Usando Python+SQL para acessar APIs

Beto Dealmeida | 2021.10.13

Quem sou eu?

- Dr. em **Oceanografia Física** pela USP
- Apaixonado por **Python** e **software livre** desde sempre
- Desenvolvedor do Apache Superset™
- Engenheiro na preset™
- Uso **barba** e **coque**

Barba + coque = más escolhas

Acessar APIs via SQL?

Quantos PR abertos/fechados no repositório do Python?

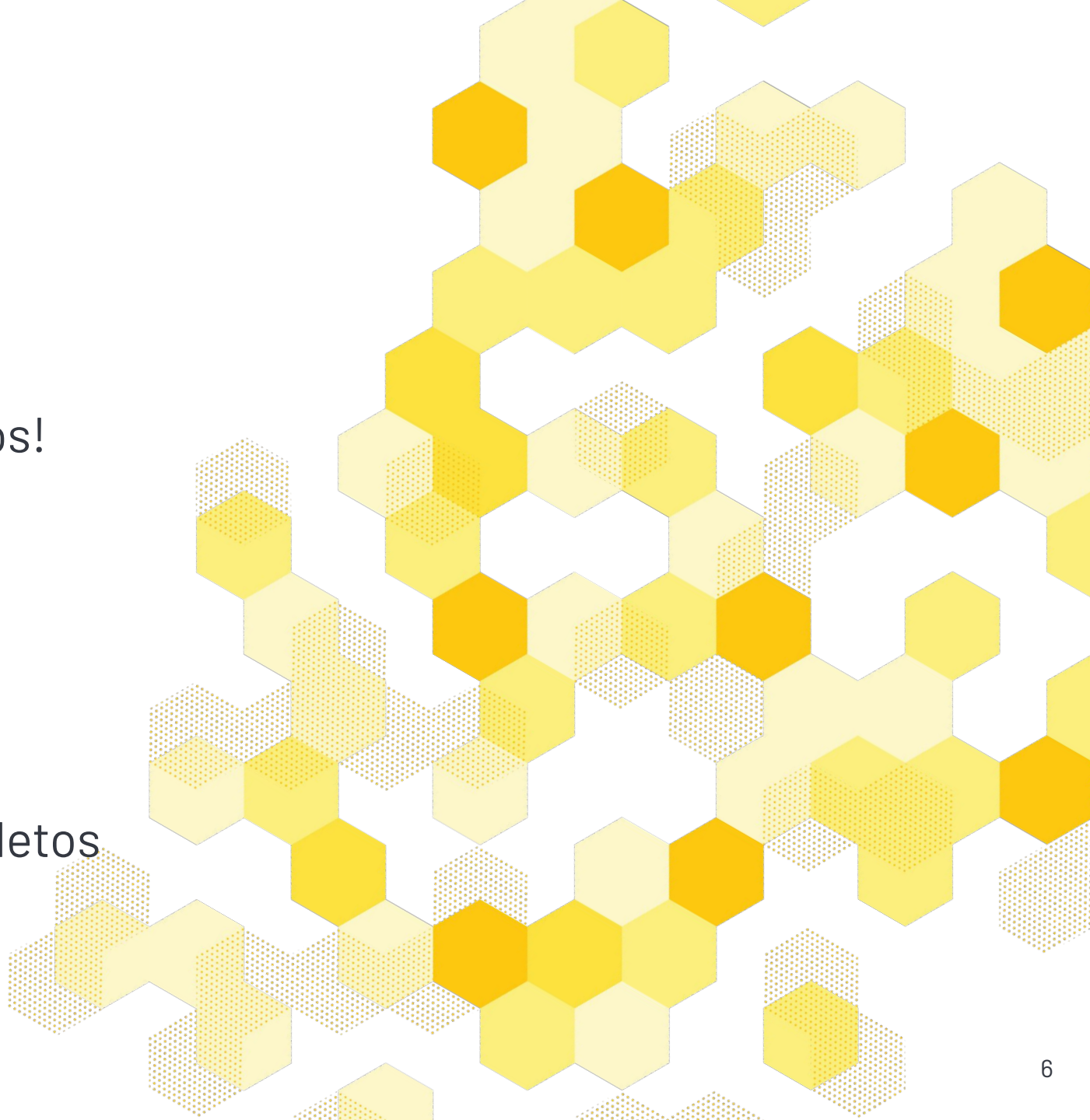
```
baseurl = "https://api.github.com/repos/python/cpython/pulls"

states = defaultdict(int)
i = 0
while True:
    i += 1
    response = requests.get(f"{baseurl}?per_page=100&page={i}")
    payload = response.json()
    if not payload:
        break
    for pr in payload:
        states[pr['state']] += 1

for key, value in states.items():
    print(f'{key}: {value}')
```

