після оновлення статусу практичного іспиту на «складений» було автоматично створено посвідчення водія (рисунок 6.4.15) та надано відповідні категорії (рисунок 6.4.16).

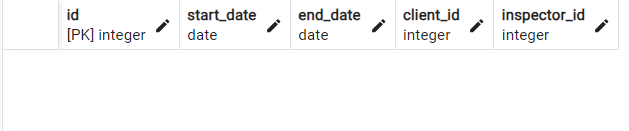


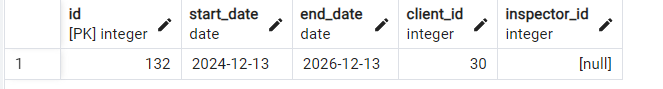
Рисунок 6.4.14 – Відсутність посвідчень у клієнта (id = 30)

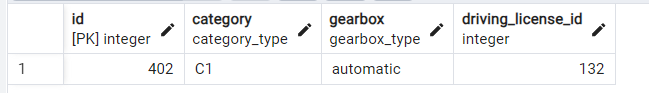
Рисунок 6.4.15 – Автоматично додане посвідчення після складання практичного іспиту 

Рисунок 6.4.16 – Автоматично додана категорія до посвідчення після складання практичного іспиту

6.5 SQL-запити

SQL-запити є ключовим інструментом для доступу до даних у базі, виконання пошукових операцій, аналізу взаємозв'язків між таблицями та формування результатів для вирішення прикладних задач.

* Кількість спроб скласти теоретичний іспит

Запит підраховує кількість невдалих спроб теоретичного іспиту для кожного клієнту, що допомагає аналізувати рівень підготовленості та виявляти тих, хто потребує додаткового навчання.

SELECT c.\*, COUNT(\*) AS attempts

FROM client c

JOIN theory\_ticket tt ON c.id = tt.client\_id

JOIN theory\_exam te ON tt.id = te.theory\_ticket\_id

WHERE get\_theory\_exam\_result(te.id) < 18

GROUP BY c.id;

На рисунку 6.5.1 зображено приклад виконання цього запиту.



Рисунок 6.5.1 – Приклад підрахунку кількості невдалих спроб теоретичного іспиту

* Клієнти з активними штрафами

Запит дозволяє отримати клієнтів, які мають невиплачені штрафи, що дозволяє відстежувати активні штрафи для інформування осіб та подальшого контролю.

SELECT c.\*, f.reason, f.sum

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

JOIN fine f ON dl.id = f.driving\_license\_id

WHERE f.status = 'pending';

На рисунку 6.5.2 зображено приклад виконання цього запиту.

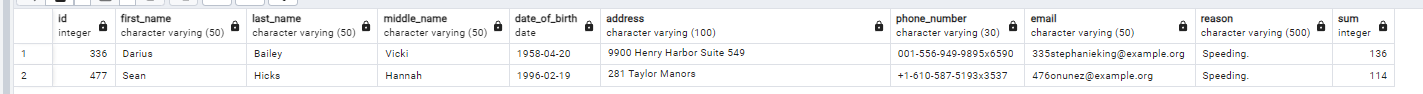


Рисунок 6.5.2 – Приклад пошуку клієнтів з активними штрафами

* Кількість нових категорій за день

Запит визначає кількість нових водійських категорій, які клієнти отримали у поточний день, що може бути використано для аналізу продуктивності сервісних центрів та популярності категорій.

SELECT category, COUNT(\*)

FROM driving\_exam de

JOIN driving\_ticket dt ON de.driving\_ticket\_id = dt.id

WHERE de.exam\_result = 'passed'

AND DATE(dt.datetime) = CURRENT\_DATE

GROUP BY category;

На рисунку 6.5.3 зображено приклад виконання цього запиту.

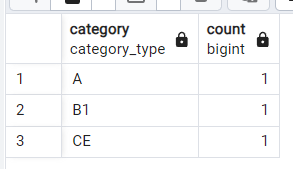


Рисунок 6.5.3 – Приклад підрахунку нових категорій за день

* Клієнти, які отримали посвідчення за останній період

Запит надає дані про клієнтів, які отримали посвідчення водія за останні три місяці, що дозволяє виявляти нових водіїв та аналізувати динаміку за останній період.

SELECT c.first\_name, c.last\_name, dl.start\_date

FROM driving\_license dl

JOIN client c ON dl.client\_id = c.id

WHERE dl.start\_date >= CURRENT\_DATE - INTERVAL '3 months';

На рисунку 6.5.4 зображено приклад виконання цього запиту.

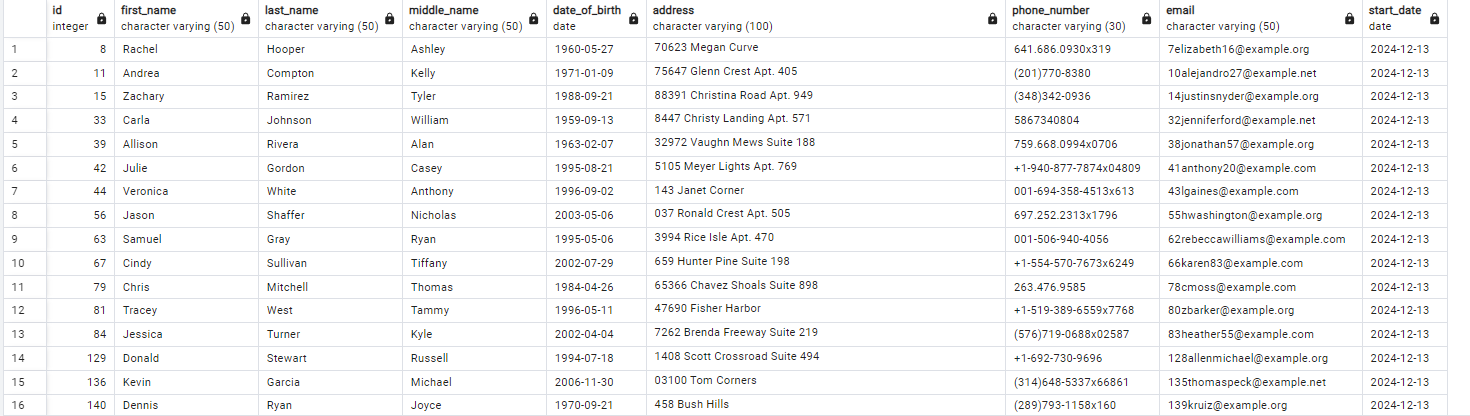


Рисунок 6.5.4 – Приклад отримання нових водіїв та дату початку їх посвідчення

* Рейтинг успішності інспекторів

Запит розраховує рейтинг інспекторів за успішністю проведення практичних іспитів, виражений у відсотках, що дозволяє оцінити якість роботи інспекторів, їх вплив на результати та, можливо, неправомірні дії.

SELECT i.\*, COUNT(CASE WHEN de.exam\_result = 'passed' THEN 1 END) \* 100. / COUNT(\*) AS success\_rate

FROM inspector i

JOIN driving\_exam de ON i.id = de.inspector\_id

GROUP BY i.id

ORDER BY success\_rate DESC;

На рисунку 6.5.5 зображено приклад виконання цього запиту.

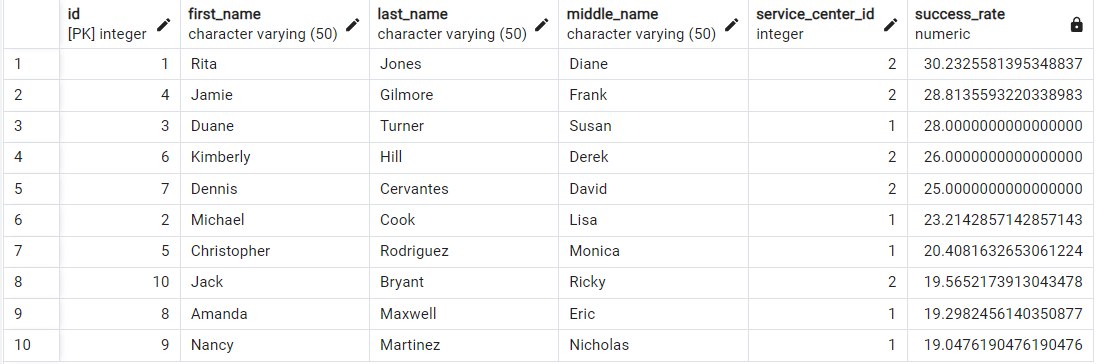


Рисунок 6.5.5 – Приклад успішності практичних іспитів у інспекторів

* Інспектори, які провели менше іспитів, ніж середній показник

Запит визначає інспекторів, кількість проведених іспитів у яких є меншою за середній показник по всіх інспекторах, що допомагає виявити інспекторів, із нижчим навантаженням та оптимізувати їх роботу.

