

# Final Project

Project 1 - Project 2 - Project 3 - Project 4

fantastic **4** ↴  
Kelompok 13



# project team ↘

- Muhammad Ramli
- Ozza Dinata
- Donna Setiawan
- Ginta Khairunisa
- Satria Said

# Project Overview ↴

Page

**04** Project 1 – Business Intelligence (BI)

**10** Project 2 – Data Build Tool (DBT)

**18** Project 3 – Airflow

**29** Project 4 – Kafka (Stream Processing)

# Project 1: Business Intelligence

## Objective Project

- Desain ERD DWH,
- Menjalankan ETL Datamart dengan python
- Desain Dashboard sesuai dengan Business Requirements Di Looker Studio

Tools yang digunakan dalam project



PostgreSQL



python

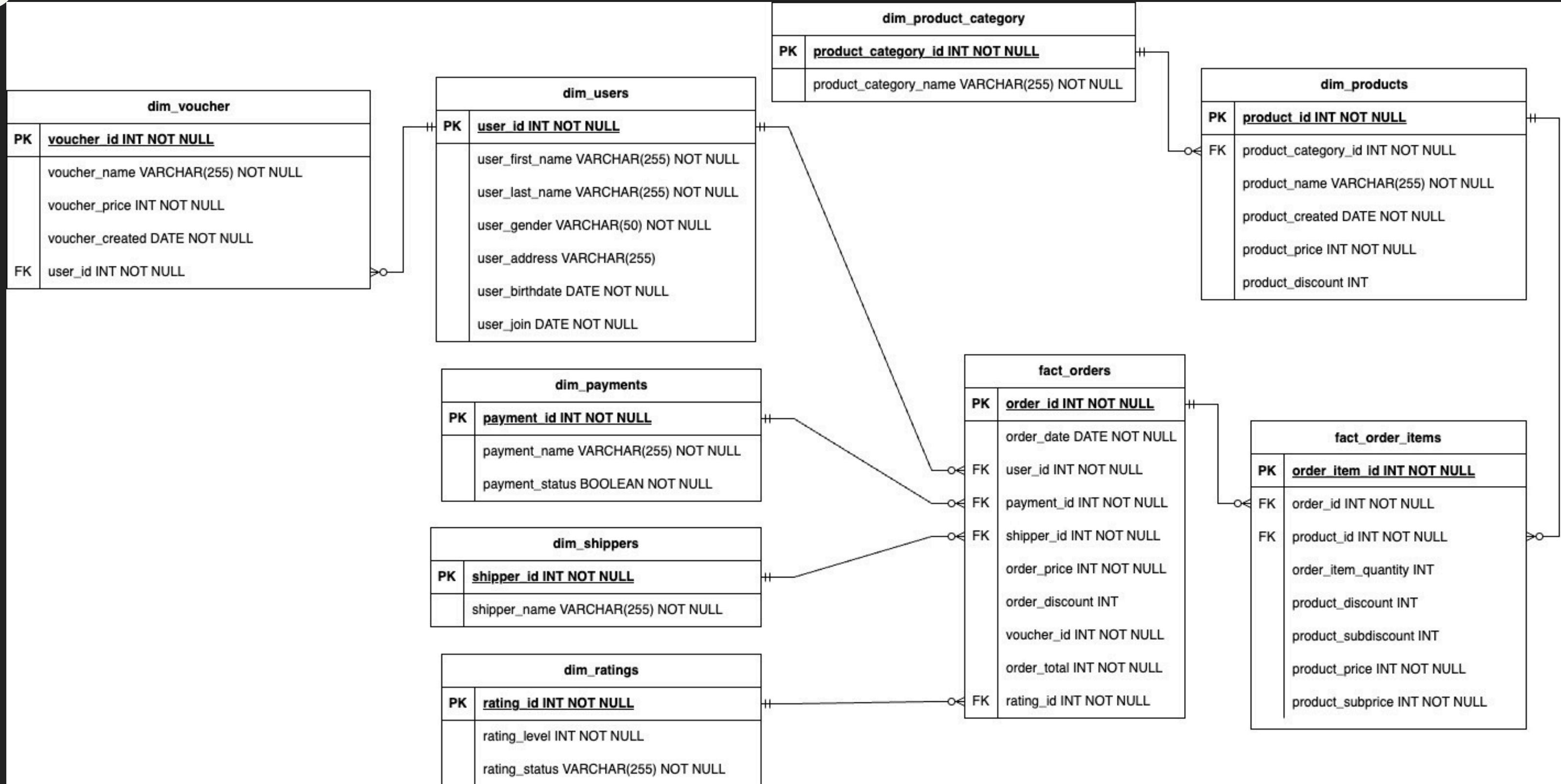


Looker Studio

# Project 1:Result

1

Membuat Desain ERD DWH untuk mengetahui alur data dan penentuan fact tables serta dim tables



# Project 1:Result

2

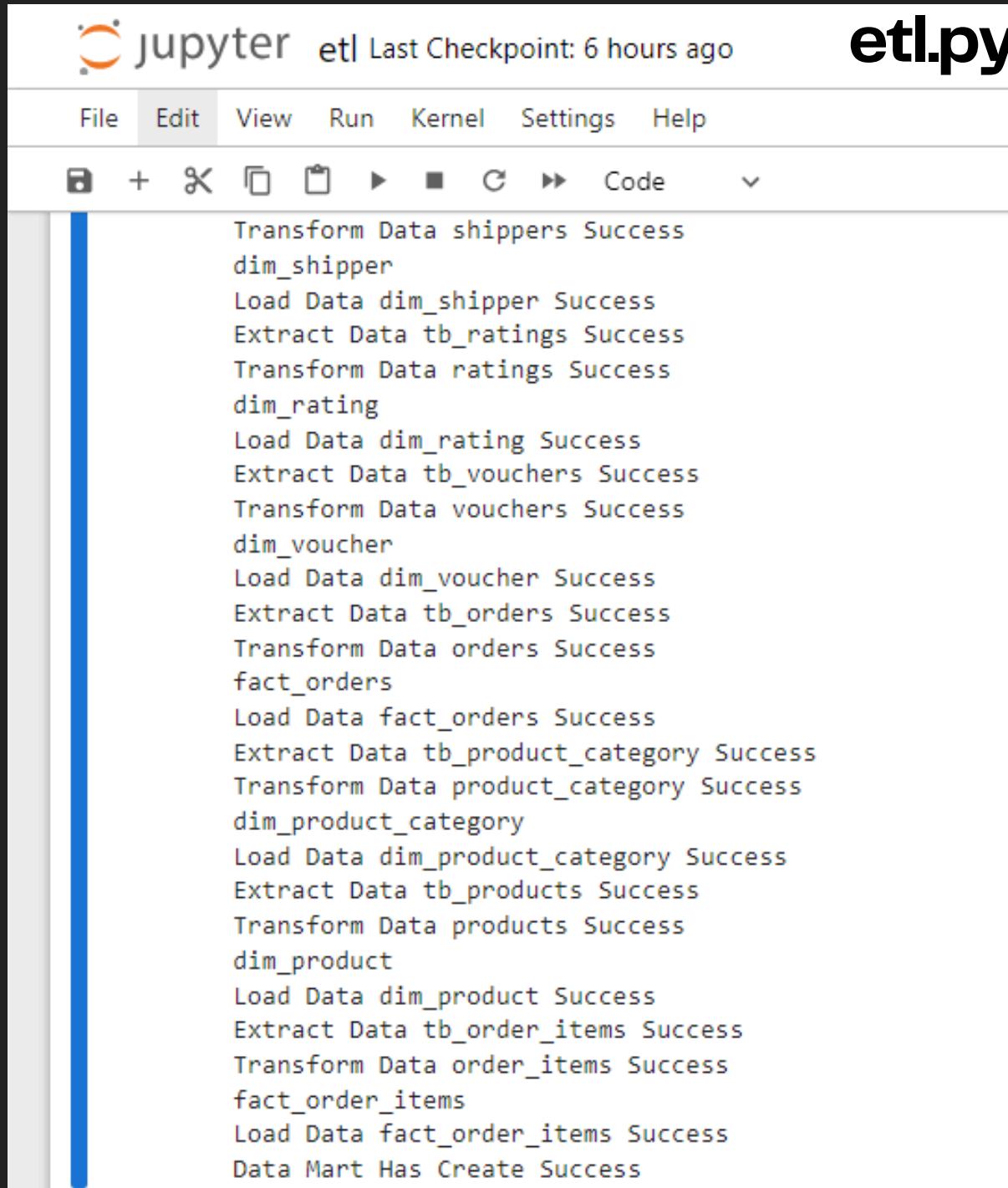
Lampiran screenshot hasil database OLTP dan script ddl oltp yg dirunning berhasil dgn toolsdbeaver

The screenshot shows the DBeaver 24.0.5 interface with the tb\_order\_items table selected. The left sidebar displays the Database Navigator with various databases and schemas listed. The main area shows the tb\_order\_items table with 22 rows of data. The columns are: order\_item\_id, order\_id, product\_id, order\_item\_quantity, product\_discount, product\_subdiscount, product\_price, and product\_subprice. The data is as follows:

	order_item_id	order_id	product_id	order_item_quantity	product_discount	product_subdiscount	product_price	product_subprice
1	90,010,001	1,110,001	31,110,002	1	15,000	15,000	250,000	250,000
2	90,010,002	1,110,002	31,110,007	1	10,000	10,000	120,000	120,000
3	90,010,003	1,110,002	31,110,002	2	15,000	30,000	250,000	500,000
4	90,010,004	1,110,003	31,110,004	1	0	0	2,000,000	2,000,000
5	90,010,005	1,110,003	31,110,005	1	1,000,000	1,000,000	4,000,000	4,000,000
6	90,010,006	1,110,004	31,110,001	2	0	0	300,000	600,000
7	90,010,007	1,110,004	31,110,002	3	15,000	45,000	250,000	750,000
8	90,010,008	1,110,004	31,110,003	1	0	0	1,800,000	1,800,000
9	90,010,009	1,110,005	31,110,005	1	1,000,000	1,000,000	4,000,000	4,000,000
10	90,010,010	1,110,006	31,110,005	1	1,000,000	1,000,000	4,000,000	4,000,000
11	90,010,011	1,110,006	31,110,002	2	15,000	30,000	250,000	500,000
12	90,010,012	1,110,007	31,110,006	1	0	0	500,000	500,000
13	90,010,013	1,110,007	31,110,007	1	10,000	10,000	120,000	120,000
14	90,010,014	1,110,007	31,110,002	1	15,000	15,000	250,000	250,000
15	90,010,015	1,110,008	31,110,004	1	0	0	2,000,000	2,000,000
16	90,010,016	1,110,009	31,110,004	1	0	0	2,000,000	2,000,000
17	90,010,017	1,110,010	31,110,002	3	15,000	45,000	250,000	750,000
18	90,010,018	1,110,010	31,110,001	1	0	0	300,000	300,000
19	90,010,019	1,110,011	31,110,001	1	0	0	300,000	300,000
20	90,010,020	1,110,011	31,110,002	1	15,000	15,000	250,000	250,000
21	90,010,021	1,110,012	31,110,002	1	15,000	15,000	250,000	250,000
22	90,010,022	1,110,012	31,110,007	2	10,000	20,000	120,000	240,000

# Project 1:Result

## 3 Menjalankan etl data warehouse menggunakan jupyter notebook



**etl.py**

```
Transform Data shippers Success
dim_shipper
Load Data dim_shipper Success
Extract Data tb_ratings Success
Transform Data ratings Success
dim_rating
Load Data dim_rating Success
Extract Data tb_vouchers Success
Transform Data vouchers Success
dim_voucher
Load Data dim_voucher Success
Extract Data tb_orders Success
Transform Data orders Success
fact_orders
Load Data fact_orders Success
Extract Data tb_product_category Success
Transform Data product_category Success
dim_product_category
Load Data dim_product_category Success
Extract Data tb_products Success
Transform Data products Success
dim_product
Load Data dim_product Success
Extract Data tb_order_items Success
Transform Data order_items Success
fact_order_items
Load Data fact_order_items Success
Data Mart Has Create Success
```

**config.py**

```
137 ddl_marts = {
138     "dim_sales": """
139         CREATE TABLE IF NOT EXISTS dm_sales (
140             order_id INT NOT NULL PRIMARY KEY,
141             order_date DATE NOT NULL,
142             month_year DATE NOT NULL,
143             user_id INT NOT NULL,
144             user_name VARCHAR(255),
145             user_gender VARCHAR(255),
146             user_address VARCHAR(255),
147             payment_type VARCHAR(255),
148             shipper_name VARCHAR(255),
149             order_price INT NOT NULL,
150             order_discount INT,
151             voucher_name VARCHAR(255),
152             order_total INT NOT NULL,
153             FOREIGN KEY (order_id) REFERENCES fact_orders(order_id),
154             FOREIGN KEY (user_id) REFERENCES dim_user(user_id)
155         );
156     """,
157     "dim_product_sales": """
158         CREATE TABLE IF NOT EXISTS dm_product (
159             order_item_id INT NOT NULL PRIMARY KEY,
160             order_id INT NOT NULL,
161             product_name VARCHAR(255) NOT NULL,
162             product_category_name VARCHAR(255) NOT NULL,
163             order_item_quantity INT,
164             product_discount INT,
165             product_subdiscount INT,
166             product_price INT NOT NULL,
167             product_subprice INT NOT NULL,
168             margin_laba INT,
169             FOREIGN KEY (order_item_id) REFERENCES fact_order_items(order_item_id),
170             FOREIGN KEY (order_id) REFERENCES fact_orders(order_id)
171         );
172     """}
```

