Повний Посібник з Оператора SELECT у PostgreSQL: Від Основ до Аналізу Даних

Вступ: Ваш Перший Крок у Світ Даних з PostgreSQL

Цей посібник створений як вичерпний ресурс для новачків, які прагнуть опанувати оператор SELECT у PostgreSQL. Мета полягає в тому, щоб провести читача від найпростіших запитів на вибірку даних до складніших технік фільтрації, агрегації та аналізу. Посібник розроблено для аудиторії без попереднього досвіду роботи з SQL, пропонуючи практичні приклади та чіткі пояснення на кожному етапі.

SQL (Structured Query Language) — це стандартна мова, що використовується для взаємодії з реляційними базами даних. Вона дозволяє створювати, оновлювати, видаляти та, що найважливіше, витягувати дані. PostgreSQL — це потужна, безкоштовна, об'єктно-реляційна система управління базами даних (СУБД), відома своєю надійністю та широким набором функцій.[1] У цьому контексті оператор

SELECT є фундаментальним інструментом, оскільки саме він відповідає за отримання інформації з бази даних.[2]

Для початку роботи та виконання прикладів з цього посібника необхідно підключитися до сервера PostgreSQL. Найпростіший спосіб зробити це — використати термінальний клієнт psql. Для підключення під стандартним користувачем postgres відкрийте термінал і введіть команду [4]:

Bash

```
psql -U postgres
```

Система попросить вас ввести пароль. Після успішного входу ви побачите запрошення командного рядка, що виглядає як postgres=#. Далі, щоб працювати з конкретною базою даних (наприклад, з назвою dvdrental), використовуйте мета-команду \c [4]:

```
SQL
```

```
\c dvdrental
```

Тепер ваше середовище налаштоване, і ви готові виконувати SQL-запити.

Розділ 1: Основи Оператора SELECT — Отримання Даних

1.1. Анатомія Запиту: Ключові слова SELECT та FROM

В основі будь-якого запиту на вибірку даних лежить проста та логічна структура, що складається з двох ключових слів: SELECT та FROM. Ця конструкція є фундаментом для отримання інформації з бази даних.[2]

- SELECT визначає, *які* дані (стовпці) необхідно повернути.
- FROM вказує, *звідки* ці дані потрібно взяти (з якої таблиці).

Загальний синтаксис виглядає так: SELECT [список стовпців] FROM [назва таблиці];. Наприклад, щоб отримати імена всіх клієнтів з таблиці customer, запит буде таким: SELECT first_name FROM customer;. Важливо зазначити, що крапка з комою (;) в кінці не є частиною самої SQL-команди, а слугує сигналом для PostgreSQL про завершення запиту, що дозволяє виконувати кілька команд послідовно.[4]

1.2. Вибірка Конкретних та Всіх (*) Стовпців

Залежно від завдання, можна вибирати один, кілька або всі стовпці з таблиці.

- Вибірка одного стовпця: Як показано вище, для отримання даних лише з одного стовпця, його назва вказується після SELECT.
  SELECT first_name FROM customer;.4
- Вибірка кількох стовпців: Щоб отримати дані з кількох стовпців, їхні назви перераховуються через кому.
  SELECT first_name, last_name, email FROM customer;.4
- **Вибірка всіх стовпців:** Для вибірки всіх стовпців з таблиці використовується символ зірочки (*). Це скорочення, яке замінює перерахування всіх назв стовпців вручну.[4]

  SELECT * FROM weather;.6

Хоча SELECT * є зручним інструментом для швидкого перегляду структури та вмісту таблиці під час розробки, його використання в робочому (production) коді вважається поганою практикою. По-перше, це призводить до вибірки зайвих даних, що збільшує навантаження на мережу та базу даних, сповільнюючи запит.[3] По-друге, це створює приховану залежність від структури таблиці. Якщо в майбутньому до таблиці буде додано новий стовпець (наприклад, з конфіденційною інформацією), застосунок, що використовує

SELECT *, автоматично почне отримувати ці дані, що може призвести до непередбачуваної поведінки та проблем з безпекою. Явне перерахування стовпців робить код більш надійним, самодокументованим та стійким до змін у схемі бази даних.

