

Міністерство освіти і науки України

Національний університет водного господарства  
та природокористування

Навчально-науковий інститут агроекології та землеустрою

Кафедра геодезії та картографії

**05-04-159М**

## **МЕТОДИЧНІ ВКАЗІВКИ**

до виконання лабораторних і самостійних робіт з навчальної  
дисципліни **«Проектування та управління базами  
геопросторових даних» (частина 2)** для здобувачів вищої  
освіти першого (бакалаврського) рівня за освітньо-професійною  
програмою «Геодезія та землеустрій» спеціальності  
193 «Геодезія та землеустрій» усіх форм навчання

Рекомендовано науково-методичною  
радою з якості ННІАЗ  
Протокол №13 від 18 лютого 2025 р.

Рівне – 2025

Методичні вказівки до виконання лабораторних і самостійних робіт з навчальної дисципліни «Проектування та управління базами геопросторових даних» (частина 2) для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Геодезія та землеустрій» спеціальності 193 «Геодезія та землеустрій» усіх форм навчання [Електронне видання] / Янчук О. Є., Прокопчук А. В. – Рівне : НУВГП, 2025. – 50 с.

Укладачі:

Янчук О. Є., канд. техн. наук, доцент кафедри геодезії та картографії;

Прокопчук А. В., ст.викл. кафедри геодезії та картографії.

Відповідальний за випуск:

Янчук Р. М., к.т.н., доцент, завідувач кафедри геодезії та картографії.

Керівник групи забезпечення спеціальності:

Янчук Р. М., к.т.н., доцент, завідувач кафедри геодезії та картографії.

© О. Є. Янчук,  
А. В. Прокопчук, 2025  
© Національний університет  
водного господарства та  
природокористування, 2025

## ЗМІСТ

ПЕРЕДМОВА .....	4
ЛАБОРАТОРНІ РОБОТИ.....	5
Лабораторна робота № 5 Створення просторової бази даних у PostgreSQL/Postgis .....	5
Лабораторна робота № 6 Робота з SQL запитами на вибірку даних .....	19
Лабораторна робота № 7 Редагування структури й вмісту бази даних та налаштування прав доступу .....	41
РЕКОМЕНДОВАНА ЛІТЕРАТУРА ТА РЕСУРСИ.....	50

## ПЕРЕДМОВА

Методичні вказівки складено відповідно до програми навчальної дисципліни «Проектування та управління базами геопросторових даних» та призначено для здобувачів вищої освіти першого (бакалаврського) рівня спеціальності 193 «Геодезія та землеустрій» усіх форм навчання.

Головним завданням курсу є навчання студентів навичкам проектування, організації функціонування і використання баз даних геоінформаційних систем, а також впровадження таких систем в виробництво і органи державного управління та у всі можливі галузі народного господарства.

У даній частині методичних вказівок викладено принципи роботи з базами геопросторових даних PostgreSQL та формування SQL запитів. Розглянуто можливості завантаження векторних даних в OSM. Значна увага приділена роботі з системою управління базами даних PostgreSQL/PostGIS та налаштуванню прав доступу її користувачів.

У результаті виконання представлених лабораторних робіт студенти повинні навчитися конвертувати дані між різними програмами, виконувати геокодування даних, формувати SQL-запити, налаштовувати права доступу для підключення до бази даних.

## ЛАБОРАТОРНІ РОБОТИ

### *Лабораторна робота № 5*

#### *Створення просторової бази даних у PostgreSQL/Postgis*

**Мета:** навчитись створювати просторову базу даних у PostgreSQL/Postgis та наповнювати її даними.

**Завдання:** з OSM завантажити карту на потрібну територію; створити на карті QGIS точкові об'єкти, які відповідають обраній предметній області; з MS Access експортувати всі корисні дані; геокодувати до точок на карті у QGIS інформацію з БД Access; створити просторову БД у Postgis; конвертувати підготовлені векторні дані у БД Postgis.

#### **1) Завантаження векторних даних з OSM**

OpenStreetMap (OSM) – міжнародний проєкт, метою якого є створення вільної, відкритої мапи світу. Дані, зібрані учасниками проєкту, є вільними, відкритими та безкоштовними. Ці дані можна вільно використовувати для будь-яких власних проєктів посилаючись на OpenStreetMap.

Для подальшої роботи на курсі нам необхідно завантажити у векторному вигляді дані про район робіт. Для навчальних цілей обмежимося наборами даних: адміністративні межі, дорожня/вулична мережа, гідрографія.

#### **Завантаження даних за допомогою плагіну QuickOSM**

Дані з OpenStreetMap можна завантажувати у векторному вигляді багатьма способами. Для випадку коли об'єкти бази даних розміщуються на території окремого населеного пункту розглянемо варіант завантаження необхідних шарів за допомогою програми QGIS.

1. Запускаємо програму QGIS та створюємо новий проєкт **Проект / Новий** або **ctrl + N**.

2. При потребі встановлюємо необхідні плагіни. Заходимо у меню **Плагіни / Управління та встановлення плагінів**. У вікні **Плагіни** шукаємо та встановлюємо плагіни **OpenLayers Plugin** та **QuickOSM** (рис. 1.1).

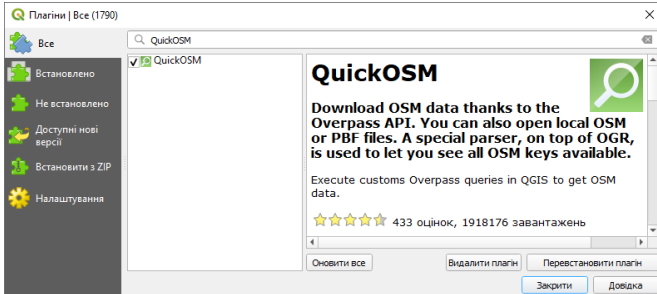



Рис. 1.1. Вікно модуля “Плагіни” з описом інстальованого розширення

3. Підключаємо базову карту через плагін **Веб / OpenLayers Plugin / OpenStreetMap / OpenStreetMap**. Збільшуємо карту для відображення на екрані необхідної території.

4. Для завантаження векторних даних запускаємо плагін **QuickOSM** натиснувши на панелі інструментів відповідну кнопку .

5. У вікні плагіна **QuickOSM** на вкладці **Швидкий запит** вводимо умови відбору (рис. 1.2). Для пошуку **адміністративних меж** задаємо ключ **boundary**, значення **administrative**. Для пошуку в межах відображення екрану задаємо **Екстент полотна**. Натискаємо **Виконати запит**.

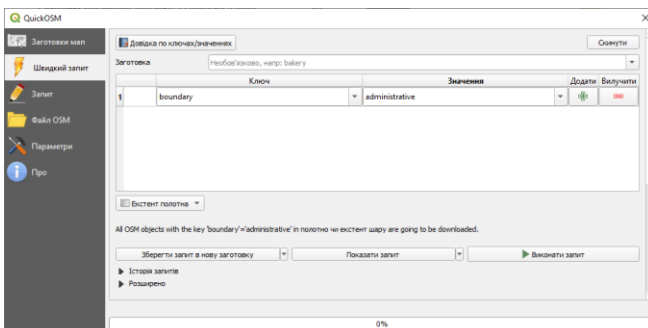


Рис. 1.2. Умови запити на отримання даних через QuickOSM

6. Результатом буде відображення на екрані даних, які відповідають умовам відбору (рис. 1.3). Виділяємо необхідні

нам об'єкти та зберігаємо їх в окремий файл формату \*.shp командою контекстного меню **Експорт / Зберегти вибрані об'єкти як** (рис. 1.4). Після збереження потрібних об'єктів тимчасові завантажені шари можна видалити з карти.

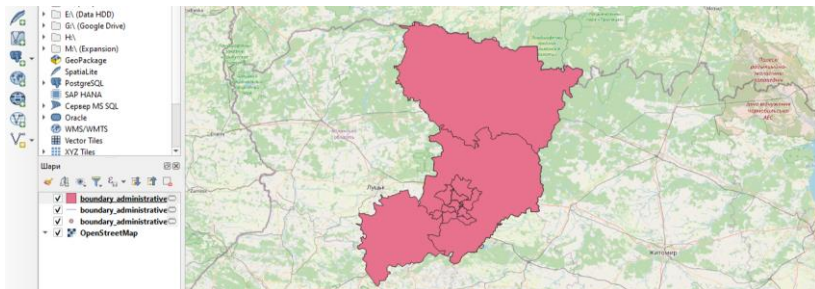


Рис. 1.3. Результат виконання запити

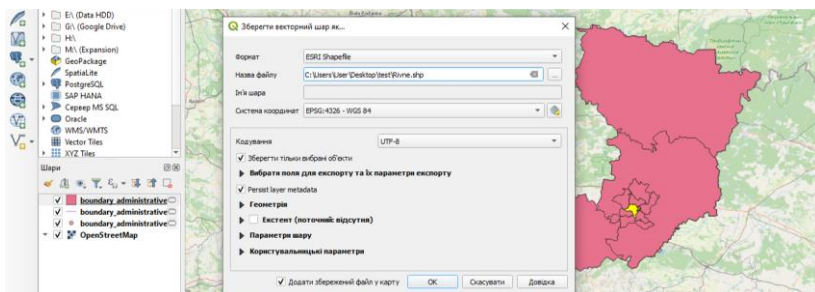


Рис. 1.4. Збереження необхідних об'єктів

7. Повторюємо запити через плагін **QuickOSM** для завантаження вуличної мережі та гідрографії змінивши відповідні запити ключ-значення. Введення значення для ключа зменшує кількість результатів пошуку, але не є обов'язковим. Тому рекомендується виконати запити для ключів highway, natural-water, waterway (рис. 1.5).

Знайти необхідний ключ-значення можна на довідковій сторінці [https://wiki.openstreetmap.org/wiki/Map\\_features](https://wiki.openstreetmap.org/wiki/Map_features).

Також для зменшення області пошуку варто замість Екстент полотна обрати **Екстент шару** та вказати шар з необхідними адміністративними межами, збережений на попередньому етапі.

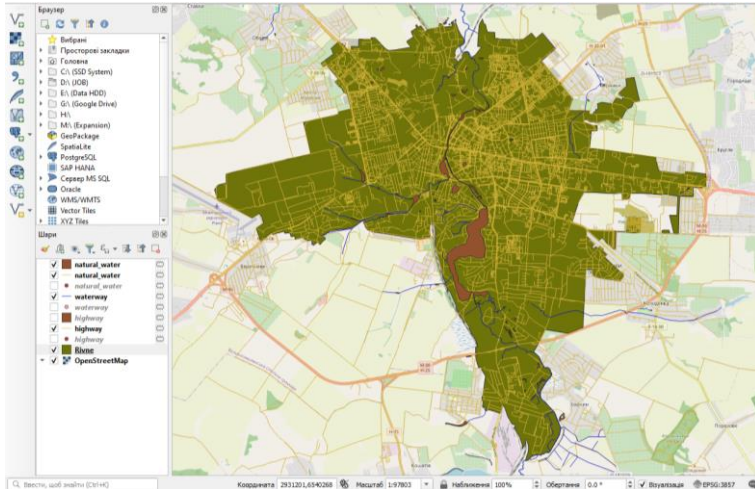


Рис. 1.5. Завантажені об'єкти вуличної мережі та гідрографії

8. Обрізаємо об'єкти за межею адміністративної одиниці командою меню **Вектор / Обрізка**.

9. Зберігаємо потрібні нам об'єкти під відповідними іменами.

### **Завантаження даних за допомогою сервісу Geofabrik**

У випадку, якщо необхідно працювати з декількома населеними пунктами або значними територіями зручніше скористатися сервісом Geofabrik <https://download.geofabrik.de/>. За згаданим посиланням шукаємо необхідний регіон, в даному випадку **Європа – Україна**, та завантажуюмо файли у форматі **[.shp.zip]**. Таким чином будуть завантажені всі шари векторних даних на територію України. Далі необхідно вибрати потрібні нам шари та обрізати їх за районом робіт.

### **2) Створення у QGIS точкових об'єктів**

У цьому пункті необхідно в середовищі програми QGIS відмітити точковими елементами розташування об'єктів, інформація про які заповнена у БД MS Access.