Lampiran screenshoot script etl yang dijalankan di jupyter notebook

# Project 1:Result

## Output Data Mart

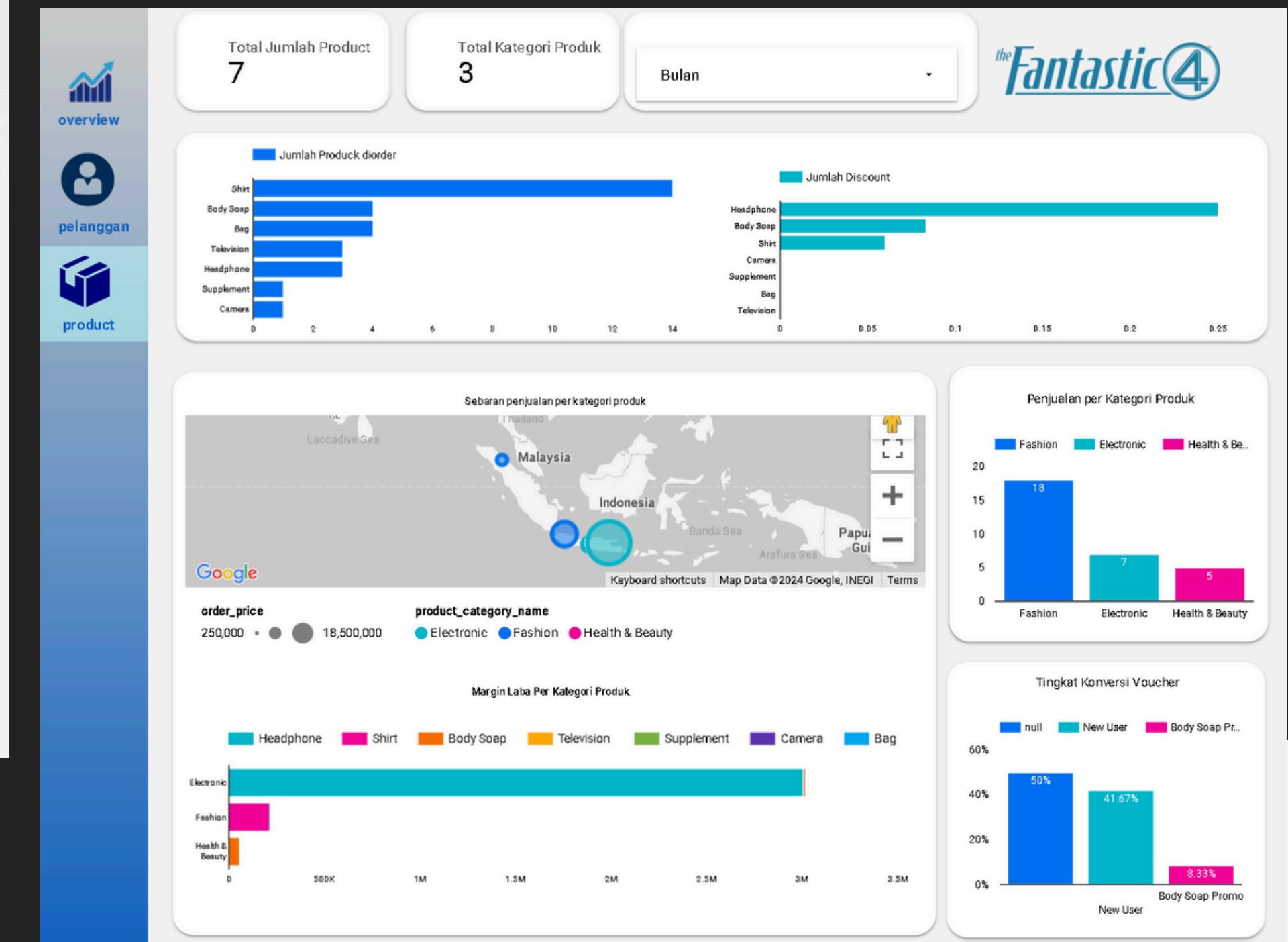
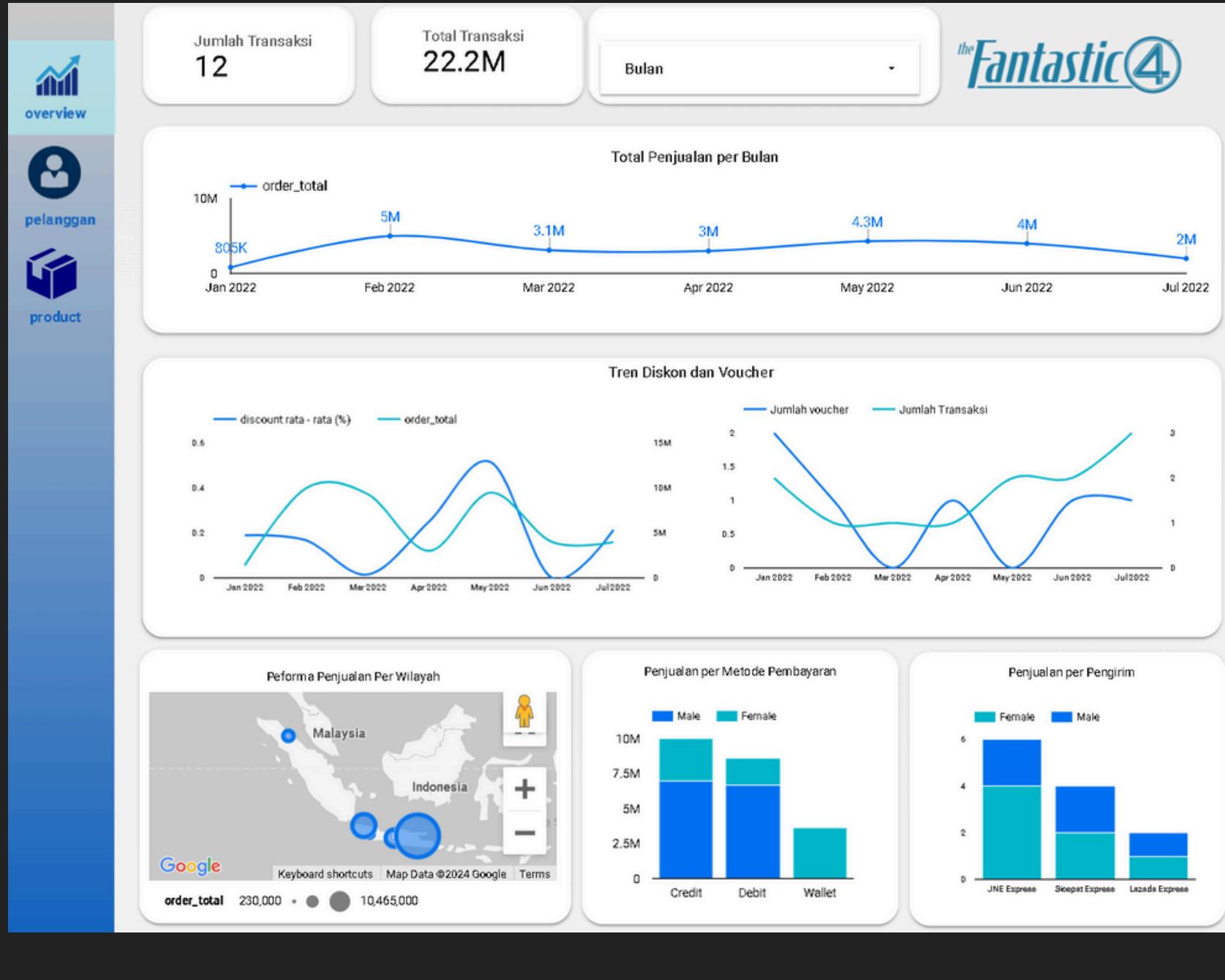
The screenshot shows the Toad Data Point interface with the 'dm\_product' table selected in the central workspace. The table has columns: order\_item\_id, order\_id, product\_name, product\_category\_name, and order\_item\_quantity. The data consists of 22 rows, each containing a unique order item ID, its corresponding order ID, the product name, its category, and the quantity ordered.

	order_item_id	order_id	product_name	product_category_name	order_item_quantity
1	90,010,001	1,110,001	Shirt	Fashion	
2	90,010,002	1,110,002	Body Soap	Health & Beauty	
3	90,010,003	1,110,002	Shirt	Fashion	
4	90,010,004	1,110,003	Television	Electronic	
5	90,010,005	1,110,003	Headphone	Electronic	
6	90,010,006	1,110,004	Bag	Fashion	
7	90,010,007	1,110,004	Shirt	Fashion	
8	90,010,008	1,110,004	Camera	Electronic	
9	90,010,009	1,110,005	Headphone	Electronic	
10	90,010,010	1,110,006	Headphone	Electronic	
11	90,010,011	1,110,006	Shirt	Fashion	
12	90,010,012	1,110,007	Supplement	Health & Beauty	
13	90,010,013	1,110,007	Body Soap	Health & Beauty	
14	90,010,014	1,110,007	Shirt	Fashion	
15	90,010,015	1,110,008	Television	Electronic	
16	90,010,016	1,110,009	Television	Electronic	
17	90,010,017	1,110,010	Shirt	Fashion	
18	90,010,018	1,110,010	Bag	Fashion	
19	90,010,019	1,110,011	Bag	Fashion	
20	90,010,020	1,110,011	Shirt	Fashion	
21	90,010,021	1,110,012	Shirt	Fashion	
22	90,010,022	1,110,012	Body Soap	Health & Beauty	

# Project 1: Result

4

Membuat report dashboard Data Penjualan E-Commerce di looker studio



Hasil Insight Report di Looker Studio

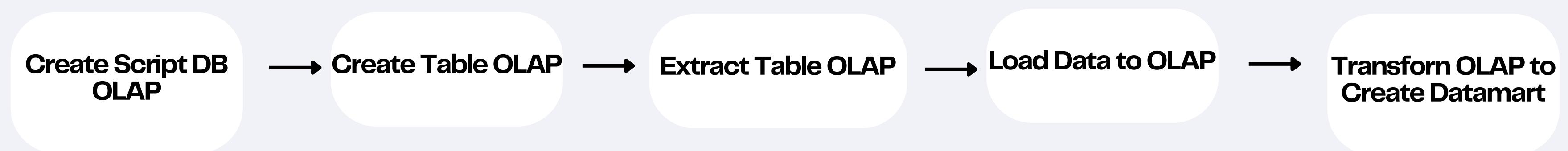
<https://lookerstudio.google.com/s/jZhxA-xFNEE>

# Project 2: DBT

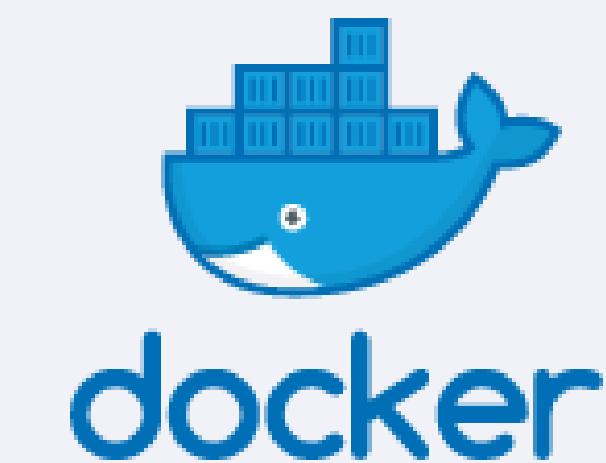
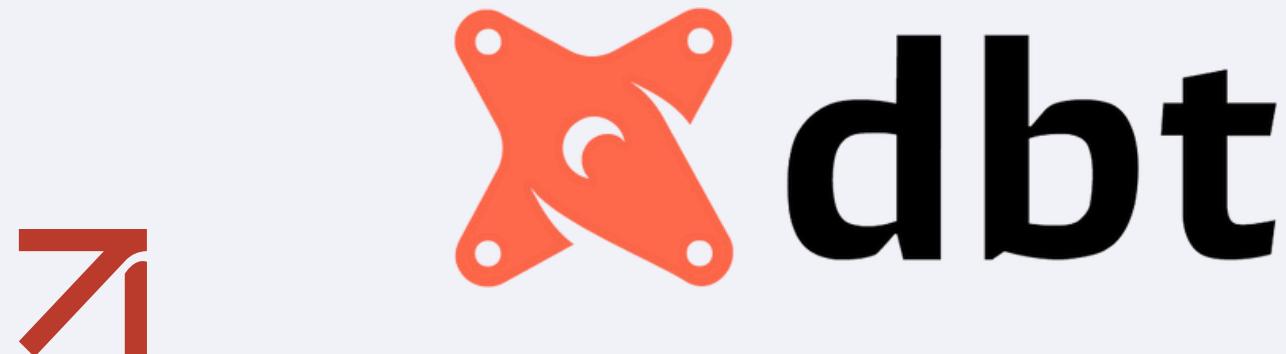
## Objective Project

Implementasi Data Pipeline End to End dengan menggunakan DBT, Docker, postgres SQL, python

### Data Pipeline



Tools yang digunakan dalam project



# Project 2 - DBT



01

## Menjalankan Docker Container

The screenshot shows the Docker Desktop application window. On the left, a sidebar includes links for Containers, Images, Volumes, Builds, Docker Scout, Extensions (with 'Add Extensions'), and Walkthroughs. The main area is titled 'Containers' and displays two entries:

