

La méthode la plus rapide
pour charger des données
dans PostgreSQL à l'aide
de Python

Qui suis-je ?



Mohamed Nossirat

Ingénieur Logiciel / E-Santé

Python / Go / Rust ?



python



TOULOUSE

Le Challenge

-  Connecter des systèmes qui n'étaient pas conçus pour communiquer entre eux ➡ Interopérabilité
-  Intégrations d'API externes partout
-  Traitement de volumes massifs de données
-  Sources de données réelles et hétérogènes : JSON / XML / CSV

```

[
- {
  id: 1,
  name: "Punk IPA 2007 - 2010",
  tagline: "Post Modern Classic. Spiky. Tropical. Hoppy.",
  first_brewed: "04/2007",
  description: "Our flagship beer that kick started the craft beer revolution. This is James and Martin's original take on an American IPA, subverted with punchy New Zealand hops. Layered with new world hops to create an all-out riot of grapefruit, pineapple and lychee before a spiky, mouth-puckering bitter finish.",
  image: "001.png",
  abv: 6,
  ibu: 60,
  target_fg: 1010,
  target OG: 1056,
  ebc: 17,
  srm: 8.5,
  ph: 4.4,
  attenuation_level: 82.14,
- volume: {
  value: 20,
  unit: "litres"
},
- boil_volume: {
  value: 25,
  unit: "litres"
},
- method: {
  - mash_temp: [
    - {
      - temp: {
        value: 65,
        unit: "celsius"
      },
      duration: 75
    },
  ],
  - fermentation: {
    - temp: {
      value: 19,
      unit: "celsius"
    }
  },
  twist: null
},
- ingredients: {
  - malt: [
    - {
      name: "Extra Pale",
      - amount: {
        value: 5.3,
        unit: "kilograms"
      }
    },
  ],
  - hops: [
    - {
      name: "Ahtanum",
      - amount: {

```

```
from typing import Iterator, Dict, Any
from urllib.parse import urlencode
import requests

def iter_beers_from_api(page_size: int = 30) -> Iterator[Dict[str, Any]]:
    session = requests.Session()
    page = 1
    while True:
        response = session.get('https://punkapi-alxiw.amvera.io/v3/beers?' + urlencode({
            'page': page,
            'per_page': page_size
        })))
        response.raise_for_status()

        data = response.json()
        if not data:
            break

        yield from data

        page += 1
```

```
>>> beers = list(iter_beers_from_api()) * 100
>>> len(beers)
41,500
```

```
>>> process(iter_beers_from_api())
```

```
services:
  db:
    image: postgres:17
    environment:
      - POSTGRES_USER=test
      - POSTGRES_PASSWORD=test
      - POSTGRES_DB=test
    ports:
      - 5432:5432
```

```
docker compose up -d
```

```
mkdir pythonpostgresql  
cd pythonpostgresql  
uv init .  
uv add "psycpg2-binary memory-profiler"
```

```
import psycopg

connection = psycopg.connect(
    host="localhost",
    dbname="test",
    user="test",
    password="test")

connection.autocommit = True

def create_staging_table(cursor) -> None:
    cursor.execute("""
        DROP TABLE IF EXISTS staging_beers;
        CREATE UNLOGGED TABLE staging_beers (
            id            INTEGER,
            name          TEXT,
            tagline       TEXT,
            first_brewed  DATE,
            description   TEXT,
            image         TEXT,
            abv           DECIMAL,
            ibu           DECIMAL,
            target_fg     DECIMAL,
            target_og     DECIMAL,
            ebc           DECIMAL,
            srm           DECIMAL,
            ph            DECIMAL,
            attenuation_level DECIMAL,
            brewers_tips  TEXT,
            contributed_by TEXT,
            volume        INTEGER
        );
    """)
```

```
$ python -m memory_profiler example.py
```