SELECT i.\*, COUNT(\*) AS total\_exams

FROM inspector i

LEFT JOIN driving\_exam de ON i.id = de.inspector\_id

GROUP BY i.id

HAVING COUNT(\*) < (

SELECT AVG(count)

FROM (

SELECT COUNT(\*)

FROM driving\_exam

GROUP BY inspector\_id

) AS subquery

);

На рисунку 6.5.6 зображено приклад виконання цього запиту.

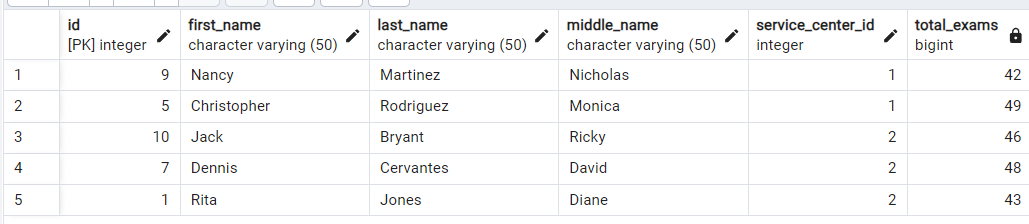


Рисунок 6.5.6 – Приклад інспекторів із меншим навантаженням

* Клієнти, які не з’явилися на запис

Запит отримує кількість неявок клієнтів, які записалися на іспит, але не з’явилися на зазначений час для реєстрації. Це може бути використано для виявлення клієнтів, що заважають ефективній роботі сервісу.

SELECT c.\*, COUNT(\*)

FROM all\_tickets at

JOIN client c ON at.client\_id = c.id

WHERE at.status = 'expired'

GROUP BY c.id;

На рисунку 6.5.7 зображено приклад виконання цього запиту.

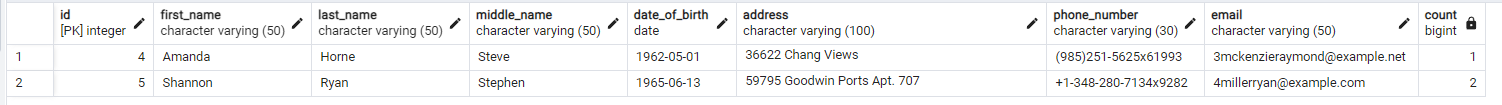


Рисунок 6.5.7 – Приклад клієнтів, що не прийшли за записом

* Клієнти з усіма категоріями посвідчення

Запит отримує клієнтів, які мають всі доступні категорії водійських посвідчень, що дозволяє знаходити найкращих водіїв для можливого подальшого їх працевлаштування.

SELECT c.\*

FROM person\_opened\_categories poc

JOIN client c ON poc.id = c.id

WHERE poc.person\_type = 'client'

GROUP BY c.id

HAVING COUNT(\*) = 14;

На рисунку 6.5.8 зображено приклад виконання цього запиту.

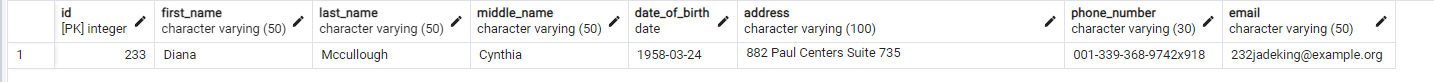


Рисунок 6.5.8 – Приклад клієнта з усіма категоріями

* Клієнти, у яких закінчився період призупинення посвідчення

Запит надає клієнтів, у яких завершився період призупинення водійського посвідчення після його тимчасового вилучення. Це дозволяє знайти відповідних водіїв, проінформувати їх та повернути фізичне посвідчення.

SELECT c.\* FROM client c

WHERE get\_person\_last\_active\_driving\_license('client', c.id) IN (

SELECT dl.id FROM driving\_license dl

JOIN driving\_license\_status dls ON dl.id = dls.driving\_license\_id

WHERE dls.status = 'suspended'

AND dls.end\_date < CURRENT\_DATE );

На рисунку 6.5.9 зображено приклад виконання цього запиту.



Рисунок 6.5.9 – Приклад клієнта, тимчасове вилучення посвідчення якого закінчилось

* Теоретичний матеріал

Запит надає дані про всі можливі тестові питання та їх відповіді, включаючи відмічену правильну відповідь.

SELECT q.text AS question, q.photo\_file\_path AS question\_photo\_file\_path,

a.text AS answer, a.photo\_file\_path AS answer\_photo\_file\_path, a.is\_correct

FROM question q

JOIN answer a ON q.id = a.question\_id;

На рисунку 6.5.10 зображено приклад виконання цього запиту.

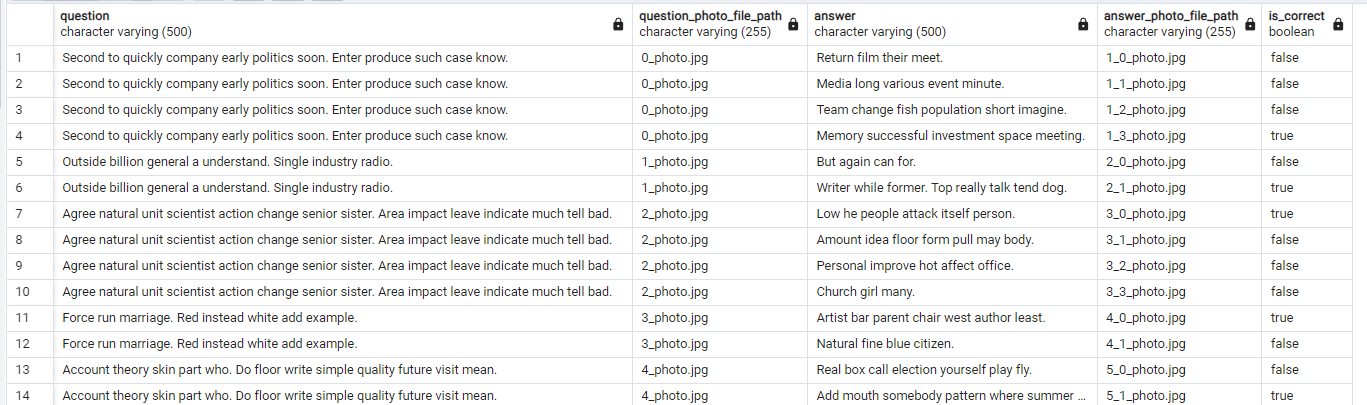


Рисунок 6.5.10 – Приклад теоретичних питань та відповідей на них

* Категорії клієнтів

Запит отримує список клієнтів із зазначенням категорій та типів коробки передач для кожного. Це дозволяє швидко дізнатися, які категорії наразі відкриті у водіїв.

SELECT first\_name, last\_name, middle\_name, category, gearbox

FROM person\_opened\_categories poc

JOIN client c ON poc.id = c.id

WHERE poc.person\_type = 'client'

ORDER BY c;

На рисунку 6.5.11 зображено приклад виконання цього запиту.



Рисунок 6.5.11 – Приклад клієнтів та їх категорій

* Кількість спроб скласти практичний іспит для різних категорій та коробок передач

Запит визначає кількість спроб та успішність клієнтів скласти практичний іспит, враховуючи категорії та коробки передач транспортних засобів. Це дозволяє аналізувати рівень підготовленості клієнтів та виявляти тих, хто потребує додаткового навчання.

SELECT c.\*, dt.category, dt.gearbox, COUNT(\*) AS total\_exams,

CASE WHEN COUNT(CASE WHEN de.exam\_result = 'passed' THEN 1 END) > 0 THEN 'Yes' ELSE 'No' END AS was\_successful

FROM driving\_exam de

JOIN driving\_ticket dt ON de.driving\_ticket\_id = dt.id

JOIN client c ON dt.client\_id = c.id

GROUP BY c.id, dt.category, dt.gearbox;

На рисунку 6.5.12 зображено приклад виконання цього запиту.