1.3. Використання Псевдонімів (AS) для Покращення Читабельності

Часто назви стовпців у базі даних є технічними або незручними для читання. Крім того, при виконанні обчислень у запиті результуючий стовпець може не мати зрозумілої назви. Для вирішення цієї проблеми використовується ключове слово AS, яке дозволяє призначити стовпцю тимчасову назву (псевдонім) у вихідному наборі даних.[8]

Наприклад, при обчисленні середньої температури, можна надати новому стовпцю зрозумілу назву temp_avg:
SELECT city, (temp_hi+temp_lo)/2 AS temp_avg, date FROM weather;.6
Псевдоніми роблять вивід запиту більш зрозумілим для людини та полегшують подальшу обробку даних в інших інструментах, таких як Python або BI-системи.[3]

1.4. Обчислення та Вирази в Запитах

Оператор SELECT не обмежується лише вибіркою існуючих стовпців. Його справжня потужність полягає в здатності виконувати перетворення даних "на льоту". Список SELECT може містити не тільки назви стовпців, але й різноманітні вирази, включаючи математичні операції та виклики функцій.[6] Це перетворює

SELECT із простого інструмента для отримання даних на динамічний калькулятор та форматувальник.

Наприклад, можна об'єднати ім'я та прізвище в один стовпець full_name за допомогою оператора конкатенації ||:
`SELECT first_name |
| ' ' |
| last_name AS full_name FROM customer;`.2
Виконання таких перетворень на рівні бази даних є значно ефективнішим, ніж завантаження "сирих" даних у застосунок для подальшої обробки. Бази даних оптимізовані для виконання операцій над великими наборами даних, і використання цієї можливості є ключем до створення продуктивних систем.

Розділ 2: Фільтрація Даних: Клауза WHERE

2.1. Призначення WHERE: Як відбирати лише потрібні рядки

Часто необхідно отримати не всі рядки з таблиці, а лише ті, що відповідають певним критеріям. Для цього використовується клауза WHERE, яка діє як фільтр. Вона перевіряє кожен рядок, що повертається з FROM, на відповідність заданій умові. Лише ті рядки, для яких умова є істинною (true), потрапляють у кінцевий результат.[6]

Синтаксично клауза WHERE завжди розміщується після FROM, але перед іншими клаузами, такими як ORDER BY.[9] Це пов'язано з логічним порядком виконання запиту, де фільтрація рядків відбувається до їх сортування.

2.2. Детальний Розгляд Операторів

Клауза WHERE підтримує широкий набір операторів для побудови умов фільтрації.

- **Оператори порівняння:** Це найпростіші оператори для порівняння значень: = (дорівнює), <> або != (не дорівнює), > (більше), < (менше), >= (більше або дорівнює), <= (менше або дорівнює).[9]

  Приклад: знайти клієнтів з іменем 'Jamie'.
  SELECT last_name, first_name FROM customer WHERE first_name = 'Jamie';.9
- BETWEEN: Перевіряє, чи знаходиться значення в заданому діапазоні (включно з межами).
  Приклад: знайти клієнтів, у яких довжина імені становить від 3 до 5 символів.
  SELECT first_name FROM customer WHERE LENGTH(first_name) BETWEEN 3 AND 5;.9
- IN: Перевіряє, чи збігається значення з будь-яким елементом у наданому списку.
  Приклад: знайти клієнтів з іменами 'Ann', 'Anne' або 'Annie'.
  SELECT first_name, last_name FROM customer WHERE first_name IN ('Ann', 'Anne', 'Annie');.9
- LIKE: Використовується для пошуку за текстовим шаблоном. Підтримує два спеціальні символи: % (відповідає будь-якій послідовності символів) та _ (відповідає одному будь-якому символу).
  Приклад: знайти клієнтів, чиє ім'я починається на 'Ann'.
  SELECT first_name, last_name FROM customer WHERE first_name LIKE 'Ann%';.9
- **IS NULL / IS NOT NULL:** Спеціальні оператори для перевірки наявності або відсутності значення NULL. NULL представляє відсутність даних і не може бути порівняний за допомогою оператора =.[3]

  Приклад: знайти товари без вказаної ціни.
  SELECT name, price FROM products WHERE price IS NULL;.3

2.3. Комбінування Умов: Логічні оператори AND, OR, NOT

Для створення складних фільтрів можна комбінувати кілька умов за допомогою логічних операторів.[6]

- AND: Повертає true лише тоді, коли обидві умови є істинними.
  Приклад: знайти клієнта з іменем 'Jamie' та прізвищем 'Rice'.
  SELECT last_name, first_name FROM customer WHERE first_name = 'Jamie' AND last_name = 'Rice';.9

- OR: Повертає true, якщо хоча б одна з умов є істинною.
  Приклад: знайти клієнтів, у яких прізвище 'Rodriguez' або ім'я 'Adam'.
  SELECT first_name, last_name FROM customer WHERE last_name = 'Rodriguez' OR first_name = 'Adam';.9
- **NOT:** Інвертує результат умови. Часто використовується в комбінації з іншими операторами, наприклад NOT IN або NOT LIKE.

