

Nama : Al Fitra Nur Ramadhani

NIM : 202210370311264

Mata Kuliah : Pemodelan dan Simulasi Data B

---

# Laporan Master Data Warehousing, Dimensional Modeling & ETL Process Sales Data

## 1. Deskripsi

Laporan ini mendokumentasikan implementasi *Master Data Warehousing* dengan pendekatan *Dimensional Modeling* dan proses *Extract, Transform, Load* (ETL) untuk data penjualan. Data sumber berasal dari dua file CSV, yaitu *Fact\_Sales\_1.csv* dan *Fact\_Sales\_2.csv*, yang diimpor ke tabel *public.sales*. Proses ETL mengubah data mentah ini menjadi struktur *data warehouse* yang terdiri dari tabel fakta (*core.sales*) dan tabel dimensi (*core.dim\_payment*, *core.dim\_product*). Selain itu, proses *incremental load* diterapkan untuk efisiensi pembaruan data. Tujuan utama adalah menyediakan data yang terstruktur untuk analisis bisnis, seperti profitabilitas, pola pembayaran, dan penggunaan kartu loyalitas.

Proses ini menggunakan *Docker* untuk mengelola lingkungan, *Python* untuk logika ETL, *PostgreSQL* sebagai DBMS, *PGAdmin 4* untuk manajemen basis data, dan *Visual Studio Code* untuk pengembangan skrip. Laporan ini akan menjelaskan setiap langkah secara rinci, termasuk kondisi *before* dan *after*, sebab-akibat, serta dampak dari pengimporan kedua file CSV.

## 2. Link GitHub = [DATA-MODELING-AND-SIMULATION/Udemy/Docker at master · alfitranurr/DATA-MODELING-AND-SIMULATION](https://github.com/alfitranurr/DATA-MODELING-AND-SIMULATION)

## 3. Tools Yang Digunakan

- **Docker:** Mengelola kontainer untuk *PostgreSQL*, *PGAdmin 4*, dan aplikasi *Python* ETL melalui *docker-compose.yml*.
- **Python 3.11:** Digunakan untuk menjalankan skrip ETL (*etl\_sales.py* untuk *full load* dan *etl\_sales\_incremental.py* untuk *incremental load*). Library *psycopg2* menghubungkan *Python* dengan *PostgreSQL*.
- **PostgreSQL:** DBMS untuk menyimpan data mentah (*public.sales*), data sementara (*staging.sales*), dan *data warehouse* (*core.sales*, *core.dim\_payment*, *core.dim\_product*).
- **PGAdmin 4:** Antarmuka untuk mengelola basis data, menjalankan query SQL, dan memeriksa struktur tabel.
- **Visual Studio Code:** IDE untuk mengembangkan skrip *Python*, mengelola file konfigurasi, dan menjalankan perintah terminal.

## 4. Implementasi Step by Step

### 1. Persiapan Lingkungan

- Konfigurasi Docker:
  - File docker-compose.yml mendefinisikan tiga layanan:
    - postgres: Menjalankan *PostgreSQL* dengan database mydb, pengguna postgres, dan kata sandi \*\*\*\*\*. Data disimpan di volume postgres-data.
    - pgadmin: Menyediakan antarmuka *PGAdmin 4* dengan email alfitranurr@gmail.com dan kata sandi yang sama.
    - python-etl: Menjalankan skrip ETL menggunakan image *Python 3.11*, dengan dependensi diinstall dari requirements.txt.
- Command untuk menjalankan layanan di VSCode:  
'docker-compose up postgres pgadmin'  
  
'docker-compose up python-etl'
- Persiapan Data:  
File CSV (Fact\_Sales\_1.csv) dan (Fact\_Sales\_2.csv) diimpor ke tabel public.sales menggunakan perintah :  
\copy public.sales (transaction\_id, transactional\_date, product\_id, customer\_id, payment, credit\_card, loyalty\_card, cost, quantity, price) FROM '/data/Fact\_Sales\_2.csv'  
DELIMITER ',' CSV HEADER;

### 2. Pembuatan Struktur Database

#### Skema dan Tabel:

- Skema public, staging, dan core dibuat di *PostgreSQL* :

```
CREATE SCHEMA IF NOT EXISTS staging;  
CREATE SCHEMA IF NOT EXISTS core;
```

- Tabel public.sales menyimpan data mentah dengan struktur:

```
CREATE TABLE public.sales  
(  
    transaction_id integer PRIMARY KEY,  
    transactional_date timestamp,  
    product_id character varying,  
    customer_id integer,  
    payment character varying,  
    credit_card bigint,  
    loyalty_card character varying,  
    cost character varying,  
    quantity integer,  
    price numeric  
);
```