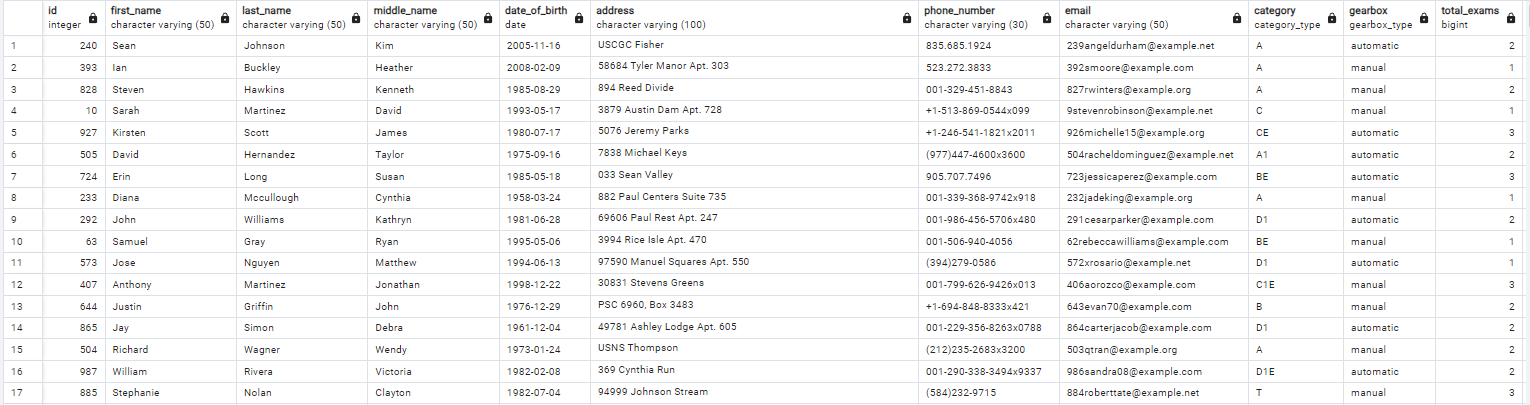


Рисунок 6.5.12 – Приклад клієнтів та їх практичних спроб

* Доступні категорії та коробки передач для сервісних центрів

Запит отримує унікальні комбінації категорій та коробок передач автомобілів, доступних у сервісних центрах для надання до практичного іспиту. Це дозволяє визначити, які категорії, враховуючи коробку передач, доступні до здачі в конкретних центрах.

SELECT DISTINCT c.category, c.gearbox, sc.address, sc.number

FROM car c

JOIN service\_center sc ON c.service\_center\_id = sc.id

WHERE car\_owner = 'service'

ORDER BY number;

На рисунку 6.5.13 зображено приклад виконання цього запиту.

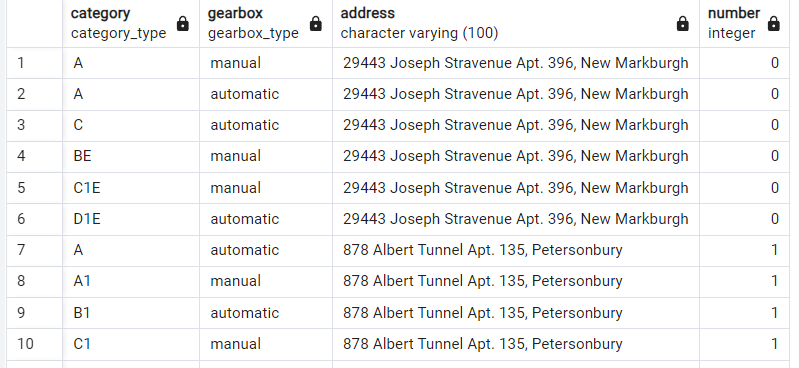


Рисунок 6.5.13 – Приклад категорій та коробок передач, доступних на сервісних центрах

* Найпопулярніша категорія та коробка передач

Запит визначає, які категорії та коробки передач є найбільш популярними серед клієнтів, що складають практичний іспит. Це допомагає організувати роботу сервісного центру, наприклад, збільшити кількість автомобілей.

SELECT category, gearbox, COUNT(\*) AS total

FROM driving\_exam de

JOIN driving\_ticket dt ON de.driving\_ticket\_id = dt.id

GROUP BY category, gearbox

ORDER BY total DESC;

На рисунку 6.5.14 зображено приклад виконання цього запиту.

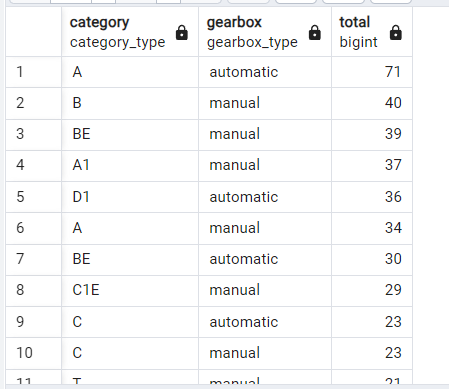


Рисунок 6.5.14 – Приклад категорій, коробок передач та їх популярності

* Небезпечні клієнти

Запит визначає клієнтів, посвідчення яких тимчасово вилучали більше одного разу, що допомагає контролювати небезпечних клієнтів.

SELECT c.\*, COUNT(\*) AS total\_suspensions

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

JOIN driving\_license\_status dls ON dl.id = dls.driving\_license\_id

WHERE dls.status = 'suspended'

GROUP BY c.id

HAVING COUNT(\*) > 1;

На рисунку 6.5.15 зображено приклад виконання цього запиту.

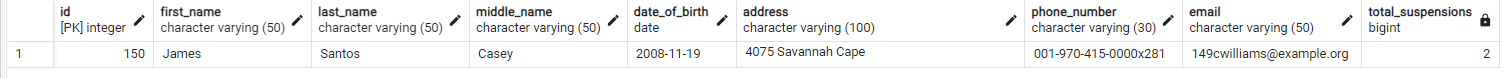


Рисунок 6.5.15 – Приклад клієнта, посвідчення якого вилучалось більше одного разу

* Тільки теоретичні сервісні центри

Запит надає список сервісних центрів, що не мають автомобілів для проведення практичного іспиту, що може вплинути на організацію роботи.

SELECT sc.\*

FROM service\_center sc

WHERE NOT EXISTS (

SELECT 1

FROM car c

WHERE c.service\_center\_id = sc.id

AND c.car\_owner = 'service'

);

На рисунку 6.5.16 зображено приклад виконання цього запиту.

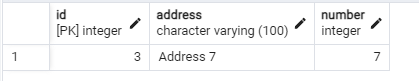


Рисунок 6.5.16 – Приклад сервісного центру без автомобілей для практичного іспиту

* Популярні сервісні центри

Запит визначає популярність сервісних центрів на основі кількості записів, що дозволяє планувати роботу сервісного центру, враховуючи навантаження.

SELECT sc.\*, COUNT(\*) AS popularity

FROM all\_tickets at

JOIN service\_center sc ON at.service\_center\_id = sc.id

GROUP BY sc.id

ORDER BY popularity DESC;

На рисунку 6.5.17 зображено приклад виконання цього запиту.

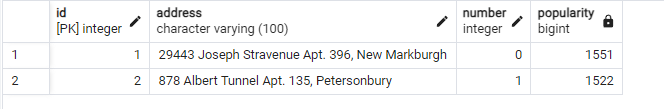


Рисунок 6.5.17 – Приклад завантаженості сервісних центрів

* Клієнти без штрафів

Запит надає список клієнтів, які не мають штрафів, пов’язаних з їх водійськими посвідченням, що дозволяє визначити законослухняних водіїв, які не порушували правила дорожнього руху.

SELECT c.\*

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

WHERE NOT EXISTS (

SELECT 1

FROM fine f

WHERE dl.id = f.driving\_license\_id

);

На рисунку 6.5.18 зображено приклад виконання цього запиту.

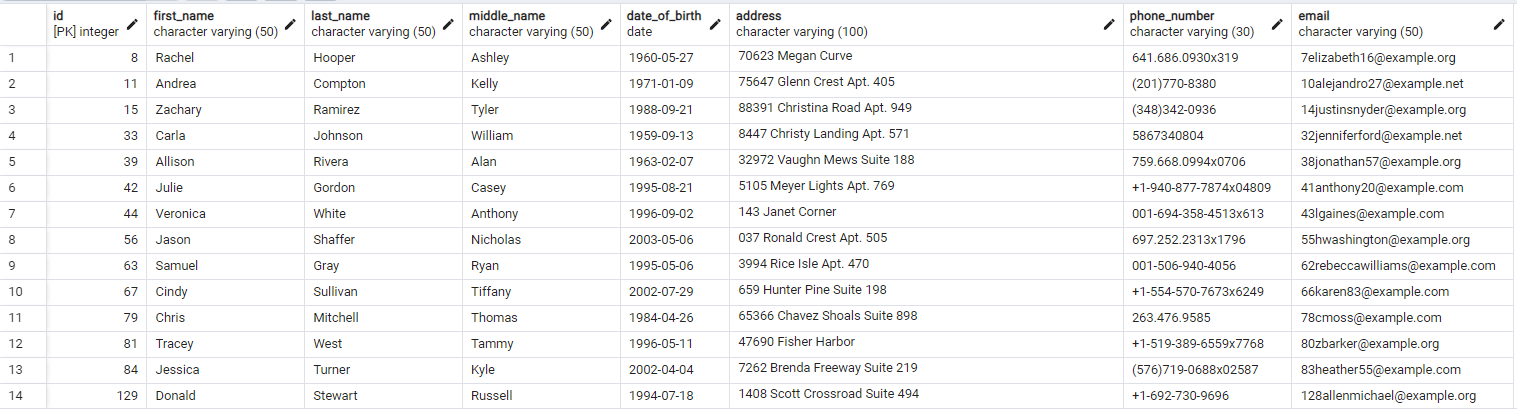


Рисунок 6.5.18 – Приклад клієнтів без штрафів

* Середній результат теоретичних іспитів за роками

Запит підраховує середнє значення результатів теоретичних іспитів для кожного року, що дозволяє аналізувати динаміку підготовленості клієнтів.

