



FastAPI & PostgreSQL

"Förbered data med FastAPI."

Table of Contents

01

Kursplan &
Översikt

03

Database &
PostgreSQL

02

Installation &
Schema

04

Uppgifter
&
Övningar

Overview



Moduler:

- PgAdmin4 install
- FastAPI
- PostgreSQL



Utbildningsmoment

- **Dataplatfformar, bakgrund och syfte**
- **Git och github i teamkontext**
- **Komponenter och teknologier i en data platform** ✓
- **ETL vs ELT**
- **Utveckling av mjukvara mot databaser** ✓
- **Använda Python mot relationsdatabaser och andra datakällor såsom csv, http xml/json** ✓
- **Använda Python mot realtidsdataströmmar såsom message queues och/eller event streaming platforms**
- **Använda Python och för att rensa, validera och transformera data**
- **Workflow processer** ✓

02

Databases

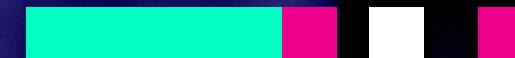
Prerequisites

Install

Prerequisites (Dependencies)

- FastApi dependency [standard]
 - Pydantic (included)
- Python project

PsycoPG3 Install

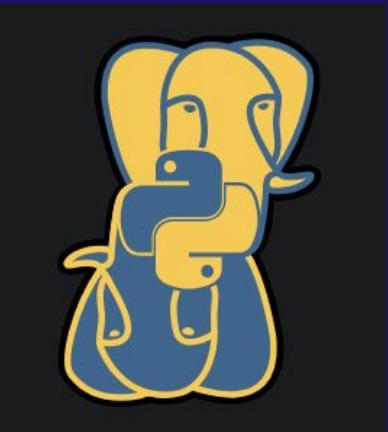


Psycopg3 (install)



```
$ pip install "psycopg[binary]"  
$ pip install psycopg[pool]
```

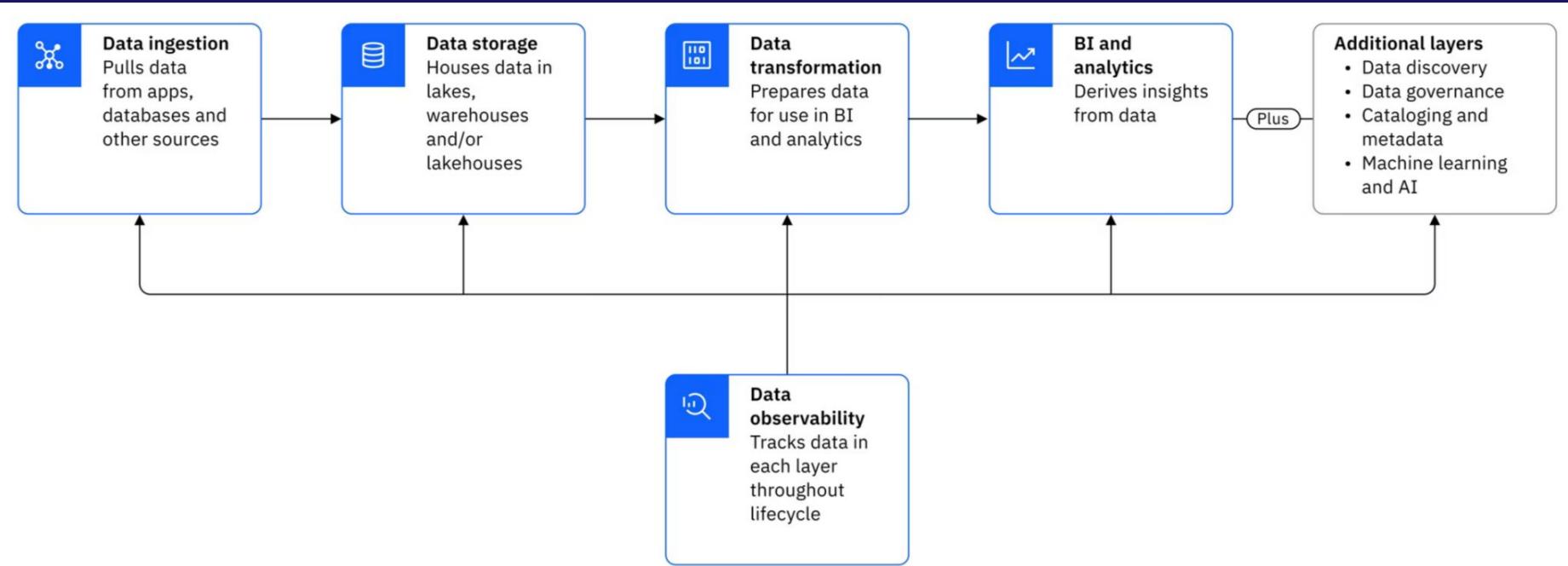
<https://www.psycopg.org/psycopg3/docs/basic/install.html>