Acessar APIs via SQL!

Por que SQL?

Por que SQL?

- Inventada em **1974** — há quase 50 anos!
  - Dinossauro vs. tubarão
- Linguagem **declarativa**
  - Relativamente fácil de aprender
- **Lingua franca** para manipular dados
  - Infelizmente existem muitos dialetos incompatíveis



Imperativa vs. declarativa

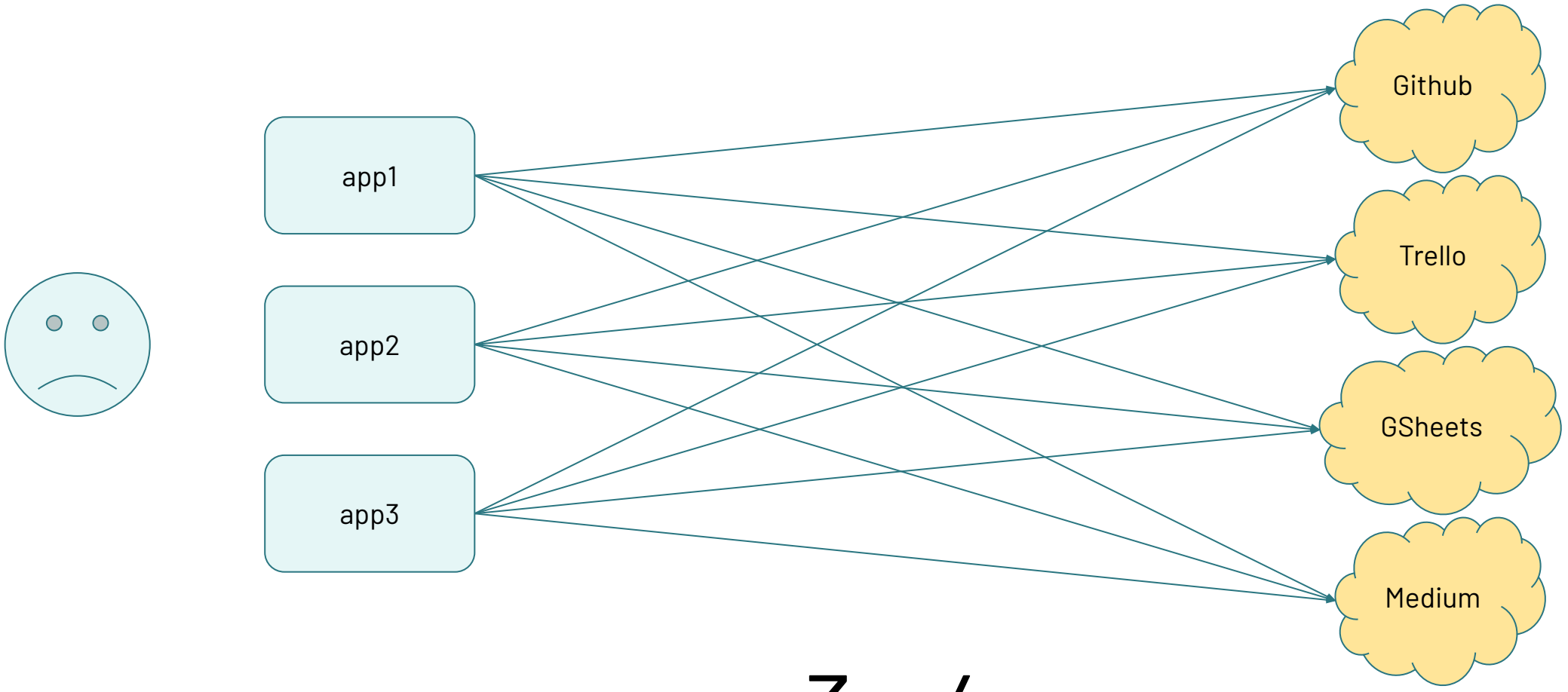
```python
baseurl = "https://api.github.com/repos/python/cpython/pulls"

states = defaultdict(int)
i = 0
while True:
    i += 1
    response = requests.get(
        f"{baseurl}?state=all&per_page=100&page={i}"
    )
    payload = response.json()
    if not payload:
        break
    for pr in payload:
        states[pr['state']] += 1

for key, value in states.items():
    print(f'{key}: {value}')
```

