TEAM 1

FINAL PROJECT

Presentation

STUDI INDEPENDEN KAMPUS MERDEKA - Batch 6

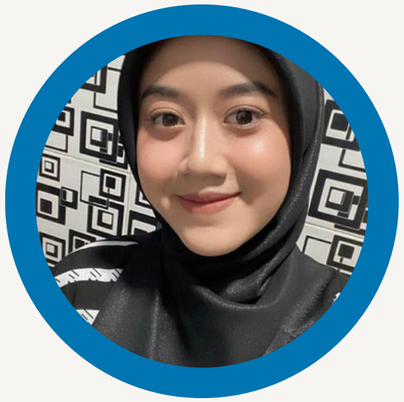
Data Warehousing

HELLO!

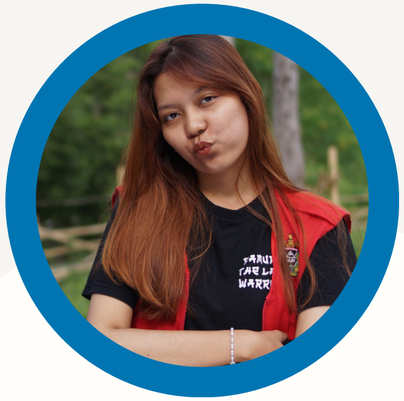
MEET OUR TEAM

INTRODUCTION

INTRODUCTION



Chika Cardealfitri



Clarisa Eudia Chesynanda

Suci Rakhmayanti

Presentation

FINAL PROJECT

01. INTRODUCTION

Penjelasan latar belakang masalah yang terjadi dalam membuat ETL, Data warehouse, Data Visualisasi.

02. ETL

Extract, transform, and load (ETL) adalah proses menggabungkan data dari berbagai sumber ke dalam repositori pusat yang besar yang disebut gudang data.

03. DATA MODELLING

Data modelling adalah proses mengumpulkan data dan mengubahnya menjadi diagram sederhana agar bisa menjadi informasi yang bisa digunakan sesuai kebutuhan bisnis.

04. DATA VISUALISATION

Visualisasi data adalah **proses menggunakan elemen visual seperti diagram, grafik, atau peta untuk merepresentasikan data.** Visualisasi data menerjemahkan yang kompleks, bervolume tinggi, atau numerik menjadi representasi visual yang lebih mudah diproses.

1.

INTRODUCTION

final project
Tugas akhir dari program msib bach 6 data warehousing.
Pada tugas ini menggunakan idx.dev, neon.db, Github dan tableu

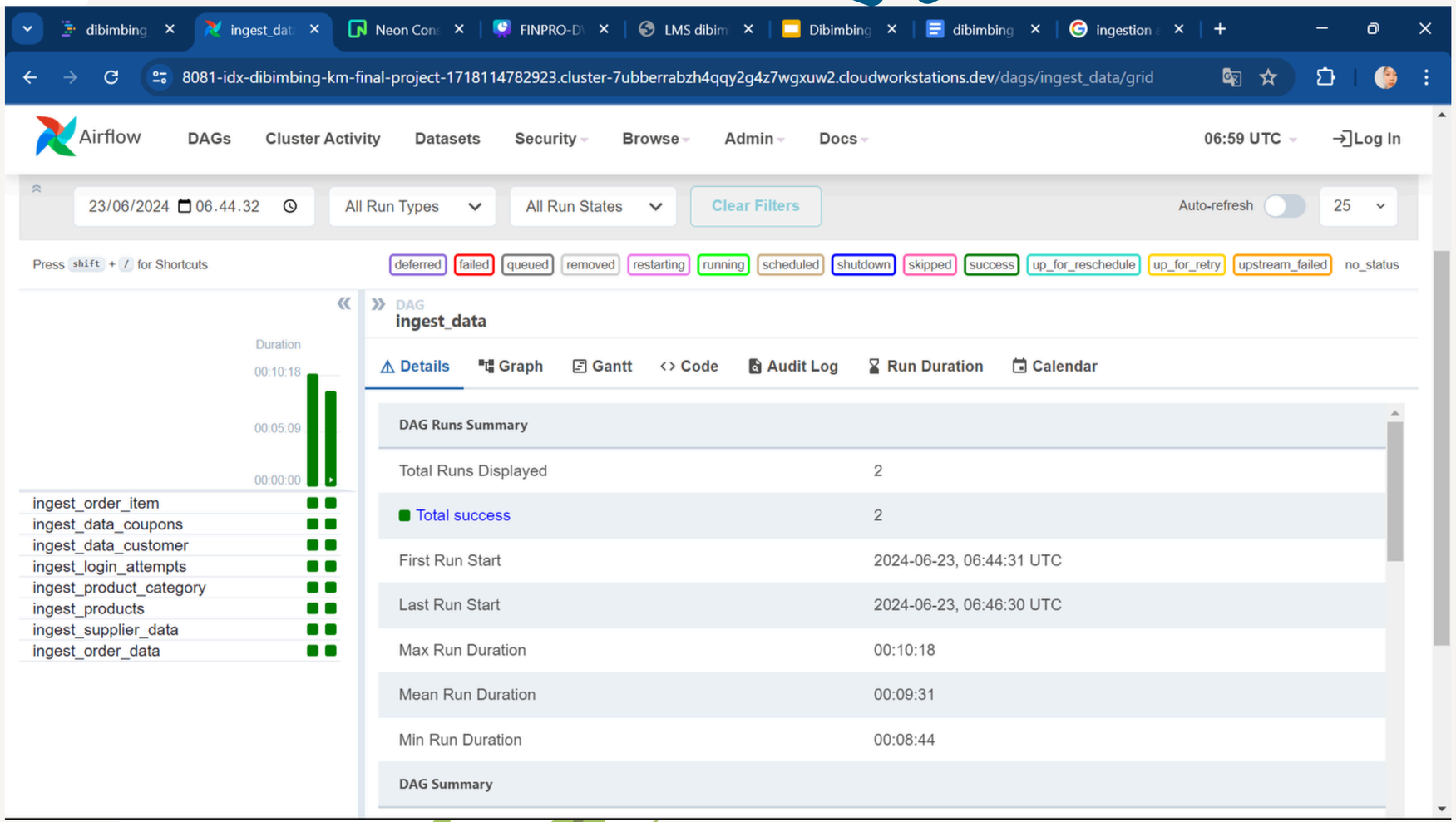
Tasks

- ETL/ELT Job Creation using Airflow
- Data Modeling in Postgres
- Dashboard Creation with Data Visualization
- Craft a Presentation Based on Your Work

Notes: All data being used is generated data crafted to mirror real-world data from online retail companies.