SELECT EXTRACT(YEAR FROM tt.datetime) AS year, AVG(get\_theory\_exam\_result(te.id)) AS average\_result

FROM theory\_exam te

JOIN theory\_ticket tt ON te.theory\_ticket\_id = tt.id

GROUP BY year;

На рисунку 6.5.19 зображено приклад виконання цього запиту.

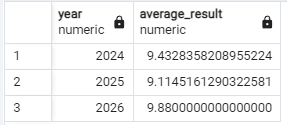


Рисунок 6.5.19 – Приклад середніх результатів теоретичних іспитів за роками

* Клієнти з недійсними документами

Запит виявляє клієнтів, чиї документи вже недійсні. Якщо особа має декілька документів одного типу, то враховується останній документ. Це дозволяє виявляти осіб, які потребують оновлення зараз або при наступній реєстрації.

SELECT c.\*, d.type, MAX(d.end\_date) AS last\_out\_of\_date

FROM client c

JOIN document d ON c.id = d.client\_id

GROUP BY c.id, d.type

HAVING MAX(d.end\_date) < CURRENT\_DATE;

На рисунку 6.5.20 зображено приклад виконання цього запиту.



Рисунок 6.5.20 – Приклад клієнтів з недійсними документами

6.6 Оптимізація

Для оптимізації швидкодії бази даних було обрано індексацію. Правильні індекси дозволяють значно зменшити час виконання пошукових запитів, особливо для великих таблиць.

Оскільки СУБД PostgreSQL автоматично створила основні індекси на унікальних полях, первинних та вторинних ключах, то нам залишилось створити більш специфічну індексацію. Тому створимо індекси для колонок, які в теорії можуть бути доволі часто використані в запитах: час іспитів, коробка передач, категорія та результат практичних іспитів, тип документу, статус штрафу та посвідчення водія, кінцева дата посвідчення.

CREATE INDEX theory\_ticket\_datetime\_index ON theory\_ticket(datetime);

CREATE INDEX driving\_ticket\_datetime\_gearbox\_category\_index ON driving\_ticket(datetime, gearbox, category);

CREATE INDEX driving\_exam\_exam\_result\_index ON driving\_exam(exam\_result);

CREATE INDEX document\_type\_index ON document(type);

CREATE INDEX fine\_status\_index ON fine(status);

CREATE INDEX driving\_license\_status\_index ON driving\_license\_status(status);

CREATE INDEX driving\_license\_end\_date\_index ON driving\_license(end\_date);

Проведемо тестування на комплексі різних запитів, оптимізованих за допомогою індексів. На рисунку 6.6.1 зображено час виконання до введення індексації. Після її введення час значно зменшився (майже в 10 разів), що можна побачити на рисунку 6.6.2.



Рисунок 6.6.1 – Комплексне тестування до індексації

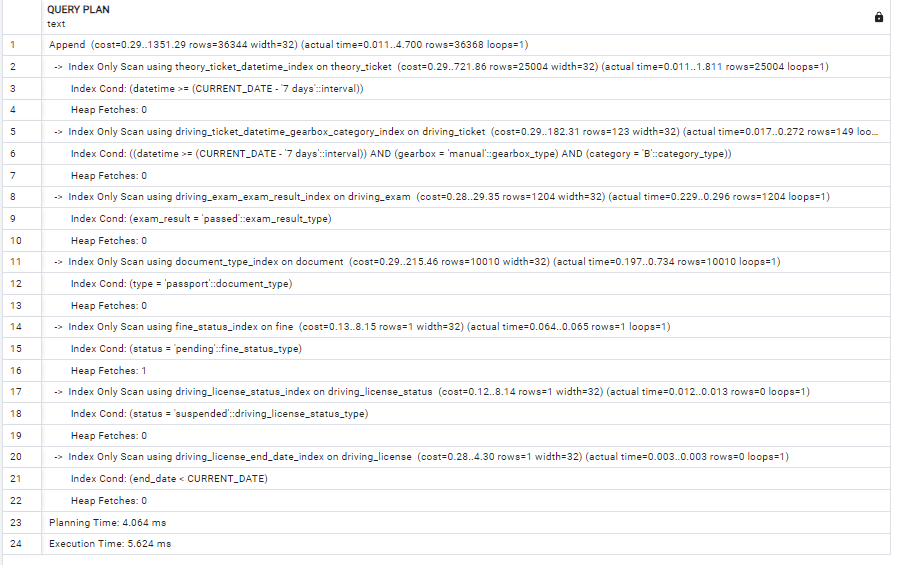


Рисунок 6.6.2 – Комплексне тестування після індексації

ВИСНОВКИ

У ході виконання курсової роботи було проведено аналіз предметного середовища для розробки бази даних для підтримки діяльності сервісного центру МВС в частині прийому іспитів та видачі посвідчень. Описано об'єкт дослідження, визначено вхідні та вихідні дані, а також основні бізнес-процеси, що потребують автоматизації в роботі сервісного центру.

Далі було проведено аналіз існуючих програмних продуктів у сфері надання аналогічних послуг. Це дозволило визначити та узагальнити найкращі практики і можливості, які можуть бути застосовані під час розробки бази даних.

Після цього було сформульовано завдання та визначено основні вимоги до розробки бази даних, що необхідно було реалізувати.

Продумавши загальну структуру, було побудовано ER-модель, у якій визначено бізнес-правила, сутності, атрибути та зв’язки між об'єктами. Ця модель стала основою для реалізації справжньої бази даних.

Тепер було реалізовано базу даних, використовуючи систему управління PostgreSQL. Детально описано структуру таблиць, методи забезпечення цілісності даних, включаючи первинні та зовнішні ключі, обмеження, а також побудовано схему бази даних засобами СУБД.

Далі було розглянуто процес роботи з базою даних. Розроблено представлення для спрощення доступу до даних, реалізовано функції та процедури для автоматизації типових операцій, створено тригери для забезпечення цілістності даних та бізнес-правил і остаточно було сформовано SQL-запити для отримання необхідної інформації для функціонування сервісу.

На завершальному етапі проведено оптимізацію бази даних: створено індекси, що дозволило значно покращити продуктивність системи при роботі з великими обсягами даних.

Таким чином, у результаті курсової роботи було створено та оптимізовано базу даних для сервісного центру МВС, яка відповідає поставленим вимогам, забезпечує надійність, швидкість і зручність роботи. Реалізована система сприятиме ефективній роботі працівників сервісного центру та підвищенню якості обслуговування громадян.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Головний сервісний центр МВС. URL: <https://hsc.gov.ua/> (дата звернення: 16.12.2024).
2. DMV USA. URL: <https://www.dmvusa.com/> (дата звернення: 16.12.2024).
3. WORD Warszawa.URL: <https://www.word.waw.pl/> (дата звернення: 16.12.2024).
4. PostgreSQL. URL: <https://www.postgresql.org/> (дата звернення: 16.12.2024).
5. PostgreSQL advantages. URL: <https://www.prisma.io/dataguide/postgresql/benefits-of-postgresql> (дата звернення: 16.12.2024).

ДОДАТОК А ТЕКСТИ SQL-СКРИПТІВ

**Створення бази даних**

CREATE DATABASE mia;

**Створення таблиць**

CREATE TYPE document\_type AS ENUM ('passport', 'medical\_certificate', 'registration\_of\_residence', 'autoschool\_certificate', 'inspector\_certificate');

CREATE TYPE gearbox\_type AS ENUM ('manual', 'automatic');

CREATE TYPE car\_owner\_type AS ENUM ('service', 'autoschool');

CREATE TYPE exam\_result\_type AS ENUM ('pending', 'passed', 'failed');

CREATE TYPE fine\_status\_type AS ENUM ('pending', 'paid');

CREATE TYPE driving\_license\_status\_type AS ENUM ('suspended', 'revoked');

CREATE TYPE ticket\_status\_type AS ENUM ('pending', 'used', 'cancelled', 'expired');

CREATE TYPE category\_type AS ENUM ('A', 'A1', 'B', 'B1', 'C', 'C1', 'D', 'D1', 'T', 'BE', 'C1E', 'CE', 'D1E', 'DE');

CREATE TABLE client (

    id SERIAL PRIMARY KEY,

    first\_name VARCHAR(50) NOT NULL,

    last\_name VARCHAR(50) NOT NULL,

    middle\_name VARCHAR(50) NOT NULL,

    date\_of\_birth DATE NOT NULL CHECK (date\_of\_birth < CURRENT\_DATE),

    address VARCHAR(100) NOT NULL,

    phone\_number VARCHAR(30) NOT NULL UNIQUE,

    email VARCHAR(50) NOT NULL UNIQUE

);