preset

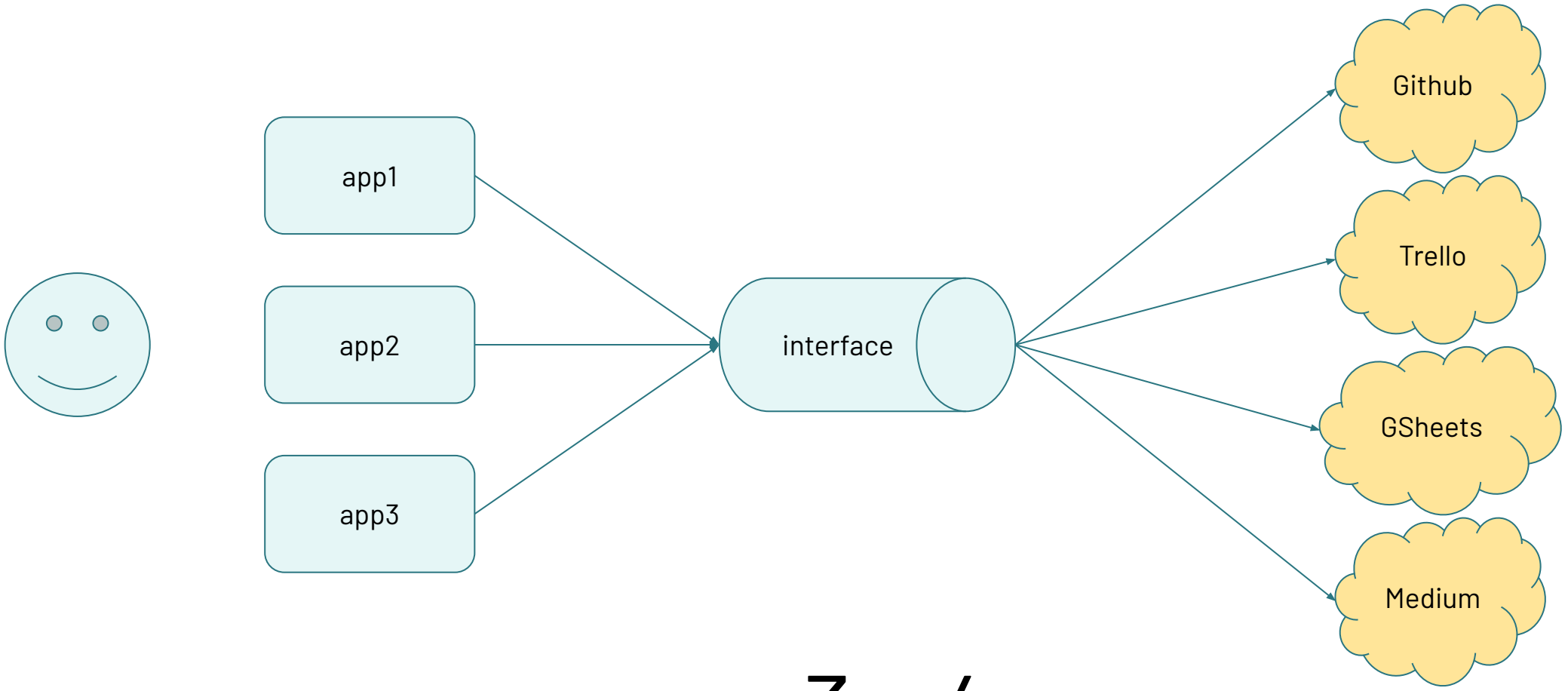
Curso rápido de SQL

```sql
CREATE TABLE animais (animal STRING, pernas INTEGER);
INSERT INTO animais (animal, pernas) VALUES ('cachorro', 4);
INSERT INTO animais (animal, pernas) VALUES ('galinha', 2);

SELECT animal AS bipede FROM animais WHERE pernas = 2;
SELECT animal AS esquisito FROM animais WHERE pernas = 3;

DELETE FROM animais WHERE pernas = 3;
UPDATE animais SET pernas = 2 WHERE pernas = 3;
```