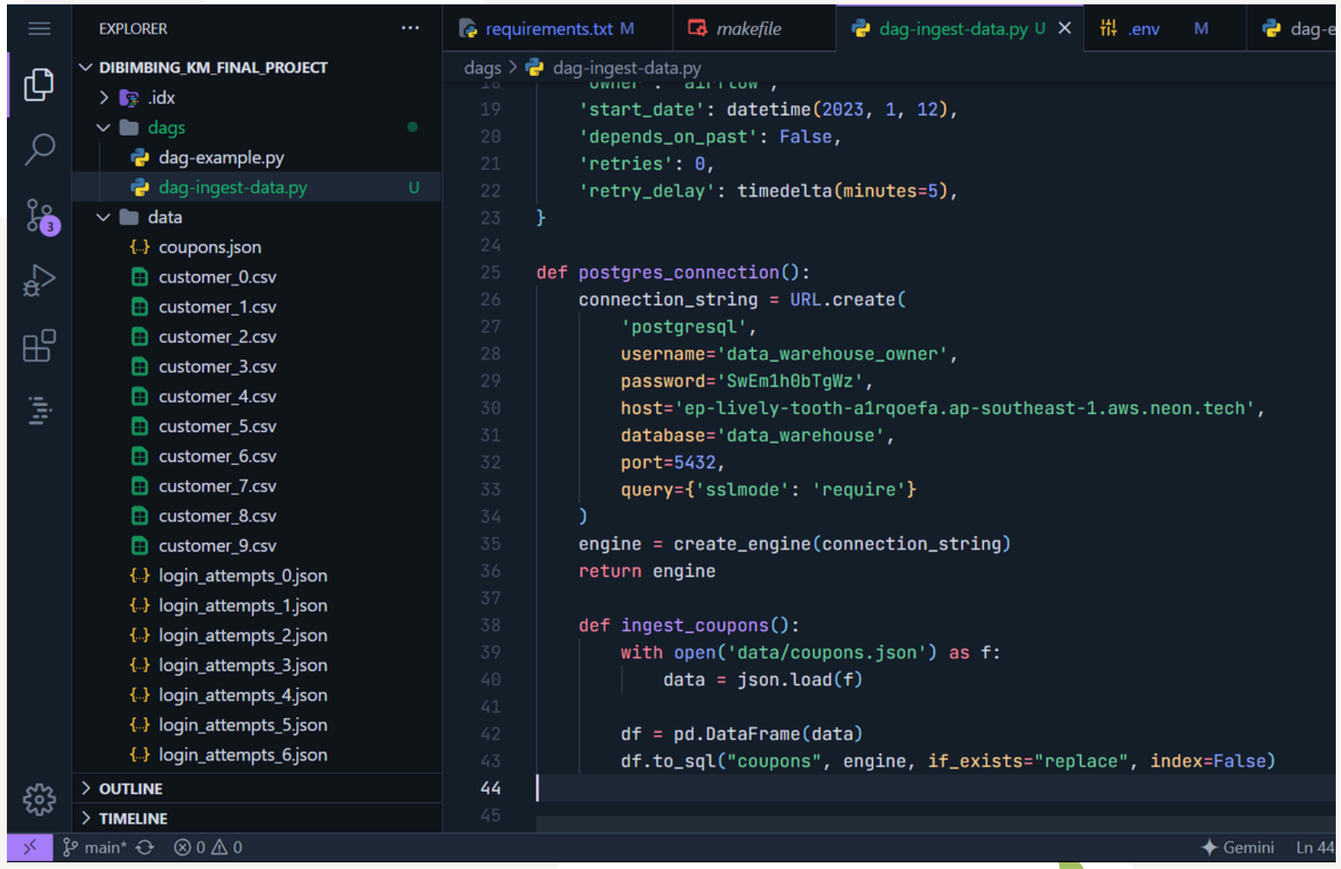
2.