CREATE TABLE service\_center (

    id SERIAL PRIMARY KEY,

    address VARCHAR(100) NOT NULL UNIQUE,

    number INT NOT NULL UNIQUE

);

CREATE TABLE inspector (

    id SERIAL PRIMARY KEY,

    first\_name VARCHAR(50) NOT NULL,

    last\_name VARCHAR(50) NOT NULL,

    middle\_name VARCHAR(50) NOT NULL,

    service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE

);

CREATE TABLE document (

    id SERIAL PRIMARY KEY,

    type document\_type NOT NULL,

    end\_date DATE NOT NULL CHECK (end\_date > CURRENT\_DATE),

    info\_file\_path VARCHAR(255) NOT NULL UNIQUE,

    client\_id INT REFERENCES client(id) ON DELETE CASCADE,

    inspector\_id INT REFERENCES inspector(id) ON DELETE CASCADE,

    CHECK (client\_id IS NOT NULL OR inspector\_id IS NOT NULL)

);

CREATE TABLE car (

    id SERIAL PRIMARY KEY,

    car\_owner car\_owner\_type NOT NULL,

    gearbox gearbox\_type NOT NULL,

    license\_plate VARCHAR(10) NOT NULL UNIQUE,

    category category\_type NOT NULL,

    service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE

);

CREATE TABLE theory\_ticket (

    id SERIAL PRIMARY KEY,

    datetime TIMESTAMP NOT NULL,

    status ticket\_status\_type NOT NULL DEFAULT 'pending',

    service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE,

    client\_id INT NOT NULL REFERENCES client(id) ON DELETE CASCADE

);

CREATE TABLE driving\_ticket (

    id SERIAL PRIMARY KEY,

    datetime TIMESTAMP NOT NULL,

    status ticket\_status\_type NOT NULL DEFAULT 'pending',

    gearbox gearbox\_type NOT NULL,

    category category\_type NOT NULL,

    car\_owner car\_owner\_type NOT NULL,

    service\_center\_id INT NOT NULL REFERENCES service\_center(id) ON DELETE CASCADE,

    client\_id INT NOT NULL REFERENCES client(id) ON DELETE CASCADE

);

CREATE TABLE theory\_exam (

    id SERIAL PRIMARY KEY,

    theory\_ticket\_id INT NOT NULL UNIQUE REFERENCES theory\_ticket(id) ON DELETE CASCADE

);

CREATE TABLE question (

    id SERIAL PRIMARY KEY,

    text VARCHAR(500) NOT NULL UNIQUE,

    photo\_file\_path VARCHAR(255) UNIQUE

);

CREATE TABLE answer (

    id SERIAL PRIMARY KEY,

    text VARCHAR(500) NOT NULL,

    photo\_file\_path VARCHAR(255) UNIQUE,

    is\_correct BOOLEAN NOT NULL,

    question\_id INT NOT NULL REFERENCES question(id) ON DELETE CASCADE,

    UNIQUE (question\_id, text)

);

CREATE TABLE theory\_exam\_result (

    id SERIAL PRIMARY KEY,

    theory\_exam\_id INT NOT NULL REFERENCES theory\_exam(id) ON DELETE CASCADE,

    question\_id INT NOT NULL REFERENCES question(id) ON DELETE CASCADE,

    selected\_answer\_id INT NOT NULL REFERENCES answer(id) ON DELETE CASCADE,

    UNIQUE (theory\_exam\_id, question\_id)

);

CREATE TABLE driving\_exam (

    id SERIAL PRIMARY KEY,

    exam\_result exam\_result\_type NOT NULL DEFAULT 'pending',

    video\_file\_path VARCHAR(255) UNIQUE,

    driving\_ticket\_id INT NOT NULL UNIQUE REFERENCES driving\_ticket(id) ON DELETE CASCADE,

    inspector\_id INT NOT NULL REFERENCES inspector(id) ON DELETE CASCADE,

    car\_id INT REFERENCES car(id) ON DELETE CASCADE

);

CREATE TABLE driving\_license (

    id SERIAL PRIMARY KEY,

    start\_date DATE DEFAULT CURRENT\_DATE CHECK (start\_date = CURRENT\_DATE),

    end\_date DATE NOT NULL,

    client\_id INT REFERENCES client(id) ON DELETE CASCADE,

    inspector\_id INT REFERENCES inspector(id) ON DELETE CASCADE,

    CHECK (

        (inspector\_id IS NOT NULL AND client\_id IS NULL) OR

        (inspector\_id IS NULL AND client\_id IS NOT NULL)

    ),

    CHECK (end\_date > start\_date)

);

CREATE TABLE driving\_license\_category (

    id SERIAL PRIMARY KEY,

    category category\_type NOT NULL,

    gearbox gearbox\_type NOT NULL,

    driving\_license\_id INT NOT NULL REFERENCES driving\_license(id) ON DELETE CASCADE,

    UNIQUE (category, gearbox, driving\_license\_id)

);

CREATE TABLE fine (

    id SERIAL PRIMARY KEY,

    status fine\_status\_type NOT NULL DEFAULT 'pending',

    issued\_date DATE DEFAULT CURRENT\_DATE CHECK (issued\_date = CURRENT\_DATE),

    reason VARCHAR(500) NOT NULL,

    sum INT NOT NULL CHECK (sum > 0),

    driving\_license\_id INT NOT NULL REFERENCES driving\_license(id) ON DELETE CASCADE

);

CREATE TABLE driving\_license\_status (

    id SERIAL PRIMARY KEY,

    status driving\_license\_status\_type NOT NULL,

    start\_date DATE DEFAULT CURRENT\_DATE CHECK (start\_date = CURRENT\_DATE),

    end\_date DATE,

    reason VARCHAR(500) NOT NULL,

    driving\_license\_id INT NOT NULL REFERENCES driving\_license(id) ON DELETE CASCADE,

    CHECK (end\_date IS NULL OR end\_date > start\_date)

);

**Створення індексів**

CREATE INDEX theory\_ticket\_datetime\_index ON theory\_ticket(datetime);

CREATE INDEX driving\_ticket\_datetime\_gearbox\_category\_index ON driving\_ticket(datetime, gearbox, category);

CREATE INDEX driving\_exam\_exam\_result\_index ON driving\_exam(exam\_result);

CREATE INDEX document\_type\_index ON document(type);

CREATE INDEX fine\_status\_index ON fine(status);

CREATE INDEX driving\_license\_status\_index ON driving\_license\_status(status);

CREATE INDEX driving\_license\_end\_date\_index ON driving\_license(end\_date);

**Створення представлень**

Відкрити категорії осіб

CREATE OR REPLACE VIEW person\_opened\_categories AS

SELECT c.id, 'client' AS person\_type, dlc.category, dlc.gearbox

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

JOIN driving\_license\_category dlc ON dl.id = dlc.driving\_license\_id

WHERE dl.end\_date > CURRENT\_DATE

UNION

SELECT i.id, 'inspector' AS person\_type, dlc.category, dlc.gearbox

FROM inspector i

JOIN driving\_license dl ON i.id = dl.inspector\_id

JOIN driving\_license\_category dlc ON dl.id = dlc.driving\_license\_id

WHERE dl.end\_date > CURRENT\_DATE;

Всі талони

CREATE OR REPLACE VIEW all\_tickets AS

SELECT dt.id, dt.client\_id, dt.datetime, dt.status, dt.service\_center\_id, 'driving' AS ticket\_type

FROM driving\_ticket dt

UNION ALL

SELECT tt.id, tt.client\_id, tt.datetime, tt.status, tt.service\_center\_id, 'theory' AS ticket\_type

FROM theory\_ticket tt;

Результати відповідей клієнтів на теоретичних іспитах

CREATE OR REPLACE VIEW theory\_exam\_result\_info AS

SELECT tt.datetime, c.id AS client\_id, ter.theory\_exam\_id, q.text AS question, a.text AS answer, a.is\_correct

FROM theory\_exam\_result ter

JOIN question q ON ter.question\_id = q.id

JOIN answer a ON ter.selected\_answer\_id = a.id

JOIN theory\_exam te ON ter.theory\_exam\_id = te.id

JOIN theory\_ticket tt ON te.theory\_ticket\_id = tt.id

JOIN client c ON tt.client\_id = c.id;

**Створення функцій**

Вік клієнта

CREATE OR REPLACE FUNCTION get\_client\_age(client\_id INT)

RETURNS INT AS $$

BEGIN

    RETURN (

        SELECT EXTRACT(YEAR FROM AGE(CURRENT\_DATE, date\_of\_birth))

        FROM client

        WHERE id = $1

    );

END;

$$ LANGUAGE plpgsql;

Перевірка, чи має клієнт активний талон

CREATE OR REPLACE FUNCTION client\_has\_active\_ticket(client\_id INT)