Для зручності, основні оператори, що використовуються в клаузі WHERE, зведені в таблицю нижче.

Таблиця 1: Зведена Таблиця Операторів WHERE

Оператор	Опис	Приклад Синтаксису
=	Перевіряє на рівність.	WHERE price = 100
<> або !=	Перевіряє на нерівність.	WHERE city <> 'Kyiv'
>	Більше ніж.	WHERE age > 18
<	Менше ніж.	WHERE stock_count < 10
>=	Більше або дорівнює.	WHERE salary >= 50000
<=	Менше або дорівнює.	WHERE rating <= 4.5
BETWEEN	Перевіряє входження в діапазон (включно).	WHERE price BETWEEN 100 AND 200
IN	Перевіряє входження у список значень.	WHERE status IN ('new', 'pending')
LIKE	Зіставляє з текстовим шаблоном.	WHERE name LIKE 'A%'
IS NULL	Перевіряє, чи є значення NULL.	WHERE manager_id IS NULL
AND	Комбінує умови; обидві	WHERE age > 18 AND city =

	мають бути істинними.	'Kyiv'
OR	Комбінує умови; хоча б одна має бути істинною.	WHERE status = 'shipped' OR status = 'delivered'

Розділ 3: Впорядкування та Організація Результатів

Після того, як дані вибрано та відфільтровано, часто виникає потреба їх організувати: відсортувати, видалити дублікати або обмежити кількість результатів. Ці операції не просто форматують вивід, а є фундаментальними інструментами для аналітичних завдань, таких як пошук лідерів продажу або реалізація посторінкового виводу в застосунках.

3.1. Сортування з ORDER BY

За замовчуванням база даних не гарантує певного порядку повернення рядків. Щоб отримати впорядкований результат, використовується клауза ORDER BY.[8]

- **Напрямок сортування:** Можна сортувати за зростанням (ASC, ascending), що є поведінкою за замовчуванням, або за спаданням (DESC, descending).[3]

  Приклад: відсортувати товари за ціною від найдорожчого до найдешевшого.
  SELECT name, price FROM products ORDER BY price DESC;.3
- Сортування за кількома стовпцями: Можна вказати кілька стовпців для сортування. Дані спочатку сортуються за першим стовпцем, а потім рядки з однаковими значеннями в першому стовпці сортуються за другим, і так далі.
  Приклад: відсортувати клієнтів за прізвищем, а потім за іменем в алфавітному порядку.
  SELECT first_name, last_name FROM customer ORDER BY last_name ASC, first_name ASC;

3.2. Усунення Дублікатів: Застосування DISTINCT

Іноді в результаті запиту з'являються однакові рядки. Щоб залишити лише унікальні значення, використовується ключове слово DISTINCT.[7] Воно застосовується до всього рядка, видаляючи повні дублікати.

Приклад: отримати список унікальних категорій товарів.
SELECT DISTINCT category FROM products;.3
PostgreSQL також пропонує розширену конструкцію DISTINCT ON (expression), яка дозволяє видаляти дублікати на основі значень у конкретних стовпцях, залишаючи при цьому перший рядок з кожної унікальної групи (відповідно до сортування ORDER BY).[2]

3.3. Обмеження Вибірки: Пагінація результатів за допомогою LIMIT та OFFSET

Для роботи з великими наборами даних часто потрібно отримувати лише їх частину.

- **LIMIT:** Обмежує максимальну кількість рядків, що повертаються запитом.[2]

  Приклад: отримати лише 10 перших товарів.
  SELECT * FROM products LIMIT 10;
- **OFFSET:** Пропускає вказану кількість рядків перед тим, як почати повертати результати.[3]

  Приклад: пропустити перші 5 товарів.
  SELECT * FROM products OFFSET 5;

Комбінація LIMIT та OFFSET є основним механізмом для реалізації пагінації (посторінкового виводу) в веб-застосунках. Наприклад, щоб показати другу сторінку товарів, де на кожній сторінці по 10 товарів, можна використати LIMIT 10 OFFSET 10.

Поєднання ORDER BY та LIMIT є потужним аналітичним інструментом. Наприклад, щоб знайти 5 найдорожчих товарів, можна виконати такий запит:
SELECT name, price FROM products ORDER BY price DESC LIMIT 5;.3

Цей запит не просто форматує вивід, а відповідає на конкретне бізнес-питання "Які наші топ-5 найдорожчих товарів?".