1. Створюємо новий точковий шар **Шар / Створити шар / Створити шар Shapefile** (рис. 1.6).



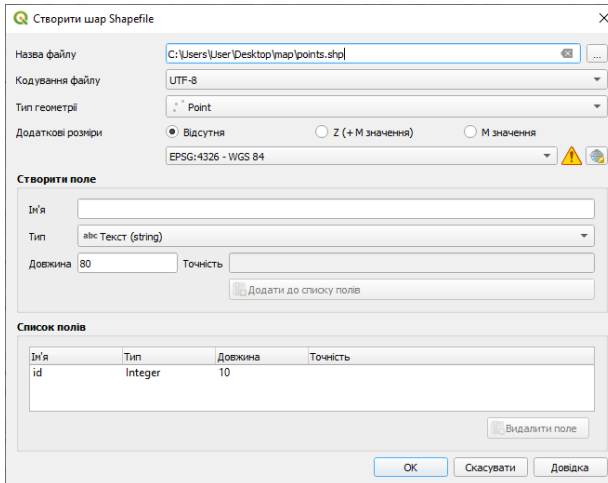


Рис. 1.6. Створення нового шейп-файлу

2. Вмикаємо можливість **редагування** створеного шару та активувавши кнопку **Додати точковий об’єкт** вказуємо місце розташування об’єктів, інформація про які заповнена у БД MS Access. *Обов’язково задаємо значення атрибуту **id** для об’єктів таке ж як задане у БД MS Access (рис. 1.7).*

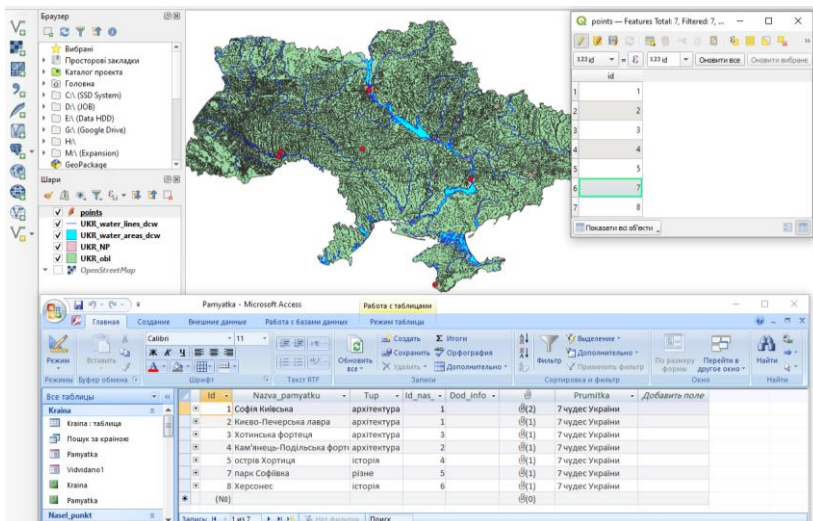


Рис. 1.7. Створення точкових об’єктів

### 3) Експорт даних з MS Access

Експортуємо дані збережені у таблицях MS Access у текстовий файл.

1. У MS Access клацаємо правою кнопкою миші на назві таблиці та з **контекстного меню** обираємо **Експорт / Текстовий файл**.

2. У першому вікні **Експорт тексту** задаємо місце збереження та ім'я й натискаємо **Ок**. У наступному вікні обираємо перемикач з **розділювачами** й натискаємо **Далі**. У наступному вікні обираємо перемикач **крапка з комою, увімкнуті імена полів у першому рядку** й натискаємо **Далі**. У наступному вікні перевіряємо адресу збереження файлу та натискаємо **Готово**.

У результаті отримуємо текстовий файл з інформацією, що містилася у БД Access (рис. 1.8).

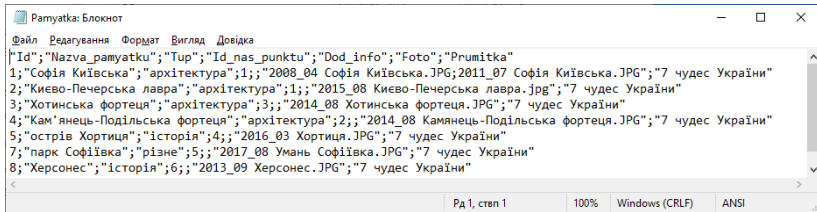


Рис. 1.8. Одержаний текстовий файл

### 4) Геокодування інформації

Наступним кроком є зв'язування атрибутивних даних з просторовими даними. Важливим є наявність поля з однаковими значеннями в обох таблицях – атрибутивній та просторовій, за якими і формуватиметься зв'язок.

*Геокодування* – метод і процес позиціонування векторних просторових об'єктів відносно деякої координатної системи і їхніх атрибутів. На практиці геокодування часто здійснюється через з'єднання або зв'язування таблиць.

1. Відкриваємо текстовий файл, експортований з MS Access, у середовищі QGIS використовуючи команду **Шар / Додати шар / Додати текстовий з роздільниками шар**.

2. У вікні **Адміністратор джерела даних | Текст з роздільниками** вказуємо розташування текстового файлу, кодування, роздільник – крапка з комою, обираємо перший запис має назви полів та не містить геометрії (рис. 1.9). Контролюємо чи правильно відображається вміст файлу у нижній частині вікна та натискаємо **Додати**.

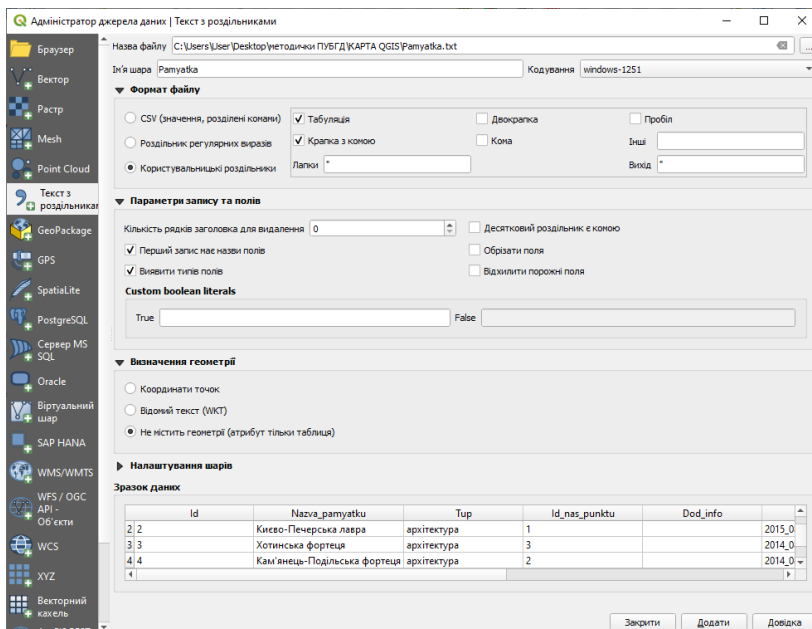


Рис. 1.9. Налаштування імпорту текстового файлу

3. З контекстного меню точкового файлу обираємо **Властивості / Об'єднує**. У нижній частині вікна натискаємо **Додати новий зв'язок**. У вікні **Додати векторне приєднання** вказуємо назву атрибутивної таблиці та її ключового поля, а також відповідне ключове поле просторової таблиці (рис. 1.10). У розділі **Приєднані поля** обираємо поля атрибутивної таблиці для приєднання до просторової таблиці.

Якщо все виконано правильно, то у атрибутивній таблиці точкових об'єктів відобразиться текстова інформація (рис. 1.11).

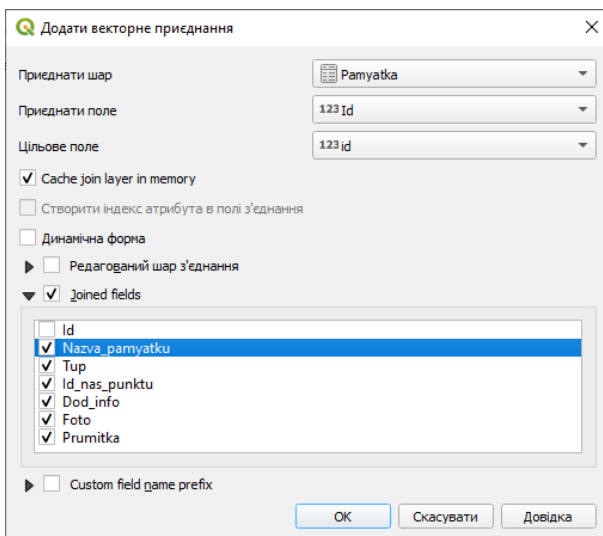


Рис. 1.10. Налаштування приєднання даних

id	yatka_Nazva_pamy	Pamyatka_Tup	nyatka_Id_nas_pun	amyatka_Dod_inf	Pamyatka_Foto	amyatka_Prumitk
1	Софія Київська	архітектура		1 NULL	2008_04 Софія ...	7 чудес України
2	2 Києво-Печерс...	архітектура		1 NULL	2015_08 Києво-...	7 чудес України
3	3 Хотинська фор...	архітектура		3 NULL	2014_08 Хотинс...	7 чудес України
4	4 Кам'янець-По...	архітектура		2 NULL	2014_08 Кам'ян...	7 чудес України
5	5 острів Хортиця	історія		4 NULL	2016_03 Хорти...	7 чудес України
6	7 парк Софіївка	різне		5 NULL	2017_08 Умань ...	7 чудес України
7	8 Херсонес	історія		6 NULL	2013_09 Херсо...	7 чудес України

Рис. 1.11. Приєднані дані

4. Дані зв'язані таким чином відображаються лише у поточному проєкті QGIS. Тому необхідно зберегти таблицю з приєднаними даними обравши з контекстного меню шару **Експорт / Зберегти об'єкти як**.

## **5) Створення просторової БД у PostgreSQL/Postgis**

PostgreSQL – кросплатформенна об'єктно-реляційна система управління базами даних (СУБД) з відкритим програмним кодом. Є альтернативою як комерційним СУБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СУБД з відкритим кодом (MySQL, Firebird, SQLite). Не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній.

PostGIS являє собою розширення СУБД PostgreSQL, призначене для зберігання в базі географічних даних (геометрії) і підтримує стандарти OGC (Open Geospatial Consortium).

Управління програмою PostgreSQL/PostGIS виконується через командний рядок, або через спеціальні оболонки (типу pgAdmin 4).

Дистрибутив програми для встановлення під відповідну операційну систему можна завантажити з вкладки **Download** сайту <https://www.postgresql.org/>. Під час встановлення необхідно задати пароль доступу для суперкористувача (superuser) та порт підключення, які в подальшому необхідні для роботи.

Встановлення розширення PostGIS можна здійснити напівавтоматично через додаток Stack Builder або безпосередньо скачавши пакет PostGIS. У першому випадку слід запустити додаток Stack Builder в меню **Поиск / PostgreSQL X.X / Application Stack Builder** та вибрати відповідний додаток для розширення (рис. 1.12).

У процесі роботи з СУБД PostgreSQL запуск на виконання більшості команд доступний як через ручне введення команд на мові SQL, так і через команди меню оболонки pgAdmin 4 та контекстні меню.

Для запуску оболонки pgAdmin 4 слід виконати команду **Поиск / PostgreSQL X.X / pgAdmin 4**.

Для підключення до СУБД необхідно двічі клацнути на **PostgreSQL X.X** у браузері об'єктів (ліва частина вікна pgAdmin 4). Далі з'явиться вікно у якому необхідно ввести пароль суперкористувача заданий під час встановлення програми. Після цього відобразиться перелік доступних баз

даних. Таблиці з даними зберігаються у розділі **Схеми** (за замовчуванням при створенні бази даних створюється схема public).

