#5.5

Data Platform Development

Psycopg3

"PostgreSQL & large data, secrets and understanding the basics"

Table of Contents

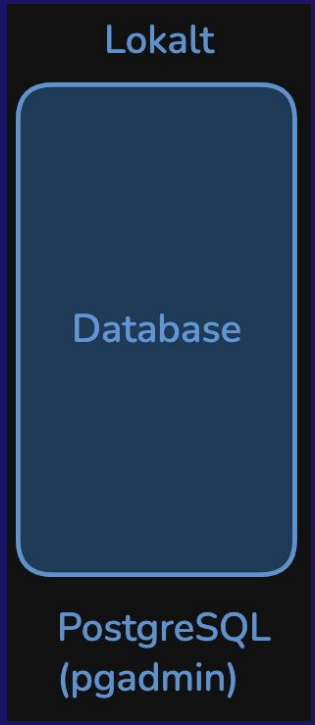
01

Workflow

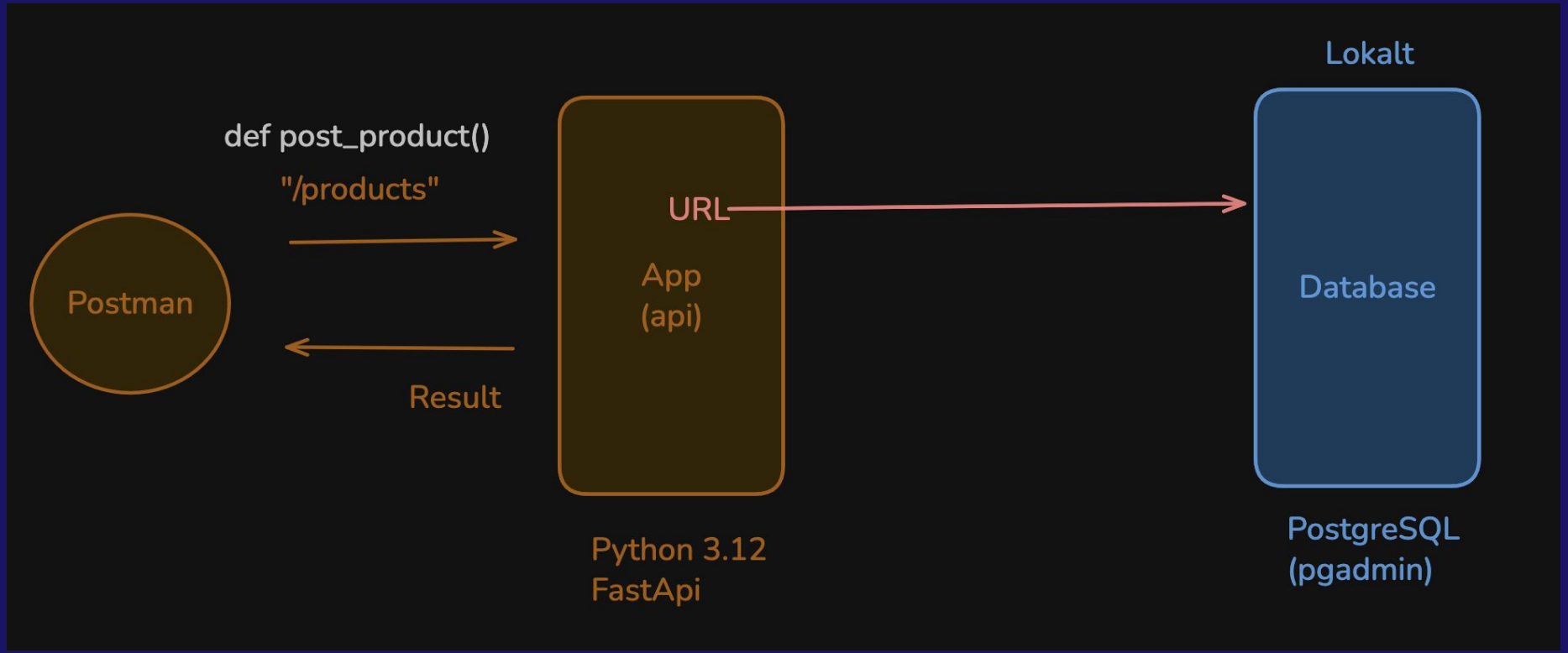

Workflow
Explained

Workflow
(How to communicate)

“A database is a separate computer.
psycopg is just a phone line to it”