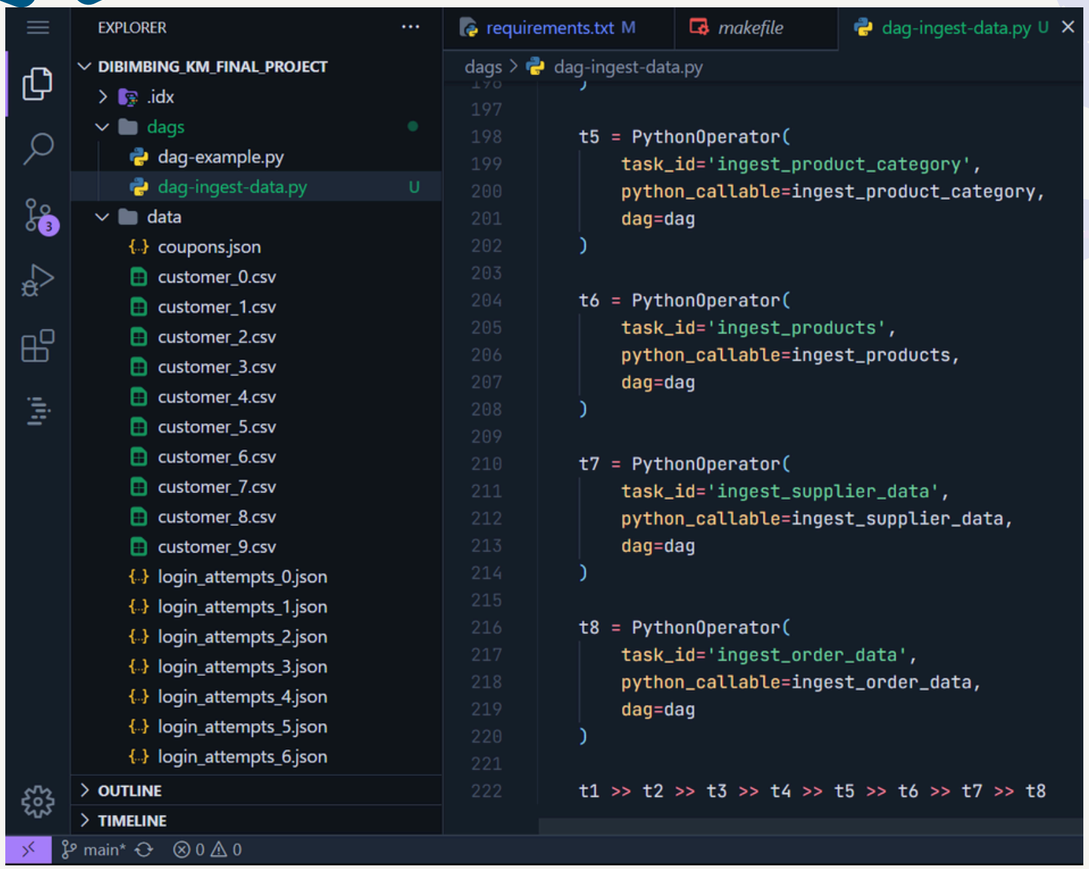
INGEST


dibimbing
ingest_dat
Neon Cons
FINPRO-D
LMS dibim
Dibimbing
dibimbing
ingestion
8081-idx-dibimbing-km-final-project-1718114782923.cluster-7ubberrabzh4qqy2g4z7wgxuw2.cloudworkstations.dev/dags/ingest_data/grid
Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
06:59 UTC
Log In
23/06/2024 06.44.32
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:10:18
00:05:09
00:00:00
ingest_order_item
ingest_data_coupons
ingest_data_customer
ingest_login_attempts
ingest_product_category
ingest_products
ingest_supplier_data
ingest_order_data
DAG
ingest_data
Details
Graph
Gantt
Code
Audit Log
Run Duration
Calendar
DAG Runs Summary
Total Runs Displayed 2
Total success 2
First Run Start 2024-06-23, 06:44:31 UTC
Last Run Start 2024-06-23, 06:46:30 UTC
Max Run Duration 00:10:18
Mean Run Duration 00:09:31
Min Run Duration 00:08:44
DAG Summary

3.

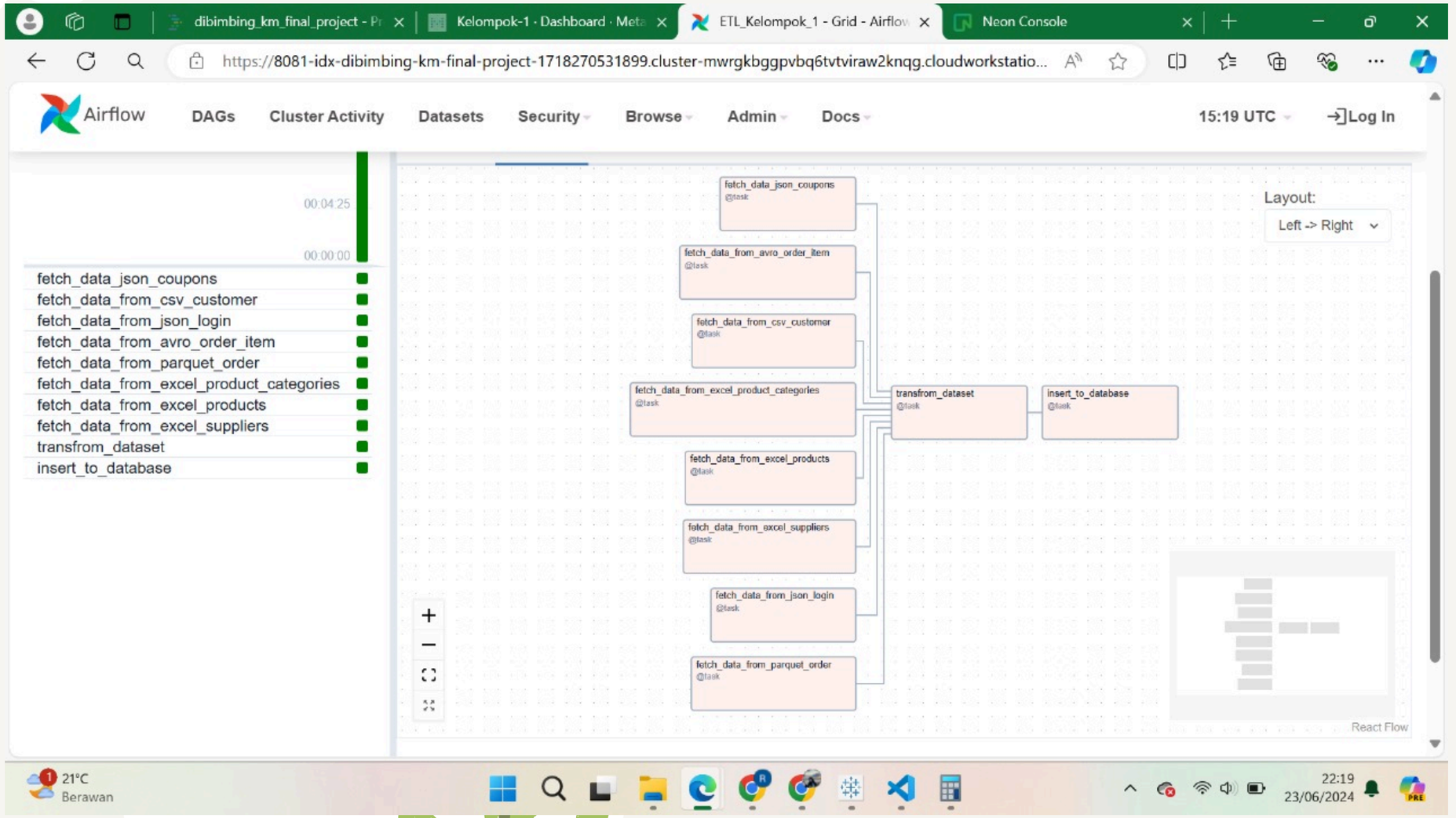
ETL


EXPLORER
DIBIMBING_KM_FINAL_PROJECT
.idx
dags
dag-example.py
dag-ingest-data.py
data
coupons.json
customer_0.csv
customer_1.csv
customer_2.csv
customer_3.csv
customer_4.csv
customer_5.csv
customer_6.csv
customer_7.csv
customer_8.csv
customer_9.csv
login_attempts_0.json
login_attempts_1.json
login_attempts_2.json
login_attempts_3.json
login_attempts_4.json
login_attempts_5.json
login_attempts_6.json
OUTLINE
TIMELINE
requirements.txt M
makefile
dag-ingest-data.py U
.env M
dags > dag-ingest-data.py
'start_date': datetime(2023, 1, 12),
'depends_on_past': False,
'retries': 0,
'retry_delay': timedelta(minutes=5),
}
def postgres_connection():
connection_string = URL.create(
'postgresql',
username='data_warehouse_owner',
password='SwEm1h0bTgWz',
host='ep-lively-tooth-a1rqoefa.ap-southeast-1.aws.neon.tech',
database='data_warehouse',
port=5432,
query={'sslmode': 'require'}
)
engine = create_engine(connection_string)
return engine
def ingest_coupons():
with open('data/coupons.json') as f:
data = json.load(f)
df = pd.DataFrame(data)
df.to_sql("coupons", engine, if_exists="replace", index=False)
main*
Gemini
Ln 44