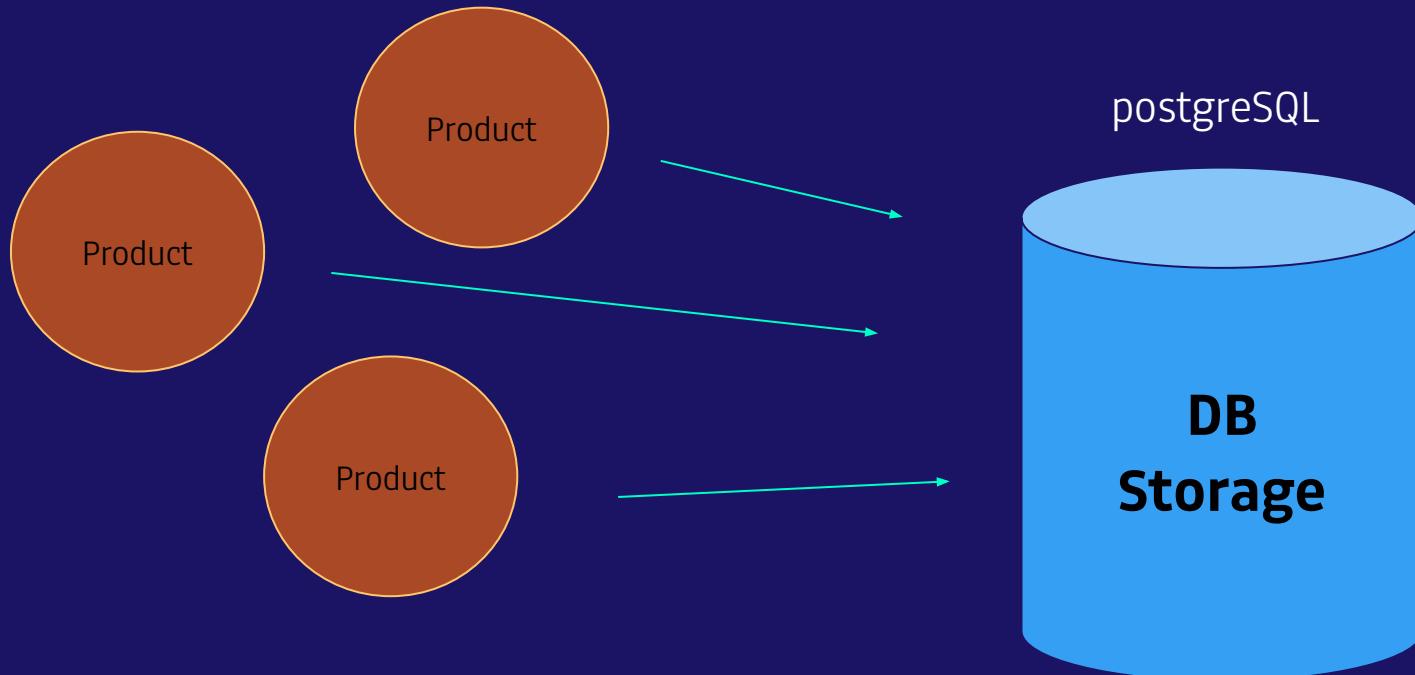
What we're building Explained



Where we are (Data Platform)