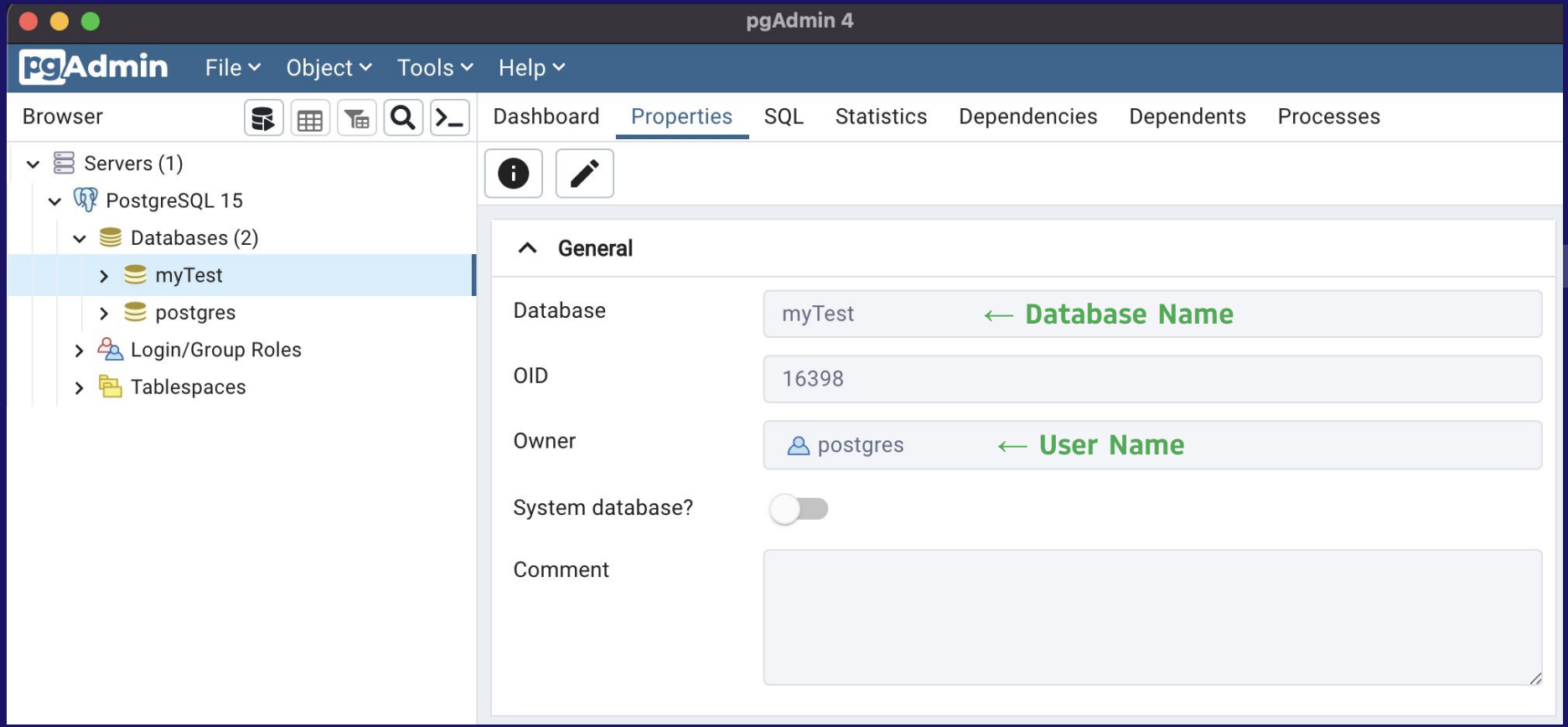


Workflow
(Visualized)



URL
(Structure)

```
"postgresql://USERNAME:PASSWORD@ADDRESS:PORT/DB_NAME"
```



PGADMIN4 example

Högerklicka på 'myTest' -> properties

02

Syntax Explained


Connection Pool
Visualized

```
8       DATABASE_URL = "postgresql://postgres:benny123@localhost:5432/lektion_5"
9       pool = ConnectionPool(DATABASE_URL)
```


```
POOL
  ├─ conn #1 (open)
  ├─ conn #2 (open)
  ├─ conn #3 (open)
  └─ conn #4 (open)
```

Opening new connections is SLOW and EXPENSIVE.
ConnectionPool object contain a list of already opened connections towards given URL

```
8   DATABASE_URL = "postgresql://postgres:benny123@localhost:5432/lektion_5"
9   pool = ConnectionPool(DATABASE_URL)
```

Pool Object

New (borrowed) Connection

```
with pool.connection() as conn:
```

Verify information by holding over

© psycopg_pool.pool.ConnectionPool

```
@contextmanager
def connection(self, timeout: float | None = None) -> Iterator[CT]
```

Context manager to obtain a connection from the pool.

Return the connection immediately if available, otherwise wait up to *timeout* or *self.timeout* seconds and throw *PoolTimeout* if a connection is not available in time.

Upon context exit, return the connection to the pool. Apply the normal :ref: `connection context behaviour <with-connection>` (commit/rollback the transaction in case of success/error). If the connection is no more in working state, replace it with a new one.

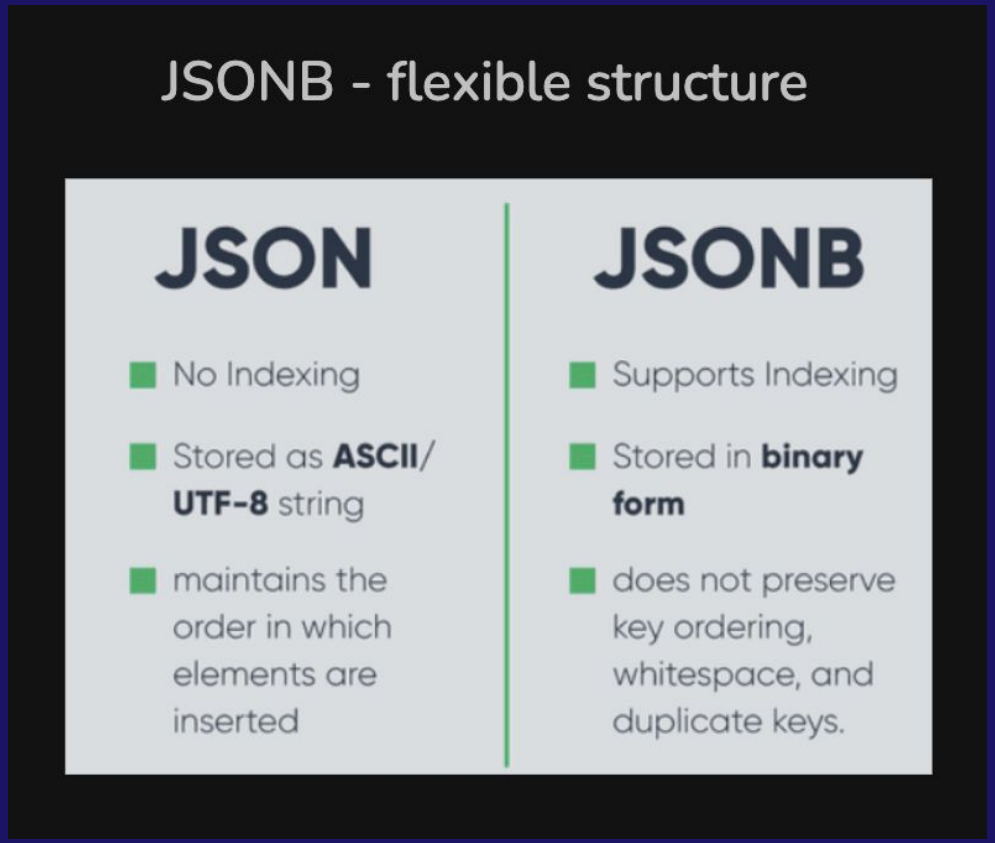
```
with pool.connection() as conn:
    with conn.transaction():
        conn.execute("QUERY")
```