EXPLORER
DIBIMBING_KM_FINAL_PROJECT
.idx
dags
dag-example.py
dag-ingest-data.py
data
coupons.json
customer_0.csv
customer_1.csv
customer_2.csv
customer_3.csv
customer_4.csv
customer_5.csv
customer_6.csv
customer_7.csv
customer_8.csv
customer_9.csv
login_attempts_0.json
login_attempts_1.json
login_attempts_2.json
login_attempts_3.json
login_attempts_4.json
login_attempts_5.json
login_attempts_6.json
OUTLINE
TIMELINE
requirements.txt M
makefile
dag-ingest-data.py U
dags > dag-ingest-data.py
t5 = PythonOperator(
task_id='ingest_product_category',
python_callable=ingest_product_category,
dag=dag
)
t6 = PythonOperator(
task_id='ingest_products',
python_callable=ingest_products,
dag=dag
)
t7 = PythonOperator(
task_id='ingest_supplier_data',
python_callable=ingest_supplier_data,
dag=dag
)
t8 = PythonOperator(
task_id='ingest_order_data',
python_callable=ingest_order_data,
dag=dag
)
t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7 >> t8
main*

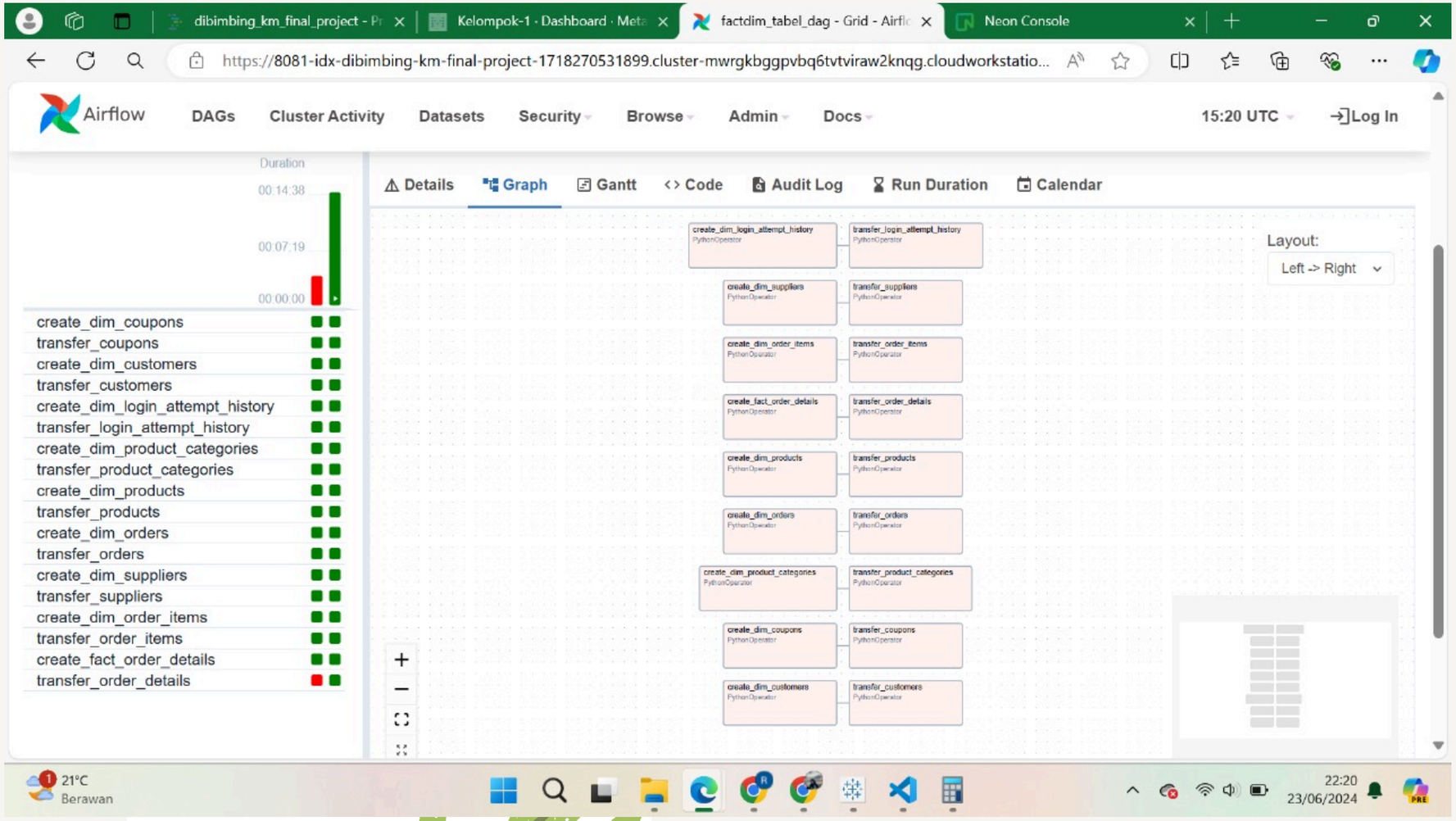
4.

ETL


dibimbing_km_final_project - Pr
Kelompok-1 · Dashboard · Meta
ETL_Kelompok_1 - Grid - Airflow
Neon Console
https://8081-idx-dibimbing-km-final-project-1718270531899.cluster-mwrgkbggpvbq6tvtviraw2knqg.cloudworkstatio...
Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
15:19 UTC
Log In
00:04:25
00:00:00
fetch_data_json_coupons
fetch_data_from_csv_customer
fetch_data_from_json_login
fetch_data_from_avro_order_item
fetch_data_from_parquet_order
fetch_data_from_excel_product_categories
fetch_data_from_excel_products
fetch_data_from_excel_suppliers
transfrom_dataset
insert_to_database
Layout:
Left -> Right
@task
React Flow
21°C
Berawan
22:19
23/06/2024

5.

DATA MODELLING



DATA MODELLING

tabel ETL yang telah berhasil diload, dilakukan transformasi dengan membentuk data modelling yang memiliki fact dan dimension table