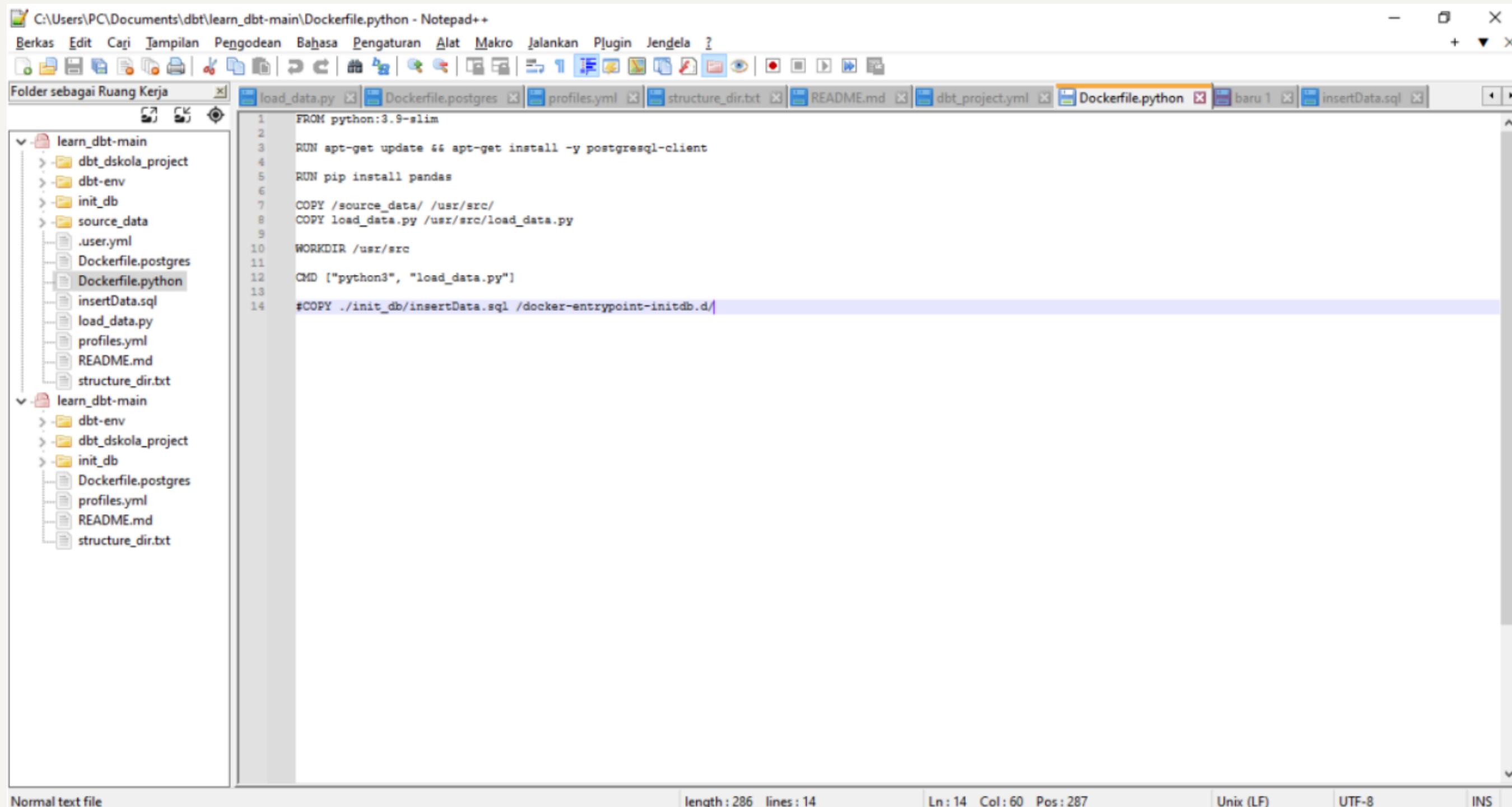
Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
python 341caeef635c8	python_script	Exited		0%	56 minutes ago	[More]
postgres 424483b6cda8	postgres_script	Running	5432:5432	0%	8 minutes ago	[More]

At the bottom, status indicators show 'Engine running', system resources (RAM 2.02 GB, CPU 0.50%), and a note 'Not signed in'. A notification bar at the bottom right indicates 'New version available' with a count of 3.

Lampiran Gambar Docker Container Run

# Project 2 - Result ↴

Lampiran Script Dockerfile Python yang akan dijalankan



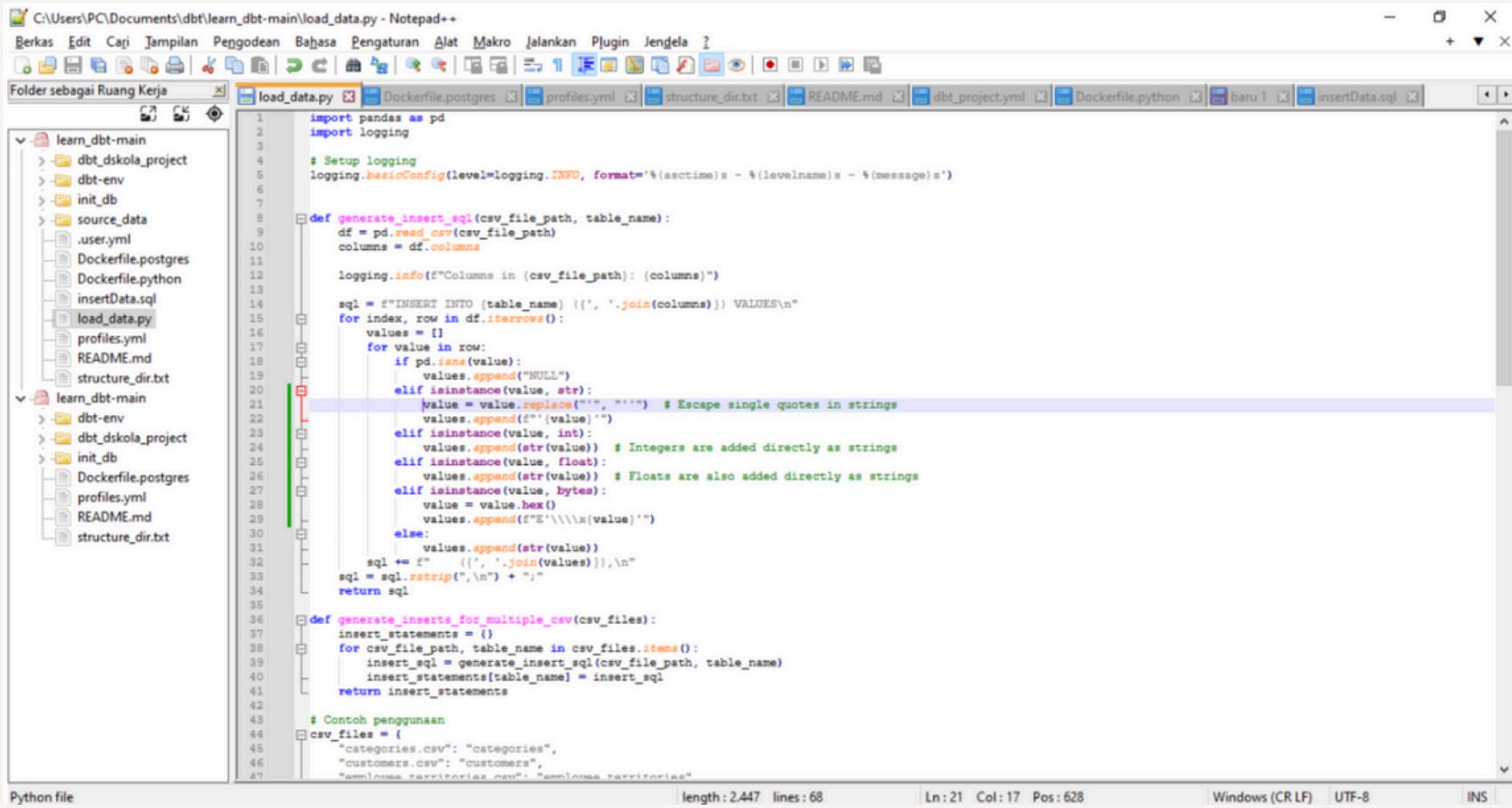
The screenshot shows a Notepad++ window with the title "C:\Users\PC\Documents\dbt\learn\_dbt-main\Dockerfile.python - Notepad++". The window displays a Dockerfile with the following content:

```
FROM python:3.9-slim
RUN apt-get update && apt-get install -y postgresql-client
RUN pip install pandas
COPY /source_data/ /usr/src/
COPY load_data.py /usr/src/load_data.py
WORKDIR /usr/src
CMD ["python3", "load_data.py"]
#COPY ./init_db/insertData.sql /docker-entrypoint-initdb.d/
```

The Notepad++ interface includes a toolbar at the top, a tab bar with multiple files, and a status bar at the bottom indicating the file type as "Normal text file" and providing line, column, and position information.

# Project 2 - Result ↴

Tampilan Script Python Load data dari file CSV



```
C:\Users\PC\Documents\dbt\learn_dbt-main\load_data.py - Notepad++
Berkas Edit Cari Tampilan Pengodean Bahasa Pengaturan Alat Makro Jalankan Pjugin Jengels I
Folder sebagai Ruang Kerja
load_data.py Dockerfile.postgres profiles.yml README.md dbt_project.yml Dockerfile.python baru 1 insertData.sql
1 import pandas as pd
2 import logging
3
4 # Setup logging
5 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
6
7
8 def generate_insert_sql(csv_file_path, table_name):
9     df = pd.read_csv(csv_file_path)
10    columns = df.columns
11
12    logging.info(f"Columns in {csv_file_path}: {columns}")
13
14    sql = f"INSERT INTO {table_name} ({', '.join(columns)}) VALUES\n"
15    for index, row in df.iterrows():
16        values = []
17        for value in row:
18            if pd.isna(value):
19                values.append("NULL")
20            elif isinstance(value, str):
21                value = value.replace("'", "''") # Escape single quotes in strings
22                values.append(f"'{value}'")
23            elif isinstance(value, int):
24                values.append(str(value)) # Integers are added directly as strings
25            elif isinstance(value, float):
26                values.append(str(value)) # Floats are also added directly as strings
27            elif isinstance(value, bytes):
28                value = value.hex()
29                values.append(f"E'{value}'")
30            else:
31                values.append(str(value))
32        sql += f" ({', '.join(values)}),\n"
33    sql = sql.rstrip(",\n") + ";"
34    return sql
35
36 def generate_inserts_for_multiple_csv(csv_files):
37     insert_statements = {}
38     for csv_file_path, table_name in csv_files.items():
39         insert_sql = generate_insert_sql(csv_file_path, table_name)
40         insert_statements[table_name] = insert_sql
41     return insert_statements
42
43 # Contoh penggunaan
44 csv_files = {
45     "categories.csv": "categories",
46     "customers.csv": "customers",
47     "employees_territories.csv": "employees_territories"
48 }
```



```
db_dskola=# \l
db_dskola=# \dt
List of relations
 Schema |      Name       | Type | Owner
-----+----------------+-----+-----
 public | categories    | table| postgres
 public | customers    | table| postgres
 public | employee_territories | table| postgres
 public | employees    | table| postgres
 public | order_details | table| postgres
 public | orders        | table| postgres
 public | products      | table| postgres
 public | regions       | table| postgres
 public | shippers      | table| postgres
 public | suppliers     | table| postgres
 public | territories   | table| postgres
(11 rows)
```

Berikut tampilan Output Table Database db\_skola

# Project 2 - Result ↘

02

Lampiran hasil data warehouse yang sudah dijalankan di DBT

The screenshot shows the dbt Docs interface for the project `dbt_dskola_project`. The left sidebar lists various models like `employees`, `order_details`, `orders`, etc., and the `dwh` model is currently selected. The main content area displays the `dwh` table details, including its type as a table owned by `postgres` in the package `dbt_dskola_project` and SQL language. A large modal window titled "Lineage Graph" is open on the right, showing a complex network of dependencies between various source tables such as `new_employees`, `new_customers`, `new_order_details`, `new_products`, `new_regions`, `new_suppliers`, and `new_territories`, all feeding into the `dwh` table.

# Project 2 - Result



03

Lampiran gambar data mart di DBT

localhost:8080/#!/model/model.dbt\_dskola\_project.datamart\_monthly\_best\_employee

dbt

Search for models...

datamart\_monthly\_best\_employee table

Details Description Columns Depends On Code

Details

TAGS	OWNER	TYPE	PACKAGE	LANGUAGE
untagged	postgres	table	dbt_dskola_project	sql

Description

This model is not currently documented

Lineage Graph

```
graph TD; raw_employees[raw.employees] --> raw_order_details[raw.order_details]; raw_order_details --> raw_orders[raw.orders]; raw_orders --> datamart_monthly_best_employee[datamart_monthly_best_employee]
```

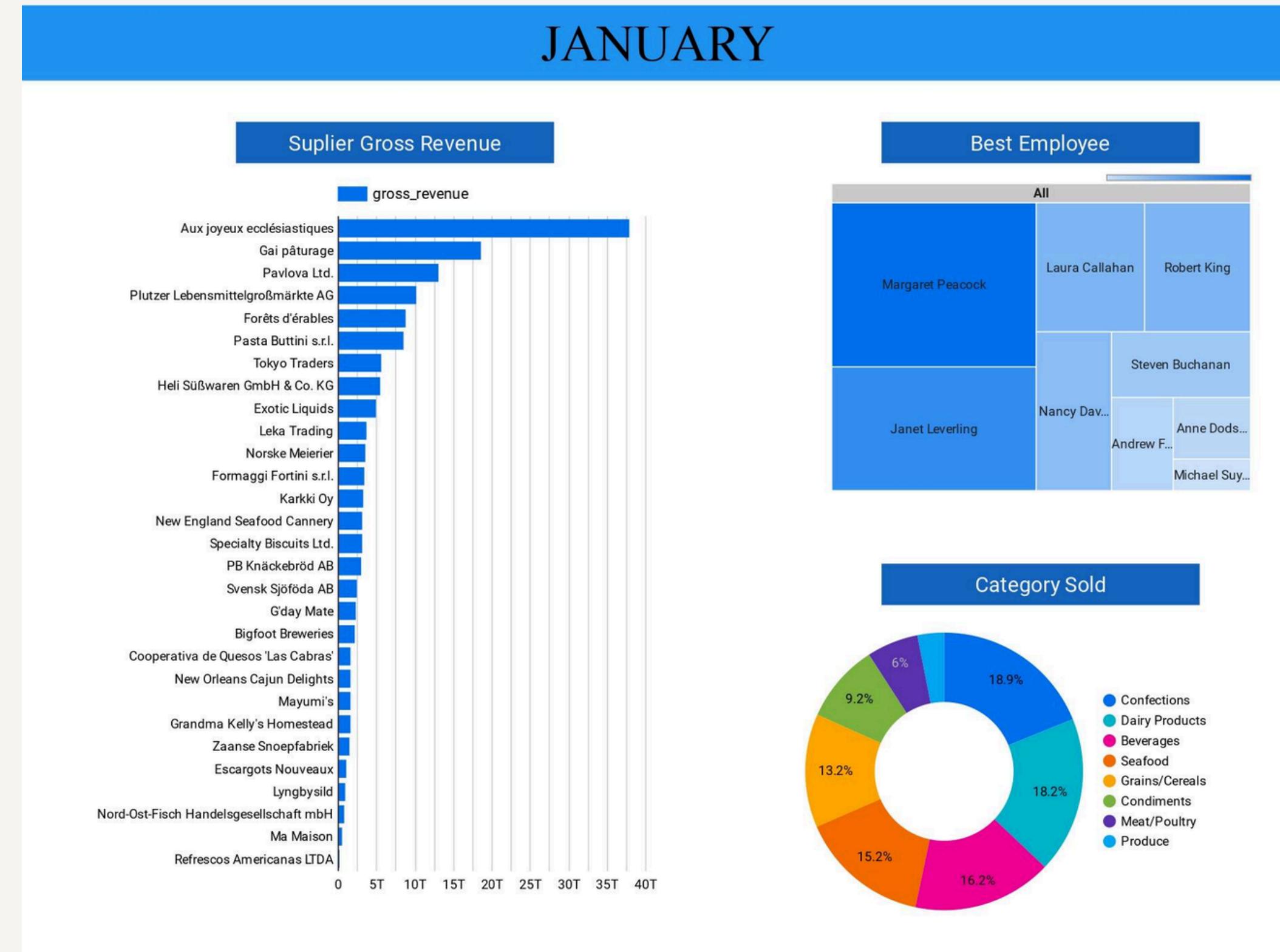
COLUMNS

COLUMN	TYPE	DESCRIPTION	CONSTRAINTS

# Project 2 - Result



04 Dashboard hasil dari data end to end proses dari dbt menggunakan Looker Studio

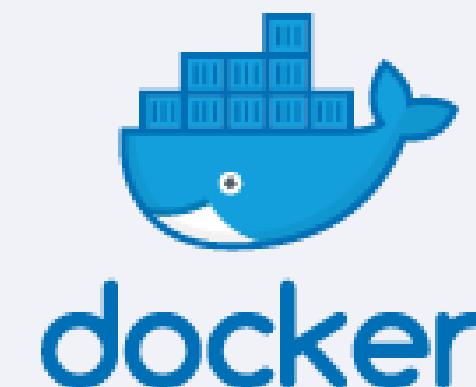


# Project 3: Airflow

## Objective Project

Implementasi Data Pipeline End to End dengan menggunakan Docker, Apache Airflow, Posgtre SQL dan Snowflake