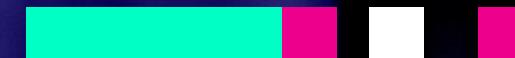


Dump all data (Preparation)



Schema Preparation

Pydantic

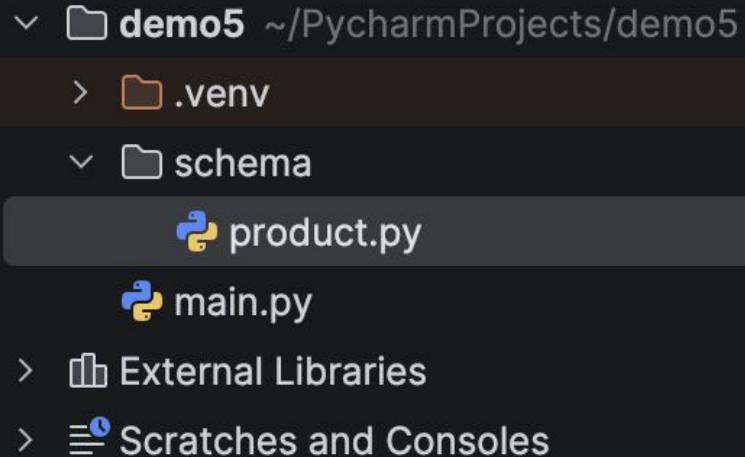


ProductSchema (Pydantic)

```
from typing import Union

from pydantic import BaseModel

class ProductSchema(BaseModel):
    product_id: str
    name: str
    price: float
    currency: str # (SEK, EUR, USD)
    category: Union[str, None]
    brand: Union[str, None]
```



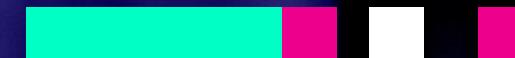
Endpoint (/products)

```
@app.post("/products", response model=ProductSchema)
def products(product: ProductSchema) -> ProductSchema:
    return product
```

03

Database & PostgreSQL

Create Table PostgreSQL



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

Database

PostgreSQL

DATABASE URL (Structure)

```
from psycopg_pool import ConnectionPool

DATABASE_URL =
"postgresql://postgres:benny123@localhost:5432/demo_5"
```

Username

password

Db name

All in All (Connection Pool)

```
from psycopg_pool import ConnectionPool

DATABASE_URL =
"postgresql://postgres:benny123@localhost:5432/demo_5"
pool = ConnectionPool(DATABASE_URL)
```

NOTE: Everytime we run a query, we'll open up a new connection and immediately close it (best practice)

Helper Function (INSERT as JSON)

```
from psycopg.types.json import Json
from psycopg import Connection

def insert_product(conn, product: dict):
    conn.execute(
        "INSERT INTO products_raw (product) VALUES (%s)",
        (Json(product),)
    )
```

Post + Query (TBD)