dags > dag-factdim.py

```python
def source_postgres_connection():
    connection_string = URL.create(
        'postgresql',
        username='data_warehouse_owner',
        password='SwEm1h0bTgWz',
        host='ep-lively-tooth-a1rqoefa.ap-southeast-1.aws.neon.tech',
        database='data_warehouse',
        port=5432,
        query={'sslmode': 'require'}
    )
    engine = create_engine(connection_string)
    return engine

def target_postgres_connection():
    connection_string = URL.create(
        drivername='postgresql',
        username='data_warehouse_owner',
        password='PQJmnIdjYf02',
        host='ep-noisy-river-a5fcgvg3.us-east-2.aws.neon.tech',
        port=5432,
        database='data_warehouse',
        query={'sslmode': 'require'}
```

```python
def create_dim_customers_table():
    logging.info("Creating dim_customers table in the target database.")
    try:
        engine = target_postgres_connection()
        with engine.connect() as connection:
            connection.execute("""
                CREATE TABLE IF NOT EXISTS dim_customers (
                    customer_id SERIAL PRIMARY KEY,
                    first_name VARCHAR,
                    last_name VARCHAR,
                    gender VARCHAR,
                    address VARCHAR,
                    zip_code VARCHAR
                );
            """)
        logging.info("Table 'dim_customers' has been successfully created in the new database.")
    except Exception as e:
        logging.error(f"Failed to create table 'dim_customers': {e}")
        raise
```

```python
def transfer_customers_data():
    logging.info("Transferring data from customer to dim_customers.")
    try:
        source_engine = source_postgres_connection()
        target_engine = target_postgres_connection()

        # Read data from the source database
        with source_engine.connect() as source_conn:
            query = "SELECT id AS customer_id, first_name, last_name, gender, address, zip_code FROM cu
            customers_df = pd.read_sql(query, source_conn)

        # Write data to the target database
        with target_engine.connect() as target_conn:
            customers_df.to_sql('dim_customers', target_conn, if_exists='replace', index=False)

        logging.info("Data has been successfully transferred from 'customer' to 'dim_customers'.")
    except Exception as e:
        logging.error(f"Failed to transfer data: {e}")
        raise
```

Menambahkan dua tabel baru, yaitu dim_customer_detail dan dim_order_transaction,

```python
# Define the DAG
with DAG(
    dag_id='modelling_dag',
    default_args=default_args,
    schedule_interval='@once',
) as dag:
    create_customer_detail_task = PostgresOperator(
        task_id='create_customer_detail_table',
        sql = """
            CREATE TABLE IF NOT EXISTS dim_customer_detail (
                customer_id INTEGER,
                full_name VARCHAR(255),
                gender VARCHAR(10)
            );
        """,
        postgres_conn_id='postgres_dw',
    )
```

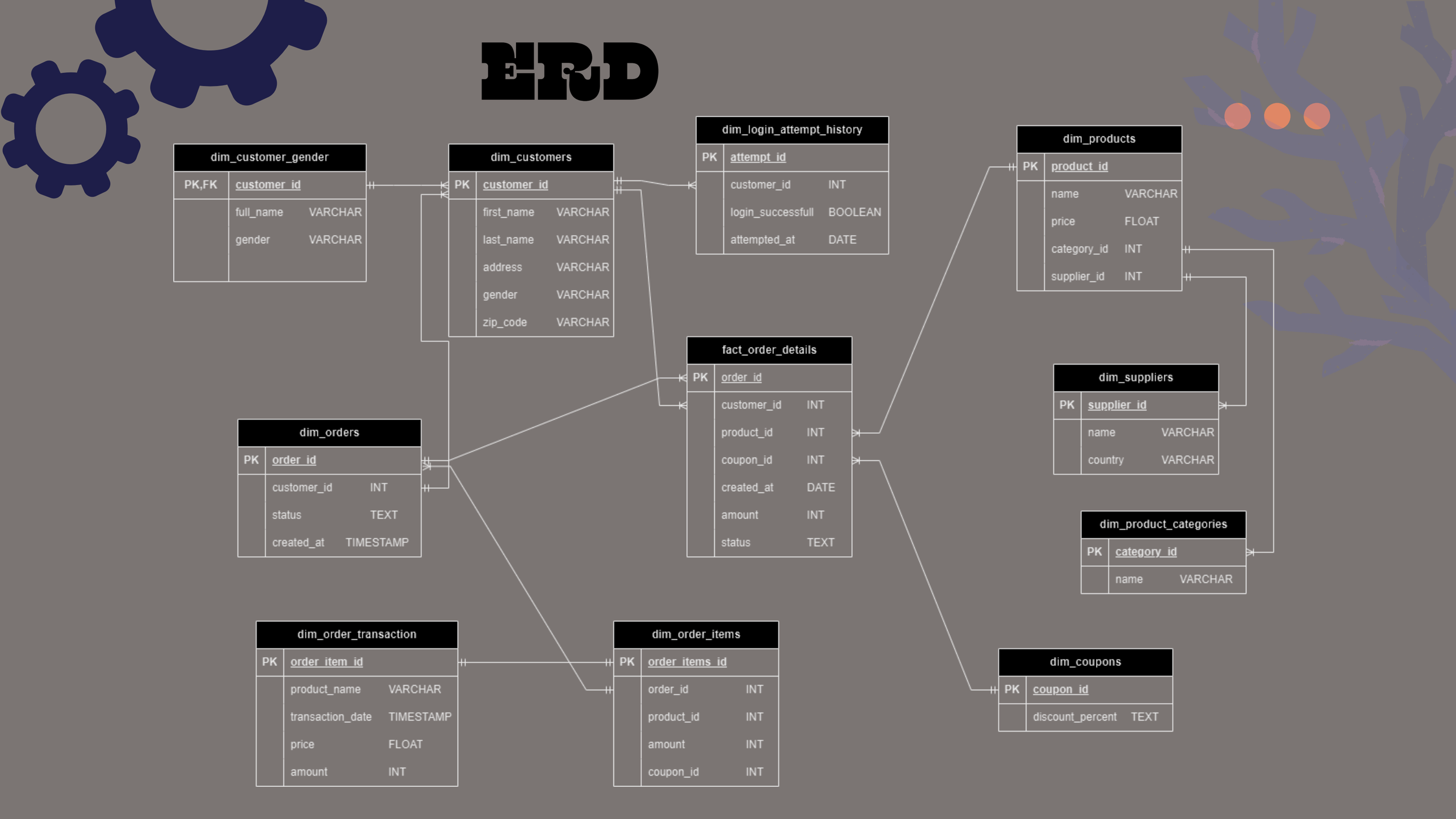
```python
load_customer_detail_task = PostgresOperator(
    task_id='load_customer_detail_table',
    sql="""
        INSERT INTO dim_customer_detail (customer_id, full_name, gender)
        SELECT customer_id, CONCAT(first_name, ' ', last_name) AS full_name,
            CASE
                WHEN gender = 'M' THEN 'Male'
                WHEN gender = 'F' THEN 'Female'
                ELSE NULL
            END AS gender
        FROM dim_customers;
    """,
    postgres_conn_id='postgres_dw',
)
```

```python
create_order_transaction_task = PostgresOperator(
    task_id='create_order_transaction_table',
    sql = """
        CREATE TABLE IF NOT EXISTS dim_order_transaction (
            order_item_id INTEGER,
            transaction_date timestamp,
            product_name VARCHAR(100),
            price FLOAT,
            amount INTEGER
        );
    """,
    postgres_conn_id='postgres_dw',
)
```

