

Архітектура та Реалізація Поліглотної Персистентності: Python та VS Code

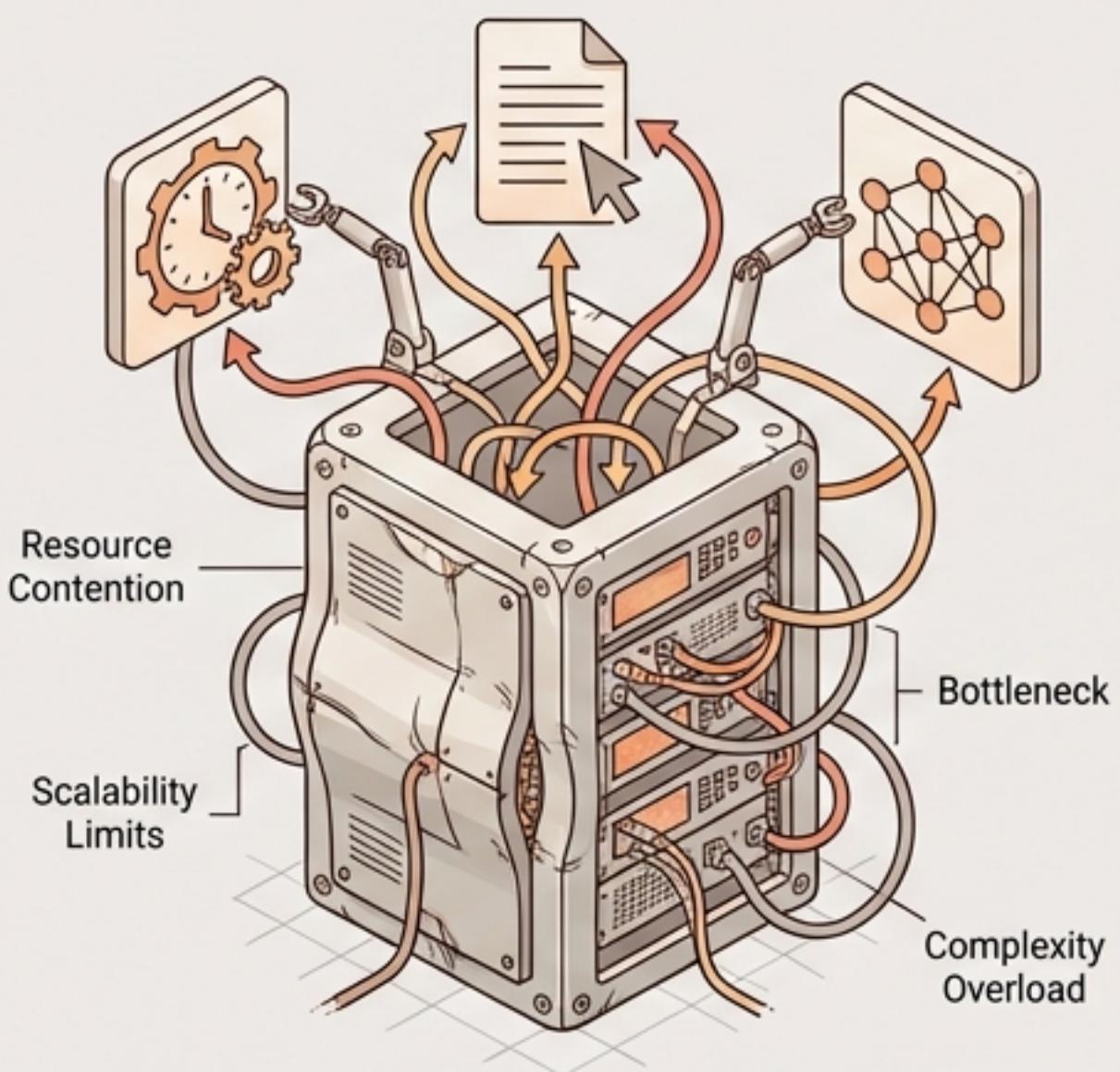
Комплексний посібник з розробки баз даних для промислових систем



Від моноліту до спеціалізації: як об'єднати SQL, NoSQL, Time-Series та Graph у єдиний пайпайн інженерії даних.

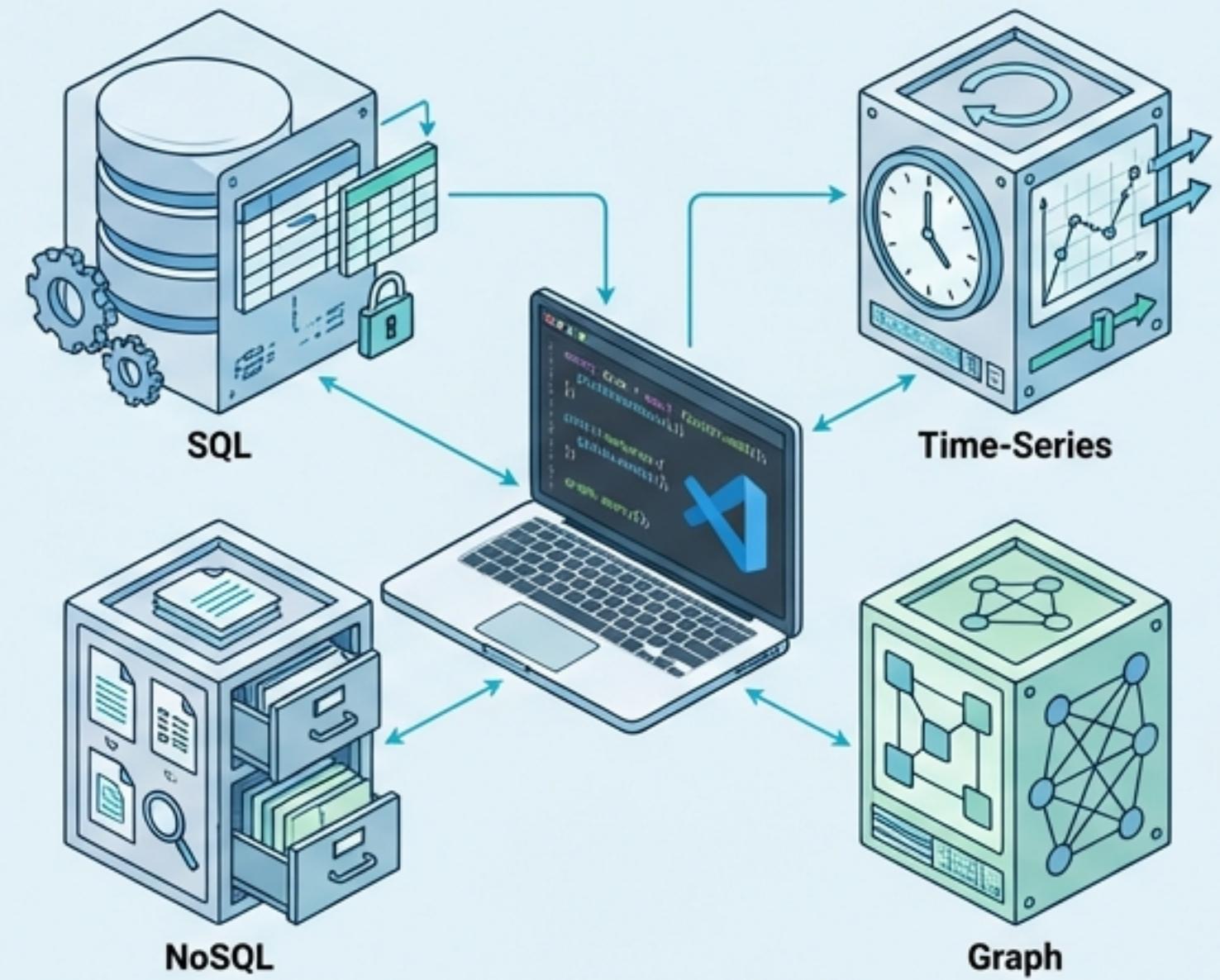
Зміна Парадигми: Від "One Size Fits All" до Спеціалізації

Минуле (The Monolith)



Складність масштабування та конфлікт ресурсів.