- Tabel staging.sales untuk data sementara dengan tipe data yang telah disesuaikan:

```
CREATE TABLE staging.sales
(
    transaction_id integer PRIMARY KEY,
    transactional_date timestamp,
    product_id character varying,
    customer_id integer,
    payment character varying,
    credit_card bigint,
    loyalty_card character varying,
    cost numeric,
    quantity integer,
    price numeric
);
```

- Tabel fakta core.sales dengan kolom tambahan untuk metrik seperti total\_cost, total\_price, dan profit:

```
CREATE TABLE core.sales
(
    transaction_id integer PRIMARY KEY,
    transactional_date timestamp,
    transactional_date_fk bigint,
    product_id character varying,
    product_fk integer,
    customer_id integer,
    payment_fk integer,
    credit_card bigint,
    cost numeric,
    quantity integer,
    price numeric,
    total_cost numeric,
    total_price numeric,
    profit numeric
);
```

- Tabel dimensi core.dim\_product untuk menyimpan ID produk unik:

```
CREATE TABLE core.dim_product (
    product_pk integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    product_id character varying UNIQUE
);
```

- Tabel dimensi core.dim\_payment untuk menyimpan kombinasi metode pembayaran dan status kartu loyalitas:

```
CREATE TABLE core.dim_payment
(
    payment_pk integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    payment character varying,
    loyalty_card character varying,
    PRIMARY KEY (payment_pk)
);
```

### 3. Proses ETL (Full Load) - etl\_sales.py

#### Extract:

- Data dari public.sales dimuat ke staging.sales setelah konversi tipe data (misalnya, cost dari *string* ke *numeric*):

```
# Step 1: Load data into staging.sales
cur.execute("TRUNCATE TABLE staging.sales;")
cur.execute("""
INSERT INTO staging.sales
SELECT
    transaction_id,
    transactional_date,
    product_id,
    customer_id,
    payment,
    credit_card,
    loyalty_card,
    CAST(cost AS numeric),
    quantity,
    price
FROM public.sales;
""")
```

### Transform:

- Memastikan nilai payment yang kosong diisi dengan 'cash' menggunakan COALESCE.
- Mengisi tabel dimensi core.dim\_payment dengan kombinasi unik metode pembayaran dan kartu loyalitas:

```
# Step 2: Populate core.dim_payment
cur.execute("""
INSERT INTO core.dim_payment (payment, loyalty_card)
SELECT DISTINCT COALESCE(payment, 'cash'), loyalty_card
FROM staging.sales s
WHERE NOT EXISTS (
    SELECT 1 FROM core.dim_payment dp
    WHERE dp.payment = COALESCE(s.payment, 'cash') AND dp.loyalty_card = s.loyalty_card
);
""")
```

- Mengisi tabel dimensi core.dim\_product dengan ID produk unik:

```
INSERT INTO core.dim_product (product_id)
SELECT DISTINCT product_id FROM public.sales;
```

- Transformasi data ke core.sales, termasuk perhitungan total\_cost, total\_price, dan profit:

```
# Step 3: Transform and load into core.sales
cur.execute("""
INSERT INTO core.sales (
    transaction_id, transactional_date, transactional_date_fk, product_id, product_fk,
    customer_id, payment_fk, credit_card, cost, quantity, price, total_cost, total_price, profit
)
SELECT
    s.transaction_id,
    s.transactional_date,
    EXTRACT(YEAR FROM s.transactional_date)::bigint * 10000 +
    EXTRACT(MONTH FROM s.transactional_date) * 100 +
    EXTRACT(DAY FROM s.transactional_date) AS transactional_date_fk,
    s.product_id,
    p.product_pk AS product_fk,
    s.customer_id,
    dp.payment_pk AS payment_fk,
    s.credit_card,
    s.cost,
    s.quantity,
    s.price,
    s.cost * s.quantity AS total_cost,
    s.price * s.quantity AS total_price,
    (s.price * s.quantity) - (s.cost * s.quantity) AS profit
FROM staging.sales s
LEFT JOIN core.dim_product p ON s.product_id = p.product_id
LEFT JOIN core.dim_payment dp ON COALESCE(s.payment, 'cash') = dp.payment AND s.loyalty_card = dp.loyalty_card
ON CONFLICT (transaction_id) DO NOTHING;
""")
```

#### Load:

- Data yang telah ditransformasi dimuat ke tabel fakta core.sales dengan kunci asing (product\_fk, payment\_fk) yang sesuai.