Tools yang digunakan dalam project



# Project 3: AIRFLOW

```
Windows PowerShell
--tls      Use TLS; implied by --tlsverify
--tlscacert string Trust certs signed only by this CA (default
                      "C:\\\\Users\\\\PC\\\\.docker\\\\ca.pem")
--tlscert string Path to TLS certificate file (default
                     "C:\\\\Users\\\\PC\\\\.docker\\\\cert.pem")
--tlskey string Path to TLS key file (default
                     "C:\\\\Users\\\\PC\\\\.docker\\\\key.pem")
--tlsverify     Use TLS and verify the remote
-v, --version   Print version information and quit

Run 'docker COMMAND --help' for more information on a command.

For more help on how to use Docker, head to https://docs.docker.com/go/guides/

[+] Running 8/8 docker> docker-compose up -d
  Network docker_default          Created
  Container docker-redis-1        Healthy
  Container docker-postgres-1     Healthy
  Container docker-airflow-worker-1 Started
  Container docker-flower-1       Started
  Container docker-airflow-webserver-1 Started
  Container docker-airflow-init-1  Started
  Container docker-airflow-scheduler-1 Started
PS C:\\\\Users\\\\PC\\\\docker>
```

1

Tampilan running airflow di docker (Windows Shell)

Docker desktop interface showing the 'Containers' tab. The sidebar includes 'Containers', 'Images', 'Volumes', 'Builds', and 'Docker Scout'. The main area displays the following table of running containers:

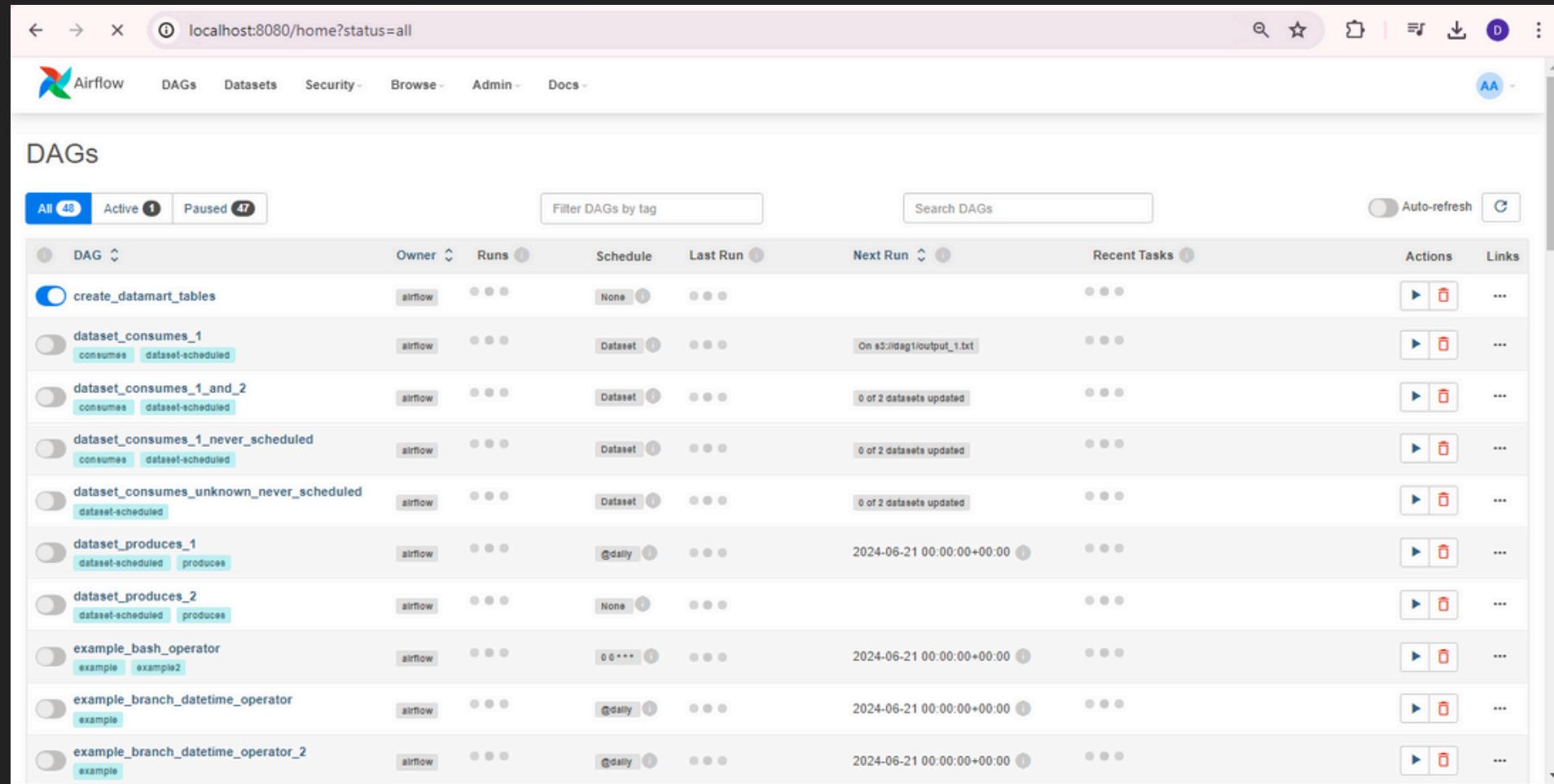
Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
docker		Running (6/7)		384.63%	42 minutes ago	[...]
redis-1	redis:latest	Running	6379:6379	2.13%	42 minutes ago	[...]
postgres-1	postgres:13	Running		0.07%	42 minutes ago	[...]
flower-1	apache/airflow:2.0.2	Running	5555:5555	84.47%	42 minutes ago	[...]
airflow-schedule	apache/airflow:2.0.2	Running		67.22%	42 minutes ago	[...]
airflow-webserve	apache/airflow:2.0.2	Running	8080:8080	93.77%	42 minutes ago	[...]
airflow-worker-1	apache/airflow:2.0.2	Running		66.05%	42 minutes ago	[...]

At the bottom, status indicators show 'Engine running', system resources (RAM 5.36 GB, CPU 64.01%), and a note 'Not signed in'. A notification bar at the bottom right indicates 'New version available'.

2

Tampilan Docker Running UI

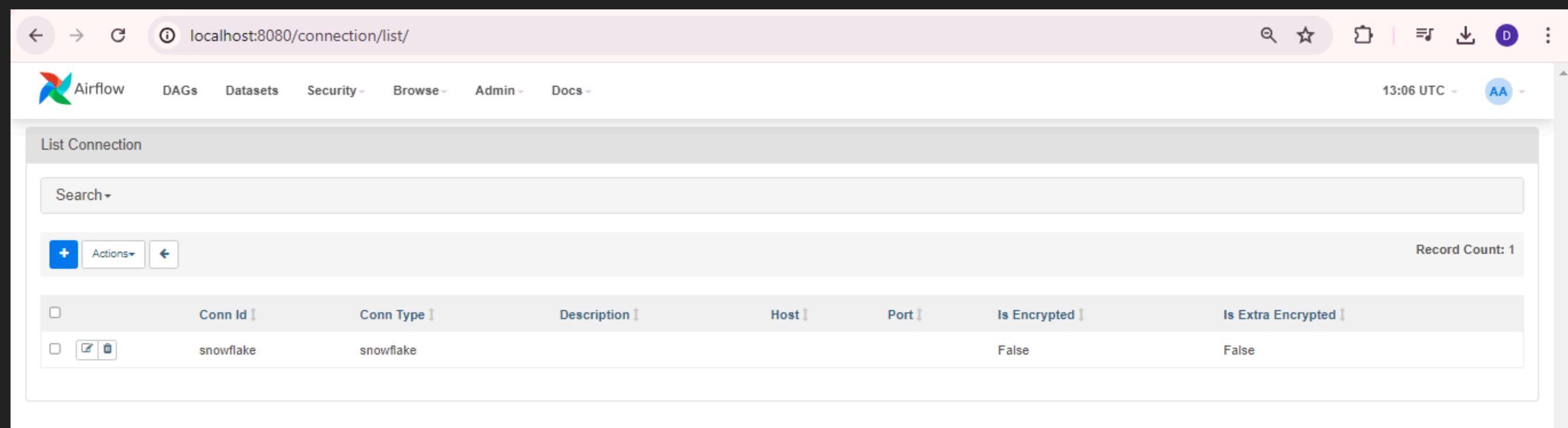
# Project 3: AIRFLOW



A screenshot of the Airflow web interface showing the 'DAGs' page. The page displays a list of 10 DAGs, each with columns for Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, Actions, and Links. The DAGs listed are: create\_datamart\_tables, dataset\_consumes\_1, dataset\_consumes\_1\_and\_2, dataset\_consumes\_1\_never\_scheduled, dataset\_consumes\_unknown\_never\_scheduled, dataset\_produces\_1, dataset\_produces\_2, example\_bash\_operator, example\_branch\_datetime\_operator, and example\_branch\_datetime\_operator\_2. Most DAGs have 'airflow' as the owner and 'None' as the schedule. The 'dataset\_consumes\_1' DAG has a 'Dataset' schedule and shows 'On s3://dag1/output\_1.txt' in the Recent Tasks column.

3

Tampilan halaman airflow



A screenshot of the Airflow web interface showing the 'List Connection' page. The page displays a single connection named 'snowflake' of type 'snowflake'. The connection details are: Conn Id: snowflake, Conn Type: snowflake, Description: (empty), Host: (empty), Port: (empty), Is Encrypted: False, and Is Extra Encrypted: False. There is a 'Record Count: 1' message at the top right of the table.

	Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
	snowflake	snowflake				False	False

4

Tampilan Koneksi ke snowflake

# Project 3: AIRFLOW

DAGs

All 48 Active 1 Paused 47

Filter DAGs by tag Search DAGs Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
create_datamart_tables	airflow	0 0 1 1	None	2024-06-22, 12:43:52		2 0 0 0 1 0 0 0 0 0 0 0 0 0	[Run]	[View]

Showing 1-1 of 1 DAGs

5 Tampilan saat running DAG datamart

DAG: create\_datamart\_tables

Grid Graph Calendar Task Duration Task Tries Landing Times Gantt Details Code Audit Log

06/22/2024 12:45:41 PM 25 All Run Types All Run States Clear Filters

Auto-refresh DAG create\_datamart\_tables

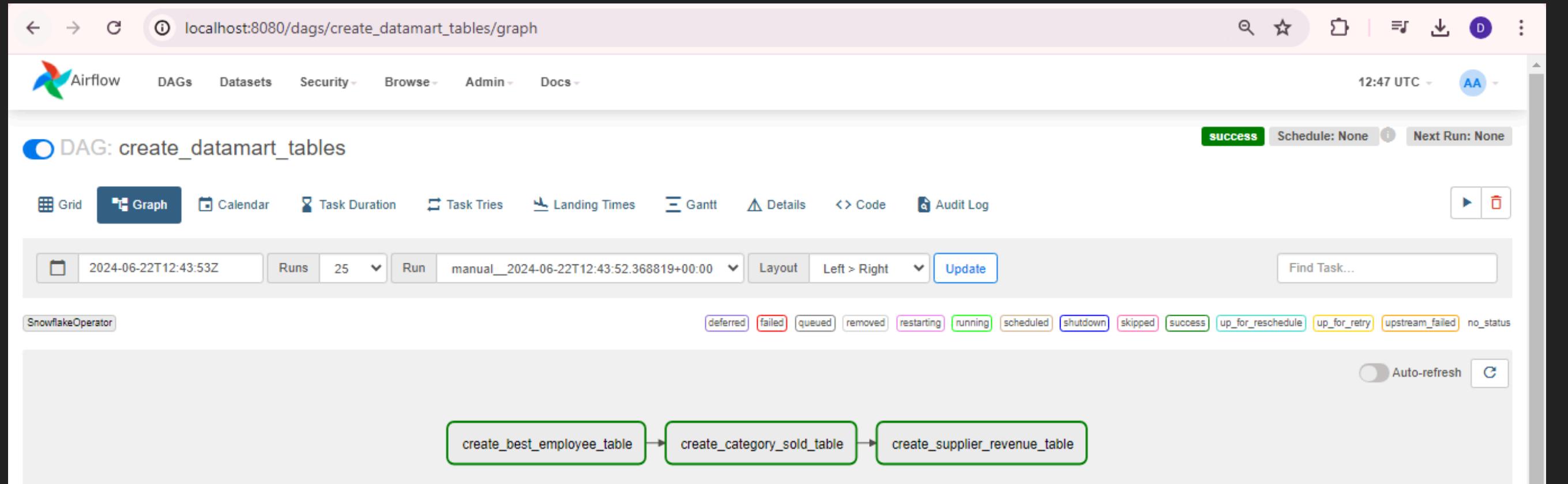
Task	Duration	Run State
create_best_employee_table	00:01:06	Success
create_category_sold_table	00:00:33	Success
create_supplier_revenue_table	00:00:00	Success