RETURNS BOOLEAN AS $$

BEGIN

    RETURN EXISTS (

        SELECT 1

        FROM all\_tickets at

        WHERE at.client\_id = $1

            AND status = 'pending'

    );

END;

$$ LANGUAGE plpgsql;

Результат теоретичного іспиту

CREATE OR REPLACE FUNCTION get\_theory\_exam\_result(theory\_exam\_id INT)

RETURNS INT AS $$

BEGIN

    RETURN (

        SELECT COUNT(\*)

        FROM theory\_exam\_result\_info teri

        WHERE teri.theory\_exam\_id = $1

            AND teri.is\_correct = TRUE

    );

END;

$$ LANGUAGE plpgsql;

Перевірка клієнта на наявність активного теоретичного іспиту

CREATE OR REPLACE FUNCTION client\_has\_active\_passed\_theory\_exam(client\_id INT)

RETURNS BOOLEAN AS $$

BEGIN

    RETURN EXISTS (

        SELECT 1

        FROM theory\_exam te

        JOIN theory\_ticket tt ON te.theory\_ticket\_id = tt.id

        WHERE tt.client\_id = $1

            AND get\_theory\_exam\_result(te.id) >= 18

            AND CURRENT\_DATE - INTERVAL '2 years' < tt.datetime

    );

END;

$$ LANGUAGE plpgsql;

Перевірка клієнта на активний документ

CREATE OR REPLACE FUNCTION person\_has\_active\_document(person\_type VARCHAR(255), person\_id INT, doc\_type document\_type)

RETURNS BOOLEAN AS $$

BEGIN

    IF person\_type = 'client' THEN

        RETURN EXISTS (

            SELECT 1

            FROM document d

            WHERE d.client\_id = $2

                AND d.type = $3

                AND end\_date > CURRENT\_DATE

        );

    ELSIF person\_type = 'inspector' THEN

        RETURN EXISTS (

            SELECT 1

            FROM document d

            WHERE d.inspector\_id = $2

                AND d.type = $3

                AND end\_date > CURRENT\_DATE

        );

    END IF;

END;

$$ LANGUAGE plpgsql;

Отримання останнього активного посвідчення особи

CREATE OR REPLACE FUNCTION get\_person\_last\_active\_driving\_license(person\_type VARCHAR(255), person\_id INT)

RETURNS INT AS $$

DECLARE

    last\_driving\_license\_id INT;

BEGIN

    IF $1 = 'client' THEN

        SELECT MAX(dl.id) INTO last\_driving\_license\_id

        FROM driving\_license dl

        WHERE dl.client\_id = $2

            AND dl.end\_date > CURRENT\_DATE

            AND NOT EXISTS (

                SELECT 1

                FROM driving\_license\_status dls

                WHERE dls.driving\_license\_id = dl.id

                    AND dls.status = 'revoked'

            );

    ELSE

        SELECT MAX(dl.id) INTO last\_driving\_license\_id

        FROM driving\_license dl

        WHERE dl.inspector\_id = $2

            AND dl.end\_date > CURRENT\_DATE

            AND NOT EXISTS (

                SELECT 1

                FROM driving\_license\_status dls

                WHERE dls.driving\_license\_id = dl.id

                    AND dls.status = 'revoked'

            );

    END IF;

    RETURN last\_driving\_license\_id;

END;

$$ LANGUAGE plpgsql;

Отримання кількості квитків у певний день

CREATE OR REPLACE FUNCTION get\_tickets\_count\_for\_day(day DATE, service\_center\_id INT)

RETURNS TABLE(theory\_tickets\_count BIGINT, driving\_tickets\_count BIGINT) AS $$

BEGIN

    RETURN QUERY SELECT

        COUNT(CASE WHEN at.ticket\_type = 'theory' THEN 1 END) AS theory\_tickets\_count, COUNT(CASE WHEN at.ticket\_type = 'driving' THEN 1 END) AS driving\_tickets\_count

        FROM all\_tickets at

        WHERE DATE(at.datetime) = $1

            AND at.service\_center\_id = $2;

END;

$$ LANGUAGE plpgsql;

Отримання доступних інспекторів для практичного іспиту

CREATE OR REPLACE FUNCTION get\_appropriate\_inspectors\_for\_driving\_exam(driving\_ticket\_id INT)

RETURNS TABLE(inspector\_id INT, inspector\_name TEXT) AS $$

BEGIN

    RETURN QUERY SELECT

        i.id, i.first\_name || ' ' || i.last\_name || ' ' || i.middle\_name

        FROM inspector i

        JOIN service\_center sc ON i.service\_center\_id = sc.id

        JOIN driving\_ticket dt ON sc.id = dt.service\_center\_id

        JOIN person\_opened\_categories poc ON i.id = poc.id

        WHERE dt.id = $1

            AND poc.person\_type = 'inspector'

            AND poc.category = dt.category

            AND poc.gearbox = dt.gearbox

            AND person\_has\_active\_document('inspector', i.id, 'inspector\_certificate');

END;

$$ LANGUAGE plpgsql;

Отримання нових питань для теоретичного іспиту клієнта

CREATE OR REPLACE FUNCTION get\_new\_theory\_exam\_questions(client\_id INT)

RETURNS TABLE(question\_id INT, question\_text VARCHAR(500), photo\_file\_path VARCHAR(255)) AS $$

BEGIN

    DROP TABLE IF EXISTS left\_questions;

    CREATE TEMP TABLE left\_questions AS

    SELECT q.id, q.text, q.photo\_file\_path

    FROM question q

    WHERE q.text NOT IN (

        SELECT question

        FROM theory\_exam\_result\_info teri

        WHERE teri.client\_id = $1

    )

    ORDER BY RANDOM()

    LIMIT 20;

    IF (SELECT COUNT(\*) FROM left\_questions) = 20 THEN

        RETURN QUERY SELECT \* FROM left\_questions;

    ELSE

        RETURN QUERY SELECT q.id, q.text, q.photo\_file\_path

            FROM question q

            ORDER BY RANDOM()

            LIMIT 20;

    END IF;

END;

$$ LANGUAGE plpgsql;

**Створення процедур**

Оновлення статусу прострочених квитків

CREATE OR REPLACE PROCEDURE update\_expired\_tickets\_status()

AS $$

BEGIN

    UPDATE theory\_ticket

    SET status = 'expired'

    WHERE datetime < CURRENT\_TIMESTAMP - INTERVAL '1 day'

      AND status = 'pending';

    UPDATE driving\_ticket

    SET status = 'expired'

    WHERE datetime < CURRENT\_TIMESTAMP - INTERVAL '1 day'

      AND status = 'pending';

END;

$$ LANGUAGE plpgsql;

Подвоєння суми неоплачених штрафів після 2 тижнів

CREATE OR REPLACE PROCEDURE double\_pending\_fines()

AS $$

BEGIN

    UPDATE fine

    SET sum = sum \* 2

    WHERE status = 'pending'

      AND issued\_date < CURRENT\_DATE - INTERVAL '2 week';

END;

$$ LANGUAGE plpgsql;

**Створення тригерів**

Перевірка адекватності часу талонів

CREATE OR REPLACE FUNCTION check\_ticket\_datetime()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_TABLE\_NAME = 'theory\_ticket' AND NEW.datetime < CURRENT\_TIMESTAMP THEN

        RAISE EXCEPTION 'Theory ticket datetime cannot be in the past';

    ELSIF TG\_TABLE\_NAME = 'driving\_ticket' AND NEW.datetime < CURRENT\_TIMESTAMP THEN

        RAISE EXCEPTION 'Driving ticket datetime cannot be in the past';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_theory\_ticket\_datetime\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ticket\_datetime();

CREATE TRIGGER check\_driving\_ticket\_datetime\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ticket\_datetime();

Перевірка віку клієнта для практичного іспиту

CREATE OR REPLACE FUNCTION check\_client\_age()

RETURNS TRIGGER AS $$

BEGIN

    IF NEW.category IN ('A', 'A1') AND get\_client\_age(NEW.client\_id) < 16 THEN

        RAISE EXCEPTION 'Client must be at least 24 years old to get category A driving ticket';

    ELSIF NEW.category IN ('B', 'B1') AND get\_client\_age(NEW.client\_id) < 18 THEN

        RAISE EXCEPTION 'Client must be at least 18 years old to get a driving ticket';

    ELSIF NEW.category IN ('BE', 'CE', 'C1E') AND get\_client\_age(NEW.client\_id) < 19 THEN

        RAISE EXCEPTION 'Client must be at least 19 years old to get category BE, CE or C1E driving ticket';

    ELSIF NEW.category IN ('D', 'D1', 'DE', 'D1E', 'T') AND get\_client\_age(NEW.client\_id) < 21 THEN

        RAISE EXCEPTION 'Client must be at least 21 years old to get category D1, D or T driving ticket';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_age\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_age();