Сучасність (Polyglot)

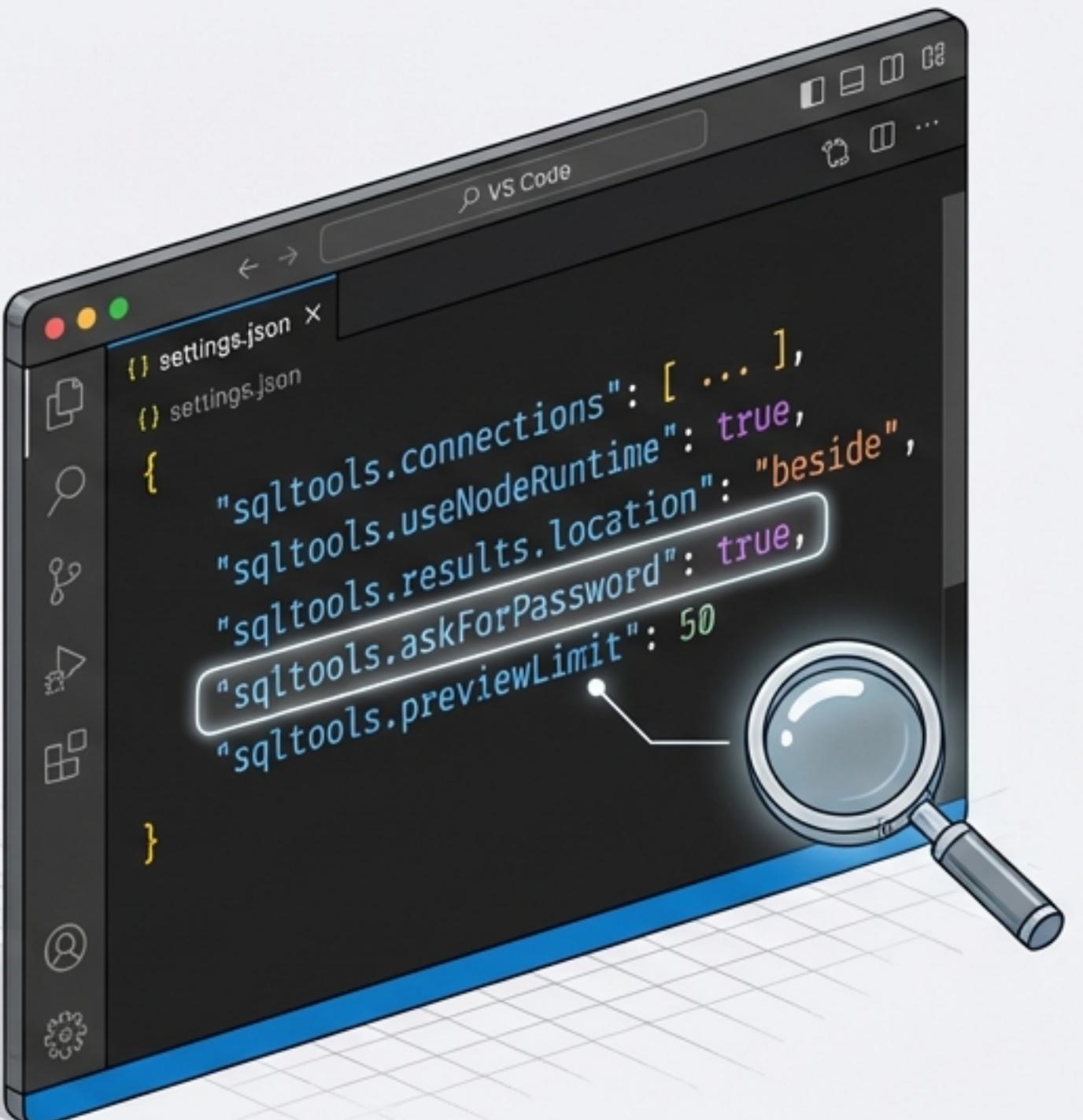


Поліглотна персистентність (Polyglot Persistence) – стратегія використання різних сховищ для різних задач.

"Поліглотна персистентність" (Polyglot Persistence)
– стратегія використання різних сховищ для різних задач.

Інженери стикаються з проблемою інтеграції гетерогенних інструментів. VS Code трансформується з редактора в "Mission Control" центр управління даними.

Інженерія Середовища: settings.json та Розширення



Pro-Tip

Безпека (Security)

askForPassword: true є критичним для виробничих середовищ. Ніколи не зберігайте паролі у відкритому вигляді в репозиторії.

Pro-Tip

Продуктивність (Performance)