DAG Details

Total Runs Displayed	2
Total success	1
Total failed	1
First Run Start	2024-06-22, 12:38:36 UTC
Last Run Start	2024-06-22, 12:43:52 UTC
Max Run Duration	00:01:06
Mean Run Duration	00:00:33

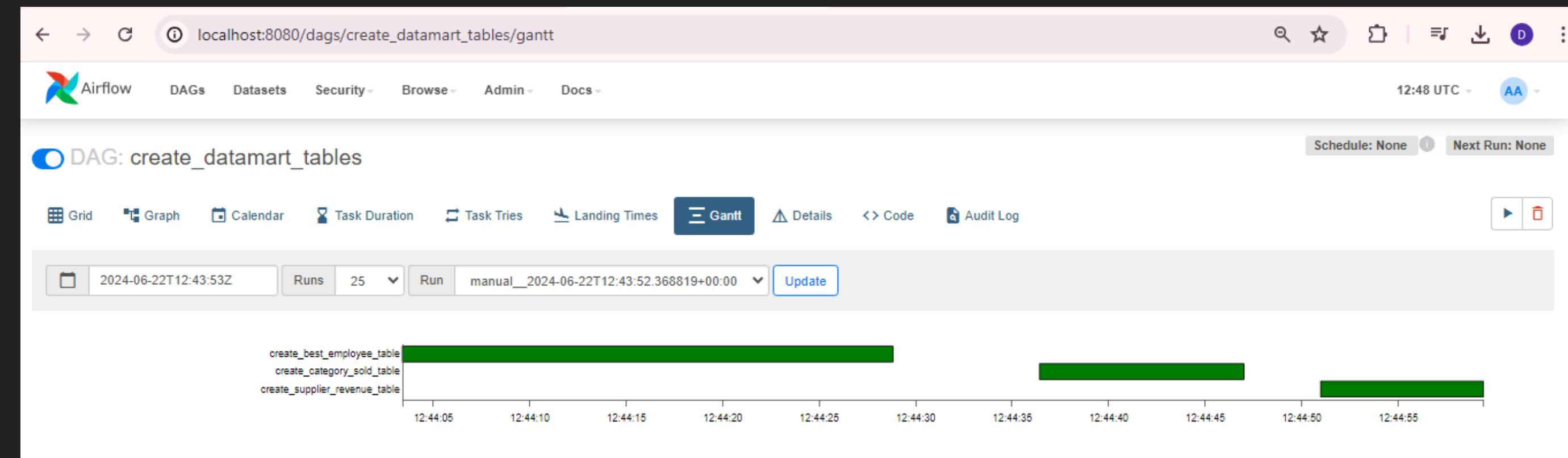
6 Tampilan di Grid Running Datamart

# Project 3: AIRFLOW



Tampilan di Graph  
Running Datamart

7



Tampilan di Grantt  
Running Datamart

8

# Project 3: AIRFLOW

DAG Details

failed 2 success 3 upstream\_failed 1

Schedule Interval None

Catchup False

Started True

End Date None

Max Active Runs 0 / 16

Concurrency 16

Default Args `{'owner': 'airflow', 'retries': 1, 'start_date': DateTime(2023, 1, 1, 0, 0, tzinfo=Timezone('UTC'))}`

Tasks Count 3

Task IDs `['create_best_employee_table', 'create_category_sold_table', 'create_supplier_revenue_table']`

Relative file location `datamart.py`

Owner airflow

Owner Links None

DAG Run Times

9

Tampilan di Detail Running Datamart

```
from airflow import DAG
from airflow.providers.snowflake.operators.snowflake import SnowflakeOperator
from datetime import datetime

# Definisikan default args
default_args = {
    'owner': 'airflow',
    'start_date': datetime(2023, 1, 1),
    'retries': 1,
}

# Definisikan query untuk membuat tabel data mart
create_datamart_best_employee_query = """
CREATE OR REPLACE TABLE PROJECT3.PUBLIC.DATAMART_MONTHLY_BEST_EMPLOYEE AS
SELECT
    e.FIRSTNAME || ' ' || e.LASTNAME AS EMPLOYEE_NAME,
    EXTRACT(MONTH FROM o.ORDERDATE) AS MONTH,
    SUM((od.UNITPRICE - (od.UNITPRICE * od.DISCOUNT)) * od.QUANTITY) AS TOTAL_GROSS_REVENUE
FROM
    PROJECT3.PUBLIC.ORDERS o
JOIN PROJECT3.PUBLIC.ORDER_DETAILS od ON o.ORDERID = od.ORDERID
JOIN PROJECT3.PUBLIC.EMPLOYEES e ON o.EMPLOYEEID = e.EMPLOYEEID
GROUP BY
    e.FIRSTNAME,
    e.LASTNAME,
    EXTRACT(MONTH FROM o.ORDERDATE)
ORDER BY
    EXTRACT(MONTH FROM o.ORDERDATE),
    TOTAL_GROSS_REVENUE DESC;
"""

create_datamart_category_sold_query = """
CREATE OR REPLACE TABLE PROJECT3.PUBLIC.DATAMART_CATEGORY_SOLD AS
SELECT
    c.CATEGORY_NAME AS CATEGORY_NAME,
    EXTRACT(MONTH FROM o.ORDERDATE) AS MONTH,
    SUM(od.QUANTITY) AS TOTAL_QUANTITY
FROM
    PROJECT3.PUBLIC.ORDERS o
JOIN PROJECT3.PUBLIC.ORDER_DETAILS od ON o.ORDERID = od.ORDERID
JOIN PROJECT3.PUBLIC.CATEGORIES c ON od.CATEGORYID = c.CATEGORYID
GROUP BY
    c.CATEGORY_NAME,
    EXTRACT(MONTH FROM o.ORDERDATE)
ORDER BY
    EXTRACT(MONTH FROM o.ORDERDATE),
    TOTAL_QUANTITY DESC;
"""

create_datamart_supplier_revenue_query = """
CREATE OR REPLACE TABLE PROJECT3.PUBLIC.DATAMART_SUPPLIER_REVENUE AS
SELECT
    s.SUPPLIER_NAME AS SUPPLIER_NAME,
    EXTRACT(MONTH FROM o.ORDERDATE) AS MONTH,
    SUM((od.UNITPRICE - (od.UNITPRICE * od.DISCOUNT)) * od.QUANTITY) AS TOTAL_GROSS_REVENUE
FROM
    PROJECT3.PUBLIC.ORDERS o
JOIN PROJECT3.PUBLIC.ORDER_DETAILS od ON o.ORDERID = od.ORDERID
JOIN PROJECT3.PUBLIC.SUPPLIERS s ON od.SUPPLIERID = s.SUPPLIERID
GROUP BY
    s.SUPPLIER_NAME,
    EXTRACT(MONTH FROM o.ORDERDATE)
ORDER BY
    EXTRACT(MONTH FROM o.ORDERDATE),
    TOTAL_GROSS_REVENUE DESC;
"""

# Define the DAG
dag = DAG(
    dag_id='create_datamart_tables',
    default_args=default_args,
    schedule_interval=None,
    start_date=datetime(2023, 1, 1),
    catchup=False,
)
```

10

Tampilan di Code Running Datamart

# Project 3: AIRFLOW

## 11 Tampilan Log Running Datamart

The screenshot shows the Airflow web interface at the URL `localhost:8080/dags/create_datamart_tables/audit_log?root=`. The top navigation bar includes links for Airflow, DAGs, Datasets, Security, Browse, Admin, and Docs, along with a search icon and a dark mode switch. The current time is listed as 12:50 UTC. The main header indicates the DAG being viewed is `create_datamart_tables`, with a status of "Schedule: None" and "Next Run: None". Below the header, there are several tabs: Grid, Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, and Audit Log. The Audit Log tab is currently selected. The title of the main content area is "Dag Audit Log". A descriptive text below the title states: "This view displays selected events and operations that have been taken on this dag. The included and excluded events are set in the Airflow configuration, which by default remove view only actions. For a full list of events regarding this DAG, click here." The main content is a table listing audit log entries:

Time	Task ID	Event	Logical Date	Owner	Details
2024-06-22, 12:44:59	create_supplier_revenue_table	success	2024-06-22 12:43:52.368819+00:00	airflow	None
2024-06-22, 12:44:51	create_supplier_revenue_table	cli_task_run	None	airflow	{"host_name": "03783357291b", "full_command": "[/home/airflow/.local/bin/airflow', 'celery', 'worker]"}
2024-06-22, 12:44:51	create_supplier_revenue_table	running	2024-06-22 12:43:52.368819+00:00	airflow	None
2024-06-22, 12:44:47	create_supplier_revenue_table	cli_task_run	None	airflow	{"host_name": "03783357291b", "full_command": "[/home/airflow/.local/bin/airflow', 'celery', 'worker]"}
2024-06-22, 12:44:47	create_category_sold_table	success	2024-06-22 12:43:52.368819+00:00	airflow	None
2024-06-22, 12:44:36	create_category_sold_table	cli_task_run	None	airflow	{"host_name": "03783357291b", "full_command": "[/home/airflow/.local/bin/airflow', 'celery', 'worker]"}
2024-06-22, 12:44:36	create_category_sold_table	running	2024-06-22 12:43:52.368819+00:00	airflow	None
2024-06-22, 12:44:36	create_category_sold_table	cli_task_run	None	airflow	{"host_name": "03783357291b", "full_command": "[/home/airflow/.local/bin/airflow', 'celery', 'worker]"}

# Project 3: AIRFLOW

12

Tampilan sinkronasi dari Postgres ke Snowflake

The screenshot shows the Airflow web interface with the following details:

**DAG: sync\_postgres\_to\_snowflake\_data** Sync data from PostgreSQL to Snowflake using alternative approach

Schedule: None | Next Run ID: None | Auto-refresh: Off | 25 | Press shift + F for Shortcuts

**DAG Runs Summary**

Total Runs Displayed	1
Total success	1
First Run Start	2024-06-23, 07:04:28 UTC
Last Run Start	2024-06-23, 07:04:28 UTC
Max Run Duration	00:00:50
Mean Run Duration	00:00:50
Min Run Duration	00:00:50

**DAG Summary**

Total Tasks	23
PythonOperators	12
PostgresOperators	11

**DAG Details**

Dag display name	sync_postgres_to_snowflake_data
Dag id	sync_postgres_to_snowflake_data
Description	Sync data from PostgreSQL to Snowflake using alternative approach
Fileloc	/opt/airflow/dags/postgres_to_snowflake.py
Has import errors	false
Has task concurrency limits	false

**Task List (Left Side)**

- create\_snowflake\_tables
- extract\_categories\_from\_postgres
- load\_categories\_info\_snowflake
- extract\_customers\_from\_postgres
- load\_customers\_info\_snowflake
- extract\_employee\_territories\_from\_postgres
- load\_employee\_territories\_info\_snowflake
- extract\_employees\_from\_postgres
- load\_employees\_info\_snowflake
- extract\_order\_details\_from\_postgres
- load\_order\_details\_info\_snowflake
- extract\_orders\_from\_postgres
- load\_orders\_info\_snowflake
- extract\_products\_from\_postgres
- load\_products\_info\_snowflake
- extract\_regions\_from\_postgres
- load\_regions\_info\_snowflake
- extract\_shippers\_from\_postgres
- load\_shippers\_info\_snowflake
- extract\_suppliers\_from\_postgres
- load\_suppliers\_info\_snowflake
- extract\_territories\_from\_postgres
- load\_territories\_info\_snowflake

# Project 3: AIRFLOW

## 13 Tampilan Data Hasil categories di Snowflake

The screenshot shows the Snowflake web interface with the following details:

- Left Sidebar:** Shows navigation links like 'Create', 'Search', 'Projects', 'Data', 'Databases' (which is selected), 'Add Data', 'Data Products', 'AI & ML', 'Monitoring', and 'Admin'. A progress bar at the bottom indicates '\$389 of \$400 left'.
- Central Navigation:** Displays the database structure: DB\_DSKOLA, PROJECT3, PUBLIC, and PUBLIC2. Under PUBLIC2, the 'Tables' section is expanded, showing 'CATEGORIES' (selected), CUSTOMERS, EMPLOYEES, EMPLOYEE\_TERRITORIES, ORDERS, ORDER\_DETAILS, PRODUCTS, REGIONS, SHIPPERS, SUPPLIERS, and TERRITORIES.
- Table Preview Area:** Titled 'PROJECT3 / PUBLIC2 / CATEGORIES'. It shows a preview of the 'CATEGORIES' table with 8 rows. The columns are: CATEGORYID, CATEGORYNAME, and DESCRIPTION. The data is as follows:

	CATEGORYID	CATEGORYNAME	DESCRIPTION
1	1	Beverages	Soft drinks coffees teas beers and ales
2	2	Condiments	Sweet and savory sauces relishes spreads and
3	3	Confections	Desserts candies and sweet breads
4	4	Dairy Products	Cheeses
5	5	Grains/Cereals	Breads crackers pasta and cereal
6	6	Meat/Poultry	Prepared meats
7	7	Produce	Dried fruit and bean curd
8	8	Seafood	Seaweed and fish

# Project 3: AIRFLOW

14

Tampilan data Hasil datamart di Snowflake

The screenshot shows the Snowflake web interface with the URL [https://app.snowflake.com/ap-southeast-3.aws/hr14309/#/data/databases/PROJECT3/schemas/PUBLIC/table/DATAMART\\_MONTHLY\\_BEST\\_EMPLOYEE/data-preview](https://app.snowflake.com/ap-southeast-3.aws/hr14309/#/data/databases/PROJECT3/schemas/PUBLIC/table/DATAMART_MONTHLY_BEST_EMPLOYEE/data-preview). The left sidebar shows the navigation menu with 'Databases' selected. The main area displays the 'Data Preview' tab for the 'DATAMART\_MONTHLY\_BEST\_EMPLOYEE' table, which contains 96 rows. The columns are 'EMPLOYEE\_NAME', 'MONTH', and 'TOTAL\_GROSS\_REVENUE'. The data is as follows:

	EMPLOYEE_NAME	MONTH	TOTAL_GROSS_REVENUE
1	Margaret Peacock	1	42982.435000000
2	Janet Leverling	1	32686.022500000
3	Laura Callahan	1	17848.682500000
4	Robert King	1	17827.340000000
5	Nancy Davolio	1	15446.961500000
6	Steven Buchanan	1	11702.800000000
7	Anne Dodsworth	1	6320.119000000
8	Michael Suyama	1	3254.830000000
9	Janet Leverling	2	31752.880000000
10	Margaret Peacock	2	22765.055000000
11	Anne Dodsworth	2	19203.340000000
12	Nancy Davolio	2	13093.905000000

# Project 3: Result

15

Tampilan table Hasil datamart di Snowflake

The screenshot shows the Snowflake web interface with a dark theme. On the left, the sidebar includes options like Create, Search, Projects, Data (with Databases selected), Data Products, AI & ML, Monitoring, and Admin. A progress bar at the bottom indicates '\$392 of \$400 left' with an Upgrade button. The main area displays the 'PROJECT3 / PUBLIC / DATAMART\_MONTHLY\_SUP...' table details. The table was created by ACCOUNTADMIN 4 hours ago, containing 338 rows and 5.0KB of data. The 'Table definition' section shows the SQL code:

```
1 create or replace TABLE
PROJECT3.PUBLIC.DATAMART_MONTHLY_SUPPLIER_GROSS_REVENUE (
2   SUPPLIER_NAME VARCHAR(256),
3   MONTH NUMBER(2,0),
4   GROSS_REVENUE NUMBER(38,9)
```

Below the table definition is a 'Privileges' section where ACCOUNTADMIN (Current Role) has ownership.