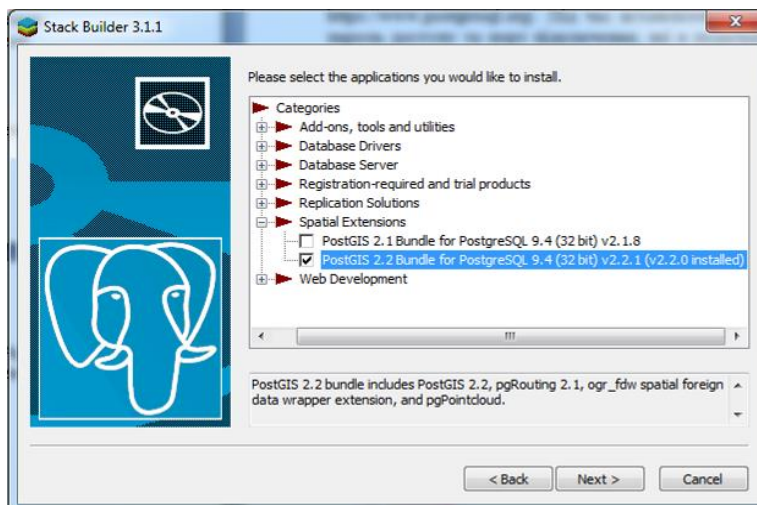


Рис. 1.12. Встановлення розширення PostGIS через додаток Stack Builder

### Створення просторової бази даних

1. У браузері об'єктів вибираємо **Бази даних (Databases)** і відкриваємо пункт меню **Об'єкт / Створити / База даних (Object / Create / Database)** або з контекстного меню **Створити / База даних (Create / Database)**.

2. На вкладці **Загальні (General)** задаємо ім'я нової бази даних та натискаємо **Зберегти (Save)**.

3. У браузері об'єктів вибираємо створену БД та обираємо або з панелі інструментів або з контекстного меню **Інструмент запитів (Query tool)**. Прописуємо запит на розширення БД до просторової **Create extension postgis** та запускаємо його на виконання.

У результаті виконання запиту у БД створюється таблиця **spatial\_ref\_sys** та представлення **geometry\_columns**.

## б) Конвертація векторних даних у БД PostgreSQL/Postgis

Необхідно імпортувати підготовлені векторні дані у БД PostgreSQL/Postgis.

1. Підключаємося до БД PostgreSQL/Postgis з QGIS. Запускаємо QGIS і вибираємо команду меню **Шар / Додати шар / Додати шар PostGIS** або комбінацію клавіш **Ctrl+Shift+D** для виклику вікна **Підключення PostgreSQL**.

2. Натискаємо кнопку **Новий** і задаємо параметри нового PostGIS з'єднання (рис. 1.13).

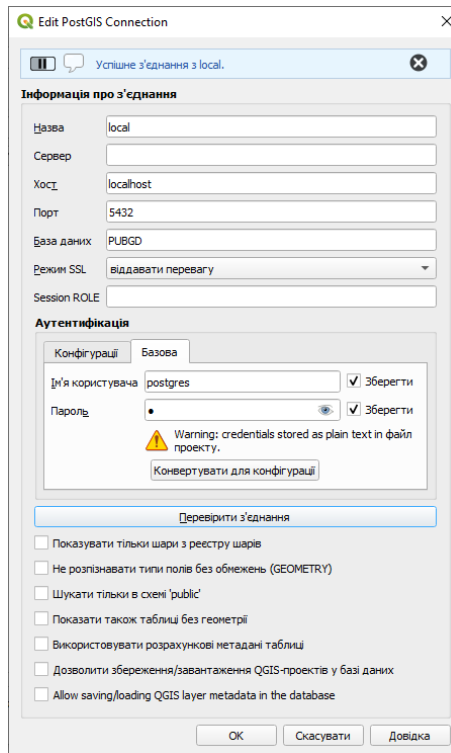


Рис. 1.13. Створення нового PostGIS-з'єднання

Достатньо заповнити наступні параметри:  
**Name (Назва)** – довільне ім'я з'єднання.

**Host (Сервер)** – адреса сервера на якому розташована база даних. Якщо база даних знаходиться на тому ж комп'ютері з якого виконується підключення необхідно прописати *localhost*.

**Database (База даних)** – назва бази даних до якої здійснюється підключення.

**Port (Порт)** – порт на якому розташований сервіс PostgreSQL, вибирається при встановленні програми (за замовчуванням 5432).

**Username (Користувач)** – ім'я користувача бази даних. Для доступу до всіх функцій роботи з базою даних слід вибрати суперкористувача, ім'я якого вказувалося при створенні програми. Для роботи з певним набором прав доступу – ім'я створеного користувача з наданими правами (див. лабораторну роботу №3).

**Password (Пароль)** – пароль для відповідного користувача, введеного на попередньому етапі.

Для навчальних цілей рекомендується встановити відмітки **Зберегти** Ім'я користувача та Пароль, щоб не вводити їх щоразу при з'єднанні з БД.

Після того як всі поля будуть заповнені варто перевірити наявність з'єднання натиснувши **Перевірити з'єднання**. Якщо всі параметри заповнені правильно має з'явитися вікно з повідомленням **Успішне з'єднання** (рис. 1.13).

3. Натискаємо **Ok** у вікні нового з'єднання, таким чином повернувшись у вікно **Підключення PostgreSQL**, яке закриваємо кнопкою **Закрити**.

4. У головному вікні QGIS вибираємо пункт меню **База даних / Менеджер БД**.

5. У лівій частині вікна розкриваємо каталог PostGIS у якому відобразиться створене нами з'єднання з переліком доступних з БД таблиць (рис. 1.14). Натискаємо кнопку **Імпортувати шар/файл**.

6. З'являється вікно **Імпортувати векторний шар** (рис.1.15). У верхній частині вікна обираємо шар, який ми будемо імпортувати у БД. Нижче обираємо схему з БД, у яку він буде імпортуватися та прописуємо його нове ім'я у БД.



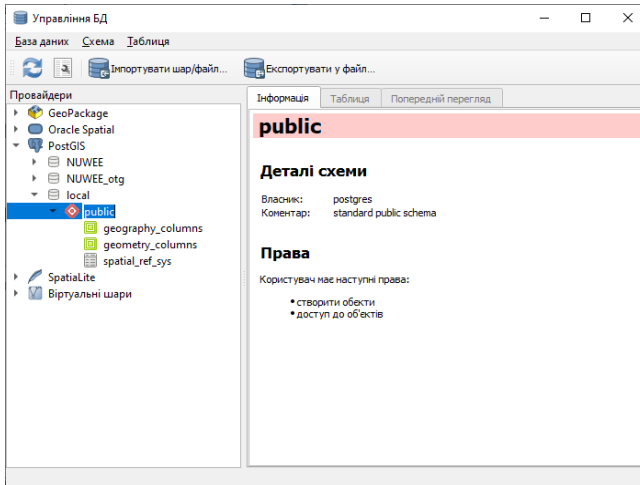


Рис. 1.14. Вікно менеджера БД

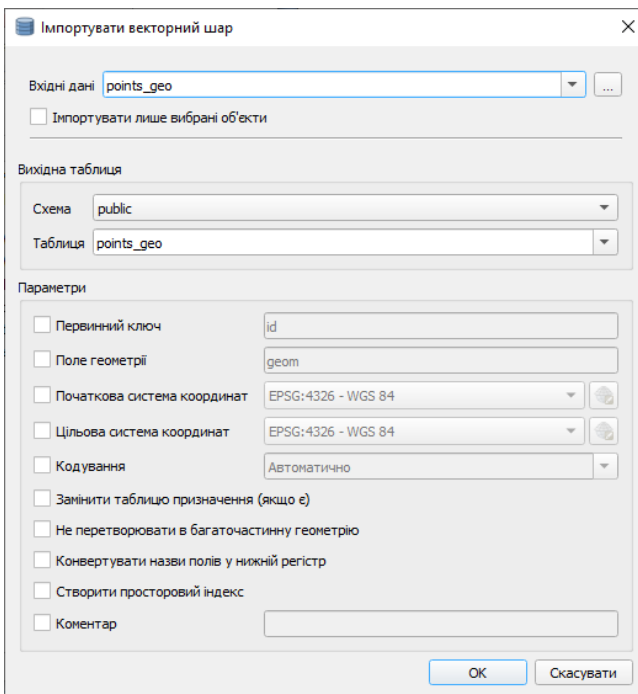


Рис. 1.15. Вікно імпорту векторного шару

7. Натискаємо **Ок**. Після завершення операції повинне з'явитися повідомлення **Імпорт був успішним**.

Таким чином імпортуємо у БД PostgreSQL/Postgis шар з геокодованими точковими об'єктами, а також підготовлені шари з адміністративними межами, дорожньою/вуличною мережею та гідрографією.

Після імпорту до PostgreSQL/Postgis всі таблиці мають відображатися у з'єднанні PostgreSQL у браузері QGIS (рис. 1.16).

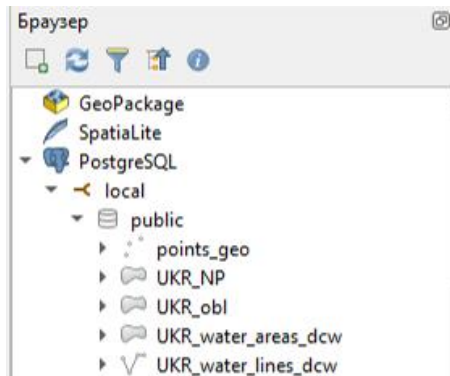


Рис. 1.16. Відображення таблиць імпортованих у PostgreSQL/Postgis

**Завдання для самостійної роботи.** Розгляньте які ще можливі варіанти завантаження векторних даних з OSM.

**Запитання для контролю:**

1. Що таке OSM?
2. Як завантажити дані OSM через QGIS?
3. Як завантажити дані OSM на територію всієї країни?
4. Що таке геокодування?
5. Як створити новий точковий шар у QGIS?
6. Як створити базу даних у PostgreSQL?
7. Як зробити базу даних просторовою у PostgreSQL/Postgis?
8. Як підключитися до БД PostgreSQL з QGIS?
9. Як імпортувати векторні дані з QGIS у БД PostgreSQL/Postgis?

## *Лабораторна робота № 6*

### *Робота з SQL запитами на вибірку даних*

**Мета:** навчитись створювати SQL запити на вибірку даних та запити для роботи з системами координат.

**Завдання:** створити SQL запити у PostgreSQL/Postgis та QGIS : на вибірку атрибутивних даних, на вибірку просторових даних, на перетворення координат, з відображенням результатів на карті.

#### **1) Запити на вибірку атрибутивних даних**

SQL (англ. Structured query language – мова структурованих запитів) – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних.

Конструкція запиту SQL складається із зарезервованих слів, а також зі слів, що визначаються користувачем.

Зарезервовані слова є постійною частиною мови SQL і мають фіксоване значення. Їх треба записувати саме так, як це регламентовано, не можна розбивати на частини для переносу з одного рядка на інший.

Слова, що визначаються користувачем, задаються ним самим (відповідно до синтаксичних правил) і становлять ідентифікатори різних об'єктів БД. Ідентифікатори мови SQL призначені для позначення об'єктів у БД і є іменами таблиць, рядків, стовпчиків та інших об'єктів бази даних. На формат ідентифікатора накладаються такі обмеження: може мати довжину до 128 символів; повинен починатись з літери; не може містити проміжки.

Стандарт SQL задає набір символів який використовуються за замовчуванням. Він включає рядкові і прописні літери латинського алфавіту (A–Z, a–z), цифри (0–9) і символ підкреслювання (\_). Вирази в SQL не чутливі до регістру та ігнорують додаткові роздільники (пробіл, табуляція, перехід на новий рядок). *Для зручності читання запитів часто*

зарезервовані слова прописують великими літерами та розділяють запит на рядки.

В оболонці pgAdmin 4 доступ до введення команд на мові SQL виконується через кнопку **Query Tool** на панелі інструментів (рис. 2.1) або через відповідну команду контекстного меню бази даних. Після введення SQL -запиту необхідно запустити його на виконання відповідною кнопкою панелі інструментів або натиснувши **F5**.