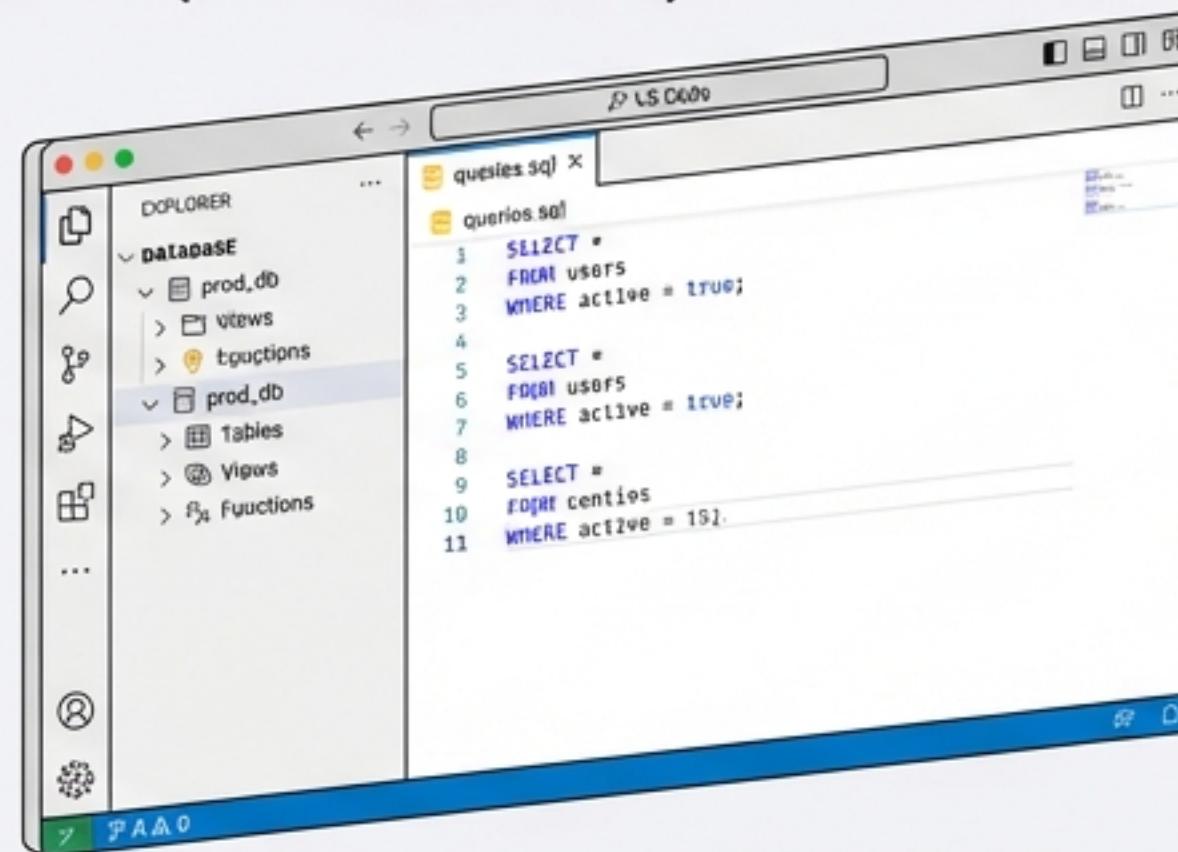
Параметр previewLimit захищає IDE від зависання при випадковому запиті великих даних.

Критичні Розширення (Extensions)

- **SQLTools (mtxr.sqltools)** – Управління з'єднаннями.
- **MongoDB for VS Code** – Перегляд колекцій та агрегацій.
- **Python (ms-python.python)** – IntelliSense та налагодження.

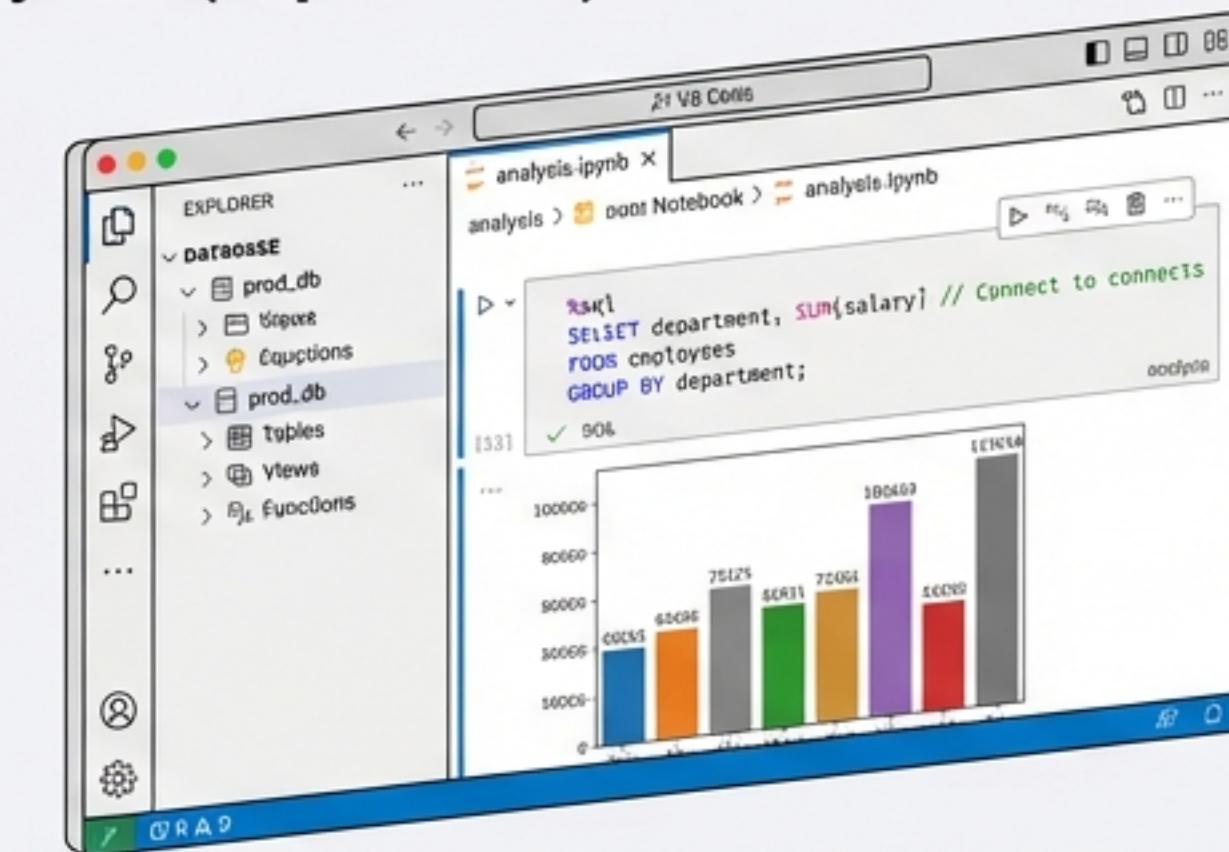
Робочий Процес: Інтерактивність проти Скриптів

SQLTools (Administration)



Для адміністрування та генерації ER-діаграм.
Інтегрується в інтерфейс редактора.

JupySQL (Exploration)



Для EDA (Exploratory Data Analysis) та візуалізації.
Комбінація SQL та Python.

```
pip install jupysql pandas matplotlib psycopg2-binary
```