Line #	Mem usage	Increment	Line Contents
=====			
3			@profile
4	5.97 MB	0.00 MB	def my_func():
5	13.61 MB	7.64 MB	a = [1] * (10 ** 6)
6	166.20 MB	152.59 MB	b = [2] * (2 * 10 ** 7)
7	13.61 MB	-152.59 MB	del b
8	13.61 MB	0.00 MB	return a

```
import time
from functools import wraps
from memory_profiler import memory_usage

def profile(fn):
    @wraps(fn)
    def inner(*args, **kwargs):
        fn_kwargs_str = ', '.join(f'{k}={v}' for k, v in kwargs.items())
        print(f'\n{fn.__name__}({fn_kwargs_str})')

        # Measure time
        t = time.perf_counter()
        retval = fn(*args, **kwargs)
        elapsed = time.perf_counter() - t
        print(f'Time {elapsed:0.4}')

        # Measure memory
        mem, retval = memory_usage((fn, args, kwargs), retval=True, timeout=200, interval=1e-7)

        print(f'Memory {max(mem) - min(mem)}')
        return retval

    return inner
```

```

@profile
def insert_one_by_one(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)
        for beer in beers:
            cursor.execute("""
                INSERT INTO staging_beers VALUES (
                    %(id)s,
                    %(name)s,
                    %(tagline)s,
                    %(first_brewed)s,
                    %(description)s,
                    %(image_url)s,
                    %(abv)s,
                    %(ibu)s,
                    %(target_fg)s,
                    %(target_og)s,
                    %(ebc)s,
                    %(srm)s,
                    %(ph)s,
                    %(attenuation_level)s,
                    %(brewers_tips)s,
                    %(contributed_by)s,
                    %(volume)s
                );
            """, {
                **beer,
                'first_brewed': parse_first_brewed(beer['first_brewed']),
                'volume': beer['volume']['value'],
            })

```

```
>>> insert_one_by_one(connection, beers)
insert_one_by_one()
Time    128.8
Memory 0.08203125
```

```

@profile
def insert_executemany(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)

        all_beers = [{
            **beer,
            'first_brewed': parse_first_brewed(beer['first_brewed']),
            'volume': beer['volume']['value'],
        } for beer in beers]

        cursor.executemany("""
            INSERT INTO staging_beers VALUES (
                %(id)s,
                %(name)s,
                %(tagline)s,
                %(first_brewed)s,
                %(description)s,
                %(image_url)s,
                %(abv)s,
                %(ibu)s,
                %(target_fg)s,
                %(target_og)s,
                %(ebc)s,
                %(srm)s,
                %(ph)s,
                %(attenuation_level)s,
                %(brewers_tips)s,
                %(contributed_by)s,
                %(volume)s
            );
        """, all_beers)

```

```
>>> insert_executemany(connection, beers)
insert_executemany()
Time    124.7
Memory 2.765625
```

```

@profile
def insert_executemany_iterator(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)
        cursor.executemany("""
            INSERT INTO staging_beers VALUES (
                %(id)s,
                %(name)s,
                %(tagline)s,
                %(first_brewed)s,
                %(description)s,
                %(image_url)s,
                %(abv)s,
                %(ibu)s,
                %(target_fg)s,
                %(target_og)s,
                %(ebc)s,
                %(srm)s,
                %(ph)s,
                %(attenuation_level)s,
                %(brewers_tips)s,
                %(contributed_by)s,
                %(volume)s
            );
            """, ({
                **beer,
                'first_brewed': parse_first_brewed(beer['first_brewed']),
                'volume': beer['volume']['value'],
            } for beer in beers))

```

```
>>> insert_executemany_iterator(connection, beers)
insert_executemany_iterator()
Time    129.3
Memory  0.0
```

```

import psycpg2.extras

@profile
def insert_execute_batch(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)

    all_beers = [{
        **beer,
        'first_brewed': parse_first_brewed(beer['first_brewed']),
        'volume': beer['volume']['value'],
    } for beer in beers]

    psycpg2.extras.execute_batch(cursor, """
        INSERT INTO staging_beers VALUES (
            %(id)s,
            %(name)s,
            %(tagline)s,
            %(first_brewed)s,
            %(description)s,
            %(image_url)s,
            %(abv)s,
            %(ibu)s,
            %(target_fg)s,
            %(target_og)s,
            %(ebc)s,
            %(srm)s,
            %(ph)s,
            %(attenuation_level)s,
            %(brewers_tips)s,
            %(contributed_by)s,
            %(volume)s
        );
    """, all_beers)

```

```
>>> insert_execute_batch(connection, beers)
insert_execute_batch()
Time    3.917
Memory 2.50390625
```

```

@profile
def insert_execute_batch_iterator(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)

        iter_beers = ({
            **beer,
            'first_brewed': parse_first_brewed(beer['first_brewed']),
            'volume': beer['volume']['value'],
        } for beer in beers)

        psycpg2.extras.execute_batch(cursor, """
            INSERT INTO staging_beers VALUES (
                %(id)s,
                %(name)s,
                %(tagline)s,
                %(first_brewed)s,
                %(description)s,
                %(image_url)s,
                %(abv)s,
                %(ibu)s,
                %(target_fg)s,
                %(target_og)s,
                %(ebc)s,
                %(srm)s,
                %(ph)s,
                %(attenuation_level)s,
                %(brewers_tips)s,
                %(contributed_by)s,
                %(volume)s
            );
            """, iter_beers)