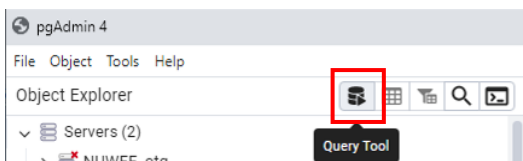


Рис. 2.1. Кнопка запуску редактора запитів SQL з панелі меню pgAdmin 4

Для вибірки даних з таблиці використовується оператор **SELECT**. У найпростішому випадку конструкція запиту на вибірку даних виглядає наступним чином:

```
SELECT column1, [column2, ...] FROM table_name;
```

Тобто необхідно перелічити імена стовпців, які ми хочемо отримати в результаті запиту та зазначити з якої таблиці їх брати. Параметри зазначені у квадратних дужках є необов'язковими для введення, тому достатньо ввести ім'я лише одного стовпця. Для вибору всіх стовпців з таблиці можна скористатися запитом:

```
SELECT * FROM table_name;
```

Оператор SELECT можна доповнювати іншими умовами:

```
SELECT список вибірки  
[ INTO нова_таблиця ]  
FROM таблиця  
[ WHERE умова_відбору ]  
[ GROUP BY умова_групування ]  
[ HAVING умова_відбору ]  
[ ORDER BY умова_сортуння [ ASC | DESC ] ]
```

INTO дозволяє помістити відібрані дані у нову таблицю. FROM вказує, з якої таблиці проводиться вибірка даних. WHERE вказує умови, за якими слід вибрати записи. GROUP BY групує рядки, за вказаною властивістю. HAVING вибирає з груп означених оператором GROUP BY. ORDER BY вказує порядок сортування записів при виведенні на екран (ASC – за зростанням; DESC – за спаданням). Запит може виконуватися без параметрів зазначених у квадратних дужках, які описуються лише при потребі.

Розглянемо декілька варіантів побудови SQL-запитів на прикладі бази даних (PUBGD) сформованої у попередній лабораторній роботі. Структура таблиці з даними про населені пункти (UKR\_NP), наведена на рисунку 2.2. Вона містить інформацію про назву (кирилицею та латиницею) населеного пункту, адміністративного району та області до яких належить населений пункт, їхні коди адміністративно-територіального устрою, тип населеного пункту (село, селище, місто), кількість населення, а також ідентифікатори об'єктів та їх геометрію.

	name_us_s	name_lat_s	koutu_s	codename_s	population	name_us_b	name_lat_b	koutu_b	name_us_1	name_lat_2	koutu_b
	character vary	character vary	character vary	character vary	double precis	character vary	character vary	character vary	character vary	character vary	character vary
1	Дібрів	Divrets	6123483102	село	188	Кременецький	Kremenets...	6123400000	Тернопіль...	Тетопіліг...	6100000000
2	Діроці	Dvortsy	6120487003	село	194	Бережанський	Berezhans...	6120400000	Тернопіль...	Тетопіліг...	6100000000
3	Діроці	Dvoryshc...	6123084005	село	61	Козівський	Kozivskiy	6123000000	Тернопіль...	Тетопіліг...	6100000000
4	Вільшанка	Vilshanka	3523683901	село	479	Новоархангел...	Novoarkha...	3523600000	Кіровоград...	Kirovohra...	3500000000
5	Вірне	Virne	3521781003	село	23	Добровеличк...	Dobrovelyc...	3521700000	Кіровоград...	Kirovohra...	3500000000
6	Владисла...	Vladyslav...	3524381302	село	87	Вільшанський	Vilshanskiy	3524300000	Кіровоград...	Kirovohra...	3500000000
7	Вовча Ба...	Vovcha B...	3524382203	село	46	Вільшанський	Vilshanskiy	3524300000	Кіровоград...	Kirovohra...	3500000000
8	Водяне	Vodiane	3521782702	село	87	Добровеличк...	Dobrovelyc...	3521700000	Кіровоград...	Kirovohra...	3500000000
9	Водяне	Vodiane	3522881101	село	349	Компанієвск...	Kompaniye...	3522800000	Кіровоград...	Kirovohra...	3500000000
10	Артиз	Artysz	5120410100	місто	15178	Арцизький	Artyszkiy	5120400000	Одеська	Odeska	5100000000
11	Жовтине	Zhovtine	6125289202	село	646	Тернопільськ...	Temopilskiy	6125200000	Тернопіль...	Тетопіліг...	6100000000
12	Вікниче	Vikniate	3521783002	село	70	Добровеличк...	Dobrovelyc...	3521700000	Кіровоград...	Kirovohra...	3500000000
13	Вільне	Vilne	3522581901	село	883	Кіровоградськ...	Kirovohrad...	3522500000	Кіровоград...	Kirovohra...	3500000000

Рис. 2.2. Структура таблиці з інформацією про населені пункти

Найпростіший запит – на вибірку всіх даних з таблиці виглядає так:

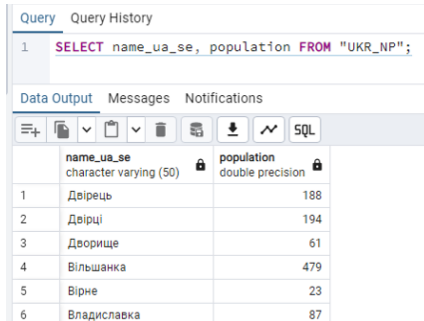
```
SELECT * FROM "UKR_NP";
```

*Примітка: оскільки назва таблиці містить великі літери, то назву потрібно взяти у подвійні лапки.*

Якщо потрібно у результаті вибірки отримати лише два стовпці – з назвами населеного пункту та кількістю населення, то необхідно прописати у запиті їхні назви, розділяючи комою:

```
SELECT name_ua_se, population FROM "UKR_NP";
```

Результат виконання запиту наведено на рисунку 2.3.



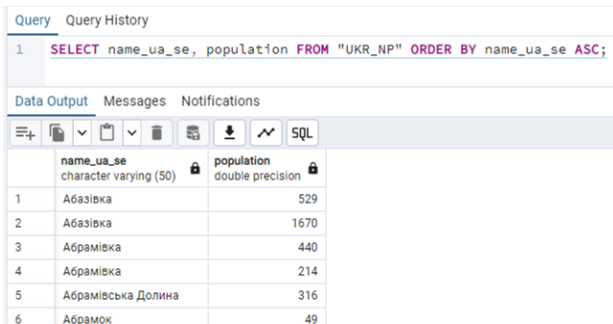
	name_ua_se character varying (50)	population double precision
1	Двірець	188
2	Двірці	194
3	Дворище	61
4	Вільшанка	479
5	Вірне	23
6	Владиславка	87

Рис. 2.3. Результат вибірки двох стовпців

Для того щоб відсортувати результат за алфавітом – додамо вираз **ORDER BY**:

```
SELECT name_ua_se, population FROM "UKR_NP"  
ORDER BY name_ua_se ASC;
```

Результат виконання запиту наведено на рисунку 2.4.



	name_ua_se character varying (50)	population double precision
1	Абазівка	529
2	Абазівка	1670
3	Абрамівка	440
4	Абрамівка	214
5	Абрамівська Долина	316
6	Абрамок	49

Рис. 2.4. Результат вибірки відсортований за алфавітом за назвами населених пунктів

Якщо потрібно відсортувати результат за кількістю населення – від найбільших населених пунктів до найменших – запит матиме вигляд:

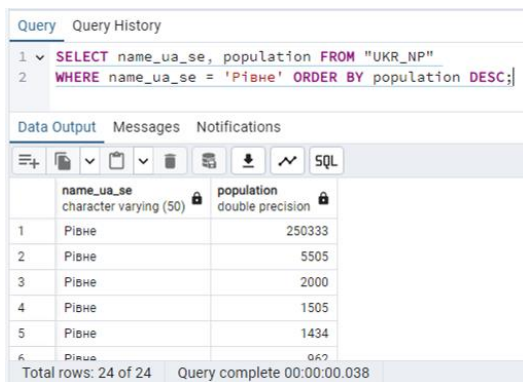
```
SELECT name_ua_se, population FROM "UKR_NP"  
ORDER BY population DESC;
```

Наведені вище запити в результаті повертають всі записи, які містяться у таблиці. Якщо нам потрібно обмежити результати певними критеріями – необхідно використати вираз **WHERE**. Наприклад, запит для вибору всіх населених пунктів з назвою Рівне виглядає так:

```
SELECT name_ua_se, population FROM "UKR_NP"  
WHERE name_ua_se = 'Рівне'  
ORDER BY population DESC;
```

*Примітка: текстові дані для пошуку необхідно виділяти одинарними лапками; розміщення даних в один рядок або декілька не впливає на виконання запиту, рекомендується запит розміщувати в декілька рядків для зручності його читання та розуміння.*

Результат виконання запиту наведено на рисунку 2.5.



	name_ua_se character varying (50)	population double precision
1	Рівне	250333
2	Рівне	5505
3	Рівне	2000
4	Рівне	1505
5	Рівне	1434
6	Рівне	062

Total rows: 24 of 24    Query complete 00:00:00.038

Рис. 2.5. Результат вибірки населеного пункту з назвою Рівне, відсортований за кількістю населення від найбільшого до найменшого

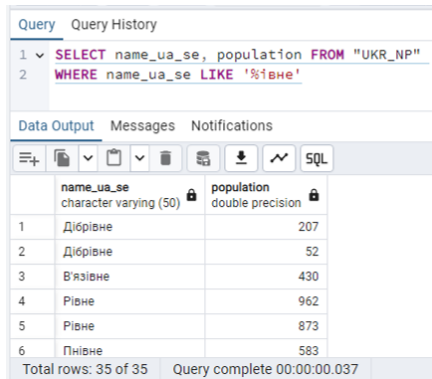
Оператор «=» підтримує лише строге співпадіння значення. Для пошуку за частиною значення комірки корисним є оператор **LIKE**. При цьому можна користуватися нижнім підкресленням «\_», що заміняє при пошуку один символ на довільний або знаком відсотків «%», що заміняє при пошуку будь-яку кількість символів на довільну.

Таким чином запит з умовою відбору '**івне**' поверне всі населені пункти, які закінчуються на «івне» та відрізняються лише першою літерою. В даному прикладі результат буде 24 записи, тобто ідентичний наведеному на рисунку 2.5.

```
SELECT name_ua_se, population FROM "UKR_NP"  
WHERE name_ua_se LIKE 'івне'
```

На противагу цьому запит з умовою відбору '**%івне**' поверне всі населені пункти, які закінчуються на «івне» та мають довільну кількість символів на початку. Результат в такому випадку становитиме 35 записів (рис. 2.6).

```
SELECT name_ua_se, population FROM "UKR_NP"  
WHERE name_ua_se LIKE '%івне'
```



	name_ua_se character varying (50)	population double precision
1	Дібрівне	207
2	Дібрівне	52
3	В'язівне	430
4	Рівне	962
5	Рівне	873
6	Пнівне	583

Total rows: 35 of 35    Query complete 00:00:00.037

Рис. 2.6. Результат вибірки населеного пункту з умовою відбору '**%івне**'

Також підтримуються математичні та статистичні оператори, зокрема дії додавання «+», віднімання «-», множення «\*», ділення «/». Наприклад, обчислимо нове значення стовпця population та відобразимо його у результаті запиту з новим ім'ям new\_popul (для цього використовується оператор **AS**):

```
SELECT name_ua_se, population, population*10 AS  
new_popul FROM "UKR_NP"
```

Результат виконання запиту наведено на рисунку 2.7.



Query		Query History	
1	<code>SELECT name_ua_se, population, population*10 AS new_popul FROM "UKR_NP"</code>		
2			
Data Output			
Messages			
Notifications			
<div style="display: flex; justify-content: space-between; align-items: center;"> <span>SQL</span> </div>			
	name_ua_se character varying (50)	population double precision	new_popul double precision
1	Двірець	188	1880
2	Двірці	194	1940
3	Дворище	61	610
4	Вільшанка	479	4790
5	Вірне	23	230
6	Владиславка	87	870

Рис. 2.7. Результат математичних обчислень

Для порівняння записів можна використовувати оператори більше «>», менше «<», більше або рівне «>=», менше або рівне «<=», не дорівнює «!=», міститься між **BETWEEN**. Наприклад, для вибору населених пунктів з населенням більше 1000000 можна використати запит:

```
SELECT name_ua_se, population FROM "UKR_NP"
WHERE population > 1000000
```

Для вибору населених пунктів з населенням у межах від 700000 до 1000000:

```
SELECT name_ua_se, population FROM "UKR_NP"
WHERE population BETWEEN 700000 AND 1000000
```

Результат виконання запиту наведено на рисунку 2.8.