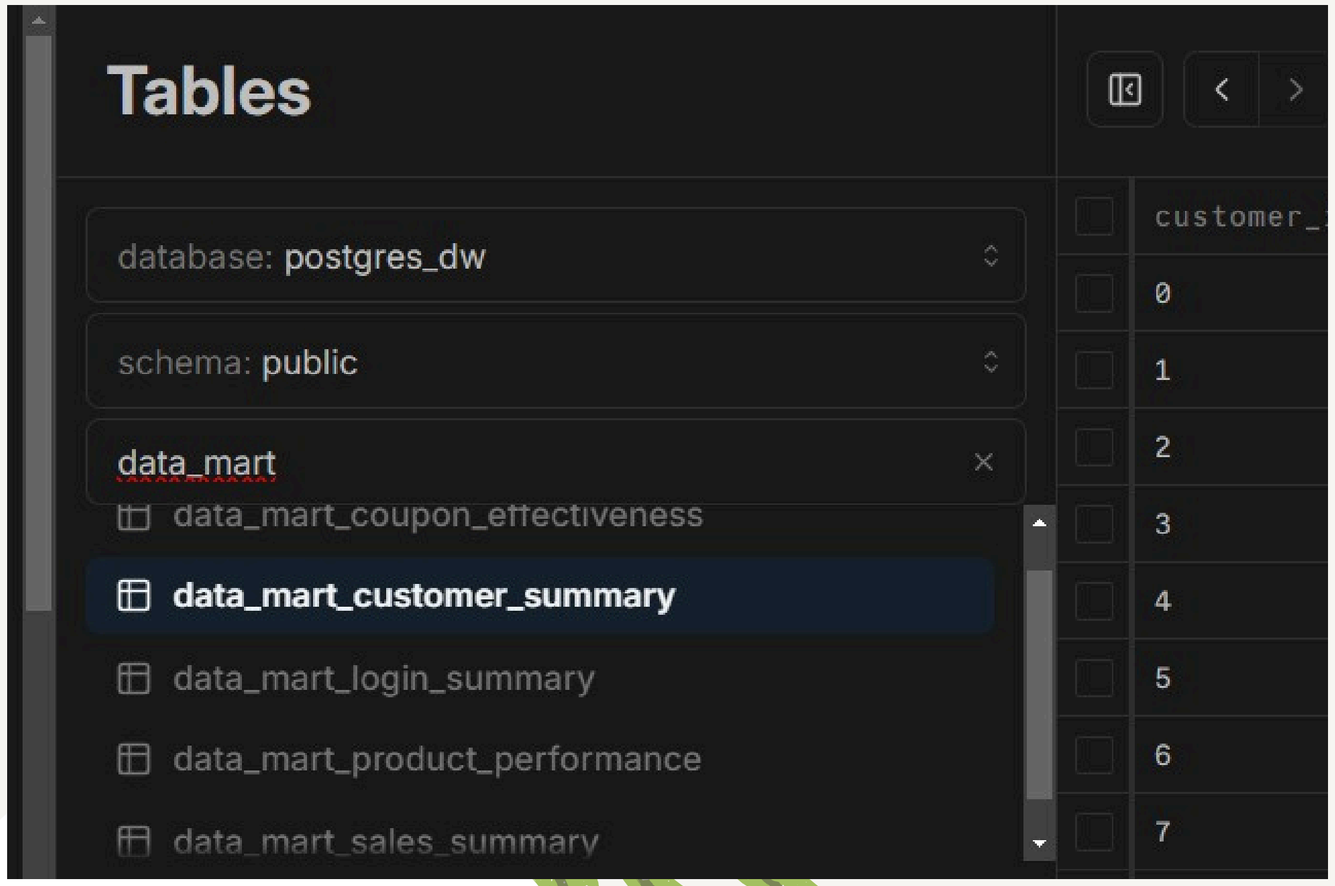
```python
load_order_transaction_task = PostgresOperator(
    task_id='load_order_transaction_table',
    sql="""
        INSERT INTO dim_order_transaction (order_item_id, transaction_date, product_name, price, amount)
        SELECT oi.order_items_id, o.created_at, p.name, p.price, oi.amount
        FROM dim_order_items oi
        JOIN dim_products p ON oi.product_id = p.product_id
        JOIN dim_orders o ON oi.order_id = o.orders_id;
    """,
    postgres_conn_id='postgres_dw',
)
```

sehingga membentuk ERD seperti di selanjutnya...