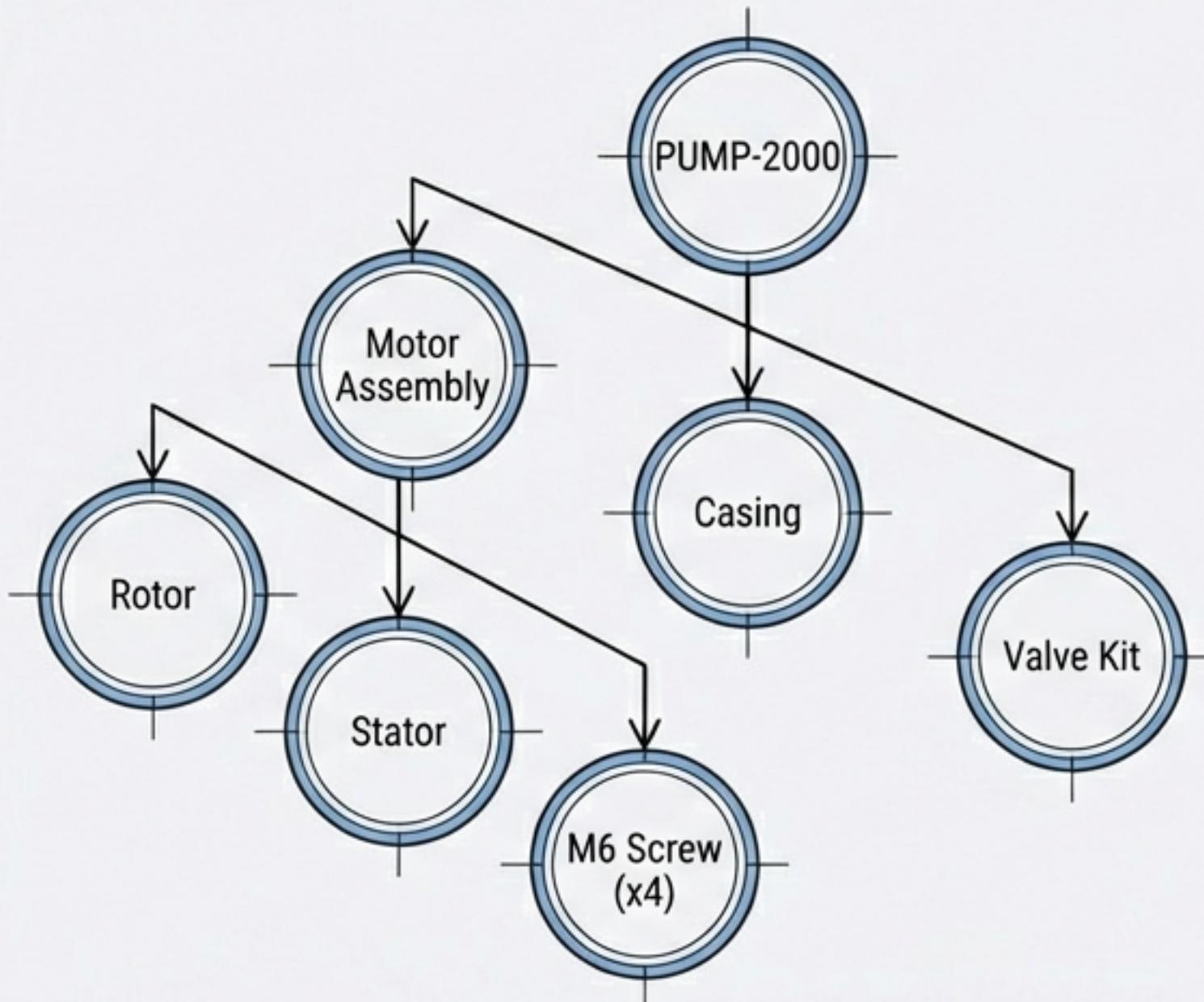
Дозволяє комбінувати SQL-логіку та графіки Matplotlib в одному звіті без перемикання вікон.



JupySQL –
рекомендовано для
Data Science проектів.

Реляційні Дані: Проблема Ієрархії (BOM)

Bill of Materials: Industrial Pump



Adjacency List (Список суміжності)

id	parent_id	name
1	NULL	PUMP-2000
2	1	Motor
3	2	Rotor

Кожен запис має посилання на parent_id.



Проблема:

Прості SELECT запити не можуть пройти все дерево. Альтернативи як Nested Sets ускладнюють запис даних, тому Adjacency List залишається стандартом.

Вирішення Рекурсії: SQLAlchemy 2.0 та СТЕ

```
# Визначення СТЕ для обходу ієрархії
bom_cte = select(BOMAssembly.child_id, BOMAssembly.
    quantity, literal(1).label('level'))\
    .where(BOMAssembly.parent_id == root_comp.id)\.
    .cte(name='bom_hierarchy', recursive=True)

# Рекурсивне з'єднання (Union All)
bom_cte = bom_cte.union_all(
    select(alias.child_id, alias.quantity,
(bom_cte.c.level + 1))
    .join(bom_cte, alias.parent_id == bom_cte.c.child_
id))
```

СТЕ (Common Table Expression):



recursive=True делегує обчислення серверу БД (PostgreSQL).

N+1 Problem:



Вирішує проблему "N+1 запитів", яка виникає при рекурсії на рівні Python.

Типи даних:



Decimal vs Float – Для ERP систем обов'язково використовувати DECIMAL для уникнення помилок округлення.

Міст до Аналітики: Pandas та SQL

	Метод	Дія	Вердикт
	<code>read_sql_table</code>	Читає всю таблицю	**Обережно** Небезпечно для великих даних. Завантажує все в RAM.
	<code>read_sql_query</code>	Виконує SQL запит	**Рекомендовано** Дозволяє фільтрацію (WHERE, JOIN) на сервері перед завантаженням.
	<code>read_sql</code>	Універсальна обгортка	Нейтрально Менш явна, може приховувати логіку.



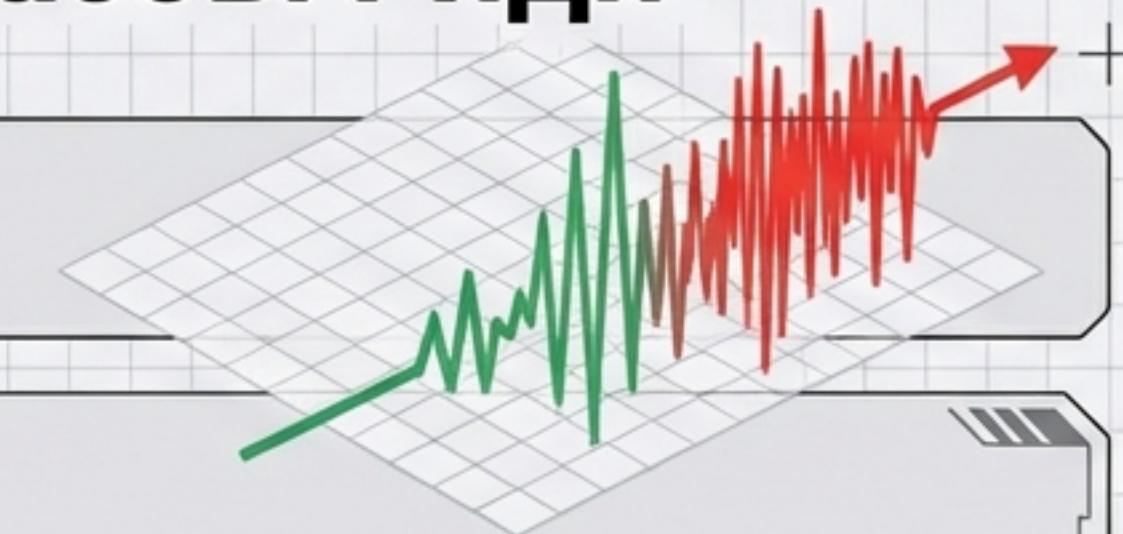
Best Practice: Передавайте об'єкт connection, а не engine, для коректного управління транзакціями.



Стратегія: Мінімізуйте передачу даних мережею. SQL – для фільтрації, Pandas – для аналізу.