Розділ 4: Агрегація Даних: Від Рядків до Підсумків

До цього моменту всі запити повертали окремі рядки з таблиці. Однак справжня сила SQL проявляється в здатності узагальнювати дані — переходити від перегляду окремих записів до аналізу підсумкових показників. Цей процес, відомий як агрегація, є основою для будь-якої аналітики та бізнес-звітності.

4.1. Вступ до Агрегатних Функцій

Агрегатні функції виконують обчислення на наборі значень і повертають єдиний результат.[11] Вони дозволяють відповідати на питання типу "Скільки?", "Яка сума?", "Яке середнє значення?".

Основні агрегатні функції:

- COUNT(): підраховує кількість рядків у групі.
- SUM(): обчислює суму значень у числовому стовпці.
- AVG(): обчислює середнє значення.
- MIN(): знаходить мінімальне значення.
- MAX(): знаходить максимальне значення.[11]

4.2. Групування Даних з GROUP BY

Щоб застосувати агрегатні функції не до всієї таблиці, а до окремих підгруп даних, використовується клауза GROUP BY. Вона об'єднує рядки з однаковими значеннями у вказаних стовпцях у групи, після чого агрегатна функція обчислюється для кожної групи окремо.[8]

GROUP BY кардинально змінює підхід до даних. Замість того, щоб дивитися на кожен окремий продукт, можна побачити загальні продажі по кожній категорії. Це перехід від мікро- до макрорівня аналізу.

Приклад: знайти середню температуру для кожного пристрою.
SELECT device_id, AVG(temperature) FROM conditions GROUP BY device_id;.8

4.3. Фільтрація Груп з HAVING

Часто виникає потреба відфільтрувати результати не на рівні окремих рядків, а на рівні цілих груп. Наприклад, знайти лише ті категорії товарів, сумарні продажі яких перевищують певну суму. Для цього використовується клауза HAVING.[13]

Приклад: знайти магазини, в яких більше 300 клієнтів.
SELECT store_id, COUNT(customer_id) FROM customer GROUP BY store_id HAVING COUNT(customer_id) > 300;.13
Новачки часто плутають WHERE та HAVING. Ключ до розуміння їхньої різниці лежить у логічному порядку виконання SQL-запиту, який виглядає так:

1. FROM: Визначаються таблиці-джерела.
2. WHERE: Фільтруються окремі рядки.
3. GROUP BY: Відфільтровані рядки групуються.
4. HAVING: Фільтруються групи, створені на попередньому етапі.
5. SELECT: Обчислюються кінцеві вирази для виводу.
6. ORDER BY: Результат сортується.[15]

З цієї послідовності стає зрозуміло, чому WHERE та HAVING не є взаємозамінними. Клауза WHERE обробляється *до* групування, тому вона не може "знати" про агреговані значення (такі як SUM() або COUNT()), оскільки вони ще не обчислені. Вона працює виключно з даними окремих рядків. Натомість клауза HAVING обробляється *після* групування і спеціально призначена для фільтрації на основі результатів агрегатних функцій.[11] Розуміння цього порядку виконання дозволяє перейти від простого запам'ятовування правил до глибокого усвідомлення логіки роботи SQL.

Розділ 5: Об'єднання Таблиць: Мистецтво JOIN

У реляційних базах даних інформація зазвичай розподілена по кількох пов'язаних таблицях для уникнення дублювання та забезпечення цілісності. Цей підхід називається нормалізацією. Щоб отримати повну картину, необхідно об'єднувати дані з цих таблиць в одному запиті. Для цього використовується оператор JOIN.[16]

5.1. Концепція Реляційних Даних та Необхідність JOIN

JOIN дозволяє комбінувати стовпці з двох або більше таблиць на основі значень у пов'язаних стовпцях, зазвичай це первинний ключ (primary key) однієї таблиці та зовнішній ключ (foreign key) іншої.[16]

5.2. INNER JOIN: Пошук спільних даних

INNER JOIN є найпоширенішим типом об'єднання. Він повертає лише ті рядки, для яких знайдено відповідність в обох таблицях на основі умови об'єднання, вказаної в клаузі ON.[17]

Приклад: отримати імена клієнтів та суми їхніх платежів, об'єднавши таблиці customer та payment.
SELECT c.first_name, c.last_name, p.amount FROM customer c INNER JOIN payment p ON c.customer_id = p.customer_id;.2

5.3. OUTER JOIN (LEFT, RIGHT, FULL): Робота з даними, що не мають збігів

Іноді необхідно отримати всі дані з однієї таблиці, навіть якщо для них немає відповідників в іншій. Для цього використовуються зовнішні об'єднання (OUTER JOIN).