Перевірка, що клієнт отримує нову категорію, яку ще немає

CREATE OR REPLACE FUNCTION check\_client\_gets\_new\_category()

RETURNS TRIGGER AS $$

BEGIN

    IF EXISTS (

        SELECT 1

        FROM person\_opened\_categories

        WHERE person\_type = 'client'

            AND id = NEW.client\_id

            AND category = NEW.category

            AND gearbox = NEW.gearbox

    ) THEN

        RAISE EXCEPTION 'Client already has the specified category and gearbox';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_gets\_new\_category\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_gets\_new\_category();

Перевірка наявності необхідних документів клієнта для іспитів

CREATE OR REPLACE FUNCTION check\_client\_has\_nessesary\_documents()

RETURNS TRIGGER AS $$

BEGIN

    IF NOT person\_has\_active\_document('client', NEW.client\_id, 'passport') OR NOT person\_has\_active\_document('client', NEW.client\_id, 'registration\_of\_residence') THEN

        RAISE EXCEPTION 'Client must have a passport and a registration of residence';

    ELSIF TG\_TABLE\_NAME = 'driving\_ticket' AND (NOT person\_has\_active\_document('client', NEW.client\_id, 'medical\_certificate') OR NOT person\_has\_active\_document('client', NEW.client\_id, 'autoschool\_certificate')) THEN

        RAISE EXCEPTION 'Client must have a medical certificate and an autoschool certificate';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_has\_nessesary\_documents\_theory\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_has\_nessesary\_documents();

CREATE TRIGGER check\_client\_has\_nessesary\_documents\_driving\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_has\_nessesary\_documents();

Перевірка тільки одного активного квитка у клієнта

CREATE OR REPLACE FUNCTION check\_only\_one\_active\_ticket()

RETURNS TRIGGER AS $$

BEGIN

    IF client\_has\_active\_ticket(NEW.client\_id) THEN

        RAISE EXCEPTION 'Client can have only one active ticket';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_only\_one\_active\_theory\_ticket\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_only\_one\_active\_ticket();

CREATE TRIGGER check\_only\_one\_active\_driving\_ticket\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_only\_one\_active\_ticket();

Перевірка 10 днів між квитками клієнта

CREATE OR REPLACE FUNCTION check\_ten\_days\_between\_tickets()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_TABLE\_NAME = 'theory\_ticket' AND EXISTS (

        SELECT 1

        FROM theory\_ticket

        WHERE client\_id = NEW.client\_id

          AND datetime > NEW.datetime - INTERVAL '10 days'

          AND status != 'cancelled'

    ) THEN

        RAISE EXCEPTION 'Client must wait at least 10 days between theory tickets';

    ELSIF TG\_TABLE\_NAME = 'driving\_ticket' AND EXISTS (

        SELECT 1

        FROM driving\_ticket

        WHERE client\_id = NEW.client\_id

          AND datetime > NEW.datetime - INTERVAL '10 days'

          AND status != 'cancelled'

    ) THEN

        RAISE EXCEPTION 'Client must wait at least 10 days between driving tickets';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_ten\_days\_between\_theory\_tickets\_trigger

BEFORE INSERT ON theory\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ten\_days\_between\_tickets();

CREATE TRIGGER check\_ten\_days\_between\_driving\_tickets\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_ten\_days\_between\_tickets();

Перевірка активного теоретичного іспиту у клієнта перед практичним

CREATE OR REPLACE FUNCTION check\_client\_has\_active\_passed\_theory\_exam()

RETURNS TRIGGER AS $$

BEGIN

    IF NOT client\_has\_active\_passed\_theory\_exam(NEW.client\_id) THEN

        RAISE EXCEPTION 'Client must have passed the theory exam';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_client\_has\_active\_passed\_theory\_exam\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_client\_has\_active\_passed\_theory\_exam();

Оновлення статусу квитка на «використний» після створення іспиту

CREATE OR REPLACE FUNCTION update\_ticket\_status\_after\_exam\_creation()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_TABLE\_NAME = 'theory\_exam' THEN

        UPDATE theory\_ticket

        SET status = 'used'

        WHERE id = NEW.theory\_ticket\_id;

    ELSIF TG\_TABLE\_NAME = 'driving\_exam' THEN

        UPDATE driving\_ticket

        SET status = 'used'

        WHERE id = NEW.driving\_ticket\_id;

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_theory\_ticket\_status\_trigger

AFTER INSERT ON theory\_exam

FOR EACH ROW

EXECUTE FUNCTION update\_ticket\_status\_after\_exam\_creation();

CREATE TRIGGER update\_driving\_ticket\_status\_trigger

AFTER INSERT ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION update\_ticket\_status\_after\_exam\_creation();

Перевірка доступності відповідного автомобіля в сервісному центрі

CREATE OR REPLACE FUNCTION check\_car\_availability()

RETURNS TRIGGER AS $$

BEGIN

    IF NEW.car\_owner = 'service' AND NOT EXISTS (

        SELECT 1

        FROM car

        WHERE service\_center\_id = NEW.service\_center\_id

          AND gearbox = NEW.gearbox

          AND category = NEW.category

          AND car\_owner = 'service'

    ) THEN

        RAISE EXCEPTION 'No available car with the specified gearbox and category at the service center';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_car\_availability\_trigger

BEFORE INSERT ON driving\_ticket

FOR EACH ROW

EXECUTE FUNCTION check\_car\_availability();

Перевірка відповідності автомобіля для іспиту

CREATE OR REPLACE FUNCTION check\_car\_is\_appropriate()

RETURNS TRIGGER AS $$

DECLARE

    driving\_ticket\_category VARCHAR(255);

    driving\_ticket\_gearbox VARCHAR(255);

    car\_category VARCHAR(255);

    car\_gearbox VARCHAR(255);

BEGIN

    IF NEW.car\_id IS NOT NULL THEN

        SELECT category, gearbox INTO driving\_ticket\_category, driving\_ticket\_gearbox

        FROM driving\_ticket

        WHERE id = NEW.driving\_ticket\_id;

        SELECT category, gearbox INTO car\_category, car\_gearbox

        FROM car

        WHERE id = NEW.car\_id;

        IF driving\_ticket\_category != car\_category OR driving\_ticket\_gearbox != car\_gearbox THEN

            RAISE EXCEPTION 'Car is not appropriate for the driving ticket';

        END IF;

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_car\_is\_appropriate\_trigger

BEFORE INSERT OR UPDATE ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION check\_car\_is\_appropriate();

Перевірка відповідності інспектора для практичного іспиту

CREATE OR REPLACE FUNCTION check\_inspector\_capability()

RETURNS TRIGGER AS $$

BEGIN

    IF NOT EXISTS (

        SELECT 1

        FROM person\_opened\_categories

        WHERE person\_type = 'inspector'

            AND id = NEW.inspector\_id

            AND category = (SELECT category FROM driving\_ticket dt WHERE dt.id = NEW.driving\_ticket\_id)

            AND gearbox = (SELECT gearbox FROM driving\_ticket dt WHERE dt.id = NEW.driving\_ticket\_id)

    ) THEN

        RAISE EXCEPTION 'Inspector does not have the necessary category';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_inspector\_capability\_trigger

BEFORE INSERT ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION check\_inspector\_capability();

Перевірка необхідних документів інспектора

CREATE OR REPLACE FUNCTION check\_inspector\_has\_necessary\_documents()

RETURNS TRIGGER AS $$

BEGIN

    IF NOT person\_has\_active\_document('inspector', NEW.inspector\_id, 'inspector\_certificate') THEN

        RAISE EXCEPTION 'Inspector must have an inspector certificate';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_inspector\_has\_neccecery\_documents\_trigger

BEFORE INSERT ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION check\_inspector\_has\_necessary\_documents();

Додавання категорії після успішого складеного практичного іспиту

CREATE OR REPLACE FUNCTION create\_driving\_license\_or\_add\_category\_after\_driving\_exam()

RETURNS TRIGGER AS $$

DECLARE

    client\_id INT;

    last\_active\_driving\_license\_id INT;

BEGIN

    IF NEW.exam\_result != 'passed' THEN

        RETURN NEW;

    END IF;

    SELECT dt.client\_id INTO client\_id

    FROM driving\_ticket dt

    WHERE id = NEW.driving\_ticket\_id;

    last\_active\_driving\_license\_id = get\_person\_last\_active\_driving\_license('client', client\_id);

    IF last\_active\_driving\_license\_id IS NULL THEN

        INSERT INTO driving\_license (end\_date, client\_id)

        VALUES (CURRENT\_DATE + INTERVAL '2 year', client\_id);

        last\_active\_driving\_license\_id = get\_person\_last\_active\_driving\_license('client', client\_id);

    END IF;

    INSERT INTO driving\_license\_category (category, gearbox, driving\_license\_id)

    VALUES ((SELECT category FROM driving\_ticket WHERE id = NEW.driving\_ticket\_id), (SELECT gearbox FROM driving\_ticket WHERE id = NEW.driving\_ticket\_id), last\_active\_driving\_license\_id);