Виклик Швидкості: IoT та Часові Ряди

Сценарій: Телеметрія вібрації турбіни (100 Гц)



TimescaleDB

- **Type:** SQL-based (PostgreSQL extension).
- **Pros:** Ідеально для інтеграції з наявними BI.
- **Key Tech:** "Hypertables" (авто-партиціонування).
- **Performance:** Стабільна при високій кардинальності (high cardinality).



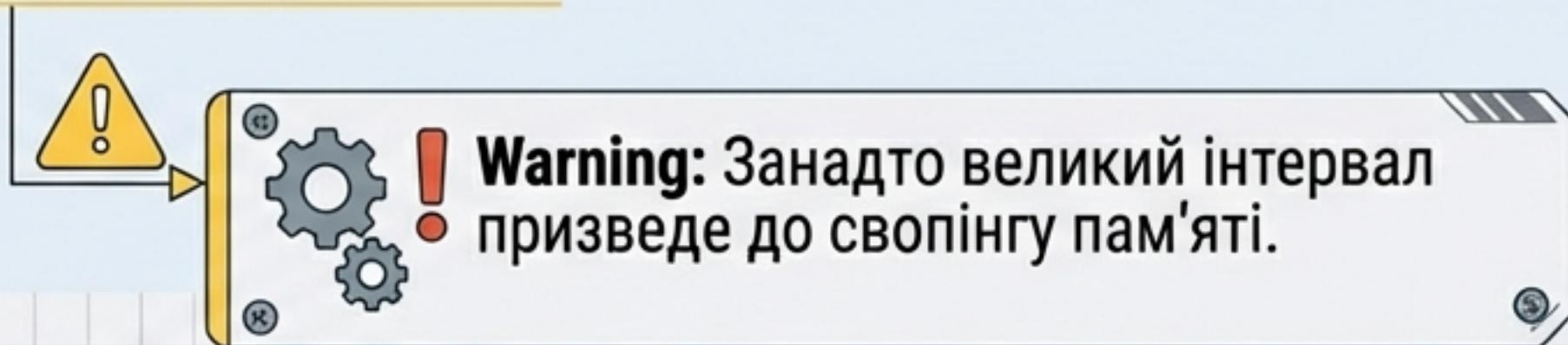
InfluxDB

- **Type:** NoSQL-based.
- **Language:** Flux.
- **Pros:** Вища швидкість запису та стиснення.
- **Warning:** "High Cardinality Problem" – втрачає продуктивність при мільйонах унікальних тегів/сенсорів.

Реалізація Коду: Hypertables проти Flux

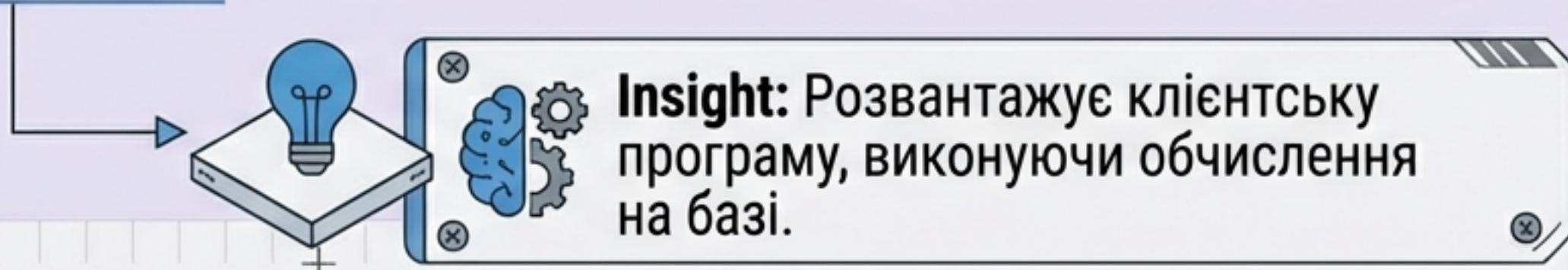
TimescaleDB
(Python/SQL)

```
# chunk_time_interval='1 day' створює новий файл для кожного дня  
conn.execute(text("SELECT create_hypertable('turbine_vibration',  
    'timestamp', chunk_time_interval => INTERVAL '1 day');"))
```

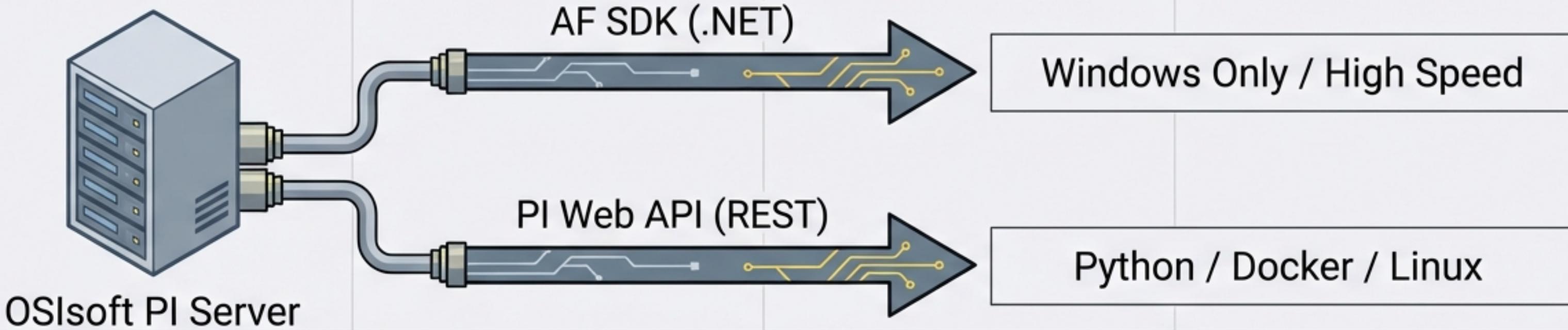


InfluxDB
(Flux Language)

```
from(bucket: "sensor_stream")  
|> filter(fn: (r) => r["_measurement"] == "vibration")  
|> aggregateWindow(every: 5m, fn: mean) // Даунсемплінг на сервері
```



Промисловий Стандарт: OSIsoft PI System



Реалізація Python (через Web API)