- **LEFT JOIN** (або LEFT OUTER JOIN): Повертає *всі* рядки з лівої (першої) таблиці та відповідні рядки з правої. Якщо для рядка з лівої таблиці не знайдено збігу в правій, стовпці правої таблиці будуть заповнені значеннями NULL.[16]
- **RIGHT JOIN** (або RIGHT OUTER JOIN): Працює дзеркально до LEFT JOIN. Повертає *всі* рядки з правої (другої) таблиці та відповідні з лівої. Якщо збігу немає, стовпці лівої таблиці заповнюються NULL.[16]
- **FULL OUTER JOIN**: Повертає всі рядки з обох таблиць. Якщо для рядка з однієї таблиці немає відповідника в іншій, відсутні стовпці заповнюються NULL. Це об'єднання результатів LEFT та RIGHT JOIN.[18]

5.4. Специфічні Випадки

- **CROSS JOIN**: Повертає декартовий добуток рядків з обох таблиць, тобто кожний рядок першої таблиці поєднується з кожним рядком другої. Якщо в таблицях N та M рядків, результат буде містити N * M рядків. Використовується рідко, переважно для генерації тестових даних.[18]
- **SELF JOIN**: Це не окремий тип JOIN, а техніка, коли таблиця об'єднується сама з собою. Це корисно для роботи з ієрархічними даними. Наприклад, якщо таблиця employees містить стовпець manager_id, який посилається на employee_id в тій же таблиці, можна отримати ім'я кожного співробітника та ім'я його керівника.[16]

Для кращого розуміння, як працюють різні типи JOIN, можна уявити їх за допомогою діаграм Венна, де кожне коло представляє таблицю.

Таблиця 2: Візуальний Довідник по Типах JOIN

Тип JOIN	Уявна Діаграма Венна	Опис	Приклад Синтаксису
INNER JOIN	Перетин двох кіл	Повертає лише рядки, що мають збіги в обох таблицях.	FROM A INNER JOIN B ON A.id = B.id
LEFT JOIN	Повне ліве коло та перетин	Повертає всі рядки з лівої таблиці та збіги з правої.	FROM A LEFT JOIN B ON A.id = B.id
RIGHT JOIN	Повне праве коло та перетин	Повертає всі рядки з правої таблиці та збіги з лівої.	FROM A RIGHT JOIN B ON A.id = B.id
FULL OUTER JOIN	Об'єднання обох кіл	Повертає всі рядки з обох таблиць, коли є збіг хоча б в одній.	FROM A FULL OUTER JOIN B ON A.id = B.id

Розділ 6: Розширені Можливості SELECT

Опанувавши основи, можна переходити до більш гнучких та потужних інструментів, які дозволяють реалізовувати складну логіку безпосередньо в SQL-запитах.

6.1. Умовна Логіка в Запитах з CASE

Вираз CASE є аналогом конструкції if/else в інших мовах програмування. Він дозволяє повертати різні значення залежно від виконання певних умов.[20]

Загальний синтаксис:
CASE WHEN condition1 THEN result1 WHEN condition2 THEN result2 ELSE result3 END
Практичний приклад: категоризація фільмів за тривалістю.

SQL

```sql
SELECT
  title,
  length,
  CASE
    WHEN length < 60 THEN 'Short'
    WHEN length BETWEEN 60 AND 120 THEN 'Medium'
    ELSE 'Long'
  END AS duration_category
FROM film;
```

PostgreSQL також пропонує кілька корисних умовних функцій:

- COALESCE(value1, value2,...): Повертає перше не-NULL значення зі списку аргументів. Дуже зручно для заміни NULL на значення за замовчуванням.[20]
- NULLIF(value1, value2): Повертає NULL, якщо value1 дорівнює value2, інакше повертає value1. Корисно для уникнення помилок, наприклад, ділення на нуль.[20]

6.2. Вступ до Вкладених Запитів (Subqueries)

Вкладений запит (або підзапит) — це SELECT-запит, що знаходиться всередині іншого SQL-запиту. Він дозволяє використовувати результат одного запиту як частину іншого.

Простий приклад: знайти всі товари, ціна яких вища за середню.
SELECT name, price FROM products WHERE price > (SELECT AVG(price) FROM products);
Тут внутрішній запит (SELECT AVG(price) FROM products) виконується першим, обчислює середню ціну, і його результат (одне число) підставляється в умову WHERE зовнішнього запиту.