# Project 4 : Kafka

## Objective Project

Streaming processing

- Check Module ML
- Masukan data ke dlm db
- Load data new ke kafka, time.sleep(10)
- Mengambil data dari kafka, procedure dan consumer dijalankan bersamaan
- Get data dalam database
- Join data kafka Dan DB postgres
- Panggil func Predict Detect
- Insert di mongodb
- Save data CSV

Tools yang digunakan dalam project



# Project 4 - Kafka

01

Modeling/Fraudmodul.py --> Check Module ML

The screenshot shows a Jupyter Notebook interface titled "DE - STREAM PROCESSING". The left sidebar displays a file tree for a project named "DE - STREAM PROCESSING". The notebook contains several cells of Python code:

```
from modelling import FraudModel

#parameter inputan data
new_data = {
    'step': 1,
    'type': 'PAYMENT',
    'amount': 9839.64,
    'oldbalanceOrg': 170136.0,
    'newbalanceOrig': 160296.36,
    'oldbalanceDest': 0.0,
    'newbalanceDest': 0.0
}

new_data
#parameter inputan path
import os

path = os.getcwd()
path = path + "\\modelling\\"
path

result = FraudModel.runModel(new_data, path)

result
```

The code imports the FraudModel class from the modelling module. It defines a new\_data dictionary with payment details. Then, it uses the FraudModel.runModel function to process this data, resulting in the output 'White List'.

# Project 4 - Kafka

02

Producer/datadump --> masukan data ke dlm db

The screenshot shows a Jupyter Notebook interface within a Visual Studio Code window. The title bar reads "DE - STREAM PROCESSING". The left sidebar displays a file tree for a project named "DE - STREAM PROCESSING". The "producer" directory contains several files: .ipynb\_checkpoints, \_\_init\_\_.py, FraudModel.py, modelling.ipynb, New\_Information.csv, Old\_Information.csv, producer.ipynb, producer.py, and some presentation files like Materi - Stream Processing.pptx and consumer.ipynb. The main editor area contains the following Python code:

```
from sqlalchemy import create_engine

def insert_data_to_postgresql(df, table_name, db_url):
    try:
        engine = create_engine(db_url)

        df.to_sql(table_name, engine, if_exists='append', index=False)
        print(f'Data telah dimasukkan ke tabel {table_name}.')
    except Exception as e:
        print(f'Terjadi kesalahan: {e}')

if __name__ == "__main__":
    csv_path = "Old_Information.csv"
    data = pd.read_csv(csv_path)

    # Informasi koneksi ke PostgreSQL
    username = "ftde01"
    password = "ftde01!@#"
    host = "34.50.87.186"
    port = "5432"
    database = "stream_processing"
    password = urllib.parse.quote_plus(password)

    # URL koneksi ke PostgreSQL
    db_url = f"postgresql://{username}:{password}@{host}:{port}/{database}"

    table_name = "old_information"
    insert_data_to_postgresql(data, table_name, db_url)
```

The code is run, and the terminal output shows:

```
... Data telah dimasukkan ke tabel old_information.
```

Below the terminal, the status bar indicates "Spaces: 4 CRLF Cell 1 of 1".

# Project 4 - Kafka

## 03 Data yang sudah masuk di DB

The screenshot shows the DBeaver 24.0.4 interface with the following details:

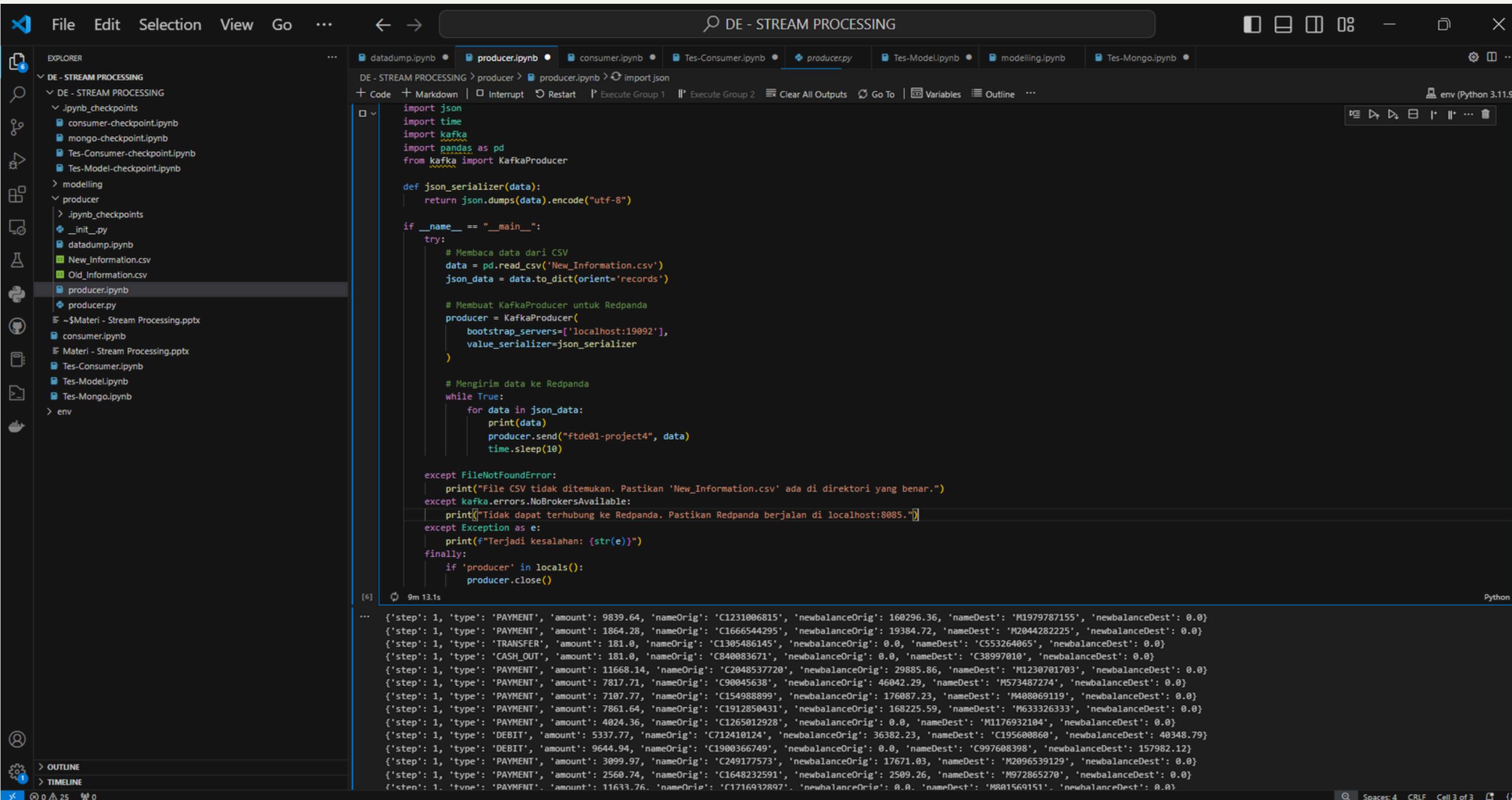
- File Bar:** File, Edit, Navigate, Search, SQL Editor, Database, Window, Help.
- Toolbar:** SQL, Commit, Rollback, Auto, stream\_processing, public@stream\_processing.
- Projects View:** Shows multiple databases: postgres, postgres 2, postgres 3, and stream\_processing (selected).
- Tables View:** Under stream\_processing > public, the 'old\_information' table is selected.
- Data View:** A large grid table showing 200 rows of data. The columns are: Grid, nameOrig, oldbalanceOrg, nameDest, and oldbalanceDest.
- Value Panel:** On the right, a panel displays the value C1231006815.
- Bottom Status Bar:** 200 row(s) fetched - 0.015s (0.006s fetch), on 2024-07-11 at 21:25:48.

Grid	nameOrig	oldbalanceOrg	nameDest	oldbalanceDest
1	C1231006815	170,136	M1979787155	0
2	C1666544295	21,249	M2044282225	0
3	C1305486145	181	C553264065	0
4	C840083671	181	C38997010	21,182
5	C2048537720	41,554	M1230701703	0
6	C90045638	53,860	M573487274	0
7	C154988899	183,195	M408069119	0
8	C1912850431	176,087.23	M633326333	0
9	C1265012928	2,671	M1176932104	0
10	C712410124	41,720	C195600860	41,898
11	C1900366749	4,465	C997608398	10,845
12	C249177573	20,771	M2096539129	0
13	C1648232591	5,070	M972865270	0
14	C1716932897	10,127	M801569151	0
15	C1026483832	503,264	M1635378213	0
16	C905080434	15,325	C476402209	5,083
17	C761750706	450	M1731217984	0
18	C1237762639	21,156	M1877062907	0
19	C2033524545	15,123	M473053293	0
20	C1670993182	705	C1100439041	22,425
21	C20804602	13,854	M1344519051	0
22	C1566511202	11,200	C1072520125	20,022

# Project 4 - Kafka

04

Producer/producer.ipynb --> lempar data new ke kafka, time.sleep(10)



The screenshot shows a Jupyter Notebook interface with the title bar "DE - STREAM PROCESSING". The left sidebar displays a file tree under "EXPLORER" with several .ipynb files, including "producer.ipynb" which is currently selected. The main area contains a code cell with the following Python script:

```
import json
import time
import kafka
import pandas as pd
from kafka import KafkaProducer

def json_serializer(data):
    return json.dumps(data).encode("utf-8")

if __name__ == "__main__":
    try:
        # Membaca data dari CSV
        data = pd.read_csv('New_Information.csv')
        json_data = data.to_dict(orient='records')

        # Membuat KafkaProducer untuk Redpanda
        producer = KafkaProducer(
            bootstrap_servers=['localhost:19092'],
            value_serializer=json_serializer
        )

        # Mengirim data ke Redpanda
        while True:
            for data in json_data:
                print(data)
                producer.send("ftde01-project4", data)
                time.sleep(10)

    except FileNotFoundError:
        print("File CSV tidak ditemukan. Pastikan 'New_Information.csv' ada di direktori yang benar.")
    except kafka.errors.NoBrokersAvailable:
        print("Tidak dapat terhubung ke Redpanda. Pastikan Redpanda berjalan di localhost:8085.")
    except Exception as e:
        print(f"Terjadi kesalahan: {str(e)}")
    finally:
        if 'producer' in locals():
            producer.close()

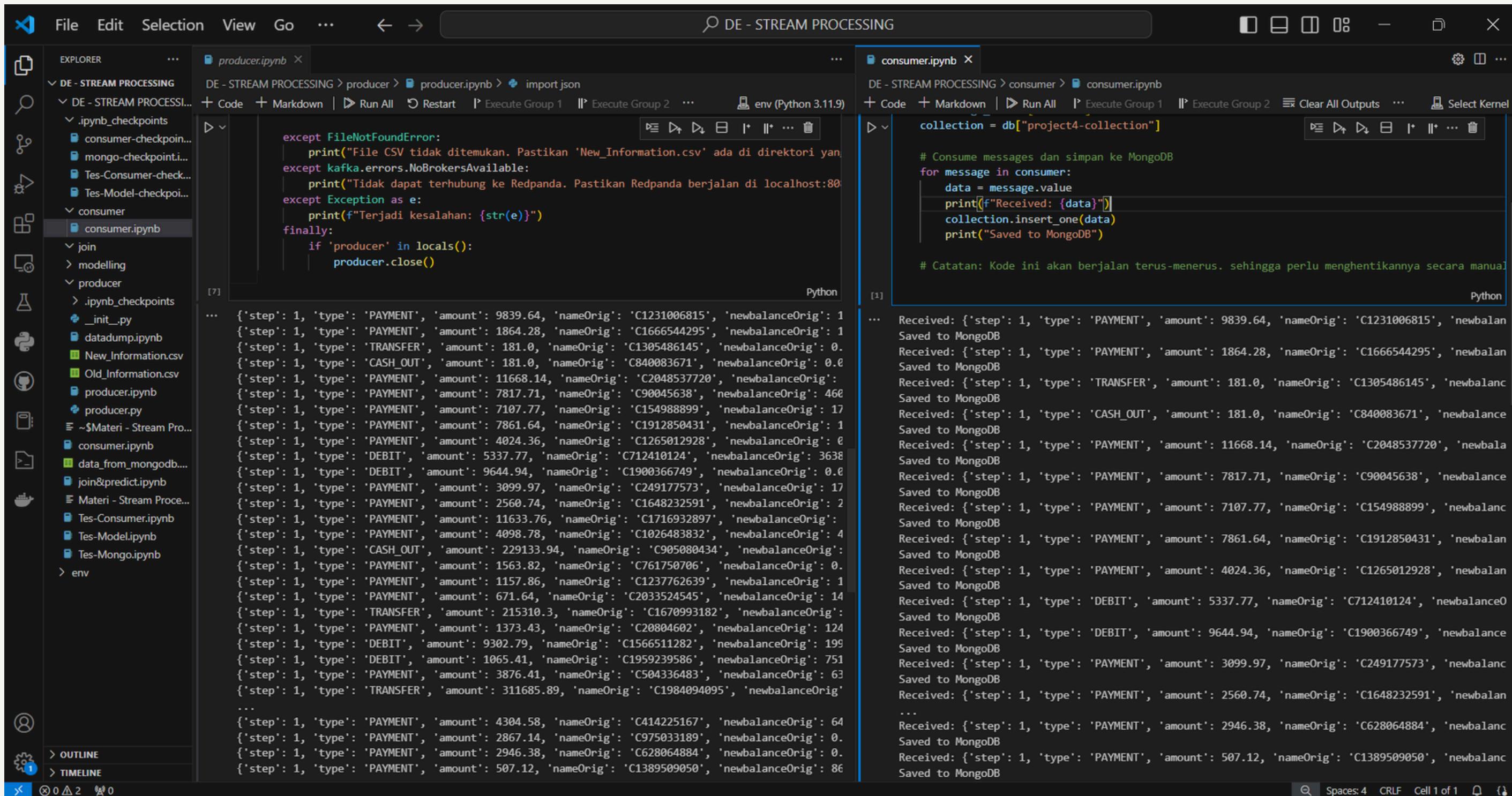
...
```

The code cell has a status bar indicating "9m 13.1s" and "Python". The bottom status bar also shows "Spaces: 4 CRLF Cell 3 of 3".