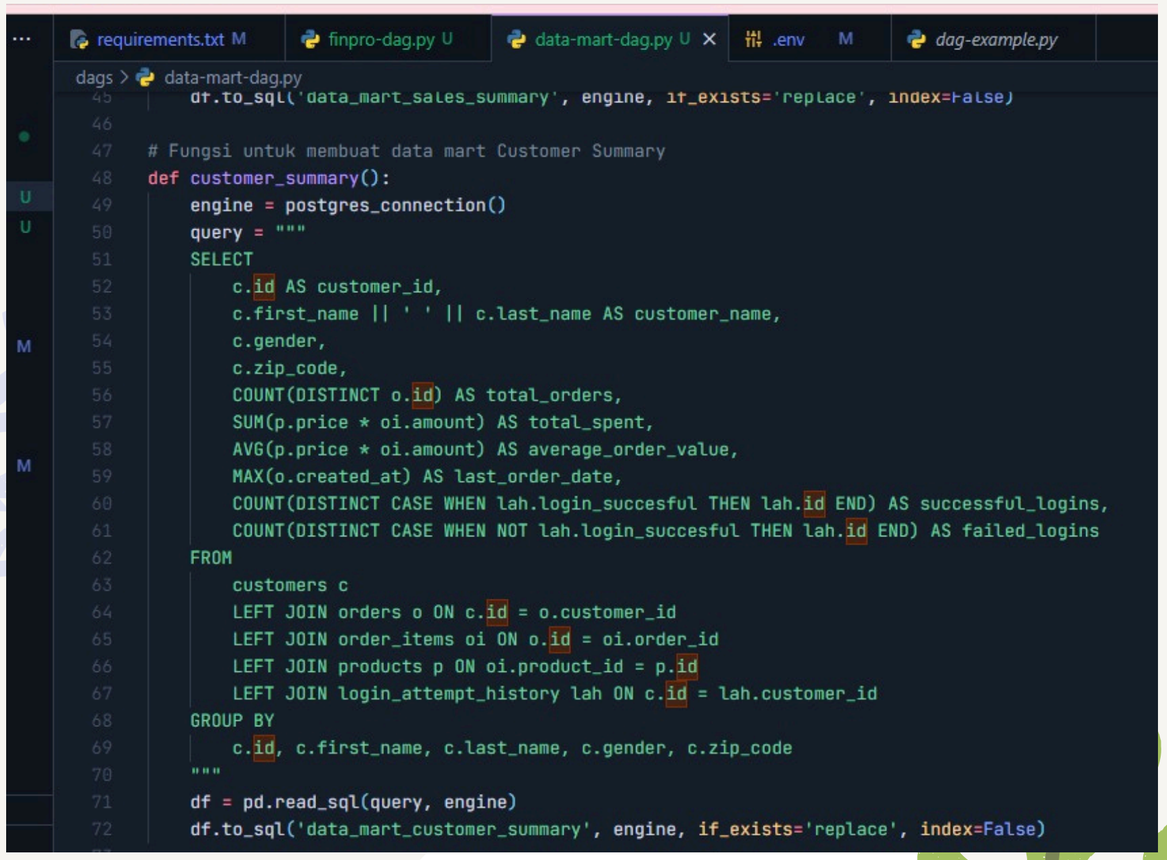
ERD


dim_customer_gender
PK,FK customer_id
full_name VARCHAR
gender VARCHAR
dim_customers
PK customer_id
first_name VARCHAR
last_name VARCHAR
address VARCHAR
gender VARCHAR
zip_code VARCHAR
dim_login_attempt_history
PK attempt_id
customer_id INT
login_successfull BOOLEAN
attempted_at DATE
dim_products
PK product_id
name VARCHAR
price FLOAT
category_id INT
supplier_id INT
fact_order_details
PK order_id
customer_id INT
product_id INT
coupon_id INT
created_at DATE
amount INT
status TEXT
dim_orders
PK order_id
customer_id INT
status TEXT
created_at TIMESTAMP
dim_suppliers
PK supplier_id
name VARCHAR
country VARCHAR
dim_product_categories
PK category_id
name VARCHAR
dim_order_transaction
PK order_item_id
product_name VARCHAR
transaction_date TIMESTAMP
price FLOAT
amount INT
dim_order_items
PK order_items_id
order_id INT
product_id INT
amount INT
coupon_id INT
dim_coupons
PK coupon_id
discount_percent TEXT

DATA MART


Tables
database: postgres_dw
schema: public
data_mart
data_mart_coupon_effectiveness
data_mart_customer_summary
data_mart_login_summary
data_mart_product_performance
data_mart_sales_summary
customer_
0
1
2
3
4
5
6
7

DATA MART



COUNT(DISTINCT o.id): Menghitung jumlah unik order untuk setiap pelanggan.
SUM(p.price * oi.amount): Menghitung total yang dibelanjakan oleh pelanggan.
AVG(p.price * oi.amount): Menghitung rata-rata nilai order.
MAX(o.created_at): Mengambil tanggal order terbaru.
COUNT(DISTINCT CASE WHEN...): Menghitung jumlah login sukses dan gagal.
LEFT JOIN: Digunakan untuk memastikan semua pelanggan termasuk dalam hasil, bahkan jika mereka belum pernah melakukan order.
GROUP BY: Mengelompokkan hasil berdasarkan informasi pelanggan.

DATA MART

requirements.txt M | finpro-dag.py U | data-mart-dag.py U | .env M | dag-example.py

dags > data-mart-dag.py

```python

# Fungsi untuk membuat data mart Sales Summary
def sales_summary():
    engine = postgres_connection()
    query = """
    SELECT
        o.id AS order_id,
        o.created_at AS order_date,
        c.id AS customer_id,
        c.first_name || ' ' || c.last_name AS customer_name,
        p.id AS product_id,
        p.name AS product_name,
        pc.name AS product_category,
        oi.amount AS quantity,
        p.price * oi.amount AS total_price,
        COALESCE(cp.discount_percent, 0) AS discount_percent,
        (p.price * oi.amount) * (1 - COALESCE(cp.discount_percent, 0)/100) AS discounted_price
    FROM
        orders o
        JOIN customers c ON o.customer_id = c.id
        JOIN order_items oi ON o.id = oi.order_id
        JOIN products p ON oi.product_id = p.id
        JOIN product_categories pc ON p.category_id = pc.id
        LEFT JOIN coupons cp ON oi.coupon_id = cp.id
    """
    df = pd.read_sql(query, engine)
    df.to_sql('data_mart_sales_summary', engine, if_exists='replace', index=False)

```

COALESCE(cp.discount_percent, 0): Menggunakan nilai discount_percent jika ada, atau 0 jika tidak ada (NULL).
(p.price * oi.amount) * (1 - COALESCE(cp.discount_percent, 0)/100): Menghitung harga setelah diskon.
JOIN customers c ON o.customer_id = c.id: Menggabungkan tabel customers dengan orders berdasarkan customer_id.
LEFT JOIN coupons cp ON oi.coupon_id = cp.id: Menggunakan LEFT JOIN untuk mengikutsertakan semua order_items, termasuk yang tidak memiliki kupon.

DATA MART

requirements.txt M | finpro-dag.py U | data-mart-dag.py U | .env M | dag-example.py

dags > data-mart-dag.py

```python
# Fungsi untuk membuat data mart Product Performance
def product_performance():
    engine = postgres_connection()
    query = """
    SELECT
        p.id AS product_id,
        p.name AS product_name,
        pc.name AS category_name,
        s.name AS supplier_name,
        s.country AS supplier_country,
        COUNT(DISTINCT oi.order_id) AS total_orders,
        SUM(oi.amount) AS total_quantity_sold,
        SUM(p.price * oi.amount) AS total_revenue,
        AVG(p.price) AS average_price
    FROM
        products p
        JOIN product_categories pc ON p.category_id = pc.id
        JOIN suppliers s ON p.supplier_id = s.id
        LEFT JOIN order_items oi ON p.id = oi.product_id
    GROUP BY
        p.id, p.name, pc.name, s.name, s.country
    """
    df = pd.read_sql(query, engine)
    df.to_sql('data_mart_product_performance', engine, if_exists='replace', index=False)
```

COUNT(DISTINCT oi.order_id): Menghitung jumlah order unik untuk setiap produk.
SUM(oi.amount): Menghitung total kuantitas produk yang terjual.
SUM(p.price * oi.amount): Menghitung total pendapatan dari produk.
AVG(p.price): Menghitung rata-rata harga produk.
LEFT JOIN order_items: Memastikan semua produk termasuk, bahkan jika belum pernah dipesan.