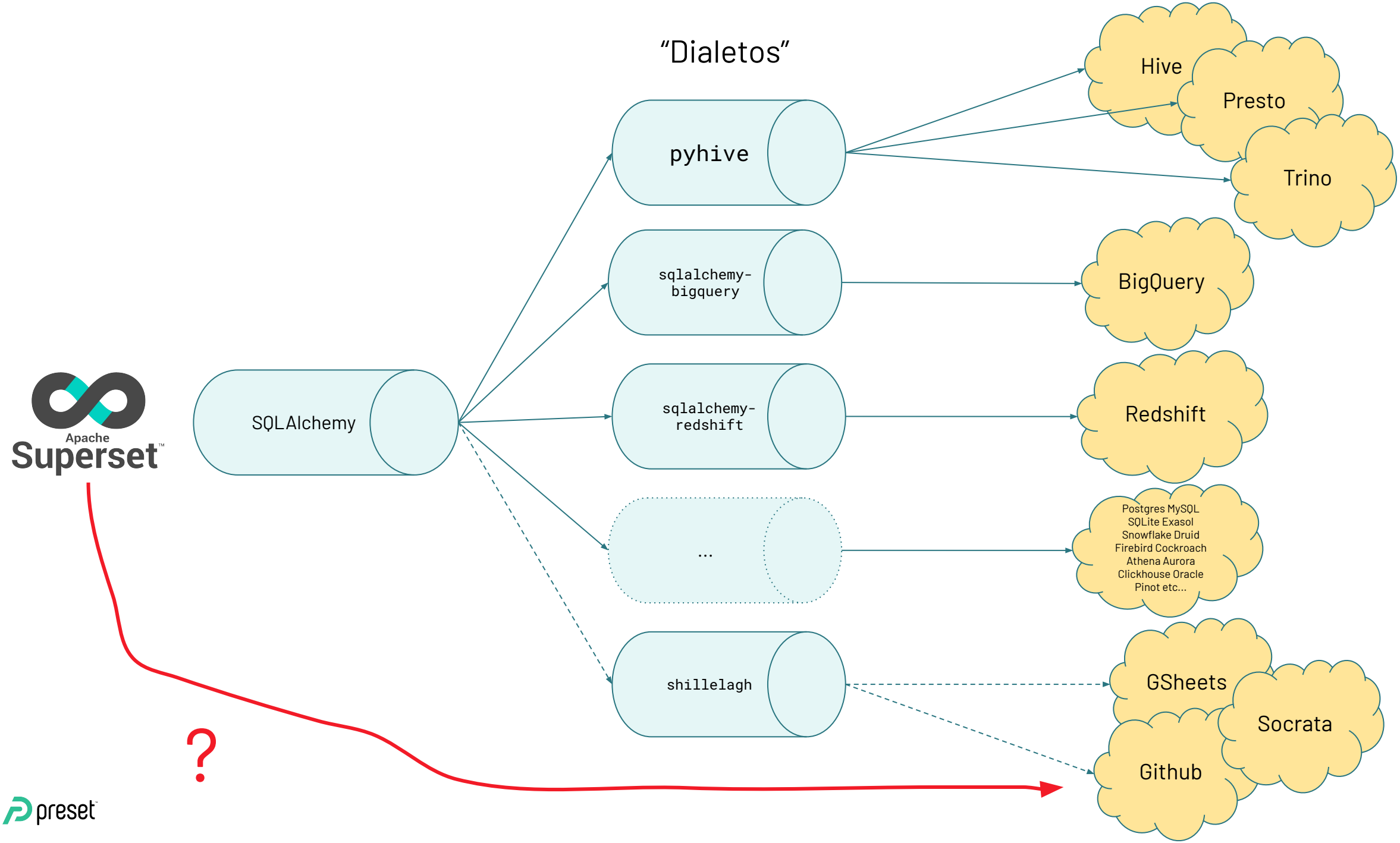
Por que SQL para APIs?