A transaction is a protected block of database work where "everything succeeds together or nothing is saved"

03

SQL & Pydantic


PostgreSQL
Features



JSONB vs JSON

(PostgreSQL - Feature)

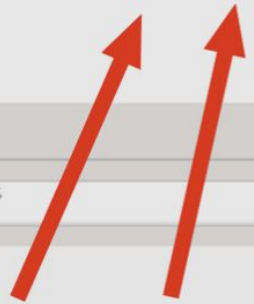
ARRAY - simple lists

```
SELECT name, tags FROM article WHERE tags && array['t1','t3','t10']::_varchar
```

article 1 ×

SELECT name, tags FROM article WHERE tags && array['t

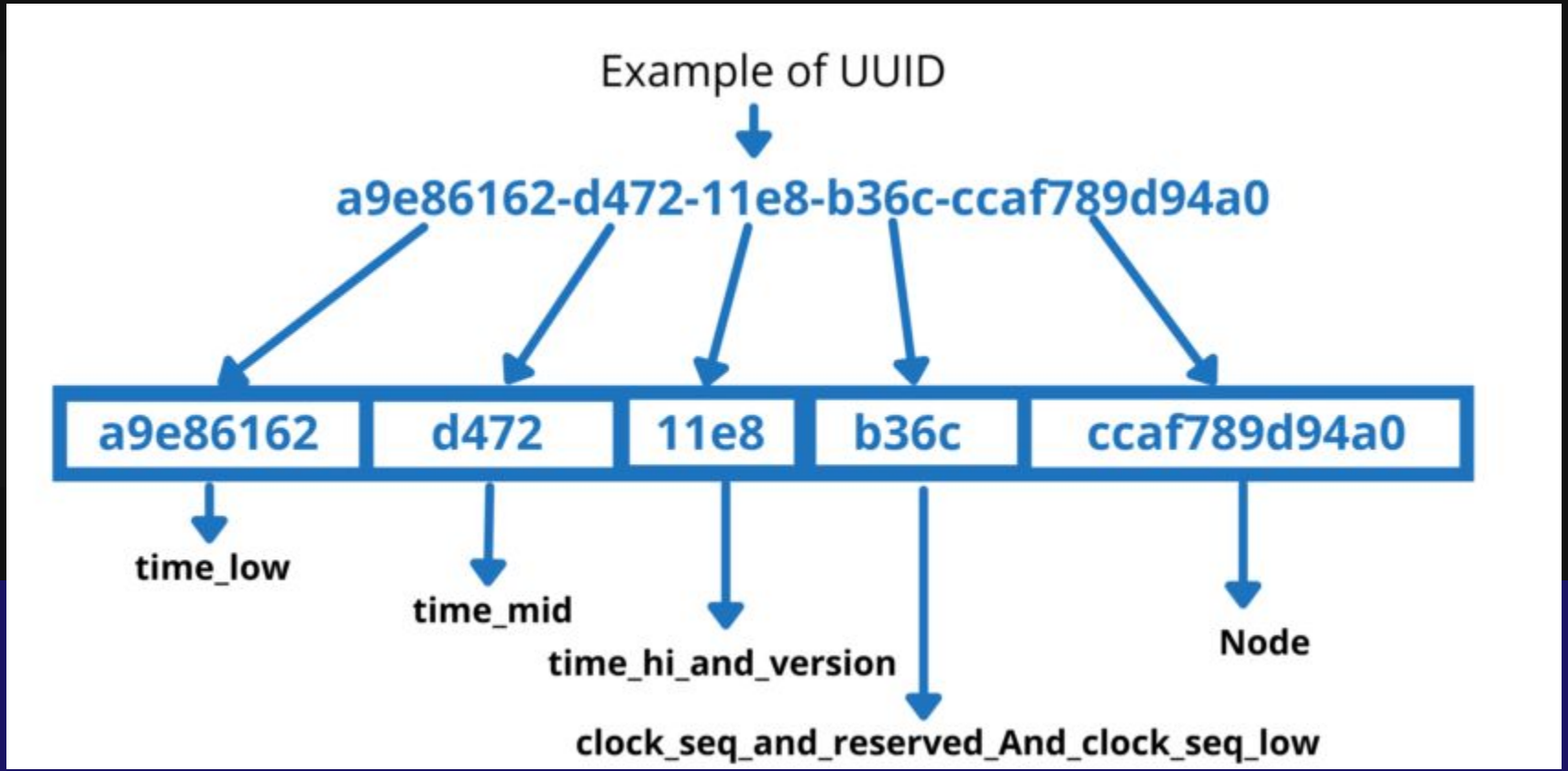
	name	tags
1	test1	{t1,t3}
2	test2	{t2,t3}
3	test3	{t3}
4	test4	{t1,t5}
5	test5	{t1,t3}
6	test6	{t2,t3}



UUID - safe IDs across systems

```sql
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL
);
```

UUID - safe IDs across systems




Primary Key
Auto Increment

```sql
CREATE TABLE IF NOT EXISTS product(
    id BIGINT GENERATED AS IDENTITY PRIMARY KEY
)
```

* Auto-incremented number
* Generated only when row is created
* Safe for concurrency (many inserts at once)
* Database-controlled (not app logic)

This works for all SQL versions...

But looks kind of long?