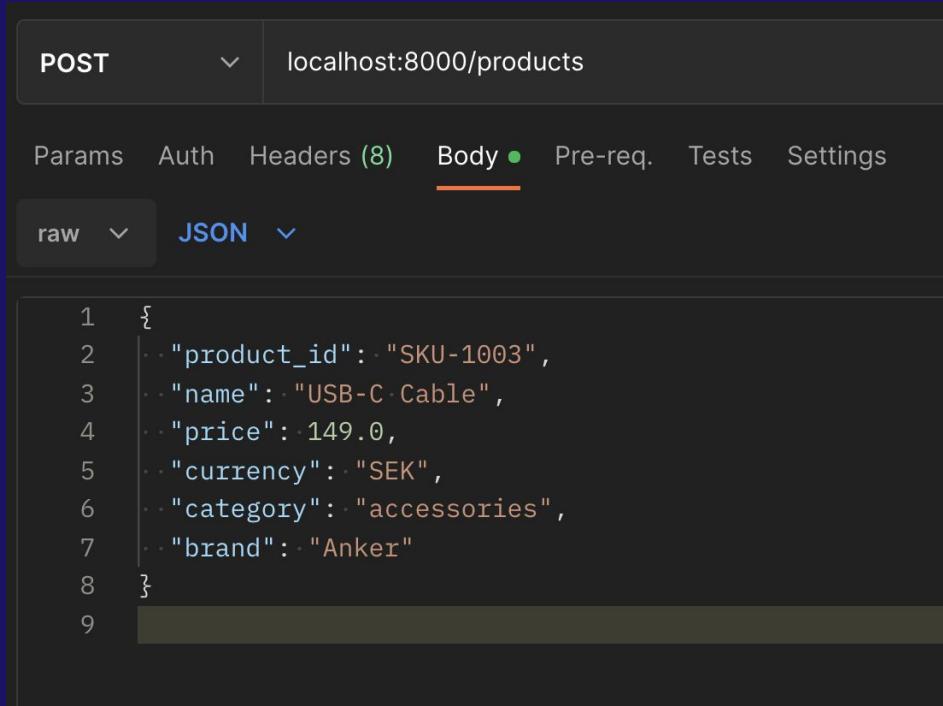
```
@app.post("/products", response_model=ProductSchema, status_code=status.HTTP_201_CREATED)
def products(product: ProductSchema) -> ProductSchema:

    with pool.connection() as conn:
        insert product(conn, product.model_dump())
        conn.commit()

    return product
```

NOTE: model_dump() removes pydantic and gives raw data

Postman (Post Product)



The screenshot shows the Postman interface with a POST request to `localhost:8000/products`. The Body tab is selected, showing a JSON payload:

```
1 {  
2   "product_id": "SKU-1003",  
3   "name": "USB-C Cable",  
4   "price": 149.0,  
5   "currency": "SEK",  
6   "category": "accessories",  
7   "brand": "Anker"  
8 }  
9
```

Copy me:

```
{  
  "product_id": "SKU-1003",  
  "name": "USB-C Cable",  
  "price": 149.0,  
  "currency": "SEK",  
  "category": "accessories",  
  "brand": "Anker"
```

Fetch Products (Result)

```
SELECT * FROM public.products_raw
ORDER BY id ASC
```

	id [PK] bigint	created_at timestamp with time zone	product jsonb	
1	1	2026-02-02 19:59:27.141988+...	{"name": "USB-C Cable", "brand": "Anker", "price": 149.0, "category": "accessories", "currency": "SEK", "product_id": "SKU-1003"}	
2	2	2026-02-02 20:01:07.459521+...	{"name": "Wireless Mouse", "brand": "Logitech", "price": 299.0, "category": "electronics", "currency": "SEK", "product_id": "SKU-2..."}	
3	3	2026-02-02 20:01:07.459521+...	{"name": "USB-C Cable", "brand": "Anker", "price": 149.0, "category": "accessories", "currency": "SEK", "product_id": "SKU-2002"}	

Frågor?



04

Uppgifter

&

Eget Arbete

Uppgifter

Välkommen till första uppgiften!

Uppifterna är till för att testa dina färdigheter och kunskaper för att både öva och repetera på det vi har arbetat med under föreläsningarna.

Dessa är **INTE** obligatoriska.
Men är ämnen ni kommer testas mot.