Підзапити є потужним інструментом, але для складних запитів, що містять кілька рівнів вкладеності, краще використовувати Загальні Табличні Вирази (Common Table Expressions або WITH clause), оскільки вони роблять код більш читабельним та структурованим.[10]

Розділ 7: Практичне Застосування: Від SQL-запиту до Візуалізації в Python

SQL є надзвичайно потужним для вибірки та попередньої обробки даних, але для глибокого аналізу, статистичного моделювання та створення складних візуалізацій часто використовують мови програмування, такі як Python. Сучасний аналіз даних — це синергія SQL та Python, де кожен інструмент виконує те, для чого він найкраще пристосований.

7.1. Налаштування Середовища: Підключення до PostgreSQL з Python

Для роботи з PostgreSQL у Python знадобляться кілька ключових бібліотек:

- pandas: для маніпуляції даними у вигляді табличних структур (DataFrame).
- SQLAlchemy: SQL-інструментарій, що спрощує взаємодію з базами даних.
- psycopg2: адаптер (драйвер), який дозволяє Python "спілкуватися" з PostgreSQL.[1]

Встановити їх можна за допомогою pip:
pip install pandas sqlalchemy psycopg2-binary.21

Після встановлення, підключення до бази даних створюється за допомогою функції create_engine з SQLAlchemy [21]:

Python

```
import pandas as pd
from sqlalchemy import create_engine

# Створіть рядок підключення, замінивши ваші дані
db_connection_str = 'postgresql+psycopg2://your_username:your_password@your_host:5432/your_database'
engine = create_engine(db_connection_str)
```

7.2. Виконання SELECT-запитів та Завантаження Даних у Pandas DataFrame

Найпростіший спосіб виконати SQL-запит і одразу завантажити його результат у pandas DataFrame — це використати функцію pandas.read_sql_query().[23]

Python

```
query = """
SELECT
    c.city,
    COUNT(cu.customer_id) AS customer_count
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON a.address_id = cu.address_id
GROUP BY c.city
ORDER BY customer_count DESC
LIMIT 10;
"""

df_top_cities = pd.read_sql_query(query, con=engine)
```

```
print(df_top_cities)
```

Цей підхід ілюструє оптимальний робочий процес: складний запит з JOIN та агрегацією виконується на стороні потужного сервера бази даних. Python отримує лише невеликий, вже оброблений результат (топ-10 міст), що є набагато ефективнішим, ніж завантажувати всі таблиці city, address та customer в пам'ять локального комп'ютера.

7.3. Аналіз Даних у Pandas: Фільтрація, сортування та групування

Pandas надає потужний інструментарій для подальшого аналізу, який концептуально схожий на операції в SQL.

- Фільтрація (SQL WHERE): В pandas для цього використовується булеве індексування.
  df_filtered = df[df['column'] > 100].24
- Сортування (SQL ORDER BY): Використовується метод .sort_values().
  df_sorted = df.sort_values(by='column', ascending=False).26
- Групування та агрегація (SQL GROUP BY): Використовується метод .groupby(), за яким слідує агрегатна функція.
  df_grouped = df.groupby('category')['sales'].sum().27

7.4. Візуалізація Результатів

Візуалізація є ключовим етапом аналізу, що дозволяє перетворити табличні дані на зрозумілі графіки.

- **Стовпчаста діаграма (Bar Chart):** Ідеально підходить для порівняння категоріальних даних. Використовуючи наш DataFrame df_top_cities, можна легко побудувати діаграму за допомогою matplotlib та seaborn.
  Python
  ```
  import matplotlib.pyplot as plt
  import seaborn as sns

  plt.figure(figsize=(12, 6))
  sns.barplot(x='customer_count', y='city', data=df_top_cities)
  plt.title('Top 10 Cities by Number of Customers')
  plt.xlabel('Number of Customers')
  plt.ylabel('City')
  plt.show()
  ```

29

- **Гістограма (Histogram):** Використовується для візуалізації розподілу однієї числової змінної.
  Python
  ```
  # Припустимо, ми завантажили дані про платежі
  query_payments = "SELECT amount FROM payment WHERE amount > 0;"
  df_payments = pd.read_sql_query(query_payments, con=engine)

  sns.histplot(df_payments['amount'], bins=30, kde=True)
  plt.title('Distribution of Payment Amounts')
  plt.show()
  ```
  31