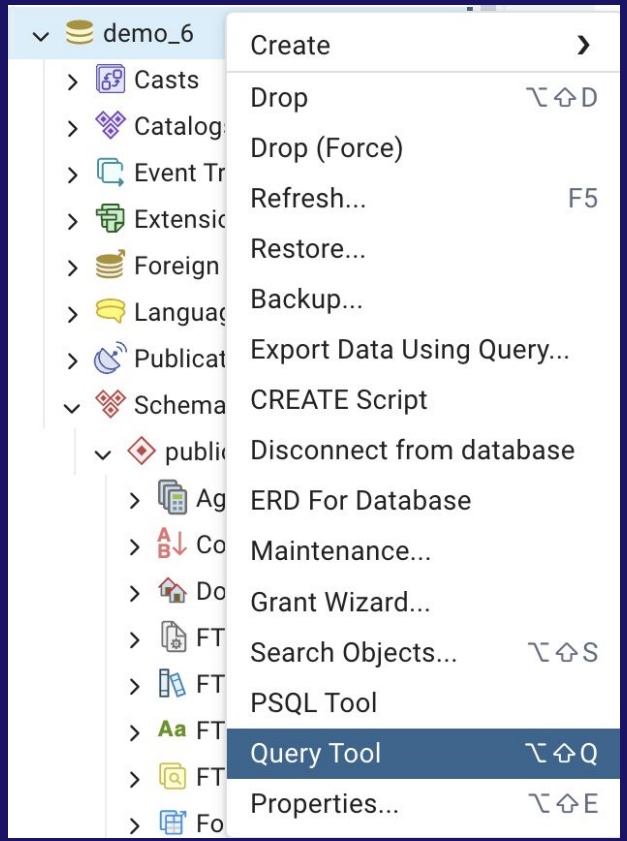
Or Postgresql shorthand:

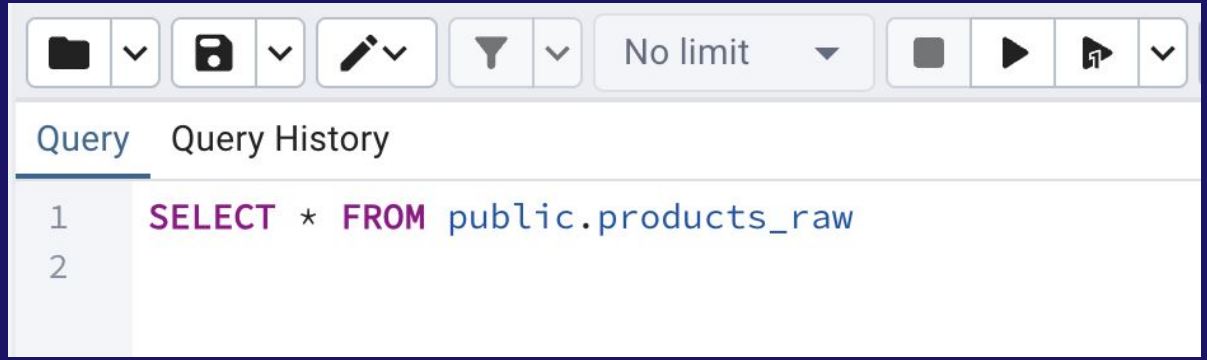
```
1  CREATE TABLE IF NOT EXISTS product (
2    id BIGSERIAL PRIMARY KEY
3  );
```

BIGSERIAL - is unique to postgresql!

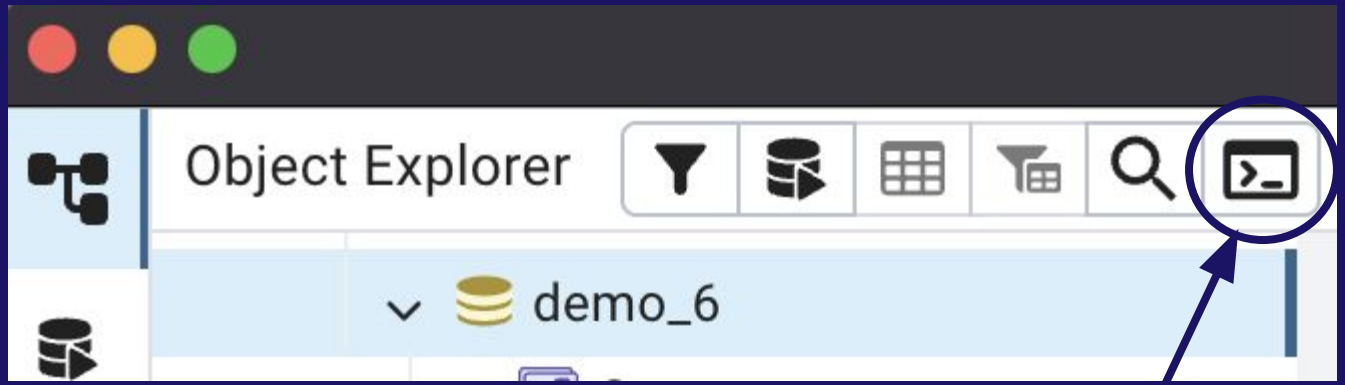

Query Tool
pgAdmin4

Write SQL Query (pgAdmin4)





Query Tool – TERMINAL (pgAdmin4)



This is great for when you want to get started with the terminal *(alternatively using a normal terminal without PGADMIN4)*

Create Table
pgAdmin4

New Table (products_raw)

```
CREATE TABLE IF NOT EXISTS products_raw (
  id BIGSERIAL PRIMARY KEY,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  product JSONB NOT NULL
);
```

Note: JSONB == json data

DEFAULT now() == current time + timezone

BIGSERIAL?
(Explained)

```
BIGSERIAL == BIGINT NOT NULL
             DEFAULT nextval('some_sequence')

// shorthand syntax
```


SQL
.execute()

```
with pool.connection() as conn:
    with conn.transaction():
        conn.execute(
            "INSERT INTO products_raw (product) VALUES (%s)",
        )
```

execute() sends ONE SQL statement to PostgreSQL and runs it.