Code Analysis

```
// Question: Where does the connection OPEN and where does it CLOSE?  
// What does: 'with' and 'as' mean in this context?  
  
import psycopg  
from psycopg_pool import ConnectionPool  
  
pool = ConnectionPool ("postgresql://postgres:password@localhost:5432/demo_5" )  
  
def store_value (value: str):  
    with pool.connection () as conn:  
  
        conn.execute (  
            "INSERT INTO demo_table (value) VALUES (%s) ",  
            (value,)  
        )  
        conn.commit ()
```

Code Analysis #2

// Question: What is 'product.model_dump()' and why is it necessary?

```
@app.post("/products", response_model=ProductSchema,  
status_code=status.HTTP_201_CREATED)  
def products(product: ProductSchema) -> ProductSchema:  
    with pool.connection() as conn:  
        insert_product(conn, product.model_dump())  
        conn.commit()  
  
    return product
```

Code Analysis #3

// Question: Just by analyzing the code... what do you think this does?

```
@app.post("/products/bulk")
def products_bulk(products: list[ProductSchema]):
    with pool.connection() as conn:
        with conn.cursor() as cur:
            cur.executemany(
                "INSERT INTO products_raw (product) VALUES (%s)",
                [Json(product.model_dump()), for product in products]
            )
            conn.commit()
    return {"inserted": len(products)}
```

```
1           // -Uppgift #1- //
2
3 /* INSTRUCTIONS
4
5 Utgå från dagens lektion
6 Ändra nu 'ProductSchema' så att den innehåller
7 ett extra värde:
8     • tags: Union[list[str], None]
9
10    Prova kör koden - fungerar det?
11 */
12
13 // HINT & Examples
14 hint("Hint: Ja")
15
16
17
18
19
20
21
22
23
```

Uppgift #1



A binary code visualization consisting of two columns of binary digits (0s and 1s). The left column has 10 digits: 0, 1, 0, 1, 0, 0, 1, 0, 1, 0. The right column has 10 digits: 0, 1, 0, 1, 0, 0, 1, 0, 1, 1. Below the columns are their decimal equivalents: 100, 5, 8, 2, 4, 16, 3, 6, 12, 21.

Kom igång enkelt med uppgift #1

```
1           // -Uppgift #2- //
2
3 /* INSTRUCTIONS
4
5     Inom ditt schema lägg till ett objekt
6     Exempelvis:
7
8     class DimensionsSchema (BaseModel):
9         width_cm: float
10        height_cm: float
11        depth_cm: float
12
13    class ProductSchema (BaseModel):
14        // old values from before remain the same
15        dimensions: Union[DimensionsSchema, None]
16
17    Påverkar detta koden när du kör?
18 */
19
20 hint("hint: nej")
21 hint("Nästa uppgift visar hur du lägger till
22 objektet in i databasen via postman")
```

Uppgift #2

0	1	0	1	0	0	1
0	1	0	1	0	0	1
0	1	0	0	0	0	1
0	0	0	0	0	0	0

```
{  
    "product_id": "SKU-123",  
    "name": "Wireless Mouse",  
    "price": 299.0,  
    "currency": "SEK",  
    "category": null,  
    "brand": null,  
    "tags": null,  
    "dimensions": {  
        "width_cm": 6.2,  
        "height_cm": 3.8,  
        "depth_cm": 10.1  
    }  
}
```

0 1 0 1 0 0
0 1 0 1 0

// -Uppgift #3- //

/* INSTRUCTIONS

Använd följande kod

```
8     @app.get("/products")
9     def get_products():
10        with pool.connection() as conn:
11            with conn.cursor() as cur:
12                cur.execute("SELECT product FROM products_raw" )
13                rows = cur.fetchall()
14
15                # rows = [ (product_dict,) , (product_dict,) , ... ]
16                return [row[0] for row in rows]
```

Testkör koden - vad tror du att 'fetchall()' returnerar?
*/

// HINT & Examples
hint("Lista av dictionaries")

THANKS !

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

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.