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER create\_driving\_license\_or\_add\_category\_after\_driving\_exam\_trig

AFTER INSERT OR UPDATE ON driving\_exam

FOR EACH ROW

EXECUTE FUNCTION create\_driving\_license\_or\_add\_category\_after\_driving\_exam();

**Створення Select-запитів**

Клієнти без штрафів

SELECT c.\*

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

WHERE NOT EXISTS (

    SELECT 1

    FROM fine f

    WHERE dl.id = f.driving\_license\_id

);

Невдалі теоретичні спроби клієнтів

SELECT c.\*, COUNT(\*) AS attempts

FROM client c

JOIN theory\_ticket tt ON c.id = tt.client\_id

JOIN theory\_exam te ON tt.id = te.theory\_ticket\_id

WHERE get\_theory\_exam\_result(te.id) < 18

GROUP BY c.id;

Клієнти з активними штрафами

SELECT c.\*, f.reason, f.sum

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

JOIN fine f ON dl.id = f.driving\_license\_id

WHERE f.status = 'pending';

Кількість нових категорій за день

SELECT category, COUNT(\*)

FROM driving\_exam de

JOIN driving\_ticket dt ON de.driving\_ticket\_id = dt.id

WHERE de.exam\_result = 'passed'

    AND DATE(dt.datetime) = CURRENT\_DATE

GROUP BY category;

Клієнти, які отримали посвідчення за останні 3 місяці

SELECT c.first\_name, c.last\_name, dl.start\_date

FROM driving\_license dl

JOIN client c ON dl.client\_id = c.id

WHERE dl.start\_date >= CURRENT\_DATE - INTERVAL '3 months';

Рейтинг успішності інспекторів

SELECT i.\*, COUNT(CASE WHEN de.exam\_result = 'passed' THEN 1 END) \* 100. / COUNT(\*) AS success\_rate

FROM inspector i

JOIN driving\_exam de ON i.id = de.inspector\_id

GROUP BY i.id

ORDER BY success\_rate DESC;

Інспектори, які провели менше іспитів, ніж середній показник

SELECT i.\*, COUNT(\*) AS total\_exams

FROM inspector i

LEFT JOIN driving\_exam de ON i.id = de.inspector\_id

GROUP BY i.id

HAVING COUNT(\*) < (

    SELECT AVG(count)

    FROM (

        SELECT COUNT(\*)

        FROM driving\_exam

        GROUP BY inspector\_id

    )

);

Клієнти, які не з’явилися на запис

SELECT c.\*, COUNT(\*)

FROM all\_tickets at

JOIN client c ON at.client\_id = c.id

WHERE at.status = 'expired'

GROUP BY c.id;

Клієнти з усіма категоріями

SELECT c.\*

FROM person\_opened\_categories poc

JOIN client c ON poc.id = c.id

WHERE poc.person\_type = 'client'

GROUP BY c.id

HAVING COUNT(\*) = 14;

Клієнти, у яких закінчився період призупинення посвідчення

SELECT c.\*

FROM client c

WHERE get\_person\_last\_active\_driving\_license('client', c.id) IN (

    SELECT dl.id

    FROM driving\_license dl

    JOIN driving\_license\_status dls ON dl.id = dls.driving\_license\_id

    WHERE dls.status = 'suspended'

        AND dls.end\_date < CURRENT\_DATE

);

Теоретичні питання та відповіді

SELECT q.text AS question, q.photo\_file\_path AS question\_photo\_file\_path, a.text AS answer, a.photo\_file\_path AS answer\_photo\_file\_path, a.is\_correct

FROM question q

JOIN answer a ON q.id = a.question\_id;

Категорії клієнтів

SELECT first\_name, last\_name, middle\_name, category, gearbox

FROM person\_opened\_categories poc

JOIN client c ON poc.id = c.id

WHERE poc.person\_type = 'client';

Кількість спроб клієнтів скласти практичний іспит

SELECT c.\*, dt.category, dt.gearbox, COUNT(\*) AS total\_exams, CASE WHEN COUNT(CASE WHEN de.exam\_result = 'passed' THEN 1 END) > 0 THEN 'Yes' ELSE 'No'

END AS was\_successful

FROM driving\_exam de

JOIN driving\_ticket dt ON de.driving\_ticket\_id = dt.id

JOIN client c ON dt.client\_id = c.id

GROUP BY c.id, dt.category, dt.gearbox;

Доступні категорії та коробки передач для сервісних центрів

SELECT DISTINCT c.category, c.gearbox, sc.address, sc.number

FROM car c

JOIN service\_center sc ON c.service\_center\_id = sc.id

WHERE car\_owner = 'service'

ORDER BY number;

Найпопулярніша категорія та коробка передач

SELECT category, gearbox, COUNT(\*) AS total

FROM driving\_exam de

JOIN driving\_ticket dt ON de.driving\_ticket\_id = dt.id

GROUP BY category, gearbox

ORDER BY total DESC;

Клієнти, з більше ніж одним призупиненням посвідчення

SELECT c.\*, COUNT(\*) AS total\_suspensions

FROM client c

JOIN driving\_license dl ON c.id = dl.client\_id

JOIN driving\_license\_status dls ON dl.id = dls.driving\_license\_id

WHERE dls.status = 'suspended'

GROUP BY c.id

HAVING COUNT(\*) > 1;

Тільки теоретичні сервісні центри

SELECT sc.\*

FROM service\_center sc

WHERE NOT EXISTS (

        SELECT 1

        FROM car c

        WHERE c.service\_center\_id = sc.id

            AND c.car\_owner = 'service'

    );

Популярність сервісних центрів

SELECT sc.\*, COUNT(\*) AS popularity

FROM all\_tickets at

JOIN service\_center sc ON at.service\_center\_id = sc.id

GROUP BY sc.id

ORDER BY popularity DESC;

Середній результат теоретичних іспитів по рокам

SELECT EXTRACT(YEAR FROM tt.datetime) AS year, AVG(get\_theory\_exam\_result(te.id)) AS average\_result

FROM theory\_exam te

JOIN theory\_ticket tt ON te.theory\_ticket\_id = tt.id

GROUP BY year

ORDER BY year;

Клієнти з недійсними документами

SELECT c.\*, d.type, MAX(d.end\_date) AS last\_out\_of\_date

FROM client c

JOIN document d ON c.id = d.client\_id

GROUP BY c.id, d.type

HAVING MAX(d.end\_date) < CURRENT\_DATE;

**Приклади додавання даних**

INSERT INTO client (first\_name, last\_name, middle\_name, date\_of\_birth, address, phone\_number, email)

VALUES ('Іван', 'Петров', 'Сергійович', '1990-05-20', 'вул. Шевченка, 1', '+380123456789', 'ivan.petrov@gmail.com');

INSERT INTO service\_center (address, number)

VALUES ('вул. Лесі Українки, 15', 123);

INSERT INTO inspector (first\_name, last\_name, middle\_name, service\_center\_id)

VALUES ('Олександр', 'Коваль', 'Іванович', 1);

INSERT INTO document (type, end\_date, info\_file\_path, client\_id)

VALUES ('passport', '2030-12-31', '/files/documents/passport1.pdf', 1);

INSERT INTO car (car\_owner, gearbox, license\_plate, category, service\_center\_id)

VALUES ('service', 'manual', 'AA1234BC', 'B', 1);

INSERT INTO theory\_ticket (datetime, service\_center\_id, client\_id)

VALUES ('2024-01-10 10:00:00', 1, 1);

INSERT INTO driving\_ticket (datetime, gearbox, category, car\_owner, service\_center\_id, client\_id)

VALUES ('2024-01-15 12:00:00', 'manual', 'B', 'service', 1, 1);

INSERT INTO theory\_exam (theory\_ticket\_id)

VALUES (1);

INSERT INTO question (text, photo\_file\_path)

VALUES ('Яка максимальна дозволена швидкість у місті?', '/files/questions/q1.jpg');

INSERT INTO answer (text, is\_correct, question\_id)

VALUES ('50 км/год', TRUE, 1);

INSERT INTO theory\_exam\_result (theory\_exam\_id, question\_id, selected\_answer\_id)

VALUES (1, 1, 1);

INSERT INTO driving\_exam (driving\_ticket\_id, inspector\_id, car\_id)

VALUES (1, 1, 1);

INSERT INTO driving\_license (end\_date, client\_id)

VALUES ('2034-01-01', 1);

INSERT INTO driving\_license\_category (category, gearbox, driving\_license\_id)

VALUES ('B', 'manual', 1);

INSERT INTO fine (reason, sum, driving\_license\_id)

VALUES ('Перевищення швидкості', 500, 1);

INSERT INTO driving\_license\_status (status, reason, driving\_license\_id)

VALUES ('revoked', 'Позбавлення прав за водіння у нетверезому стані', 1);