3 x 4



3 + 4


"Dialetos"
Apache Superset
SQLAlchemy
pyhive
sqlalchemy-bigquery
sqlalchemy-redshift
...
shillelagh
Hive
Presto
Trino
BigQuery
Redshift
Postgres MySQL
SQLite Exasol
Snowflake Druid
Firebird Cockroach
Athena Aurora
Clickhouse Oracle
Pinot etc...
GSheets
Socrata
Github
?

preset

SQLAlchemy e DB API 2.0

```
from mylib import run_query
access = run_query("SELECT * FROM users")
results = access.read_data()


from otherlib import execute
data = execute("SELECT * FROM users")
```

PEP 249

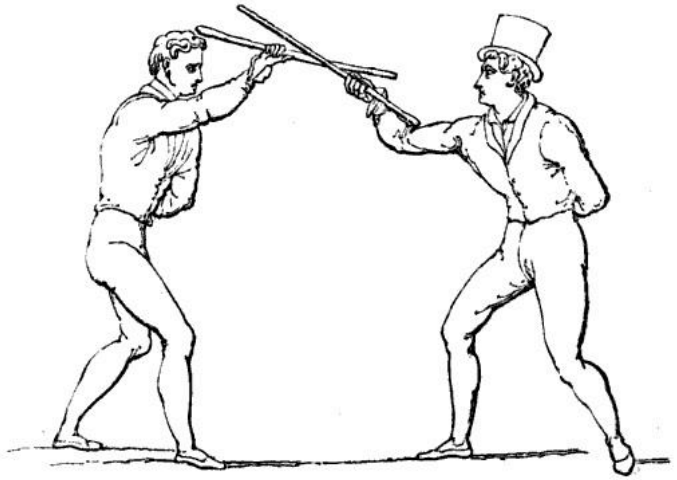
```
from mylib import connection
conn = connection()
cursor = conn.cursor()
cursor.execute("SELECT * FROM users WHERE id=?")
results = cursor.fetchall()

from otherlib import connection
…
cursor.execute("SELECT * FROM users WHERE id=:id")
```