- **Кругова діаграма (Pie Chart):** Показує долі частин у цілому. Для її побудови зручно використовувати метод .value_counts() в pandas для підготовки даних.
  Python
  ```
  # Припустимо, ми завантажили дані про рейтинг фільмів
  query_ratings = "SELECT rating FROM film;"
  df_films = pd.read_sql_query(query_ratings, con=engine)

  rating_counts = df_films['rating'].value_counts()

  plt.figure(figsize=(8, 8))
  plt.pie(rating_counts, labels=rating_counts.index, autopct='%1.1f%%')
  plt.title('Distribution of Film Ratings')
  plt.show()
  ```
  33

Висновок та Рекомендації для Подальшого Навчання

Цей посібник охопив фундаментальні аспекти оператора SELECT у PostgreSQL, провівши вас від простої вибірки стовпців до складних операцій, таких як агрегація, об'єднання таблиць та аналіз результатів у Python. Розуміння цих концепцій є основою для будь-якої роботи з реляційними базами даних.

Найкращі практики, які варто запам'ятати:

- **Уникайте SELECT * у робочому коді:** Завжди явно вказуйте стовпці, які вам потрібні. Це робить ваші запити більш продуктивними, безпечними та легкими для розуміння.[3]
- **Використовуйте псевдоніми:** AS робить ваші запити, особливо складні, значно читабельнішими.[3]
- **Розрізняйте WHERE та HAVING:** Пам'ятайте, що WHERE фільтрує рядки до агрегації, а HAVING — групи після неї. Це ключова концепція для правильної побудови аналітичних запитів.

Напрямки для подальшого вивчення:
SQL — це глибока та багатогранна мова. Опанувавши основи, варто звернути увагу на більш просунуті теми, які значно розширять ваші аналітичні можливості:

- **Віконні функції (Window Functions):** Дозволяють виконувати складні обчислення над набором рядків, пов'язаних з поточним рядком (наприклад, ковзне середнє або ранжування).
- **Загальні Табличні Вирази (Common Table Expressions, WITH clause):** Покращують структуру та читабельність складних запитів, розбиваючи їх на логічні блоки.[10]
- **Оптимізація запитів (EXPLAIN):** Аналіз планів виконання запитів для виявлення вузьких місць та підвищення продуктивності.
- **Транзакції та цілісність даних:** Розуміння того, як забезпечити надійність та послідовність даних за допомогою транзакцій (BEGIN, COMMIT, ROLLBACK).[12]

Постійна практика та вивчення нових можливостей PostgreSQL перетворять вас з новачка на впевненого фахівця, здатного ефективно працювати з даними.

Джерела

1. How to insert a pandas DataFrame to an existing PostgreSQL table? - GeeksforGeeks, доступ отримано вересня 10, 2025, https://www.geeksforgeeks.org/python/how-to-insert-a-pandas-dataframe-to-an-existing-postgresql-table/
2. PostgreSQL SELECT Statement: Syntax, Parameters, Examples - phoenixNAP, доступ отримано вересня 10, 2025, https://phoenixnap.com/kb/postgresql-select
3. PostgreSQL SELECT - DataCamp, доступ отримано вересня 10, 2025, https://www.datacamp.com/doc/postgresql/select
4. PostgreSQL SELECT - Neon, доступ отримано вересня 10, 2025, https://neon.com/postgresql/postgresql-tutorial/postgresql-select
5. Documentation: 17: psql - PostgreSQL, доступ отримано вересня 10, 2025, https://www.postgresql.org/docs/current/app-psql.html
6. Documentation: 17: 2.5. Querying a Table - PostgreSQL, доступ отримано вересня 10, 2025, https://www.postgresql.org/docs/current/tutorial-select.html