Query		Query History	
1	<code>SELECT name_ua_se, population FROM "UKR_NP"</code>		
2	<code>WHERE population BETWEEN 700000 AND 1000000</code>		
3			
Data Output			
Messages			
Notifications			
<div style="display: flex; justify-content: space-between; align-items: center;"> <span>SQL</span> </div>			
	name_ua_se character varying (50)	population double precision	
1	Дніпропетровськ	997754	
2	Львів	730272	
3	Донецьк	953217	
4	Запоріжжя	770672	

Рис. 2.8. Результат вибірки значень у заданому діапазоні

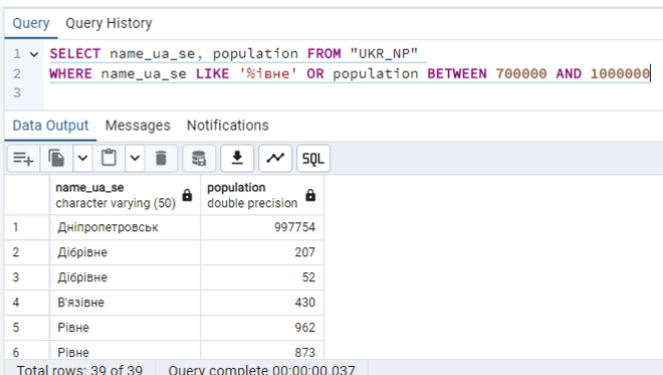
Для об'єднання умов відбору використовуються оператори **AND** та **OR**. Оператор **AND** повертає дані тільки, якщо всі умови виконуються. Оператор **OR** повертає дані, якщо хоча б одна умова виконується.

Наприклад, наведений нижче запит з **AND** не поверне жодного результату, оскільки немає жодного населеного пункту з закінченням «івне», яке має населення у межах від 700000 до 1000000.

```
SELECT name_ua_se, population FROM "UKR_NP"  
WHERE name_ua_se LIKE '%івне' AND population  
BETWEEN 700000 AND 1000000
```

Тоді як наведений нижче запит з **OR** поверне 39 результатів, вивівши всі населені пункти з закінченням «івне» та ті, що мають населення у межах від 700000 до 1000000 (рис. 2.9).

```
SELECT name_ua_se, population FROM "UKR_NP"  
WHERE name_ua_se LIKE '%івне' OR population  
BETWEEN 700000 AND 1000000
```



The screenshot shows a SQL query execution interface. The query is: `SELECT name_ua_se, population FROM "UKR_NP" WHERE name_ua_se LIKE '%івне' OR population BETWEEN 700000 AND 1000000`. The results table has two columns: `name_ua_se` (character varying (50)) and `population` (double precision). The results are as follows:

	name_ua_se	population
1	Дніпропетровськ	997754
2	Дібрівне	207
3	Дібрівне	52
4	В'язівне	430
5	Рівне	962
6	Рівне	873

Total rows: 39 of 39 Query complete 00:00:00.037

Рис. 2.9. Результат вибірки значень з оператором **OR**

При цьому необхідно пам'ятати, що SQL має пріоритети виконання команд. Так оператор **AND** має вищий пріоритет, ніж оператор **OR**, тому при використанні їх в одному запиті спочатку буде перевірятися виконання умови **AND**. Для задання правильної послідовності виконання команд можна

скористатися **дужками**, як у математиці. Тоді, спочатку будуть опрацьовані оператори в дужках, а потім - всі решта.

Наприклад, наведений нижче запит виведе всі населені пункти з назвою Рівне, а також зі співпадінням умов назва – Львів та населення більше 200000. Всього 25 результатів (рис. 2.10).

```
SELECT name_ua_se, population FROM "UKR_NP"  
WHERE name_ua_se = 'Рівне' OR name_ua_se = 'Львів'  
AND population >200000  
ORDER BY name_ua_se ASC;
```

	name_ua_se character varying (50)	population double precision
1	Львів	730272
2	Рівне	873
3	Рівне	169
4	Рівне	76
5	Рівне	667
6	Рівне	354

Total rows: 25 of 25    Query complete 00:00:00.137

Рис. 2.10. Результат вибірки без використання дужок

Якщо необхідно знайти всі населені пункти з назвами Рівне або Львів, які мають понад 200000 населення, то потрібно умову OR взяти в дужки. Результатом такого запиту буде 2 записи (рис. 2.11).

```
SELECT name_ua_se, population FROM "UKR_NP"  
WHERE (name_ua_se = 'Рівне' OR name_ua_se = 'Львів')  
AND population >200000  
ORDER BY name_ua_se ASC;
```

Query Query History

```

1 SELECT name_ua_se, population FROM "UKR_NP"
2 WHERE (name_ua_se = 'Рівне' OR name_ua_se = 'Львів') AND population >200000
3 ORDER BY name_ua_se ASC;

```

Data Output Messages Notifications

	name_ua_se character varying (50)	population double precision
1	Львів	730272
2	Рівне	250333

Рис. 2.11. Результат вибірки з використанням дужок

Також варто згадати поширені функції:

**IS NULL** – для вибірки порожніх записів (WHERE *name\_ua\_se* IS NULL);

**IS NOT NULL** – для вибірки НЕ порожніх записів (WHERE *name\_ua\_se* IS NOT NULL);

**IN** – для вибірки даних, що відповідають хоча б одному з перелічених (WHERE *name\_ua\_se* IN ('Рівне', 'Львів')). Принцип дії подібний до оператора OR, але має ряд переваг: при роботі з довгими списками, речення з IN легше читати; використовується менша кількість операторів, що пришвидшує обробку запиту; можна використовувати додаткову конструкцію SELECT, що відкриває великі можливості для створення складних підзапитів.

**NOT table\_name IN** дозволяє забрати непотрібні значення з вибірки (WHERE NOT *name\_ua\_se* IN ('Рівне', 'Львів')) – обирає населені пункти, крім Рівне і Львова.

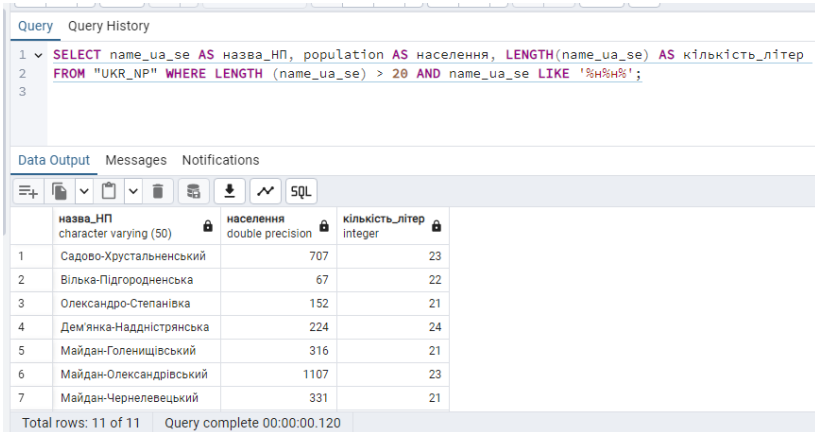
**NOT LIKE** обирає значення, які НЕ відповідають умові (WHERE *name\_ua\_se* NOT LIKE 'Рівне') – обирає населені пункти, крім Рівне.

**LENGTH** – функція, що рахує довжину тексту (WHERE LENGTH (*name\_ua\_se*) > 15) – виведе населені пункти з назвами довше 15 літер.

Для прикладу сформуємо запит, що виведе всі населені пункти з назвою понад 20 літер та у назві яких є дві літери «н» (не обов'язково послідовно). У результаті запиту виведемо стовпці з назвою населеного пункту, кількістю населення та

кількістю літер у назві відобразивши їх імена кирилицею (рис. 2.12).

```
SELECT name_ua_se AS назва_НП, population AS населення, LENGTH(name_ua_se) AS кількість_літер
FROM "UKR_NP" WHERE LENGTH (name_ua_se) > 20
AND name_ua_se LIKE '%н%н%';
```



The screenshot shows a database query interface. At the top, the query is displayed in a text area. Below it, the results are shown in a table with columns for the settlement name, population, and the number of characters in the name. The results are sorted in descending order of the number of characters.

назва_НП	населення	кількість_літер
character varying (50)	double precision	integer
1 Садово-Хрустальненський	707	23
2 Вілька-Підгородненська	67	22
3 Олександро-Степанівка	152	21
4 Дем'янка-Наддністрянська	224	24
5 Майдан-Голенищівський	316	21
6 Майдан-Олександрівський	1107	23
7 Майдан-Чернелевецький	331	21

Total rows: 11 of 11    Query complete 00:00:00.120

Рис. 2.12. Результат вибірки населених пунктів з назвами понад 20 літер та двома літерами «н»

Часто існує потреба у визначенні загальних статистичних даних. Функції, які повертають один результат, називаються "агрегатні" функції:

**SUM()** обчислює сумарне значення;

**COUNT()** обчислює кількість рядків;

**AVG()** обчислює середнє значення;

**STDDEV()** обчислює стандартне відхилення.

Для прикладу сформуємо запит, що обчислює по кожній області (стовпець name\_ua\_\_1) загальну кількість населення, кількість населених пунктів та середню кількість населення у населеному пункті. Імена стовпців у результаті запиту відобразимо під новими назвами та відсортуємо результат за загальною кількістю населення від найбільш населеної області до найменш населеної (рис. 2.13).

```

SELECT name_ua__1 AS назва_області, SUM(population)
AS всього_в_області,
COUNT(name_ua_se) AS кількість_НП, AVG(population)
AS у_середньому_в_НП
FROM "UKR_NP"
GROUP BY name_ua__1
ORDER BY всього_в_області DESC

```

The screenshot shows a database query interface with a 'Query' tab selected. The SQL query is displayed in a text area, and below it, the 'Data Output' tab shows the results of the query in a table. The table has six columns: 'назва\_області' (character varying (100)), 'всього\_в\_області' (double precision), 'кількість\_НП' (bigint), and 'у\_середньому\_в\_НП' (double precision). The results are sorted in descending order of population.

	назва_області character varying (100)	всього_в_області double precision	кількість_НП bigint	у_середньому_в_НП double precision
1	Київська	4677833	1183	3954.2121724429417
2	Донецька	4446161	1301	3417.4950038431975
3	Дніпропетровська	3377090	1503	2246.8995342648036
4	Харківська	2831518	1755	1613.4005698005699
5	Львівська	2616540	1928	1357.1265560165975
6	Одеська	2459801	1177	2089.8903993203057

Рис. 2.13. Результат вибірки з використанням статистичних функцій

Запити можуть бути вкладеними. Наприклад, отримаємо список населених пунктів Рівненської області з населенням більшим за середнє значення по області (рис. 2.14).

```

SELECT name_ua_se, population FROM "UKR_NP"
WHERE name_ua__1 LIKE 'Рівненська' AND population >
(SELECT AVG(population) FROM "UKR_NP" WHERE
name_ua__1 LIKE 'Рівненська')
ORDER BY name_ua_se

```

У випадку не правильно сформованого запиту або наявності синтаксичних помилок – виводиться відповідне повідомлення з описом помилки (рис. 2.15).