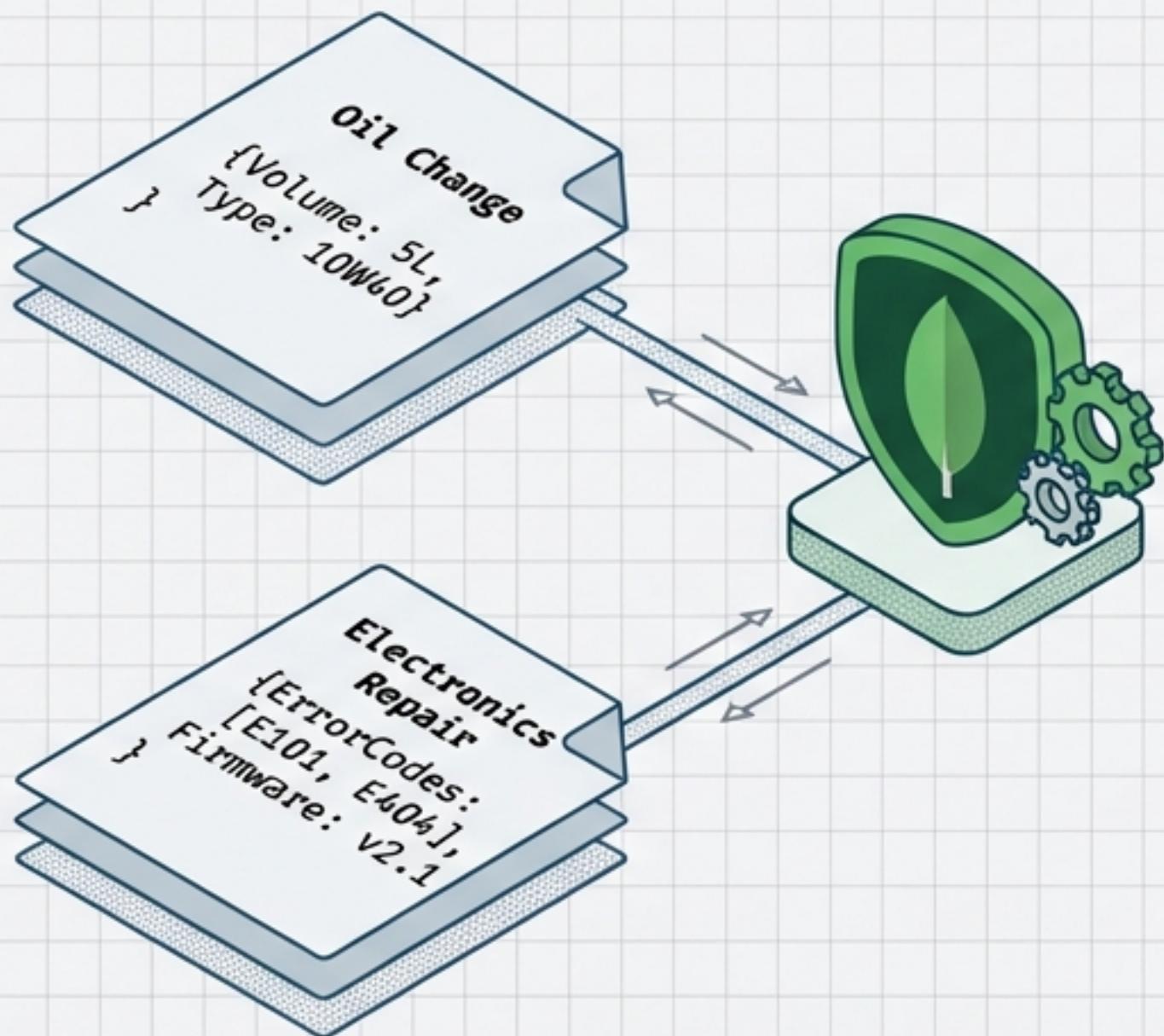
- Використовує library `requests` (Kerberos/Basic Auth).
- `urllib3.disable_warnings` (тільки для dev!).

****Performance Warning: Кешування WebID****

Постійний пошук тегу за текстовим шляхом (string path lookup) є дорогою операцією. Зберігайте WebID для повторних запитів.

Гнучкість Схеми: Документоорієнтовані Логи

Динамічні Логи Обслуговування



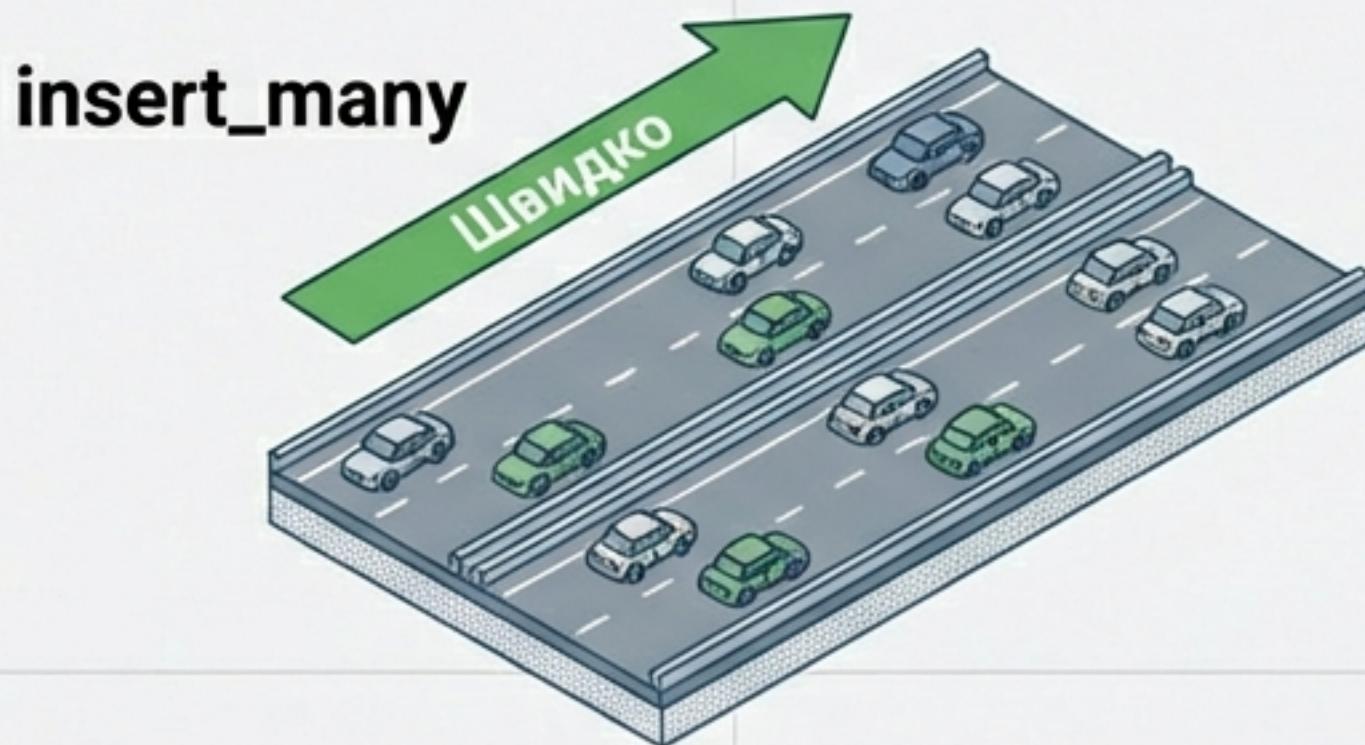
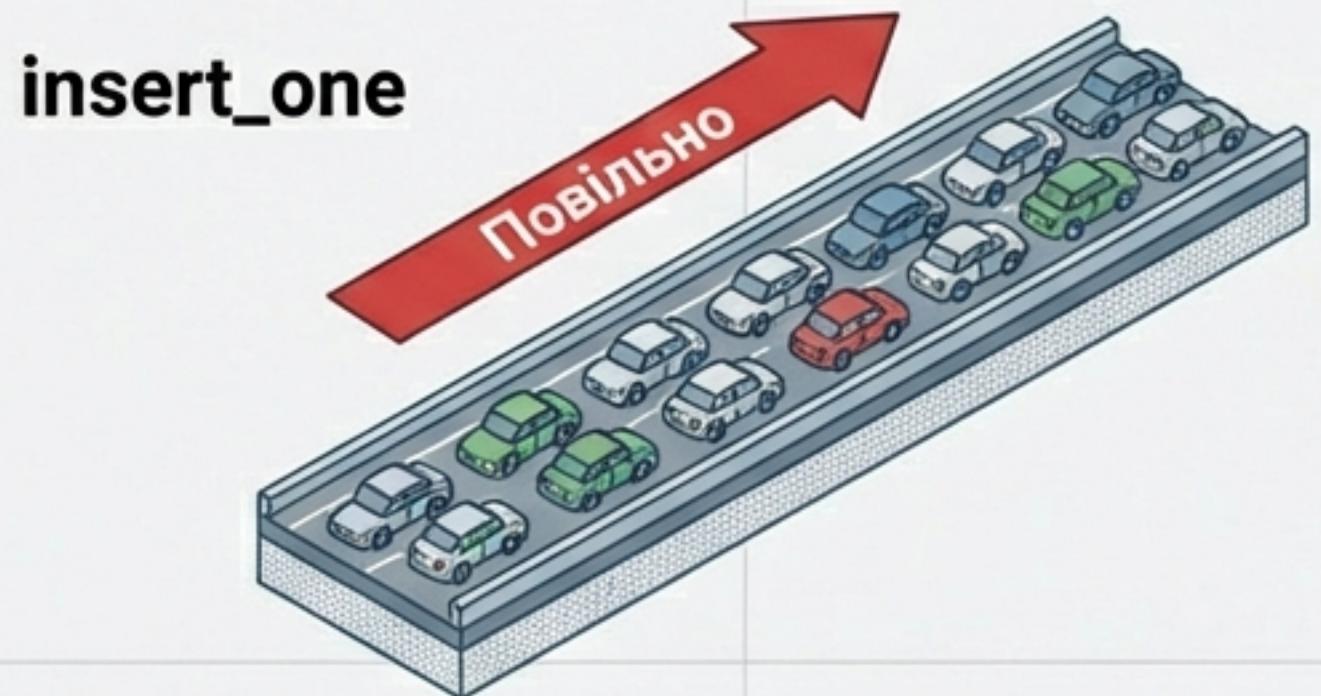
MongoDB & Python Logging Handler

```
class MongoHandler(logging.Handler):
    def emit(self, record):
        log_entry = { ... }
        if hasattr(record, 'details'):
            # Збереження extra полів як вкладених документів
            log_entry['details'] = record.details
        self.collection.insert_one(log_entry)
```