SQLAlchemy

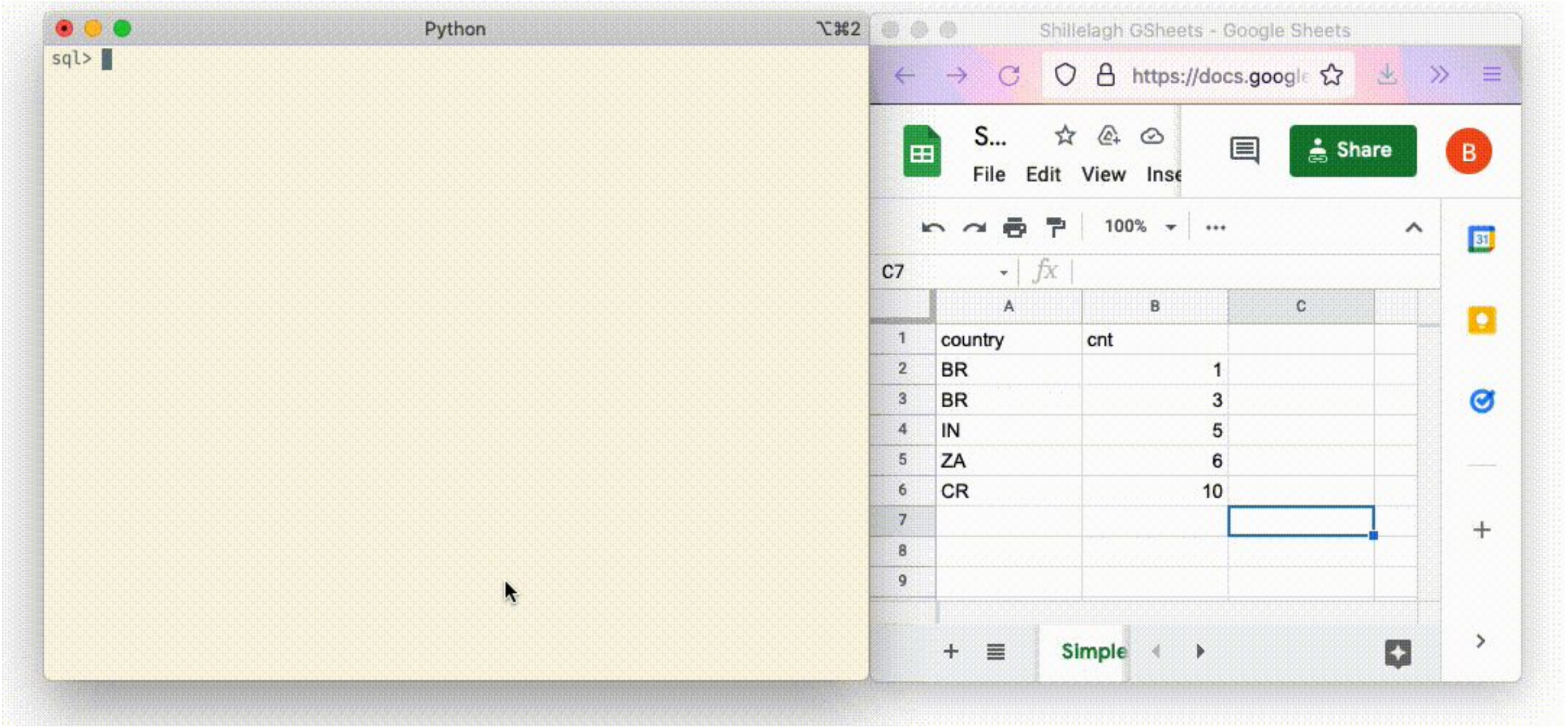
```
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String)

session.query(User).filter_by(id=12)



s = select(users).where(users.c.id == 12)
for row in conn.execute(s):
    print(row)
```

Shillelagh
(xi-LÊI-li)

1. **Cajado** irlandês, usado para combate e apoio
2. Magia no jogo **Dungeons & Dragons**
3. Biblioteca/CLI Python para acessar **APIs** via **SQL**



Exemplo


Python
⌥⌘2
sql>
Shillelagh GSheets - Google Sheets
https://docs.google
S...
File
Edit
View
Inse
Share
B
100%
C7
fx
A
B
C
1 country cnt
2 BR 1
3 BR 3
4 IN 5
5 ZA 6
6 CR 10
7
8
9
Simple

APIs compatíveis

- Github
- Google Sheets
- **WeatherAPI**
- Socrata Open Data API
- Datassette
- Pandas
- CSV

```
from datetime import datetime, timedelta
from shillelagh.backends.apsw.db import connect

connection = connect(":memory:")
cursor = connection.cursor()

three_days_ago = datetime.now() - timedelta(days=3)

sql = """
SELECT *
FROM "https://api.weatherapi.com/v1/history.json?q=94923"
WHERE time >= ?
"""
for row in cursor.execute(sql, (three_days_ago,)):
    print(row)
```