#### 4. Proses ETL (Incremental Load) - etl\_sales\_incremental.py

##### Extract:

- Memuat hanya data baru berdasarkan last\_load\_date dari tabel etl\_metadata:

```
# Load new data into staging.sales
cur.execute("""
INSERT INTO staging.sales
SELECT
    transaction_id,
    transactional_date,
    product_id,
    customer_id,
    payment,
    credit_card,
    loyalty_card,
    CAST(cost AS numeric),
    quantity,
    price
FROM public.sales p
WHERE p.transactional_date > %s
ON CONFLICT (transaction_id) DO NOTHING;
""", (last_load_date,))
```

##### Transform:

- Memperbarui core.dim\_payment dengan kombinasi baru:

```
# Update dim_payment with new combinations
cur.execute("""
INSERT INTO core.dim_payment (payment, loyalty_card)
SELECT DISTINCT COALESCE(payment, 'cash'), loyalty_card
FROM staging.sales s
WHERE NOT EXISTS (
    SELECT 1 FROM core.dim_payment dp
    WHERE dp.payment = COALESCE(s.payment, 'cash') AND dp.loyalty_card = s.loyalty_card
);
""")
```

- Transformasi data baru ke core.sales dengan logika yang sama seperti full load, tetapi hanya untuk data dengan transactional\_date > last\_load\_date.

##### Load:

- Memuat data baru ke core.sales dan memperbarui last\_load\_date di etl\_metadata:

```
# Update last_load_date
cur.execute("SELECT MAX(transactional_date) FROM staging.sales WHERE transactional_date > %s", (last_load_date,))
new_last_load_date = cur.fetchone()[0]
if new_last_load_date:
    cur.execute("UPDATE etl_metadata SET last_load_date = %s WHERE table_name = 'sales'", (new_last_load_date,))
```

#### 5. Verifikasi Data

##### Memeriksa Tabel Dimensi:

- Query SELECT \* FROM core.dim\_payment; digunakan untuk memverifikasi isi tabel core.dim\_payment. Contoh hasil berdasarkan dim\_payment.csv:

	payment_pk [PK] integer	payment character varying	loyalty_card character varying
1	1	visa	F
2	2	cash	T
3	3	americanexpress	T
4	4	mastercard	T
5	5	visa	T
6	6	mastercard	F
7	7	cash	F
8	8	americanexpress	F

Penjelasan: Tabel ini berisi kombinasi unik metode pembayaran dan status kartu loyalitas. Kolom payment\_pk adalah kunci utama yang dihasilkan secara otomatis, digunakan sebagai kunci asing di core.sales.

#### Memeriksa Tabel Fakta:

- Query `SELECT * FROM core.sales WHERE transactional_date >= '2022-05-01' LIMIT 5;` digunakan untuk memeriksa data di core.sales. Contoh hasil setelah proses ETL:

	transactional_date_fk bigint	product_id character varying	product_fk integer	customer_id integer	payment_fk integer	credit_card bigint	cost numeric	quantity integer	price numeric	total_cost numeric	total_price numeric	profit numeric
1	20220501	P0305	2386	6	7	30271790522719	8.74	3	9.99	26.22	29.97	3.75
2	20220501	P0242	2328	7	1	4041591026711540	1.6	2	1.79	3.2	3.58	0.38
3	20220501	P0529	2595	2	6	5048373491517664	17.59	2	18.79	35.18	37.58	2.4
4	20220501	P0336	2412	7	4	5048377130633352	8.17	4	8.89	32.68	35.56	2.88
5	20220501	P0399	2471	6	1	4041595611872	9.83	2	11.29	19.66	22.58	2.92

Penjelasan: Tabel core.sales berisi data transaksi yang telah ditransformasi, dengan kolom seperti transactional\_date\_fk (format YYYYMMDD untuk analisis waktu), product\_fk (kunci asing ke core.dim\_product), payment\_fk (kunci asing ke core.dim\_payment), serta metrik total\_cost, total\_price, dan profit.

## 5. Kesimpulan

- Proses ETL berhasil mengubah data mentah dari public.sales menjadi struktur *data warehouse* yang terdiri dari tabel fakta (core.sales) dan tabel dimensi (core.dim\_payment, core.dim\_product).
- Pendekatan *dimensional modeling* memungkinkan analisis data yang efisien dengan pemisahan data transaksional dan atribut (metode pembayaran, produk).
- Proses *incremental load* meningkatkan efisiensi dengan hanya memproses data baru, cocok untuk pembaruan data secara berkala.
- Penggunaan *Docker* memastikan lingkungan yang konsisten dan portabel, sementara *PostgreSQL* dan *Python* mendukung skalabilitas dan fleksibilitas proses ETL.