TTL Index (expireAfterSeconds):
Автоматичне видалення старих логів базою даних. Zero maintenance.

Масова Вставка: Оптимізація PyMongo

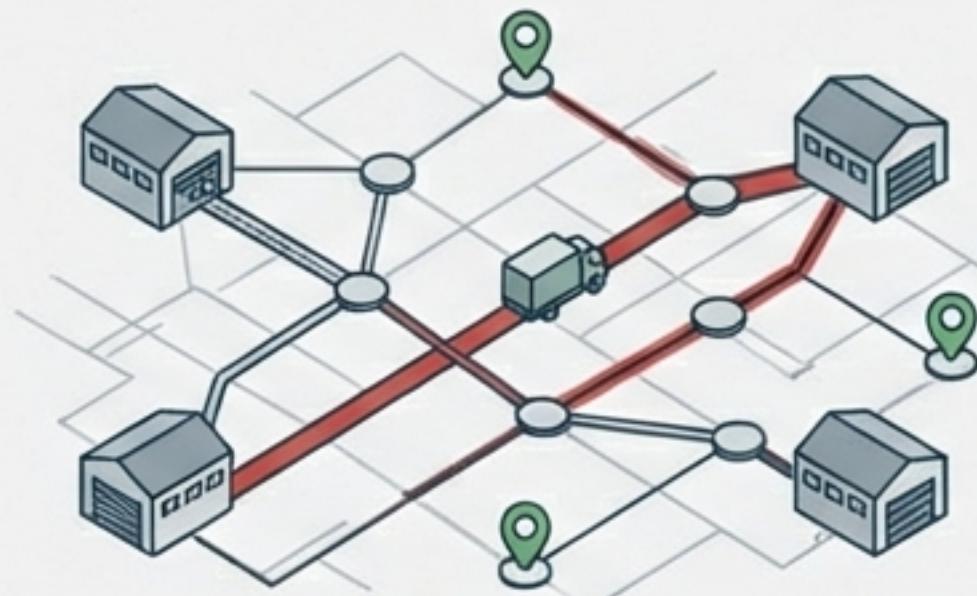


```
# ordered=False дозволяє продовжувати вставку навіть при помилках  
collection.insert_many(logs_list, ordered=False)
```

- ⚙️ 1. **Паралелізм:** Використання усіх шардів кластера.
- ➡️ 2. **Ізоляція помилок:** Один "битий" документ не зупиняє весь процес імпорту.
- ➡️ 3. **Швидкість:** Зменшення кількості мережевих запитів (round-trips).

Графові Дані: Оптимізація Ланцюгів Постачання

Логістика та Маршрутизація



Задача: Пошук маршруту доставки через склади.

Вузьке Місце (Bottleneck):

SQL: Глибокі JOINs



Експоненційне зростання часу.

Graph (Neo4j): Pointer Hopping



$O(1)$ перехід між вузлами.

Code Implementation (Neo4j Driver):

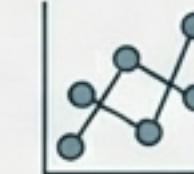
```
# Пошук найкоротшого шляху з урахуванням ваги (часу)
# Вимагає GDS (Graph Data Science) library
query = """
    MATCH p = shortestPath((start)-[:LINK*]->(end))
    RETURN p
    """
```



Алгоритмічна Нотатка: Для реального часу використовуйте алгоритм Дейкстри (``gds.shortestPath.dijkstra``), а не стандартний ``shortestPath``.