# Project 4 - Kafka

05

producer/producer.ipynb & consumer/consumer.ipynb--> Mengambil data dari kafka, producer dan consumer dijalankan bersamaan serta menyimpan consume dari kafka ke mongodb



The screenshot shows a Jupyter Notebook interface with two open files:

- producer.ipynb**: Contains Python code for producing data to Kafka. It includes error handling for file not found, broker availability, and general exceptions. It also includes a section for closing the producer.
- consumer.ipynb**: Contains Python code for consuming data from Kafka and saving it to MongoDB. It uses a loop to consume messages, extract the value, print it, and then insert it into a MongoDB collection named "project4-collection". A note at the bottom of this file states: "# Catatan: Kode ini akan berjalan terus-menerus. sehingga perlu menghentikannya secara manual".

Both notebooks are running in a "DE - STREAM PROCESSING" environment, as indicated by the tabs and the title bar. The output pane for the consumer notebook shows a series of log entries indicating successful message consumption and insertion into MongoDB.

```
except FileNotFoundError:
    print("File CSV tidak ditemukan. Pastikan 'New_Information.csv' ada di direktori yang sama dengan file ini")
except kafka.errors.NoBrokersAvailable:
    print("Tidak dapat terhubung ke Redpanda. Pastikan Redpanda berjalan di localhost:8091")
except Exception as e:
    print(f"Terjadi kesalahan: {str(e)}")
finally:
    if 'producer' in locals():
        producer.close()

...
{'step': 1, 'type': 'PAYMENT', 'amount': 9839.64, 'nameOrig': 'C1231006815', 'newbalanceOrig': 1}
{'step': 1, 'type': 'PAYMENT', 'amount': 1864.28, 'nameOrig': 'C1666544295', 'newbalanceOrig': 1}
{'step': 1, 'type': 'TRANSFER', 'amount': 181.0, 'nameOrig': 'C1305486145', 'newbalanceOrig': 0}
{'step': 1, 'type': 'CASH_OUT', 'amount': 181.0, 'nameOrig': 'C840083671', 'newbalanceOrig': 0.0}
{'step': 1, 'type': 'PAYMENT', 'amount': 11668.14, 'nameOrig': 'C2048537720', 'newbalanceOrig': 0.0}
{'step': 1, 'type': 'PAYMENT', 'amount': 7817.71, 'nameOrig': 'C90045638', 'newbalanceOrig': 466}
{'step': 1, 'type': 'PAYMENT', 'amount': 7107.77, 'nameOrig': 'C154988899', 'newbalanceOrig': 17}
{'step': 1, 'type': 'PAYMENT', 'amount': 7861.64, 'nameOrig': 'C1912850431', 'newbalanceOrig': 1}
{'step': 1, 'type': 'PAYMENT', 'amount': 4024.36, 'nameOrig': 'C1265012928', 'newbalanceOrig': 0}
{'step': 1, 'type': 'DEBIT', 'amount': 5337.77, 'nameOrig': 'C712410124', 'newbalanceOrig': 3638}
{'step': 1, 'type': 'DEBIT', 'amount': 9644.94, 'nameOrig': 'C1900366749', 'newbalanceOrig': 0.0}
{'step': 1, 'type': 'PAYMENT', 'amount': 3099.97, 'nameOrig': 'C249177573', 'newbalanceOrig': 17}
{'step': 1, 'type': 'PAYMENT', 'amount': 2560.74, 'nameOrig': 'C1648232591', 'newbalanceOrig': 2}
{'step': 1, 'type': 'PAYMENT', 'amount': 11633.76, 'nameOrig': 'C1716932897', 'newbalanceOrig': 0}
{'step': 1, 'type': 'PAYMENT', 'amount': 4098.78, 'nameOrig': 'C1026483832', 'newbalanceOrig': 4}
{'step': 1, 'type': 'CASH_OUT', 'amount': 229133.94, 'nameOrig': 'C905080434', 'newbalanceOrig': 0}
{'step': 1, 'type': 'PAYMENT', 'amount': 1563.82, 'nameOrig': 'C761750706', 'newbalanceOrig': 0}
{'step': 1, 'type': 'PAYMENT', 'amount': 1157.86, 'nameOrig': 'C1237762639', 'newbalanceOrig': 1}
{'step': 1, 'type': 'PAYMENT', 'amount': 671.64, 'nameOrig': 'C2033524545', 'newbalanceOrig': 14}
{'step': 1, 'type': 'TRANSFER', 'amount': 215310.3, 'nameOrig': 'C1670993182', 'newbalanceOrig': 0}
{'step': 1, 'type': 'PAYMENT', 'amount': 1373.43, 'nameOrig': 'C20804682', 'newbalanceOrig': 124}
{'step': 1, 'type': 'DEBIT', 'amount': 9302.79, 'nameOrig': 'C1566511282', 'newbalanceOrig': 199}
{'step': 1, 'type': 'DEBIT', 'amount': 1065.41, 'nameOrig': 'C1959239586', 'newbalanceOrig': 751}
{'step': 1, 'type': 'PAYMENT', 'amount': 3876.41, 'nameOrig': 'C504336483', 'newbalanceOrig': 63}
{'step': 1, 'type': 'TRANSFER', 'amount': 311685.89, 'nameOrig': 'C1984094095', 'newbalanceOrig': 0}
...
{'step': 1, 'type': 'PAYMENT', 'amount': 4304.58, 'nameOrig': 'C414225167', 'newbalanceOrig': 64}
{'step': 1, 'type': 'PAYMENT', 'amount': 2867.14, 'nameOrig': 'C975033189', 'newbalanceOrig': 0}
{'step': 1, 'type': 'PAYMENT', 'amount': 2946.38, 'nameOrig': 'C628064884', 'newbalanceOrig': 0}
{'step': 1, 'type': 'PAYMENT', 'amount': 507.12, 'nameOrig': 'C1389509050', 'newbalanceOrig': 86}
```

```
collection = db["project4-collection"]

# Consume messages dan simpan ke MongoDB
for message in consumer:
    data = message.value
    print(f"Received: {data}")
    collection.insert_one(data)
    print("Saved to MongoDB")

# Catatan: Kode ini akan berjalan terus-menerus. sehingga perlu menghentikannya secara manual
```

# Project 4 - Kafka

## 06 Consume messages dan simpan ke MongoDB

The screenshot shows the MongoDB Compass interface connected to localhost:27017. The left sidebar lists databases: admin, config, dskola, ftde-01-result, ftde01, local, reactjs, and testDatabase. The ftde01 database is selected, and its project4-collection collection is highlighted. The main pane displays two documents in the project4-collection collection. The first document has the following fields:

```
_id: ObjectId('66913cb197376ebdb144e888')
step : 1
type : "PAYMENT"
amount : 9839.64
nameOrig : "C1231006815"
newbalanceOrig : 160296.36
nameDest : "M1979787155"
newbalanceDest : 0
```

The second document has the following fields:

```
_id: ObjectId('66913cb197376ebdb144e889')
step : 1
type : "PAYMENT"
amount : 1864.28
nameOrig : "C1666544295"
newbalanceOrig : 19384.72
nameDest : "M2044282225"
newbalanceDest : 0
```

# Project 4 - Kafka

07

Join&predict.ipynb --> ambil data dalam database postgres

The screenshot shows a Jupyter Notebook interface with the title bar 'DE - STREAM PROCESSING'. The left sidebar displays a file tree under 'EXPLORER' for a project named 'DE - STREAM PROCESSING'. The main area contains several code cells:

- Cell [1]:

```
import pandas as pd
import urllib.parse

from sqlalchemy import create_engine

# Informasi koneksi ke PostgreSQL
username = "ftde01"
password = "ftde01!@#"
host = "34.50.87.186"
port = "5432"
database = "stream_processing"
password = urllib.parse.quote_plus(password)

# URL koneksi ke PostgreSQL
db_url = f"postgresql://{username}:{password}@{host}:{port}/{database}"
engine = create_engine(db_url)
```
- Cell [2]:

```
df = pd.read_sql_query('SELECT * FROM old_information;', engine)
df.head(5)
```

	nameOrig	oldbalanceOrg	nameDest	oldbalanceDest
0	C1231006815	170136.0	M1979787155	0.0
1	C1666544295	21249.0	M2044282225	0.0
2	C1305486145	181.0	C553264065	0.0
3	C840083671	181.0	C38997010	21182.0
4	C2048537720	41554.0	M1230701703	0.0
- Cell [3]:

```
df.columns
```

	0.0s
...	Index(['nameOrig', 'oldbalanceOrg', 'nameDest', 'oldbalanceDest'], dtype='object')
- Cell [4]:

```
from pymongo import MongoClient
import pandas as pd

# Menghubungkan ke MongoDB
mongo_client = MongoClient("mongodb://localhost:27017/")
db = mongo_client["ftde01"]
collection = db["project4-collection"]

# Mengambil data dari koleksi MongoDB
data = collection.find()
```

# Project 4 - Kafka

08

Join&predict.ipynb --> ambil data dari consume kafka yang telah disimpan ke mongodb

The screenshot shows a Jupyter Notebook interface with the title 'join&predict.ipynb'. The code cell contains the following Python script:

```
from pymongo import MongoClient
import pandas as pd

# Menghubungkan ke MongoDB
mongo_client = MongoClient("mongodb://localhost:27017/")
db = mongo_client["ftde01"]
collection = db["project4-collection"]

# Mengambil data dari koleksi MongoDB
data = list(collection.find())

# Memasukkan data ke dalam DataFrame Pandas
producer = pd.DataFrame(data)

producer
```

Below the code cell, a table displays the data from the MongoDB collection:

	id	step	type	amount	nameOrig	newbalanceOrig	nameDest	newbalanceDest
0	66913cb197376ebdb144e888	1	PAYMENT	9839.64	C1231006815	160296.36	M1979787155	0.00
1	66913cb197376ebdb144e889	1	PAYMENT	1864.28	C1666544295	19384.72	M2044282225	0.00
2	66913cb197376ebdb144e88a	1	TRANSFER	181.00	C1305486145	0.00	C553264065	0.00
3	66913cb197376ebdb144e88b	1	CASH_OUT	181.00	C840083671	0.00	C38997010	0.00
4	66913cb197376ebdb144e88c	1	PAYMENT	11668.14	C2048537720	29885.86	M1230701703	0.00
...	...	...	...	...	...	...	...	...
116	6691409797376ebdb144e8fc	1	PAYMENT	3705.83	C671596011	38197.63	M1925352804	0.00
117	669140a197376ebdb144e8fd	1	CASH_OUT	419801.40	C1687354037	0.00	C33524623	1517262.16
118	669140b977376ebdb144e8fe	1	CASH_OUT	335416.51	C743778731	0.00	C575355780	52415.15
119	669140b977376ebdb144e8ff	1	PAYMENT	3372.29	C967323951	38025.71	M1600594643	0.00
120	669140b977376ebdb144e900	1	PAYMENT	661.43	C743648472	13416.57	M692998280	0.00

121 rows x 8 columns

```
[121] # Menghitung duplikasi dalam df
dups_count_df = df.duplicated(subset=['nameOrig', 'nameDest']).sum()
print(f"Jumlah duplikasi di df: {dups_count_df}")

[122] ... Jumlah duplikasi di df: 45000

[123] # Menghitung duplikasi dalam producer
dups_count_producer = producer.duplicated(subset=['nameOrig', 'nameDest']).sum()
print(f"Jumlah duplikasi di producer: {dups_count_producer}")

[124] ... Jumlah duplikasi di producer: 0

[125] # Menghapus duplikasi dalam df
df_cleaned = df.drop_duplicates(subset=['nameOrig', 'nameDest'])
```

\*Cek duplicate data, ternyata database postgre terdapat data duplikat

# Project 4 - Kafka

09

Join&predict.ipynb --> Join data dari kakfa Dan DB postgres

The screenshot shows a Jupyter Notebook interface with the title bar 'DE - STREAM PROCESSING'. The left sidebar contains an 'EXPLORER' section with a tree view of files and folders related to 'DE - STREAM PROCESSING'. The main area displays Python code in three code cells:

```
# Menghapus duplikasi dalam df
df_cleaned = df.drop_duplicates(subset=['nameOrig', 'nameDest'])

[24]
```

```
data = produder.merge(df_cleaned, how='inner', on=['nameOrig','nameDest'])
predict = data.drop(['nameOrig','nameDest'], axis=1)
predict = predict.to_dict('index')[0]
predict

[29]
```

```
{'_id': ObjectId('66913cb197376ebdb144e888'),
 'step': 1,
 'type': 'PAYMENT',
 'amount': 9839.64,
 'newbalanceOrig': 160296.36,
 'newbalanceDest': 0.0,
 'oldbalanceOrg': 170136.0,
 'oldbalanceDest': 0.0}
```

```
predict

[30]
```

```
{'_id': ObjectId('66913cb197376ebdb144e888'),
 'step': 1,
 'type': 'PAYMENT',
 'amount': 9839.64,
 'newbalanceOrig': 160296.36,
 'newbalanceDest': 0.0,
 'oldbalanceOrg': 170136.0,
 'oldbalanceDest': 0.0}
```

The code performs data cleaning by dropping duplicates from the DataFrame 'df'. It then merges this cleaned DataFrame with a producer's data ('produder') using an inner join on 'nameOrig' and 'nameDest'. The resulting DataFrame is converted to a dictionary where the index is the row number. Finally, the variable 'predict' is assigned to this dictionary.