DATA MART

```python

# Fungsi untuk membuat data mart Login Summary
def login_summary():
    engine = postgres_connection()
    query = """
    SELECT
        c.id AS customer_id,
        c.first_name || ' ' || c.last_name AS customer_name,
        COUNT(lah.id) AS total_login_attempts,
        SUM(CASE WHEN lah.login_succesful THEN 1 ELSE 0 END) AS successful_logins,
        SUM(CASE WHEN NOT lah.login_succesful THEN 1 ELSE 0 END) AS failed_logins,
        MAX(lah.attempted_at) AS last_login_attempt
    FROM
        customers c
        LEFT JOIN login_attempt_history lah ON c.id = lah.customer_id
    GROUP BY
        c.id, c.first_name, c.last_name
    """
    df = pd.read_sql(query, engine)
    df.to_sql('data_mart_login_summary', engine, if_exists='replace', index=False)

```

COUNT(lah.id): Menghitung total percobaan login.
SUM(CASE WHEN...): Menghitung jumlah login sukses dan gagal.
MAX(lah.attempted_at): Mengambil waktu percobaan login terakhir.
LEFT JOIN: memastikan bahwa semua pelanggan dari tabel 'customers' dimasukkan ke dalam hasil, bahkan jika mereka tidak memiliki catatan login di tabel 'login_attempt_history'.

DATA MART

```python
# Fungsi untuk membuat data mart Coupon Effectiveness
def coupon_effectiveness():
    engine = postgres_connection()
    query = """
    SELECT
        cp.id AS coupon_id,
        cp.discount_percent,
        COUNT(DISTINCT oi.order_id) AS orders_used,
        COUNT(DISTINCT oi.product_id) AS products_discounted,
        SUM(p.price * oi.amount) AS total_pre_discount_value,
        SUM(p.price * oi.amount * (1 - cp.discount_percent/100)) AS total_post_discount_value,
        SUM(p.price * oi.amount * (cp.discount_percent/100)) AS total_discount_amount
    FROM
        coupons cp
        JOIN order_items oi ON cp.id = oi.coupon_id
        JOIN products p ON oi.product_id = p.id
    GROUP BY
        cp.id, cp.discount_percent
    """
    df = pd.read_sql(query, engine)
    df.to_sql('data_mart_coupon_effectiveness', engine, if_exists='replace', index=False)
```

COUNT(DISTINCT oi.order_id): Menghitung jumlah order yang menggunakan kupon.
COUNT(DISTINCT oi.product_id): Menghitung jumlah produk yang didiskon.
SUM(p.price * oi.amount): Menghitung total nilai sebelum diskon.
SUM(p.price * oi.amount * (1 - cp.discount_percent/100)): Menghitung total nilai setelah diskon.
SUM(p.price * oi.amount * (cp.discount_percent/100)): Menghitung total jumlah diskon yang diberikan.
JOIN: Hanya akan menampilkan kupon yang benar-benar digunakan dalam pesanan yang akan muncul

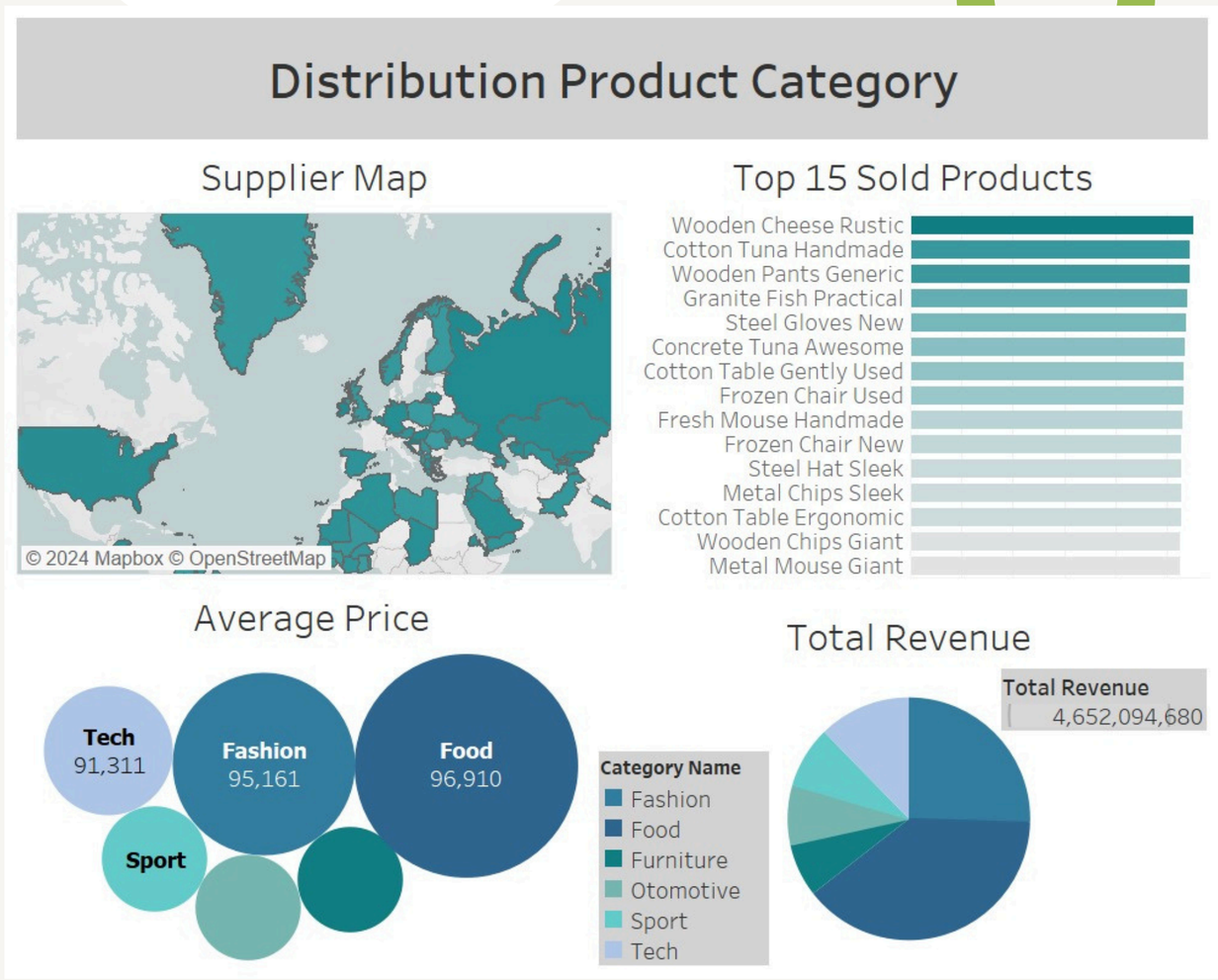
Final project

Distribution Product Category

DASHBOARD

Data dashboard adalah tampilan visual yang menyajikan berbagai jenis data dalam satu tempat secara terpusat

- Supplier map menunjukan distribusi banyak order yang terjadi di setiap negara
- Top 15 Product menunjukan 15 produk yang memiliki penjualan tertinggi
- Average price menunjukan rata-rata harga pada setiap kategori produk
- Total revenue menunjukan besar keseluruhan pendapatan berdasarkan kategori produk



THANK YOU!

FOR YOUR ATTENTION