Візуалізація Мережі: PyVis у VS Code

NetworkX + Matplotlib

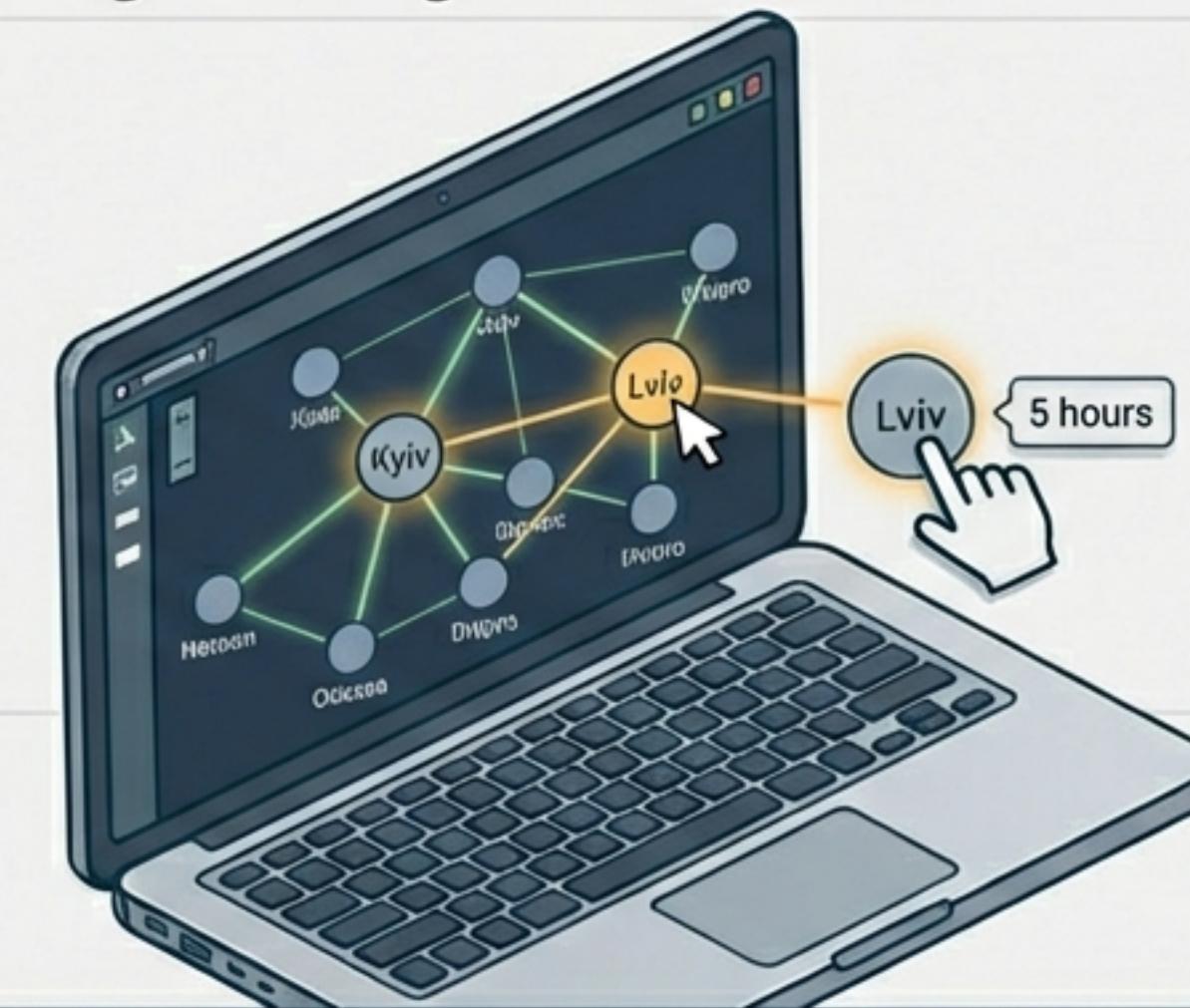


- Статичні зображення.
- Для наукового аналізу.

PyVis



- Інтерактивний HTML/JS.
- Для презентації та дослідження.



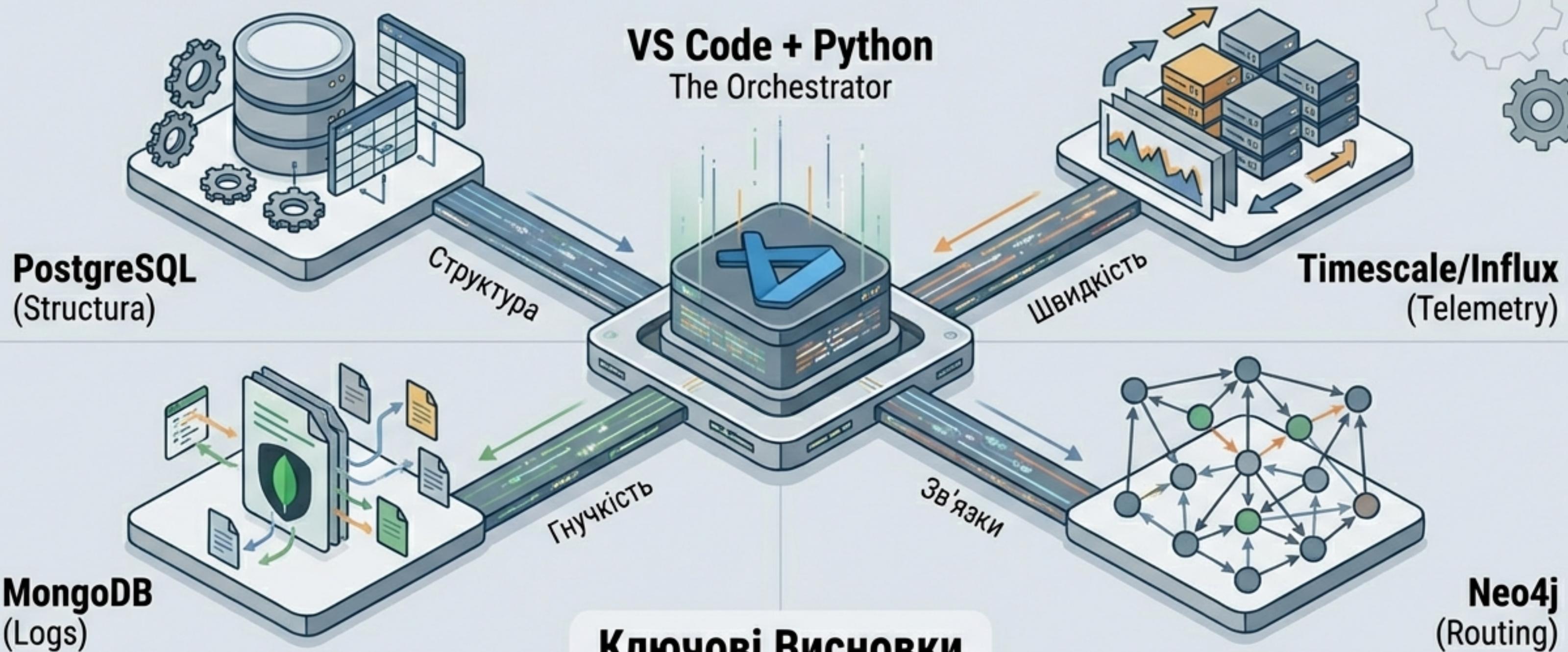
Робочий Процес:

1. Генерація HTML через Python.
2. Відкриття через "Live Preview" у VS Code.

```
net = Network(notebook=True)
net.add_edge("Kyiv", "Lviv", title="5 hours")
net.show("map.html")
```

Примітка: PyVis використовує physics engine для розміщення вузлів у реальному часі.

Єдина Архітектура: Синтез Технологій



Ключові Висновки

Правильний інструмент для правильної роботи.

Централізація налаштувань та коду у VS Code мінімізує перемикання контексту.

Python як єдиний "клей" для всіх типів баз даних.