# Project 4 - Kafka

10

join&predict.ipynb -->panggil func Predict Detect

The screenshot shows a Jupyter Notebook interface titled "DE - STREAM PROCESSING". The left sidebar displays a file tree under "EXPLORER" for a project named "DE - STREAM PROCESSING". The "join&predict.ipynb" file is selected and open in the main notebook area. The code in the notebook is as follows:

```
from modelling import FraudModel
#parameter inputan path
import os
path = os.getcwd()
path = path + "\\modelling\\"
path
'd:\\Data-Engineer-Skola\\Task\\project\\New folder\\DE - STREAM PROCESSING\\DE - STREAM PROCESSING\\modelling\\'
result = FraudModel.runModel(predict, path)
data['predict'] = result
data
```

Below the code, a table is displayed showing 121 rows by 11 columns of data. The columns are labeled: \_id, step, type, amount, nameOrig, newbalanceOrig, nameDest, newbalanceDest, oldbalanceOrg, oldbalanceDest, and predict. The data includes various transaction types like PAYMENT, TRANSFER, and CASH\_OUT, along with their corresponding amounts and balance details.

	_id	step	type	amount	nameOrig	newbalanceOrig	nameDest	newbalanceDest	oldbalanceOrg	oldbalanceDest	predict
0	66913cb197376ebdb144e888	1	PAYMENT	9839.64	C1231006815	160296.36	M1979787155	0.00	170136.00	0.0	White List
1	66913cb197376ebdb144e889	1	PAYMENT	1864.28	C1666544295	19384.72	M2044282225	0.00	21249.00	0.0	White List
2	66913cb197376ebdb144e88a	1	TRANSFER	181.00	C1305486145	0.00	C553264065	0.00	181.00	0.0	White List
3	66913cb197376ebdb144e88b	1	CASH_OUT	181.00	C840083671	0.00	C38997010	0.00	181.00	21182.0	White List
4	66913cb197376ebdb144e88c	1	PAYMENT	11668.14	C2048537720	29885.86	M1230701703	0.00	41554.00	0.0	White List
...	...	...	...	...	...	...	...	...	...	...	...
116	6691409797376ebdb144e8fc	1	PAYMENT	3705.83	C671596011	38197.63	M1925352804	0.00	41903.46	0.0	White List
117	669140a197376ebdb144e8fd	1	CASH_OUT	419801.40	C1687354037	0.00	C33524623	151726.16	38197.63	499962.0	White List
118	669140ab97376ebdb144e8fe	1	CASH_OUT	335416.51	C743778731	0.00	C575335780	52415.15	144478.00	295.0	White List
119	669140b597376ebdb144e8ff	1	PAYMENT	3372.29	C967323951	38025.71	M1600594643	0.00	41398.00	0.0	White List
120	669140bf97376ebdb144e900	1	PAYMENT	661.43	C743648472	13416.57	M692998280	0.00	14078.00	0.0	White List

# Project 4 - Kafka

11

Join&predict.ipynb --> insert hasil prediksi di mongodb

The screenshot shows a Jupyter Notebook interface with the title bar 'DE - STREAM PROCESSING'. The left sidebar displays a file tree under 'EXPLORER' for a project named 'DE - STREAM PROCESSING'. The current notebook tab is 'Tes-Consumer.ipynb'. The notebook contains two code cells:

**Code Cell 1 (Python):**

```
from pymongo import MongoClient

# Mengatur koneksi ke MongoDB lokal tanpa autentikasi
mongo_client = MongoClient("mongodb://localhost:27017/")

db = mongo_client["ftde01"]
collection = db["project4-collection"]

# Asumsikan 'data' adalah DataFrame pandas yang sudah didefinisikan sebelumnya

for _, row in data.iterrows():
    document = row.to_dict()
    collection.insert_one(document)

print("Data telah disimpan ke MongoDB")
```

**Code Cell 2 (Python):**

```
mongo_client = MongoClient("mongodb://localhost:27017/")
db = mongo_client["ftde01"]
collection = db["project4-collection"]

# Membaca data dari MongoDB ke dalam list of dictionaries
data_from_mongo = list(collection.find())

# Membaca data ke dalam DataFrame
df = pd.DataFrame(data_from_mongo)
df
```

The output of the first cell shows the message "Data telah disimpan ke MongoDB". The output of the second cell shows the data from MongoDB as a DataFrame:

	_id	step	type	amount	nameOrig	newbalanceOrig	nameDest	newbalanceDest	oldbalanceOrg	oldbalanceDest	predict
0	669128897460fec20029f857	1	PAYOUT	3671.32	C361380654	96422.68	M631673932	0.0	100094.0	0.0	White List
1	669128897460fec20029f858	1	PAYOUT	3671.32	C361380654	96422.68	M631673932	0.0	100094.0	0.0	White List
2	669128897460fec20029f859	1	PAYOUT	3671.32	C361380654	96422.68	M631673932	0.0	100094.0	0.0	White List
3	669128897460fec20029f85a	1	PAYOUT	3671.32	C361380654	96422.68	M631673932	0.0	100094.0	0.0	White List

# Project 4 - Kafka

12

Tampilan pada Mongo DB

The screenshot shows the MongoDB Compass interface connected to localhost:27017. The database selected is ftde-01-result, and the collection is project4-result. There are two documents displayed:

```
_id: ObjectId('66913cb197376ebdb144e888')
step : 1
type : "PAYMENT"
amount : 9839.64
nameOrig : "C1231006815"
newbalanceOrig : 160296.36
nameDest : "M1979787155"
newbalanceDest : 0
oldbalanceOrg : 170136
oldbalanceDest : 0
predict : "White List"

_id: ObjectId('66913cb197376ebdb144e889')
step : 1
type : "PAYMENT"
amount : 1864.28
nameOrig : "C1666544295"
newbalanceOrig : 19384.72
nameDest : "M2044282225"
```

# Project 4 - Result

13

Join&predict.ipynb Save data CSV

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, ...
- Toolbar:** Back, Forward, Search bar, and other standard icons.
- Explorer:** Shows the project structure with files like 'join&predict.ipynb', 'consumer.ipynb', and 'data\_from\_mongodb.csv'.
- Code Cell:** Contains Python code for connecting to MongoDB, reading data from a specific collection, and saving it to a CSV file named 'data\_from\_mongodb.csv'. The code includes comments explaining the steps.
- Output Cell:** Shows the message "Data telah disimpan ke data\_from\_mongodb.csv" followed by the first few lines of the generated CSV file.
- CSV Preview:** A preview of the CSV file is shown on the right side of the interface.

```
from pymongo import MongoClient
mongo_client = MongoClient("mongodb://localhost:27017/")

db = mongo_client["ftde-01-result"]
collection = db["project4-result"]

# Membaca data dari MongoDB ke dalam list of dictionaries
data_from_mongo = list(collection.find())

# Membaca data ke dalam DataFrame
df = pd.DataFrame(data_from_mongo)
df

# Menyimpan DataFrame ke file CSV
csv_filename = "data_from_mongodb.csv"
df.to_csv(csv_filename, index=False)

print(f"Data telah disimpan ke {csv_filename}")

[5]:    ✓ 0.2s
... Data telah disimpan ke data_from_mongodb.csv
```

_id	step	type	amount	nameOrig	newbalanceOrig	nameDest	newbalanceDest	oldbalanceOrg	oldbalanceDest	predic
66913cb197376ebdb144e888	1	PAYMENT	9839.64	C1231006815	160296.36	M1979787155	0.0	170136.0	0.0	White List
66913cb197376ebdb144e889	1	PAYMENT	1864.28	C1666544295	19384.72	M2044282225	0.0	21249.0	0.0	White List
66913cb197376ebdb144e88a	1	TRANSFER	181.0	C1305486145	0.0	C553264065	0.0	181.0	0.0	White List
66913cb197376ebdb144e88b	1	CASH_OUT	181.0	C840083671	0.0	C38997010	0.0	181.0	21182.0	White List
66913cb197376ebdb144e88c	1	PAYMENT	11668.14	C2048537720	29885.86	M1230701703	0.0	41554.0	0.0	White List
66913cb197376ebdb144e88d	1	PAYMENT	7817.71	C90045638	46042.29	M573487274	0.0	53860.0	0.0	White List
66913cb197376ebdb144e88e	1	PAYMENT	7107.77	C154988899	176087.23	M408069119	0.0	183195.0	0.0	White List
66913cb197376ebdb144e88f	1	PAYMENT	7861.64	C1912850431	168225.59	M633326333	0.0	176087.23	0.0	White List
66913cb297376ebdb144e890	1	PAYMENT	4024.36	C1265012928	0.0	M1176932104	0.0	2671.0	0.0	White List
66913cb297376ebdb144e891	1	DEBIT	5337.77	C712410124	36382.23	C195600860	40348.79	41720.0	41898.0	White List
66913cb297376ebdb144e892	1	DEBIT	9644.94	C1900366749	0.0	C979608398	157982.12	4465.0	10845.0	White List
66913cb297376ebdb144e893	1	PAYMENT	3099.97	C249177573	17671.03	M2096539129	0.0	20771.0	0.0	White List
66913cb297376ebdb144e894	1	PAYMENT	2560.74	C1648232591	2509.26	M972865270	0.0	5070.0	0.0	White List
66913cb297376ebdb144e895	1	PAYMENT	11633.76	C1716932897	0.0	M801569151	0.0	10127.0	0.0	White List
66913cb297376ebdb144e896	1	PAYMENT	4098.78	C1026483832	499165.22	M1635378213	0.0	503264.0	0.0	White List
66913cb297376ebdb144e897	1	CASH_OUT	229133.94	C905080434	0.0	C476402209	51513.44	15325.0	5083.0	White List
66913cb297376ebdb144e898	1	PAYMENT	1563.82	C761750706	0.0	M1731217984	0.0	450.0	0.0	White List
66913cb997376ebdb144e899	1	PAYMENT	1157.86	C1237762639	19998.14	M1877062907	0.0	21156.0	0.0	White List
66913cc397376ebdb144e89a	1	PAYMENT	671.64	C2033524545	14451.36	M473053293	0.0	15123.0	0.0	White List
66913cc97376ebdb144e89b	1	TRANSFER	215310.3	C1670993182	0.0	C1100439041	0.0	705.0	22425.0	White List
66913cd797376ebdb144e89c	1	PAYMENT	1373.43	C20804602	12480.57	M1344519051	0.0	13854.0	0.0	White List
66913ce197376ebdb144e89d	1	DEBIT	9302.79	C1566511282	1996.21	C1973538135	16896.7	11299.0	29832.0	White List
66913ceb97376ebdb144e89e	1	DEBIT	1065.41	C1959239586	751.59	C515132998	0.0	1817.0	10330.0	White List
66913cf597376ebdb144e89f	1	PAYMENT	3876.41	C504336483	63975.59	M1404932042	0.0	67852.0	0.0	White List
66913cff97376ebdb144e8a0	1	TRANSFER	311685.89	C1984094095	0.0	C932583850	2719172.89	10835.0	6267.0	White List
66913d0997376ebdb144e8a1	1	PAYMENT	6061.13	C1043358826	0.0	M1558079303	0.0	443.0	0.0	White List
66913d1397376ebdb144e8a2	1	PAYMENT	9478.39	C1671590089	107015.61	M58488213	0.0	116494.0	0.0	White List
66913d1d97376ebdb144e8a3	1	PAYMENT	8009.09	C1053967012	2958.91	M295304806	0.0	10968.0	0.0	White List
66913d2797376ebdb144e8a4	1	PAYMENT	8901.99	C1632497828	0.0	M33419717	0.0	2958.91	0.0	White List
66913d3197376ebdb144e8a5	1	PAYMENT	9920.52	C764826684	0.0	M1940055334	0.0	0.0	0.0	White List
66913d3b97376ebdb144e8a6	1	PAYMENT	3448.92	C2103763750	0.0	M335107734	0.0	0.0	0.0	White List
66913d4597376ebdb144e8a7	1	PAYMENT	4206.84	C215078753	0.0	M1757317128	0.0	0.0	0.0	White List
66913d4f97376ebdb144e8a8	1	PAYMENT	5885.56	C840514538	0.0	M1804441305	0.0	0.0	0.0	White List
66913d5997376ebdb144e8a9	1	PAYMENT	5307.88	C1768242710	0.0	M1971783162	0.0	0.0	0.0	White List
66913d6397376ebdb144e8aa	1	PAYMENT	5031.22	C247113419	0.0	M151442075	0.0	0.0	0.0	White List
66913d6d97376ebdb144e8ab	1	PAYMENT	24213.67	C1238616099	0.0	M70695990	0.0	0.0	0.0	White List
66913d7797376ebdb144e8ac	1	PAYMENT	8603.42	C1608633989	0.0	M1615617512	0.0	253.0	0.0	White List
66913d8197376ebdb144e8ad	1	PAYMENT	2791.42	C923341586	297689.58	M107994825	0.0	300481.0	0.0	White List
66913d8b97376ebdb144e8ae	1	PAYMENT	7413.54	C1470868839	290276.03	M1426725223	0.0	297689.58	0.0	White List
66913d9597376ebdb144e8af	1	PAYMENT	3295.19	C711197015	230337.81	M1384454980	0.0	233633.0	0.0	White List
66913d9f97376ebdb144e8b0	1	PAYMENT	1684.81	C1481594086	0.0	M1569435561	0.0	297.0	0.0	White List
66913da997376ebdb144e8b1	1	DEBIT	5758.59	C1466917878	26845.41	C1297685781	16997.22	32604.0	209699.0	White List
66913db397376ebdb144e8b2	1	CASH_OUT	110414.71	C768216420	0.0	C1509514333	2415.16	26845.41	288800.0	White List

**Sekian & Terima Kasih**



Kelompok 13

**fantastic 4**