Query Query History

```

1 SELECT name_ua_se, population FROM "UKR_NP"
2 WHERE name_ua__1 LIKE 'Рівненська' AND population >
3 (SELECT AVG(population) FROM "UKR_NP" WHERE name_ua__1 LIKE 'Рівненська')
4 ORDER BY name_ua_se
5

```

Data Output Messages Notifications

	name_ua_se character varying (50)	population double precision
1	Антонівка	1405
2	Бабин	2367
3	Балашівка	1735
4	Бережки	1630
5	Березне	13444
6	Березове	2599
7	Берестя	2719

Total rows: 165 of 165 Query complete 00:00:00.080

Рис. 2.14. Результат вибірки з вкладеним запитом

Query Query History

```

1 SELECT name_ua_se, population FROM "UKR_NP"
2 WHERE name_ua__1 LIKE 'Рівненська' AND population >;
3 (SELECT AVG(population) FROM "UKR_NP" WHERE name_ua__1
4 ORDER BY name_ua_se
5

```

Data Output Messages Notifications

ERROR: синтаксична помилка в або поблизу ";"  
LINE 2: WHERE name\_ua\_\_1 LIKE 'Рівненська' AND population >;  
A

ПОМИЛКА: синтаксична помилка в або поблизу ";"  
SQL state: 42601  
Character: 98

Query Query History

```

1 SELECT name_ua_se, population FROM "UKR_NP"
2 WHERE name_ua__1 LIKE 'Рівненська' AND population >
3 (SELECT AVG(population) FROM "UKR_NP" WHERE name_ua__1
4 ORDER BY name_ua_s
5

```

Data Output Messages Notifications

ERROR: стовпець "name\_ua\_s" не існує  
LINE 4: ORDER BY name\_ua\_s  
A

HINT: Можливо, передбачалося посилання на стовпець "UKR\_NP.name\_u

ПОМИЛКА: стовпець "name\_ua\_s" не існує  
SQL state: 42703  
Character: 183

Рис. 2.15. Приклади повідомлень про наявність помилок у запиті

## 2) Запити при роботі з геометричними даними

Специфікація OpenGIS визначає два стандартні способи опису просторових об'єктів – Well-Known Text (WKT) та Well-Known Binary (WKB), які містять інформацію про тип і координати об'єкта. Формат WKT є зрозумілим людині, тоді як WKB є машинно-орієнтованим. Дані у СУБД PostgreSQL/Postgis за замовчуванням відображаються у форматі WKB.

Для прикладу, точка з координатами 1.0,1.0 у форматі WKT описується як POINT(1 1), а у форматі WKB має вигляд 0101000000000000000000F03F000000000000F03

Згідно стандарту – функції для роботи з просторовими даними починаються з префікса `ST_`.

Наприклад, функції для конвертації координат у поширені формати мають вигляд:

**`ST_AsText()`** – відображає координати у форматі WKT;

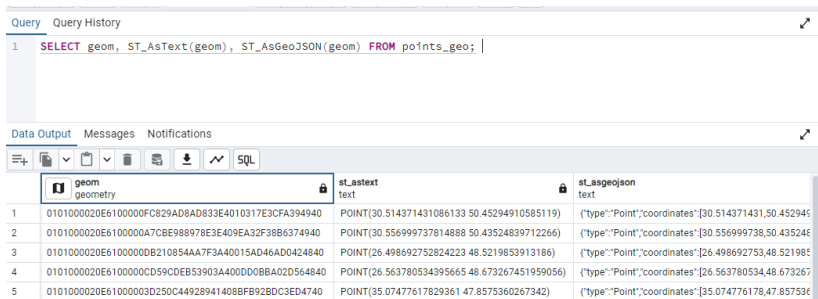
**`ST_AsGML()`** – відображає координати у форматі GML;

**`ST_AsKML()`** – відображає координати у форматі KML;

**`ST_AsGeoJSON()`** – відображає координати у форматі GeoJSON.

Для наочного порівняння переглянемо інформацію про геометрію точкових об'єктів з таблиці `point_geo` (створеної у минулій лабораторній роботі) у різних форматах (WKB, WKT, GeoJSON) (рис. 2.16).

```
SELECT geom, ST_AsText(geom), ST_AsGeoJSON(geom)
FROM points_geo;
```



	geom geometry	st_astext text	st_asgeojson text
1	0101000020E6100000FC829AD8A83E4010317E3CFA394940	POINT(30.514371431086133 50.45294910585119)	{"type":"Point","coordinates":[30.514371431,50.452945
2	0101000020E6100000A7CBE9889787E3E409EA32F38B6374940	POINT(30.556999737814888 50.43524839712266)	{"type":"Point","coordinates":[30.556999738,50.435248
3	0101000020E6100000DB210854AA7F34A0015AD46AD0424840	POINT(26.498692752824223 48.5219853913186)	{"type":"Point","coordinates":[26.498692753,48.521985
4	0101000020E6100000CD59CDEB53903A400D08BA02D0564840	POINT(26.563780534395665 48.673267451959056)	{"type":"Point","coordinates":[26.563780534,48.673267
5	0101000020E61000003D250C44928941408BF8928DC3ED4740	POINT(35.07477617829361 47.8575360267342)	{"type":"Point","coordinates":[35.074776178,47.857536

Рис. 2.16. Результат вибірки даних про геометрію

Як видно з результатів стовпця `st_astext` (рис. 2.16) координати об'єктів відображаються у градусній мірі. Однак, залишається питання про систему координат, яка використана. Переглянути дані про ідентифікатор системи координат таблиці з геометрією можна за допомогою функції `ST_SRID()` (рис. 2.17).

```
SELECT ST_SRID(geom) FROM points_geo LIMIT 1
```

*Примітка: функція `LIMIT` використана, щоб обмежити кількість результатів для виведення, оскільки всі об'єкти у таблиці збережені в однаковій системі координат.*



st_srid	integer
1	4326

Рис. 2.17. Результат вибірки даних про систему координат таблиці

Координати об’єктів можна перераховувати в інші системи. Трансформацію координат в іншу систему “на льоту” можна виконати за допомогою функції `ST_Transform()`. В якості вхідних даних для неї потрібно задати назву стовпця з геометрією та код системи координат у яку потрібно перерахувати координати (рис. 2.18).

```
SELECT ST_AsText(geom) AS original,
       ST_AsText(ST_Transform(geom, 3857)) AS transformed
FROM points_geo
```

	original text	transformed text
1	POINT(30.514371431086133 50.45294910585119)	POINT(3396844.2895853226 6525091.329990164)
2	POINT(30.556999737814888 50.43524839712266)	POINT(3401589.6509837476 6521997.205094072)
3	POINT(26.498692752824223 48.5219853913186)	POINT(2949820.9839318013 6194138.133692451)
4	POINT(26.563780534395665 48.673267451959056)	POINT(2957066.5226331977 6219602.470619949)
5	POINT(35.07477617829361 47.8575360267342)	POINT(3904506.223855687 6083186.546526557)

Рис. 2.18. Результат перерахунку координат “на льоту”

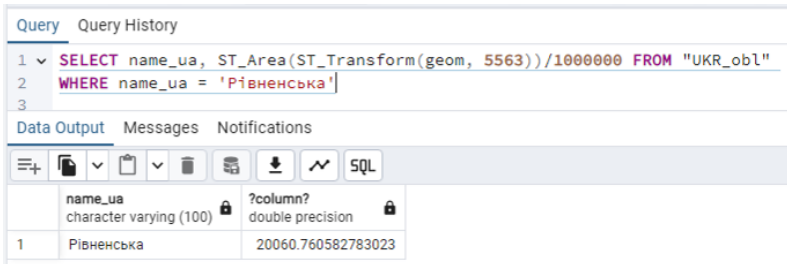
При потребі можна додати новий стовпець з геометрією у таблицю та помістити у нього координати перераховані у потрібну систему.

```
SELECT AddGeometryColumn ('points_geo','geom3857',
                          3857, 'POINT', 2);
UPDATE points_geo SET geom3857 = ST_Transform
(geom,3857);
```

У SQL доступний великий набір функцій для роботи з геометрією. Розглянемо приклади з використанням найпоширеніших з них.

Розрахувати площу Рівненської області можна використовуючи функцію обчислення площі **ST\_Area()**. При цьому ми виконали перерахунок координат з системи WGS84 (EPSG:4326) в УСК2000 5 зона (EPSG:5563) та з м<sup>2</sup> у км<sup>2</sup> (рис. 2.19).

```
SELECT name_ua, ST_Area(ST_Transform(geom,
5563))/1000000 FROM "UKR_obl"
WHERE name_ua = 'Рівненська'
```



The screenshot shows a SQL query execution window with the following content:

```
Query Query History
1 SELECT name_ua, ST_Area(ST_Transform(geom, 5563))/1000000 FROM "UKR_obl"
2 WHERE name_ua = 'Рівненська'
3
```

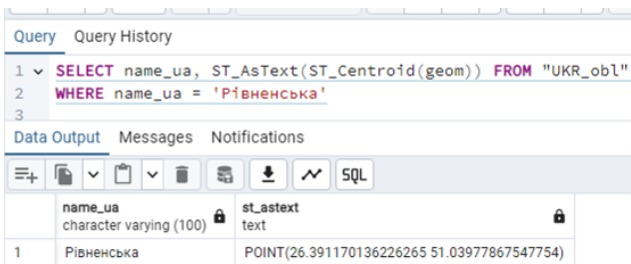
Below the query is a table with the following data:

	name_ua character varying (100)	?column? double precision
1	Рівненська	20060.760582783023

Рис. 2.19. Результат обчислення площі області

Знайти геометричний центр Рівненської області можна за допомогою функції **ST\_Centroid()** (рис. 2.20).

```
SELECT name_ua, ST_AsText(ST_Centroid(geom)) FROM
"UKR_obl"
WHERE name_ua = 'Рівненська'
```



The screenshot shows a SQL query execution window with the following content:

```
Query Query History
1 SELECT name_ua, ST_AsText(ST_Centroid(geom)) FROM "UKR_obl"
2 WHERE name_ua = 'Рівненська'
3
```

Below the query is a table with the following data:

	name_ua character varying (100)	st_astext text
1	Рівненська	POINT(26.391170136226265 51.03977867547754)

Рис. 2.20. Результат обчислення геометричного центру області

Обчислити довжину об'єктів на карті можна за допомогою функції **ST\_Length()**. Для прикладу визначимо довжину річок (у кілометрах) з відповідної таблиці БД "UKR\_water\_lines\_dcw" (рис. 2.21).

```
SELECT nam, ST_Length(ST_Transform(geom,5563))/1000
FROM "UKR_water_lines_dcw"
```

	nam	?column?
	character varying (254)	double precision
1	Припять	230.97745502510386
2	Уж	223.89834948152233
3	Стохід	166.07826763059433
4	Турія	181.3330971363066
5	Зах. Буг	354.29605350956837
6	Горинь	455.478066067256

Рис. 2.21. Результат визначення довжини річок

Для визначення відстані між точками використовується функція **ST\_Distance(geometry1, geometry2)**. Для прикладу визначимо відстань з Рівного до Києва (у метрах) використовуючи таблицю з межами населених пунктів "UKR\_NP". При цьому необхідно прописати вкладені запити на визначення координат центрів Рівного та Києва, й перерахувати координати у метричну проекцію (рис. 2.22).

```
SELECT ST_Distance(rr, kk) FROM
  (SELECT ST_Transform(ST_Centroid(geom),5563) AS rr
   FROM "UKR_NP"
   WHERE name_ua_se = 'Рівне' AND population >200000)
AS t1,
  (SELECT ST_Transform(ST_Centroid(geom),5563) AS kk
   FROM "UKR_NP"
   WHERE name_ua_se = 'Київ' AND population >200000)
AS t2
```

Query Query History

```

1 SELECT ST_Distance(rr, kk) FROM
2 (SELECT ST_Transform(ST_Centroid(geom),5563) AS rr FROM "UKR_NP"
3 WHERE name_ua_se = 'Рівне' AND population >200000) AS t1,
4 (SELECT ST_Transform(ST_Centroid(geom),5563) AS kk
5 FROM "UKR_NP"
6 WHERE name_ua_se = 'Київ' AND population >200000) AS t2

```

Data Output Messages Notifications

	st_distance	
	double precision	
1	305637.1954383909	

Рис. 2.22. Результат визначення відстані між населеними пунктами

Для знаходження найзахіднішої пам'ятки з таблиці "points\_geo" можна використати функцію **ST\_X()** (рис. 2.23).

```

SELECT pamyatka_n, ST_X(geom) FROM points_geo
ORDER BY ST_X(geom) LIMIT 1

```

Query Query History

```

1 SELECT pamyatka_n, ST_X(geom) FROM points_geo
2 ORDER BY ST_X(geom) LIMIT 1
3