APIs compatíveis

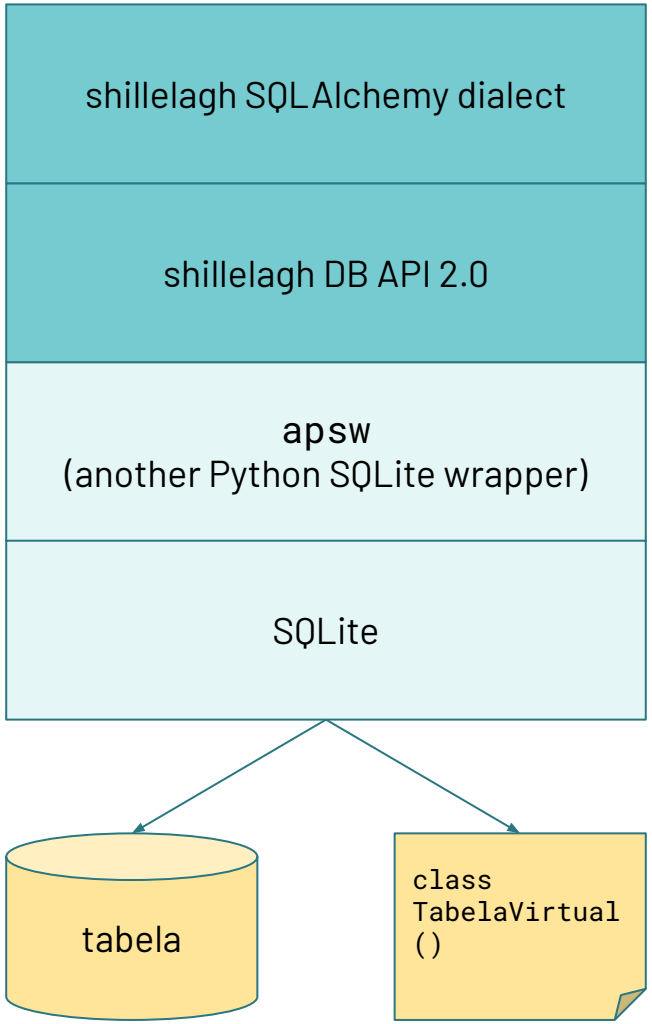
- Github
- Google Sheets
- WeatherAPI
- Socrata Open Data API
- Datassette
- **Pandas**
- CSV

```
import pandas as pd
from shillelagh.backends.apsw.db import connect

connection = connect(":memory:")
cursor = connection.cursor()

mydf = pd.DataFrame({"a": [1, 2, 3]})

sql = "SELECT SUM(a) FROM mydf"
for row in cursor.execute(sql):
    print(row)
```

Como funciona a biblioteca?

Como funciona?

Tabelas virtuais



Como funciona?

Tabelas virtuais

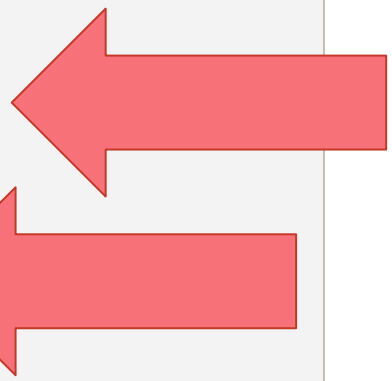
```
import apsw

connection = apsw.Connection(":memory:")
connection.createmodule("gsheetsapi", MyModule)

cursor = connection.cursor()
cursor.execute("""
  CREATE VIRTUAL TABLE my_table
  USING gsheetsapi('https://docs.google.com/spreadsheets/d/XXX/edit#gid=0');
""")
cursor.execute("SELECT * FROM my_table")
```

```
tabela = MyModule('https://docs.google.com/spreadsheets/d/XXX/edit#gid=0')
tabela.BestIndex(...)
cursor = tabela.Open(...)
cursor.Filter(...)
```

Como funciona?

Tabelas virtuais

```
import apsw

from shillelagh.adapters.api.gsheets.adapter import GSheetsAdapter
from shillelagh.backends.apsw.vt import VTModule

connection = apsw.Connection(":memory:")
connection.createmodule("gsheetsapi", VTModule(GSheetsAdapter))

cursor = connection.cursor()
cursor.execute("""
  CREATE VIRTUAL TABLE "https://docs.google.com/spreadsheets/d/XXX/edit#gid=0"
  USING gsheetsapi('https://docs.google.com/spreadsheets/d/XXX/edit#gid=0');
""")
cursor.execute(
    'SELECT * FROM "https://docs.google.com/spreadsheets/d/XXX/edit#gid=0"'
)
```



Funciona, mas é bastante trabalho!

Se uma pessoa precisa digitar ***a mesma coisa*** *todas as vezes que ela usa seu produto... você pode digitar por ela*

Como funciona?