Tuple &
.execute()

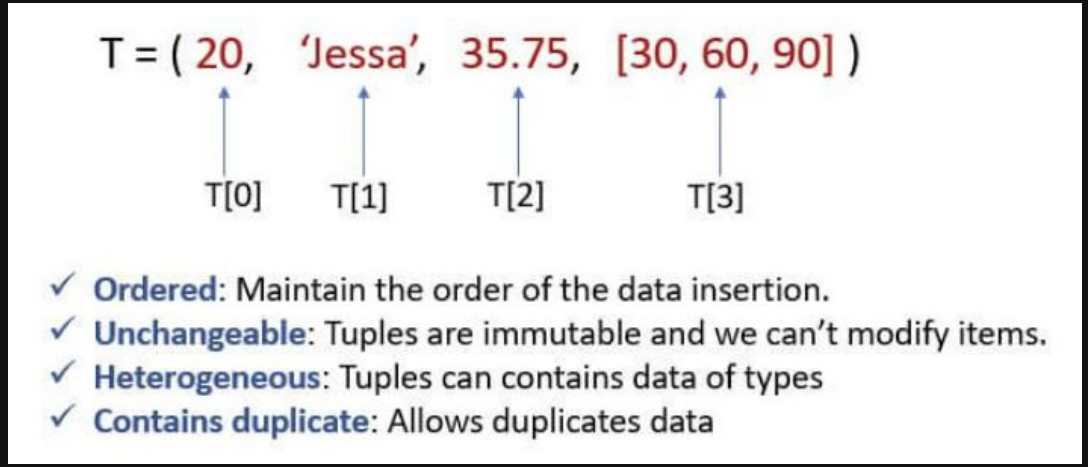
```python
with pool.connection() as conn:
    with conn.transaction():
        conn.execute(
            "INSERT INTO products_raw (product) VALUES (%s)",
            product
        )
```

```
TypeError: execute() got an unexpected keyword argument 'func'
```

```
test_tuple = (10, 20)
test_array = [30, 40]
```

Notera att det inte gick även med array[ ]
Den kräver alltså en TUPLE

https://www.psycopg.org/psycopg3/docs/basic/usage.html#main-objects-in-psycopg-3



```python
with pool.connection() as conn:
    with conn.transaction():
        conn.execute(
            "INSERT INTO products_raw (product) VALUES (%s)",
            (product)
        )
```

Notera: (1) är inte en tuple, (1,) ÄR

```python
tuple_example = (1)
```

This works!

```
with pool.connection() as conn:
    with conn.transaction():
        conn.execute(
            "INSERT INTO products_raw (product) VALUES (%s)",
            (product,)
        )
```


Tuples & Array
.execute()

What is Pydantic...?

```python
from pydantic import BaseModel


class ProductSchema(BaseModel):
    name: str
    description: str
    price: float
```

- str --> TEXT
- int --> INTEGER / BIGINT
- float --> NUMERIC / REAL
- bool --> BOOLEAN
- dict --> JSON / JSONB
- list --> ARRAY or JSONB
- datetime --> TIMESTAMPTZ
- Pydantic model --> must be converted

SQL blir förvirrad över datatypen!

```python
conn.execute(
    "INSERT INTO products_raw (product) VALUES (%s)",
    (product,)
)
```

```python
conn.execute(
    "INSERT INTO products_raw (product) VALUES (%s)",
    [product]
)
```

```
psycopg.ProgrammingError: cannot adapt type 'ProductSchema' using placeholder '%s' (format: AUTO)
```

Samma problem, bästa praxis är Tuples, så låt oss förhålla oss till dessa

(fixes size == more secure)

Detta exempel visar bara på att det tekniskt möjligt med array!

```python
with pool.connection() as conn:
    with conn.transaction():
        conn.execute(
            "INSERT INTO products_raw (product) VALUES (%s)",
            [product]
        )
```

```
psycopg.ProgrammingError: cannot adapt type 'ProductSchema' using placeholder '%s' (format: AUTO)
```

```python
# Connect to an existing database
with psycopg.connect("dbname=test user=postgres") as conn:

    # Open a cursor to perform database operations
    with conn.cursor() as cur:

        # Execute a command: this creates a new table
        cur.execute("""
            CREATE TABLE test (
                id serial PRIMARY KEY,
                num integer,
                data text)
            """)

        # Pass data to fill a query placeholders and let Psycopg perform
        # the correct conversion (no SQL injections!)
        cur.execute(
            "INSERT INTO test (num, data) VALUES (%s, %s)",
            (100, "abc'def"))
```

Documentation

https://www.psycopg.org/psycopg3/docs/basic/usage.html#main-objects-in-psycopg-3


Tillvägagångssätt…?
Explained

MAC: COMMAND + CLICK
WINDOWS: CTRL + CLICK

```
conn.execute(
    "INSERT INTO products_raw (product) VALUES (%s)",
    [product]
)
```

```python
def execute(
    self,
    query: Query,
    params: Params | None = None,
    *,
    prepare: bool | None = None,
    binary: bool = False,
```

Samma sak igen

```python
Params: TypeAlias = Union[Sequence[Any], Mapping[str, Any]]
```

Sequence = An **ordered collection of values**, accessed by position (array, tuple)
Mapping = A **collection of named values**, accessed by key (dictionary)


Pydantic → JSON
Explained

What is Pydantic...?

```python
from pydantic import BaseModel


class ProductSchema(BaseModel):
    name: str
    description: str
    price: float
```

- str --> TEXT
- int --> INTEGER / BIGINT
- float --> NUMERIC / REAL
- bool --> BOOLEAN
- dict --> JSON / JSONB
- list --> ARRAY or JSONB
- datetime --> TIMESTAMPTZ
- Pydantic model --> must be converted

Vi kan konvertera till JSONB

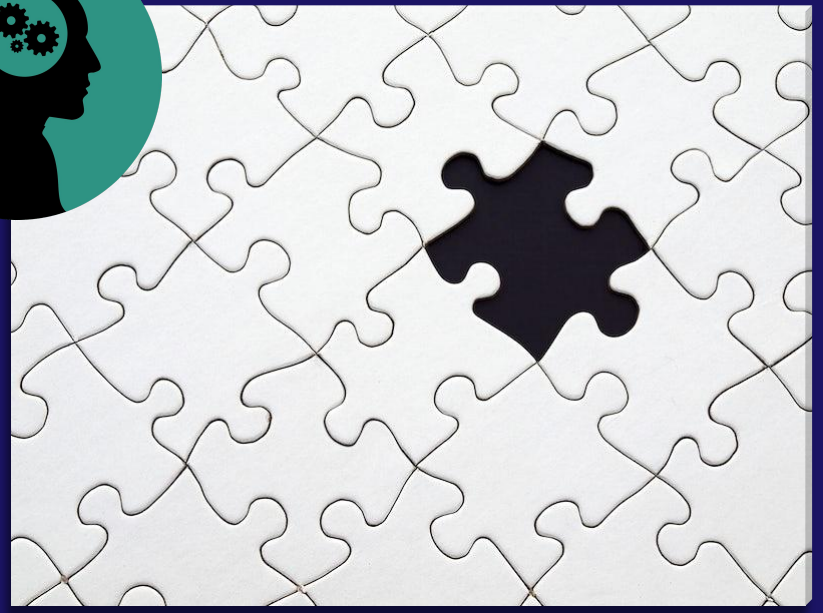
```python
with pool.connection() as conn:
    with conn.transaction():
        conn.execute(
            "INSERT INTO products_raw (product) VALUES (%s)",
            (Json(product), )
        )
```