```

Data Output Messages Notifications

	pamyatka_n	st_x
	character varying (254)	double precision
1	Хотинська фортеця	26.498692752824223

Рис. 2.23. Результат пошуку найзахіднішої пам'ятки

Для визначення населеного пункту у якому розташовується Хотинська фортеця можна скористатися функцією **ST\_Contains(geometry1, geometry2)**. Знову ж таки скористаємося вкладеним запитом для визначення координат фортеці (рис. 2.24).

```

SELECT name_ua_se FROM "UKR_NP"
WHERE ST_Contains(geom,
(SELECT ST_Transform(geom, 4326) FROM points_geo
WHERE pamyatka_n = 'Хотинська фортеця'));

```

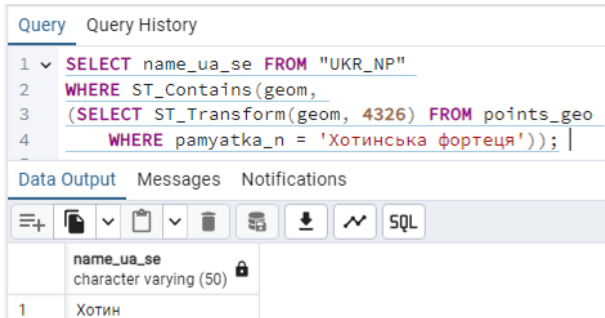


Рис. 2.24. Результат пошуку населеного пункту у якому міститься Хотинська фортеця

Цю ж дію можна виконати розділивши на дві частини. Спочатку через запит знайти координати Хотинської фортеці, а потім знайти населений пункт, який містить ці координати скориставшись функцією **ST\_GeomFromText()**.

```

SELECT name_ua_se FROM "UKR_NP"
WHERE ST_Contains(geom,
ST_GeomFromText('POINT(26.498692752824223
48.5219853913186)', 4326));

```

Визначимо які населені пункти знаходяться на відстані 5 км від Хотинської фортеці. Для цього підійде функція **ST\_Dwithin(geometry1, geometry2, distance)**. Щоб не ускладнювати запит використаємо вже відомі координати фортеці у системі WGS84 'POINT(26.498692752824223 48.5219853913186)' (рис. 2.25).

```

SELECT name_ua_se FROM "UKR_NP"
WHERE ST_Dwithin(ST_Transform(geom,5563),
ST_Transform(ST_GeomFromText('POINT(26.4986927528
24223 48.5219853913186)',4326),5563), 5000);

```

Змінивши одну складову у наведеному запиті можна визначити скільки людей живе у населених пунктах, що знаходяться на відстані 5 км від Хотинської фортеці (рис. 2.26).

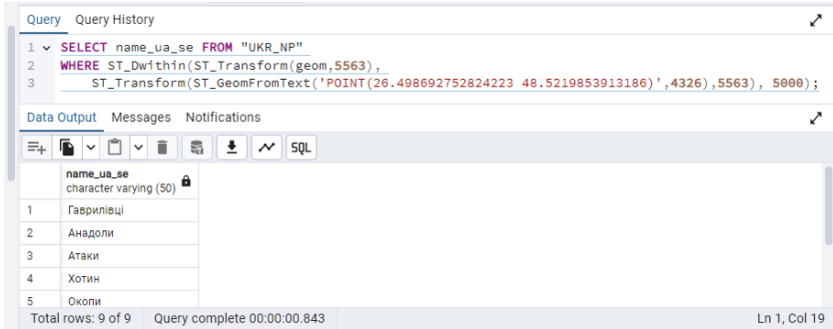


Рис. 2.25. Результат пошуку населених пунктів на відстані 5000 метрів від Хотинської фортеці

```

SELECT SUM(population) FROM "UKR_NP"
WHERE ST_Dwithin(ST_Transform(geom,5563),
ST_Transform(ST_GeomFromText('POINT(26.49869275282
4223 48.5219853913186)',4326),5563), 5000);

```

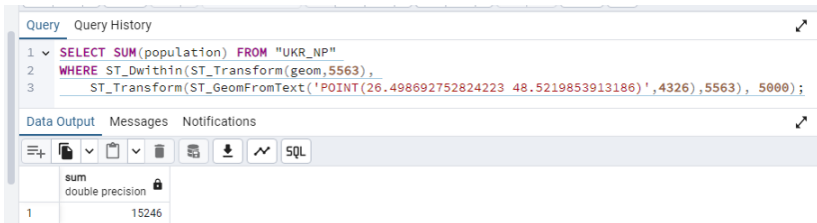


Рис. 2.26. Результат обчислення кількості населення у населених пунктах на відстані 5000 метрів від Хотинської фортеці

Визначимо які області перетинають 26 меридіан. Для цього підійде функція **ST\_Intersects(geometry1, geometry2)** (рис. 2.27).

```

SELECT name_ua FROM "UKR_obl"
WHERE ST_Intersects(geom,
ST_GeomFromText('LINESTRING(26 20, 26 70)',4326))


```



Рис. 2.27. Результат визначення областей, які перетинають 26 меридіан

### 3) SQL запити до бази даних через інтерфейс QGIS

При роботі з базами даних через інтерфейс QGIS теж можна використовувати SQL запити. В першу чергу необхідно встановити підключення до БД, як описано у питанні 6 минулої лабораторної роботи (рис. 1.13).

Далі у головному вікні QGIS вибираємо пункт меню **База даних / Менеджер БД**. У лівій частині вікна розкриваємо каталог PostGIS у якому відобразиться створене нами з'єднання з переліком доступних з БД таблиць (рис. 1.14). На панелі інструментів вікна **Управління БД** натискаємо кнопку **Вікно SQL** .

Відкривається вкладка **Запит** з інтерфейсом схожим на редактор запитів PostgreSQL/Postgis – у верхній частині вводим текст запиту, у нижній – одержуємо результат (рис. 2.28). Крім того доступний перемикач **Завантажити як новий шар**. Активація цієї опції дає можливість **Завантажити** результат запиту як окремий шар на карту QGIS (рис. 2.29). При бажанні результуючий шар можна зберегти в окремий файл скориставшись командою з контекстного меню **Експорт / Зберегти об'єкти як**.

**Завдання для самостійної роботи.** Розгляньте інші можливості оператора SELECT та інші просторові функції.

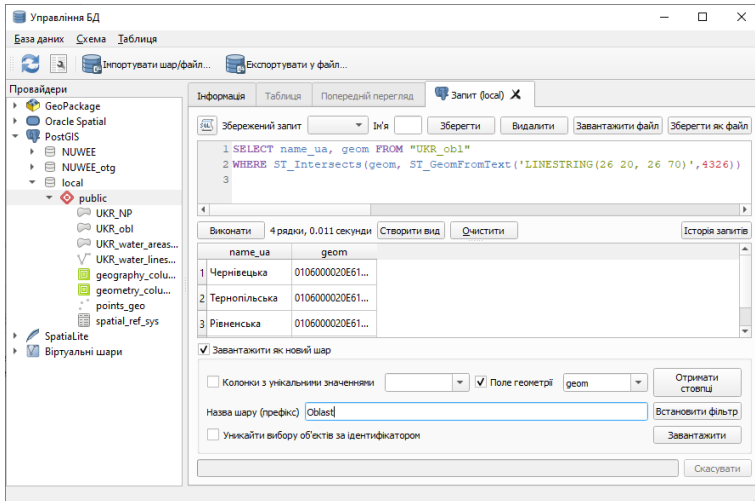


Рис. 2.28. Інтерфейс редактора SQL запитів у QGIS

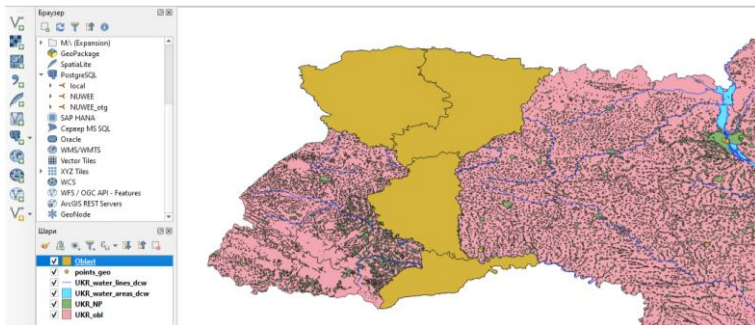


Рис. 2.29. Відображення результату запиту на карті

### Запитання для контролю:

1. Що таке SQL?
2. Як називається оператор SQL для вибірки даних?
3. Якими умовами можна доповнювати оператор Select?
4. Чи підтримуються вкладені запити?
5. Як виконувати запити у середовищі PostgreSQL/Postgis?
6. Як ви знаєте функції для роботи з геометричними об'єктами?
7. Як виконувати запити у середовищі QGIS?
8. Як відобразити результат запиту на карті?



## Лабораторна робота № 7

### Редагування структури й вмісту бази даних та налаштування прав доступу

**Мета:** навчитись редагувати структуру бази даних у PostgreSQL/Postgis, її наповнення та керувати правами доступу.

**Завдання:** додати/видалити поля таблиці засобами SQL, pgAdmin та QGIS; внести/змінити дані у БД, що відповідають певній умові; створити трьох користувачів БД та надати їм права: на повний доступ до даних; на перегляд даних і додавання нових даних (але без можливості видалення існуючих даних); лише на перегляд даних.

#### 1) Редагування структури та вмісту бази даних

Команди для роботи зі структурою бази даних дозволяють створювати і змінювати структуру об'єктів бази даних (таблиця, схема, індекс тощо): CREATE – створення об'єкта; ALTER – зміна об'єкта; DROP – видалення об'єкта.

Команди для роботи з записами даних використовуються для маніпулювання інформацією всередині об'єктів бази даних: INSERT — вставка; UPDATE — зміна; DELETE — видалення.

Розглянемо приклади SQL запитів із використанням цих команд.

Запит на **створення бази даних** містить команду **CREATE**, тип об'єкта який створюється та його назву, наприклад:

```
CREATE DATABASE pubgd;
```

При потребі запит може бути доповнений додатковими характеристиками, наприклад власник та його ім'я, бажаний тип кодування:

```
CREATE DATABASE pubgd OWNER postgres ENCODING  
'UTF8';
```

Також за допомогою команди можна **CREATE** створювати інші об'єкти бази даних, наприклад **схеми** або

таблиці. Перед виконанням запитів слід пересвідчитися, що запит виконується до потрібної БД/схеми/таблиці...

```
CREATE SCHEMA test AUTHORIZATION postgres;
```

При створенні таблиці необхідно крім типу створюваного об'єкта та його імені задати імена полів, їх типи даних та особливості (якщо назва таблиці чи атрибуту містить великі літери – назва береться у подвійні лапки).

```
CREATE TABLE "Pamyatka" (  
    "Id" integer PRIMARY KEY,  
    geom geometry,  
    "Nazva_pam" varchar(200) NOT NULL,  
    tup varchar(50),  
    dod_info varchar(250),  
    foto bytea )
```

*Примітка: щоб мати можливість використовувати поля типу geometry спочатку необхідно перетворити базу на просторову запитом create extension postgis; для текстових полів змінної довжини varchar у дужках слід зазначити бажану довжину поля.*

Для видалення об'єктів бази даних використовується команда **DROP**. Необхідно зазначити тип об'єкта та його назву, наприклад:

```
DROP DATABASE pubgd;
```

або

```
DROP TABLE "Pamyatka"
```

Для внесення даних у таблицю використовується команда **INSERT**. Необхідно зазначити назву таблиці у яку слід додавати дані, назви полів та значення для введення. Якщо у якесь поле не потрібно вносити значення, то його не згадуємо у переліку. Назви ідентифікаторів БД (таблиця, стовпці), якщо вони містять великі літери – беруться у подвійні лапки; значення, що вносяться виділяються *одинарними лапками* незалежно від вмісту.

```
INSERT INTO "Pamyatka"  
("Id", geom, "Nazva_pam", tup)  
VALUES  
(3, 'POINT(26.498692752824223 48.5219853913186)',  
'Хотинська фортеця', 'архітектура');
```

Для редагування властивостей об'єктів бази даних використовується команда **ALTER**. Спектр її застосування в комбінації з іншими операторами доволі широкий. Розглянемо деякі варіанти.