7. How to Use the PostgreSQL SELECT Statement?[Syntax+Examples] - Hevo Data, доступ отримано вересня 10, 2025, https://hevodata.com/learn/what-is-postgresql-select-statement/
8. Understanding PostgreSQL SELECT - TigerData, доступ отримано вересня 10, 2025, https://www.tigerdata.com/learn/understanding-postgresql-select
9. PostgreSQL WHERE: Filtering Rows of a Query - Neon, доступ отримано вересня 10, 2025, https://neon.com/postgresql/postgresql-tutorial/postgresql-where
10. Documentation: 17: SELECT - PostgreSQL, доступ отримано вересня 10, 2025, https://www.postgresql.org/docs/current/sql-select.html
11. SQL GROUP BY and HAVING Clauses: A Comprehensive Guide - Secoda, доступ отримано вересня 10, 2025, https://www.secoda.co/learn/sql-group-by-and-having-clauses-a-comprehensive-guide
12. HAVING clause - PostgreSQL - GeeksforGeeks, доступ отримано вересня 10, 2025, https://www.geeksforgeeks.org/postgresql/postgresql-having-clause/
13. PostgreSQL HAVING - Neon, доступ отримано вересня 10, 2025, https://neon.com/postgresql/postgresql-tutorial/postgresql-having
14. Understanding HAVING in PostgreSQL (With Examples) - TigerData, доступ отримано вересня 10, 2025, https://www.tigerdata.com/learn/understanding-having-in-postgresql-with-examples
15. How to Use GROUP BY and HAVING in SQL - DataCamp, доступ отримано вересня 10, 2025, https://www.datacamp.com/tutorial/group-by-having-clause-sql
16. A Visual Explanation of PostgreSQL Joins - Neon, доступ отримано вересня 10, 2025, https://neon.com/postgresql/postgresql-tutorial/postgresql-joins
17. Joins in Postgres | Tutorials - Crunchy Data, доступ отримано вересня 10, 2025, https://www.crunchydata.com/developers/playground/joins-in-postgres
18. PostgreSQL - JOINS - Tutorials Point, доступ отримано вересня 10, 2025, https://www.tutorialspoint.com/postgresql/postgresql_using_joins.htm
19. PostgreSQL Joins Tutorial with Examples (INNER, LEFT, RIGHT, FULL, CROSS) - YouTube, доступ отримано вересня 10, 2025, https://www.youtube.com/watch?v=MIOv8qJQaqA
20. Documentation: 17: 9.18. Conditional Expressions - PostgreSQL, доступ отримано вересня 10, 2025, https://www.postgresql.org/docs/current/functions-conditional.html
21. pandas postgresql: Mastering Data Manipulation | by Hey Amit - Medium, доступ отримано вересня 10, 2025, https://medium.com/@heyamit10/pandas-postgresql-mastering-data-manipulation-79f55f84a7d4
22. Managing PostgreSQL Databases in Python with psycopg2 - DataCamp, доступ отримано вересня 10, 2025, https://www.datacamp.com/tutorial/tutorial-postgresql-python
23. Psycopg2 connection sql database to pandas dataframe - Stack Overflow,

доступ отримано вересня 10, 2025, https://stackoverflow.com/questions/70892143/psycopg2-connection-sql-database-to-pandas-dataframe
24. Python pandas Tutorial: The Ultimate Guide for Beginners - DataCamp, доступ отримано вересня 10, 2025, https://www.datacamp.com/tutorial/pandas
25. The pandas DataFrame: Make Working With Data Delightful - Real Python, доступ отримано вересня 10, 2025, https://realpython.com/pandas-dataframe/
26. pandas.DataFrame.sort_values — pandas 2.3.2 documentation - PyData |, доступ отримано вересня 10, 2025, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html
27. pandas.DataFrame.groupby — pandas 2.3.2 documentation - PyData |, доступ отримано вересня 10, 2025, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html
28. pandas GroupBy: Your Guide to Grouping Data in Python, доступ отримано вересня 10, 2025, https://realpython.com/pandas-groupby/
29. How to Use Matplotlib to Plot Multiple Columns of Pandas Data Frame on a Bar Chart, доступ отримано вересня 10, 2025, https://saturncloud.io/blog/how-to-use-matplotlib-to-plot-multiple-columns-of-pandas-data-frame-on-a-bar-chart/
30. Plot Multiple Columns of Pandas Dataframe on Bar Chart with Matplotlib - GeeksforGeeks, доступ отримано вересня 10, 2025, https://www.geeksforgeeks.org/python/plot-multiple-columns-of-pandas-dataframe-on-bar-chart-with-matplotlib/
31. Basic histogram with Seaborn - Python Graph Gallery, доступ отримано вересня 10, 2025, https://python-graph-gallery.com/20-basic-histogram-seaborn/
32. How to Make a Seaborn Histogram: A Detailed Guide - DataCamp, доступ отримано вересня 10, 2025, https://www.datacamp.com/tutorial/how-to-make-a-seaborn-histogram
33. Creating Pie Charts with Matplotlib | CodeSignal Learn, доступ отримано вересня 10, 2025, https://codesignal.com/learn/courses/introduction-to-basic-plots-with-matplotlib/lessons/creating-pie-charts-with-matplotlib
34. How to Plot Value Counts in Pandas - GeeksforGeeks, доступ отримано вересня 10, 2025, https://www.geeksforgeeks.org/python/how-to-plot-value-counts-in-pandas/