```

```
>>> insert_execute_batch_iterator(connection, beers)
insert_execute_batch_iterator()
Time    4.333
Memory 0.2265625
```

```

import psycopg2.extras

@profile
def insert_execute_values(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)
        psycopg2.extras.execute_values(cursor, """
            INSERT INTO staging_beers VALUES %s;
            """, [(
                beer['id'],
                beer['name'],
                beer['tagline'],
                parse_first_brewed(beer['first_brewed']),
                beer['description'],
                beer['image_url'],
                beer['abv'],
                beer['ibu'],
                beer['target_fg'],
                beer['target_og'],
                beer['ebc'],
                beer['srm'],
                beer['ph'],
                beer['attenuation_level'],
                beer['brewers_tips'],
                beer['contributed_by'],
                beer['volume']['value'],
            ) for beer in beers])

```

```
>>> insert_execute_values(connection, beers)
insert_execute_values()
Time    3.666
Memory 4.50390625
```

@profile

```
def insert_execute_values_iterator(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)
        psycopg2.extras.execute_values(cursor, """
            INSERT INTO staging_beers VALUES %s;
        """, ((
            beer['id'],
            beer['name'],
            beer['tagline'],
            parse_first_brewed(beer['first_brewed']),
            beer['description'],
            beer['image_url'],
            beer['abv'],
            beer['ibu'],
            beer['target_fg'],
            beer['target_og'],
            beer['ebc'],
            beer['srm'],
            beer['ph'],
            beer['attenuation_level'],
            beer['brewers_tips'],
            beer['contributed_by'],
            beer['volume']['value'],
        )) for beer in beers))
```

```
>>> insert_execute_values_iterator(connection, beers)
insert_execute_values_iterator()
Time    3.677
Memory  0.0
```

COPY

- **Performance maximale** : Selon la documentation officielle de PostgreSQL, c'est la méthode la plus rapide pour peupler une base de données.
- **Efficacité** : Réduit considérablement la charge CPU et le temps de transaction par rapport à des multiples commandes `INSERT`.
- **Standard CSV** : Utilise le format CSV comme pivot pour le transfert de données.

```

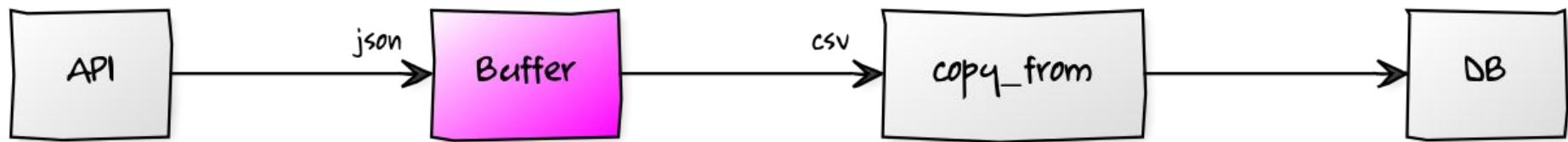
import io

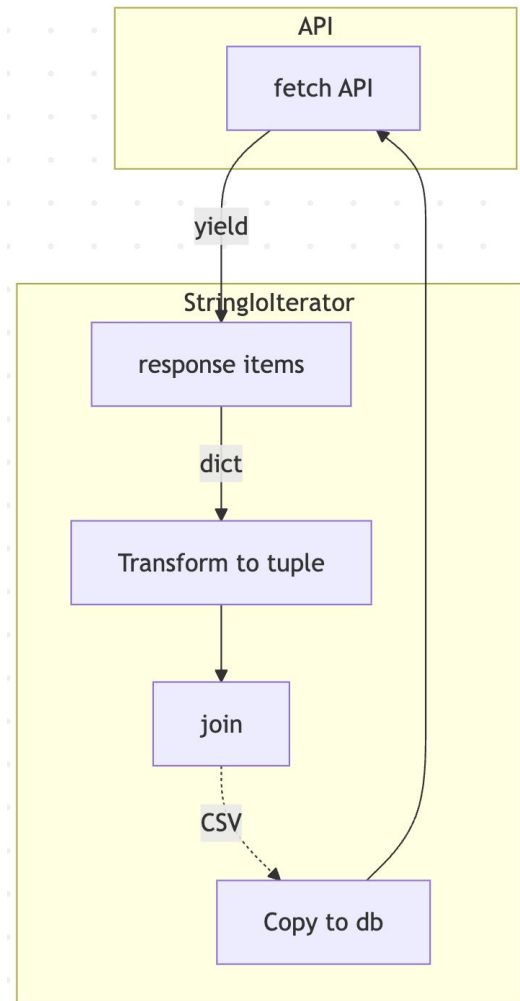
def clean_csv_value(value: Optional[Any]) -> str:
    if value is None:
        return r'\N'
    return str(value).replace('\n', '\\n')

@profile
def copy_stringio(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)
        csv_file_like_object = io.StringIO()
        for beer in beers:
            csv_file_like_object.write(''.join(map(clean_csv_value, (
                beer['id'],
                beer['name'],
                beer['tagline'],
                parse_first_brewed(beer['first_brewed']),
                beer['description'],
                beer['image_url'],
                beer['abv'],
                beer['ibu'],
                beer['target_fg'],
                beer['target_og'],
                beer['ebc'],
                beer['srm'],
                beer['ph'],
                beer['attenuation_level'],
                beer['contributed_by'],
                beer['brewers_tips'],
                beer['volume']['value'],
            ))) + '\n')
        csv_file_like_object.seek(0)
        cursor.copy_from(csv_file_like_object, 'staging_beers', sep='|')