**Перейменування таблиці RENAME** у базі даних – зазначається існуюче та бажане нове ім'я таблиці.

```
ALTER TABLE "Pamyatka" RENAME TO pamyatka;
```

**Перейменування поля RENAME** у таблиці – зазначається ім'я таблиці, яка містить поле, а також існуюче та бажане нове ім'я поля.

```
ALTER TABLE pamyatka  
RENAME COLUMN tup TO tup_pamyatku;
```

**Додавання нового поля ADD** у таблицю – зазначається ім'я таблиці, а також ім'я нового поля та його тип даних.

```
ALTER TABLE pamyatka  
ADD prumitka CHARACTER VARYING(100);
```

Якщо при додаванні нового поля необхідно наголосити, що значення не може бути пустим додається обмеження.

```
ALTER TABLE pamyatka  
ADD prumitka CHARACTER VARYING(100) NOT NULL;
```

Якщо у таблиці вже є записи, то наведений вище запит видасть помилку. Потрібно при додаванні нового поля призначити значення за замовчуванням **DEFAULT** для всіх новостворених значень.

```
ALTER TABLE pamyatka  
ADD prumitka1 CHARACTER VARYING(100) NOT NULL  
DEFAULT 'Невідомо';
```

**Видалення поля DROP** у таблиці – зазначається ім'я таблиці та ім'я поля.

```
ALTER TABLE пам'ятка
DROP COLUMN примітка1;
```

**Зміна типу даних поля** – зазначається ім'я таблиці, яка містить поле, ім'я поля та бажаний тип даних.

```
ALTER TABLE пам'ятка
ALTER COLUMN примітка TYPE integer;
```

Якщо у таблиці вже є записи, то наведений вище запит видасть помилку. Потрібно при **зміні типу поля** додати вираз **USING**.

```
ALTER TABLE пам'ятка
ALTER COLUMN примітка TYPE integer USING
(примітка::integer);
```

З застосуванням виразу **USING** також виконується **приєднання даних з інших стовпців** – зазначається ім'я таблиці, ім'я поля та його тип даних, а також імена полів, значення яких буде об'єднуватися та розділовий знак між ними.

```
ALTER TABLE пам'ятка
ALTER COLUMN дод_інфо TYPE varchar(250) USING
(дод_інфо || '-' || примітка);
```

**Додавання обмеження для існуючого поля**, наприклад, що воно **НЕ може бути пустим**.

```
ALTER TABLE пам'ятка
ALTER COLUMN geom SET NOT NULL;
```

**Видалення обмеження для існуючого поля**, наприклад, що воно **може бути пустим**.

```
ALTER TABLE пам'ятка
ALTER COLUMN geom DROP NOT NULL;
```

Додавання обмеження для існуючого поля, наприклад, перевірка певної умови CHECK (в даному прикладі поле «Вік пам'ятки Age» має бути більше нуля).

```
ALTER TABLE patuyatka  
ADD CHECK (Age > 0);
```

Додавання обмеження для існуючого поля, наприклад, перевірка унікальності значення поля UNIQUE.

```
ALTER TABLE patuyatka  
ADD UNIQUE ( "Nazva_pam" );
```

Оновлення значень у полі виконується за допомогою команди UPDATE. При цьому необхідно зазначити умову, при якій буде відбуватися оновлення, а також ім'я поля, значення якого буде змінюватися. Можна одночасно змінювати значення для декількох полів перелічивши їх через кому після оператора SET. Наведений запит оновить значення у полі *prumitka* на '*пам`ятка старовини*' для тих записів, у яких значення поля *Age* перевищує 100 років.

```
UPDATE patuyatka  
SET prumitka = 'пам`ятка старовини'  
WHERE Age > 100;
```

Якщо необхідно змінити значення всіх записів у таблиці, то не потрібно зазначати умову WHERE. У такому випадку для всіх записів таблиці значення у полі *prumitka* буде змінено на '*пам`ятка старовини*'.

```
UPDATE patuyatka  
SET prumitka = 'пам`ятка старовини'
```

Більшість розглянутих функцій доступна безпосередньо через засоби pgAdmin4 з контекстного меню бази даних або таблиці. А також з відповідних контекстних меню через інтерфейс QGIS при роботі з таблицями баз даних.

## 2) Налаштування прав доступу до бази даних

У базах даних можна створювати нових користувачів та призначати їм права доступу. Знову ж таки така можливість доступна через інтерфейс команд pgAdmin або безпосередньо через SQL запити. Розглянемо декілька прикладів додавання нових користувачів та присвоєння їм прав доступу через редактор SQL.

Для **створення нового користувача** використовується вже відома нам команда **CREATE**. Необхідно зазначити ім'я нового користувача та пароль для нього. У наведеному запиті створюється (*create*) новий користувач (*user*) *user1* з паролем доступу '*111*'.

```
CREATE USER user1 WITH PASSWORD '111';
```

Можна створювати користувача з паролем доступу, який буде **дійсний лише до заданого терміну**.

```
CREATE USER user2 WITH PASSWORD '222' VALID UNTIL '2026-01-01';
```

Для **налаштування прав доступу** слід використовувати команду **GRANT**. У запиті необхідно зазначити тип прав, які надаються, тип та ім'я об'єкта для доступу та ім'я користувача який отримує права.

```
GRANT SELECT ON TABLE pamyatka TO user1;
```

У даному прикладі надаються права (*grant*) тільки на читання (перегляд) даних (*select*) з таблиці *pamyatka* користувачу *user1*.

Основні типи прав доступу до даних у таблиці наступні:

**SELECT** – можливість перегляду існуючих записів;

**UPDATE** – можливість редагування (зміни) вже існуючих записів;

**INSERT** – можливість додавання нових записів;

**DELETE** – можливість видалення існуючих записів;

**ALL** – всі можливі зміни.

Важливо зауважити, що якщо ми хочемо надати користувачу можливість редагування якогось шару просторових

даних, то крім доступу на редагування (*update*) слід дати йому доступ і на перегляд даних (*select*), інакше шар для редагування об'єктів може бути недоступний для перегляду зі сторонніх ГІС програм. Теж стосується комбінації інших видів прав. У такому випадку запит на можливість редагування даних матиме вигляд:

```
GRANT SELECT ON TABLE pamyatka TO user1;  
GRANT UPDATE ON TABLE pamyatka TO user1;
```

Крім того, для прав, що потребують створення або видалення об'єктів, тобто зміни їх кількості, необхідно дати доступ на використання послідовності створеної на основі потрібної таблиці.

```
GRANT USAGE ON SEQUENCE pamyatka_Id_seq TO  
user1;
```

Назву послідовності для потрібної таблиці можна переглянути у відповідному розділі браузера об'єктів у pgAdmin.

Щоб мати можливість створювати нові шари з просторовими об'єктами, або видаляти чи редагувати їх, працювати з системами координат, слід дати права доступу на таблиці з метаданими:

```
GRANT ALL ON geometry_columns TO user1;  
GRANT ALL ON spatial_ref_sys TO user1;
```

Для видалення наданих прав доступу слід використовувати команду **REVOKE**. У запиті необхідно зазначити тип прав, які видаляються, тип та ім'я об'єкта з доступом та ім'я користувача у якого забираються права.

```
REVOKE SELECT ON TABLE pamyatka TO user1;
```

Можна видалити відразу всі надані права на конкретну таблицю.

```
REVOKE ALL ON TABLE pamyatka FROM user1;
```

У випадку наявності багатьох таблиць у БД є можливість видалити права відразу на всі таблиці у конкретній схемі даних.

```
REVOKE ALL ON ALL TABLES IN SCHEMA public
FROM user2;
```

**Переглянути список всіх користувачів (ролей)** на сервері можна за допомогою запити

```
SELECT rolname FROM pg_roles;
```

**Переглянути права конкретного користувача** можна за допомогою запити

```
SELECT table_catalog, table_schema, table_name,
privilege_type, grantee
FROM information_schema.table_privileges
WHERE grantee = 'user1';
```

### **3) Корисні SQL запити**

Крім того в нагоді можуть стати наступні SQL запити.

Дізнатися **розмір бази даних** у “людиночитабельному” форматі:

```
SELECT pg_size_pretty (pg_database_size('pubgd'));
```

Замість імені бази даних можна використати функцію **current\_database()**, яка повертає розмір поточної бази даних.

Дізнатися **розмір таблиці** у “людиночитабельному” форматі:

```
SELECT pg_size_pretty(pg_relation_size('pamyatka'))
```

Вивести **перелік таблиць** у БД:

```
SELECT table_name FROM information_schema.tables
WHERE table_schema NOT IN ('information_schema',
'pg_catalog');
```

Порахувати **кількість записів у таблиці**:

```
SELECT count(*) FROM pamyatka;
```

Порахувати **кількість записів у таблиці** при умові, що **задане поле не містить NULL**:

```
SELECT count("Nazva_pam") FROM pamyatka;
```



Порахувати кількість унікальних записів у таблиці по заданому полю:

```
SELECT count(distinct "Nazva_pam") FROM pamyatka;
```

Змінити систему координат таблиці:

```
SELECT UpdateGeometrySRID ('public', 'pamyatka', 'geom',  
3857);
```

**Завдання для самостійної роботи.** У лабораторній роботі розглянуто редагування структури й вмісту бази даних та налаштування прав доступу з використанням SQL запитів. Розгляньте можливості таких налаштувань використовуючи засоби pgAdmin4 та QGIS.

**Запитання для контролю:**

1. Які дії можна виконати за допомогою команди CREATE?
2. Які дії можна виконати за допомогою команди ALTER?
3. Які дії можна виконати за допомогою команди INSERT?
4. Які дії можна виконати за допомогою команди UPDATE?
5. Яка різниця між використанням команд DROP та DELETE?
6. Яка команда надає права доступу користувачу?
7. Яка команда видаляє права доступу користувачу?
8. Як надати право на редагування даних у таблиці?
9. Як надати право на додавання нових записів у таблицю?
10. Як переглянути права користувача у БД?
11. Як дізнатися розмір бази даних?
12. Як дізнатися кількість записів у таблиці?
13. Як дізнатися кількість унікальних записів у таблиці?
14. Як змінити систему координат для таблиці?

## РЕКОМЕНДОВАНА ЛІТЕРАТУРА ТА РЕСУРСИ

1. Геоінформаційні системи і бази даних : монографія / В. І. Зацерковний, В. Г. Бурачек, О. О. Железняк, А. О. Терещенко. Ніжин : НДУ ім. М. Гоголя, 2014. 492 с.
2. Геоінформаційні технології та інфраструктура геопросторових даних: у шести томах. Том 2: Системи керування базами геоданих для інфраструктури просторових даних : навчальний посібник / Кейк Д., Лященко А. А., Путренко В. В., Хмелевський Ю., Дорошенко К. С., Говоров М. К. : Планета-Прінт, 2017. 456 с.
3. Методичні вказівки до виконання лабораторних робіт з дисципліни «Побудова та управління банками геоінформації» студентами напряму підготовки 6.080101 «Геодезія, картографія та землеустрій» професійного спрямування «Геоінформаційні системи і технології» Частина 2 «Робота з геопросторовими даними» 05-04-40 / О. Є. Янчук, Н. В. Левчук, А. В. Прокопчук. Рівне : НУВГП, 2016. 56 с. URL: <http://ep3.nuwm.edu.ua/4293/>
4. Основи створення інтегрованих геопросторових даних. / Ю. О. Карпінський та ін. Київ : КНУБА, 2023. 302 с.
5. Open Geospatial Consortium. Standards. Simple Feature Access – Part 1: Common Architecture URL: <http://www.opengeospatial.org/standards/sfa>
6. OpenStreetMap Україна. Вільна мапа — створювати яку може кожен. URL: <https://openstreetmap.org.ua/>
7. Postgis. Documentation URL: <https://postgis.net/documentation/>
8. QGIS - провідна вільна настільна ГІС. URL: <https://www.qgis.org/uk/site/about/index.html>
9. SQL. URL: <https://uk.wikipedia.org/wiki/SQL>