Converting a PYDANTIC -> JSON
Doesn't work...

```python
conn.execute(
    "INSERT INTO products_raw (product) VALUES (%s)",
    (Json(product.model_dump()),),
)
```

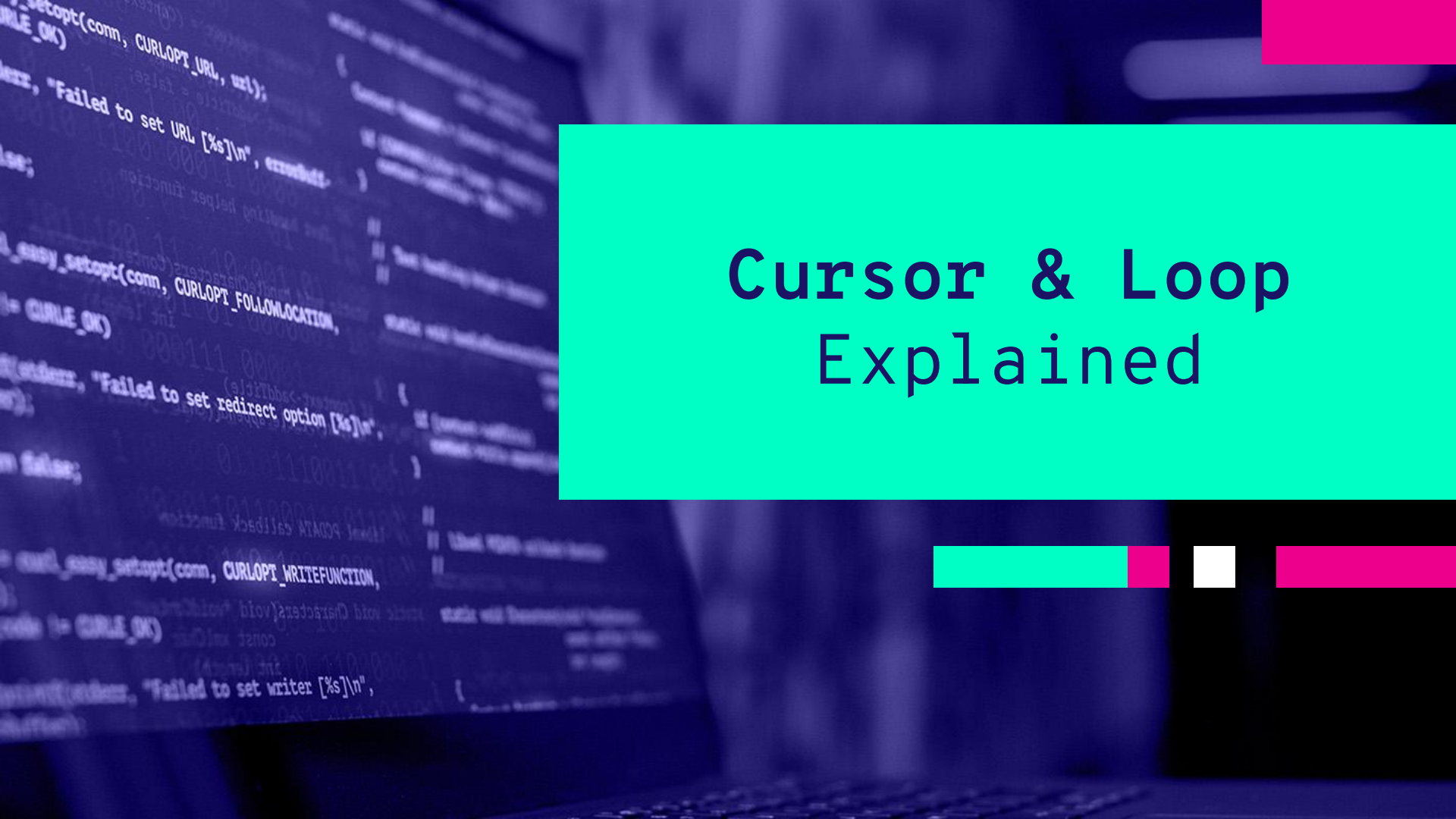
Pydantic has a built-in function for converting
PYDANTIC -> Dictionary -> JSON

Frågor?



04

Cursor, Control & Looping


Cursor & Loop
Explained

```
with conn.cursor() as cur:
    cur.execute("SELECT * FROM products")
    for row in cur:
        if some_complex_rule(row):
            do_something(row)
```