Tabelas virtuais

```
import apsw

from shillelagh.adapters.api.gsheets.adapters import GSheetsAdapter
from shillelagh.backends.apsw.vt import VTModule

connection = apsw.Connection(":memory:")
connection.createmodule("gsheetsapi", VTModule(GSheetsAdapter))

cursor = connection.cursor()
cursor.execute("""
  CREATE VIRTUAL TABLE "https://docs.google.com/spreadsheets/d/XXX/edit#gid=0"
  USING gsheetsapi('https://docs.google.com/spreadsheets/d/XXX/edit#gid=0');
""")
cursor.execute(
    'SELECT * FROM "https://docs.google.com/spreadsheets/d/XXX/edit#gid=0"'
)
```

Ninguém tem tempo pra isso...

```
while True:
    try:
        self._cursor.execute(operation, parameters)
        self.description = self._get_description()
        self._results = self._convert(self._cursor)
        break
    except apsw.SQLError as ex:
        message = ex.args[0]
        if not message.startswith(NO_SUCH_TABLE):
            raise ProgrammingError(message) from ex

        # create the virtual table
        uri = message[len(NO_SUCH_TABLE) :]
        self._create_table(uri)
```

```
SELECT * FROM "https://api.example.com/"
```

```
NO SUCH TABLE: https://api.example.com/
```

```
from shillelagh.lib import find_adapter

adapter = find_adapter('https://api.example.com/')

connection.createmodule(
    adapter.__name__,
    VTModule(adapter)
)
cursor.execute(
    f'CREATE VIRTUAL TABLE "{table_name}" '
    f'USING {adapter.__name__}({formatted_args})',
)
```

3 maneiras de usar

```
from shillelagh.backends.apsw.db import connect

connection = connect(":memory:")
cursor = connection.cursor()

query = 'SELECT * FROM "https://api.example.com/"'
for row in cursor.execute(query):
    print(row)
```

```
$ shillelagh
sql> SELECT * FROM "https://api.example.com/"
```

```
from sqlalchemy.engine import create_engine

engine = create_engine("shillelagh://")
connection = engine.connect()

query = 'SELECT * FROM "https://api.example.com/"'
for row in connection.execute(query):
    print(row)
```

Implementando novas APIs

APIs compatíveis

- Github
- Google Sheets
- WeatherAPI
- Socrata Open Data API
- Datassette
- Pandas
- CSV
- **<sua API aqui>**

Um “adaptador” para weatherapi.com

$ curl https://api.weatherapi.com/v1/history.json?
key=**XXX**&
q=**94401**&
dt=**2021-10-13**

retorna JSON com 20+ variáveis; nesse exemplo só queremos **time** e **temp_c**

Criando um “adaptador”

```python
class WeatherAPI(Adapter):

    time = DateTime(filters=[Range], order=Order.ASCENDING, exact=False
    temp_c = Float()

    def get_data(self, bounds: Dict[str, Filter]) -> Iterator[Dict[str, Any]]:
        start, end = get_range(bounds["time"])

        while start <= end:
            resp = requests.get(
                f"https://api.weatherapi.com/v1/history.json&dt={start
            )
            data = resp.json()
            for i, row in enumerate(data["forecast"]["forecastday"][0]["hour"]):
                yield dict(rowid=i, time=row["time"], temp_c=row["temp_

            start += timedelta(days=1)
```

```
SELECT *
FROM
"https://api.weatherapi.com/v1/history.json?q=Paris"
WHERE
  time >= '2021-10-13T12:00:00Z' AND
  time <= '2021-10-13T14:00:00Z'
```

```python
bounds = {
    'time': Range(
        datetime(2021, 10, 13, 12),
        datetime(2021, 10, 13, 14),
    ),
}
```

```python
start = date(2021, 10, 13)
end = date(2021, 10, 13)
```

Definindo responsabilidades

```
@staticmethod
def supports(uri: str) -> bool:
    """https://api.weatherapi.com/v1/history.json?key=XXX&q=94923"""
    parsed = urllib.parse.urlparse(uri)
    query_string = urllib.parse.parse_qs(parsed.query)
    return (
        parsed.netloc == "api.weatherapi.com"
        and parsed.path == "/v1/history.json"
        and "q" in query_string
        and "key" in query_string
    )
```

Usando o adaptador

Obrigado!
Perguntas?