```

```
>>> copy_stringio(connection, beers)
copy_stringio()
Time    0.6274
Memory 99.109375
```





```

from typing import Iterator, Optional
import io

class StringIteratorIO(io.TextIOBase):
    def __init__(self, iter: Iterator[str]):
        self._iter = iter
        self._buff = ''

    def readable(self) -> bool:
        return True

    def _read1(self, n: Optional[int] = None) -> str:
        while not self._buff:
            try:
                self._buff = next(self._iter)
            except StopIteration:
                break
        ret = self._buff[:n]
        self._buff = self._buff[len(ret):]
        return ret

    def read(self, n: Optional[int] = None) -> str:
        line = []
        if n is None or n < 0:
            while True:
                m = self._read1()
                if not m:
                    break
                line.append(m)
        else:
            while n > 0:
                m = self._read1(n)
                if not m:
                    break
                n -= len(m)
                line.append(m)
        return ''.join(line)

```

```
from typing import Iterator, Optional
import io

class StringIteratorIO(io.TextIOBase):
    def __init__(self, iter: Iterator[str]):
        self._iter = iter
        self._buff = ''

    def readable(self) -> bool:
        return True

    def _read1(self, n: Optional[int] = None) -> str:
        while not self._buff:
            try:
                self._buff = next(self._iter)
            except StopIteration:
                break
        ret = self._buff[:n]
        self._buff = self._buff[len(ret):]
        return ret

    def read(self, n: Optional[int] = None) -> str:
        line = []
        if n is None or n < 0:
            while True:
                m = self._read1()
                if not m:
                    break
                line.append(m)
        else:
            while n > 0:
                m = self._read1(n)
                if not m:
                    break
                n -= len(m)
                line.append(m)
        return ''.join(line)
```



```

@profile
def copy_string_iterator(connection, beers: Iterator[Dict[str, Any]]) -> None:
    with connection.cursor() as cursor:
        create_staging_table(cursor)
        beers_string_iterator = StringIteratorIO((
            '|'.join(map(clean_csv_value, (
                beer['id'],
                beer['name'],
                beer['tagline'],
                parse_first_brewed(beer['first_brewed']).isoformat(),
                beer['description'],
                beer['image_url'],
                beer['abv'],
                beer['ibu'],
                beer['target_fg'],
                beer['target_og'],
                beer['ebc'],
                beer['srm'],
                beer['ph'],
                beer['attenuation_level'],
                beer['brewers_tips'],
                beer['contributed_by'],
                beer['volume']['value'],
            ))) + '\n'
            for beer in beers
        ))
        cursor.copy_from(beers_string_iterator, 'staging_beers', sep='|')

```

```
>>> copy_string_iterator(connection, beers)
copy_string_iterator()
Time    0.4596
Memory  0.0
```

FUNCTION	TIME (SECONDS)	MEMORY (MB)
insert_one_by_one()	128.8	0.08203125
insert_executemany()	124.7	2.765625
insert_executemany_iterator()	129.3	0.0
insert_execute_batch()	3.917	2.50390625
insert_execute_batch_iterator(page_size=1)	130.2	0.0
insert_execute_batch_iterator(page_size=100)	4.333	0.0
insert_execute_batch_iterator(page_size=1000)	2.537	0.2265625
insert_execute_batch_iterator(page_size=10000)	2.585	25.4453125
insert_execute_values()	3.666	4.50390625
insert_execute_values_iterator(page_size=1)	127.4	0.0
insert_execute_values_iterator(page_size=100)	3.677	0.0
insert_execute_values_iterator(page_size=1000)	1.468	0.0
insert_execute_values_iterator(page_size=10000)	1.503	2.25
copy_stringio()	0.6274	99.109375
copy_string_iterator(size=1024)	0.4536	0.0
copy_string_iterator(size=8192)	0.4596	0.0
copy_string_iterator(size=16384)	0.4649	0.0
copy_string_iterator(size=65536)	0.6171	0.0

Merci pour votre attention

Questions ?