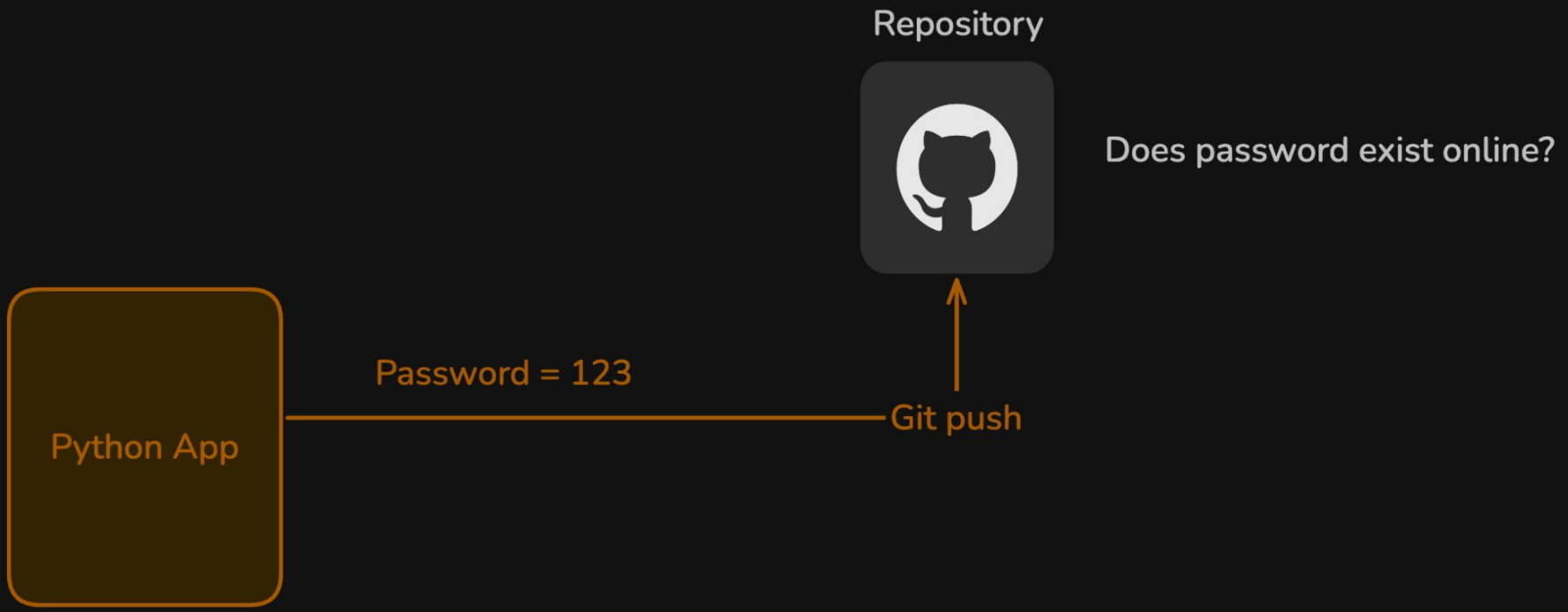
Sometimes you want to convert EACH item in a LIST
Or perform something PER ITEM (ex: print)

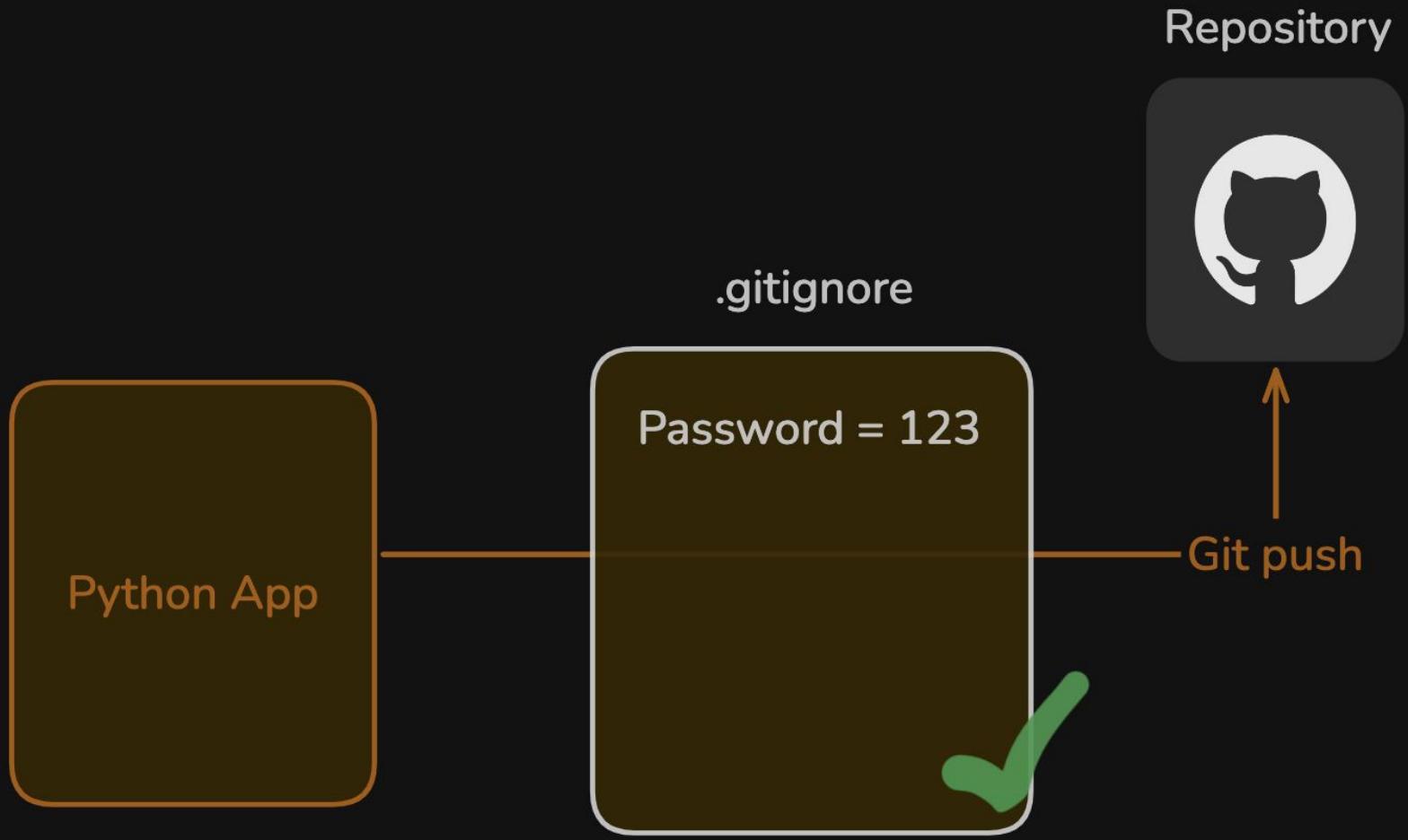
cur (cursor) helps with this!

Filtrera i SQL om du kan.
Filtrera med cursor om du måste.

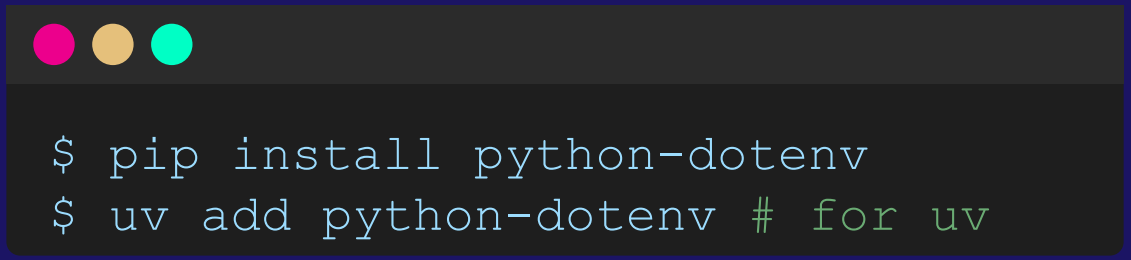
```
with conn.cursor() as cur:
    cur.execute("""
        SELECT * FROM products
        WHERE active = true
    """)
    for row in cur:
        if complex_python_rule(row):
            handle(row)
```


Environment
Variables
Explained

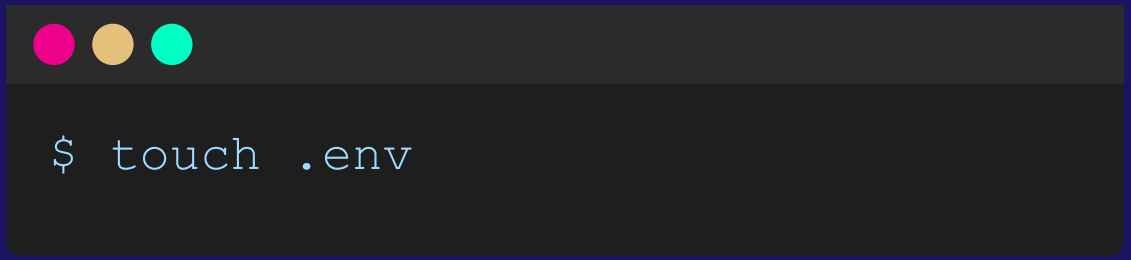

Repository
Does password exist online?
Python App
Password = 123
Git push


Repository
.gitignore
Password = 123
Python App
Git push

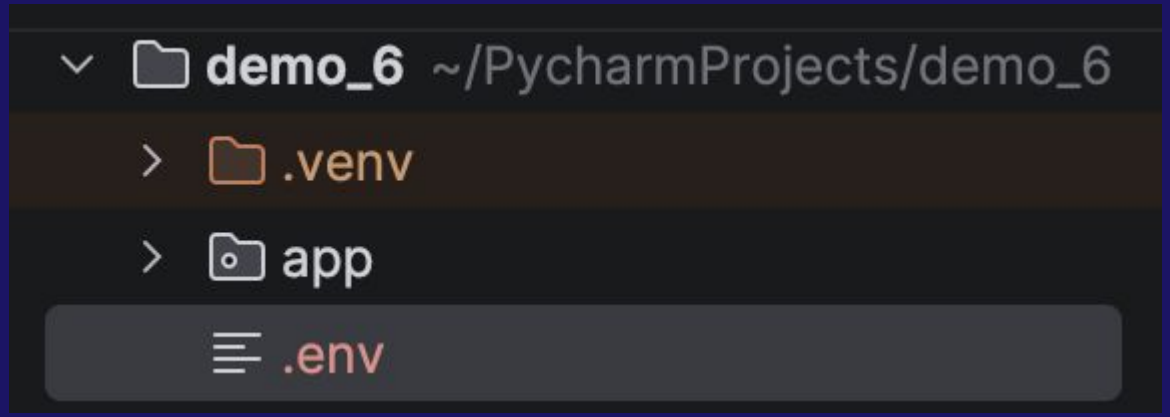
Environment Variables
(Dependency)



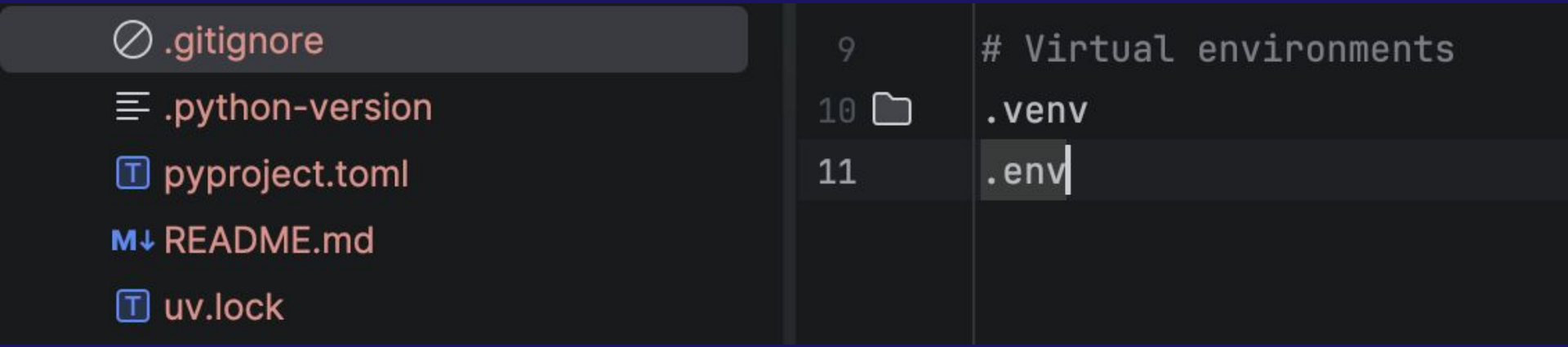
Environment Variables
(File Creation)



Environment Variables
(Results)



.gitignore
(example)



.env
(Content)

```
DATABASE_URL=postgres://postgres:benny123@localhost:5432/demo_6
```

Environment Variables (Results)

```
from dotenv import load_dotenv
import os

load_dotenv()  # reads .env from project root

DATABASE_URL = os.getenv("DATABASE_URL")
```

THANKS !

Do you have any questions?
kristoffer.johansson@sti.se