ETL Process with Stored Procedure

Efficient Data Transfer and Loading

Hijir Della Wirasti

Business Intelligence
Dibimbing Batch 13

Github: https://github.com/hijirdella/ETL-Data-Warehouse-Generate-Sales

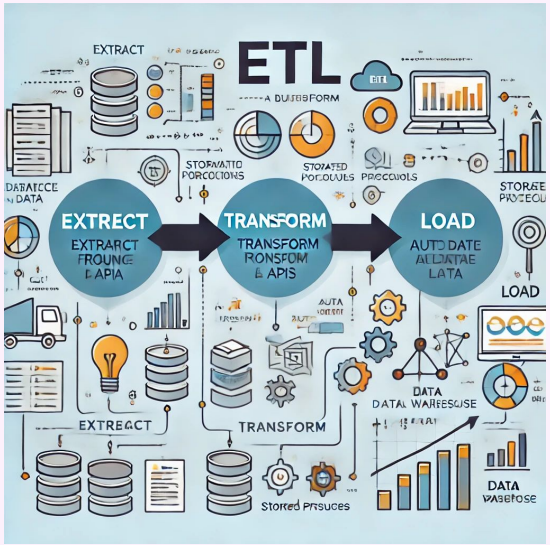
https://www.linkedin.com/in/hijirdella/

Table of contents

02

Introduction

Introduction to the ETL Process



ETL (Extract, Transform, Load) is a crucial process in data management that involves:

- **Extracting** raw data from various sources such as databases, APIs, or flat files.
- **Transforming** the data into a suitable format or structure for analysis, applying cleaning, aggregation, or enrichment.
- **Loading** the transformed data into a target system, such as a data warehouse, for reporting and analysis.

Purpose:

- To ensure data consistency, accuracy, and availability for decision-making.
- To integrate disparate data sources into a unified system for better insights.

The goal

Definition: A stored procedure is a precompiled collection of SQL statements that can be executed as a single unit.
Why Use Stored Procedures for ETL?:

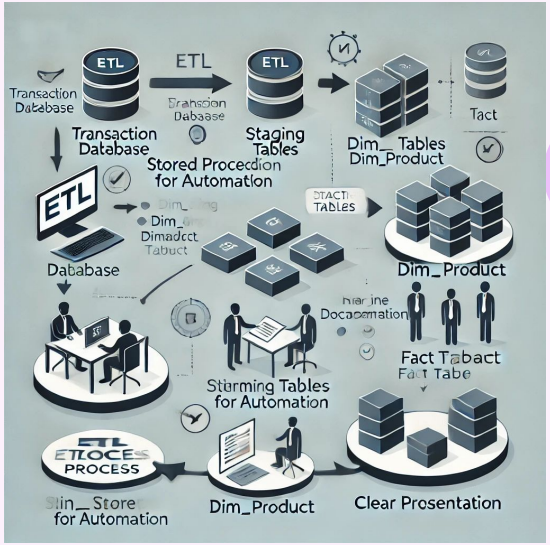
- **Automation**: Automates repetitive ETL tasks, such as data validation, transformation, and loading.
- **Efficiency**: Reduces processing time by running directly within the database engine.
- **Scalability**: Handles large volumes of data efficiently, making it ideal for enterprise-level ETL operations.
- **Error Handling**: Includes mechanisms for logging and handling errors during data processing.

03

Task Overview

Brief Overview of the Task

- **Objective**:
  - Design a stored procedure named `dwh.generate_sales()` to automate ETL processes.
- **Scope**:
  - Move data from a transactional database to staging tables.
  - Load data from staging tables into dimensional tables in the data warehouse.
  - Populate fact tables with processed data for analytical purposes.



Stored Procedure: dwh.generate_sales()
Goal: Automate the ETL (Extract, Transform, Load) process to move data from source tables to staging tables as the first step in the ETL pipeline.

Assignment Requirements

- **ETL Process**:
  a. Extract data from transaction tables and load it into staging tables.
  b. Transform and organize the data into dimension tables (`dim_store`, `dim_product`, etc.).
  c. Aggregate and load transformed data into a fact table (`fact_sales_transaction`).
- **Stored Procedure**:
  a. Name: `dwh.generate_sales()`
  b. Automate the ETL steps using SQL in PostgreSQL.
  c. Include clear in-line documentation within the procedure to explain each step.
- **Presentation**:
  a. Create a simple presentation that explains the steps you took to design the procedure.
  b. Highlight your approach to solving the problem and the significance of each step in the ETL process.

Schema Creation Flow

Staging

Area sementara untuk menyimpan data yang diekstraksi dari berbagai sumber sebelum diproses lebih lanjut.

Data Warehouse

Untuk menyimpan data yang telah diproses, diorganisasi, dan disusun untuk keperluan analisis jangka panjang.

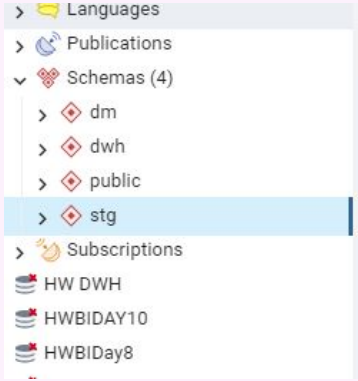
Data Mart

Subset dari Data Warehouse yang berfokus pada kebutuhan analisis spesifik, seperti penjualan, pemasaran, atau keuangan.

Public

Skema default dalam PostgreSQL yang biasanya digunakan untuk menyimpan data umum atau tabel yang bersifat publik

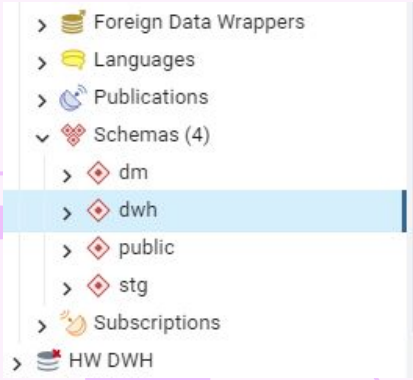
Schema – stg




```sql
-- Check if the schema exists, if not, create it
DO
$$
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM information_schema.schemata
        WHERE schema_name = 'stg'
    ) THEN
        CREATE SCHEMA stg;
    END IF;
END
$$;
```

Create stg schema

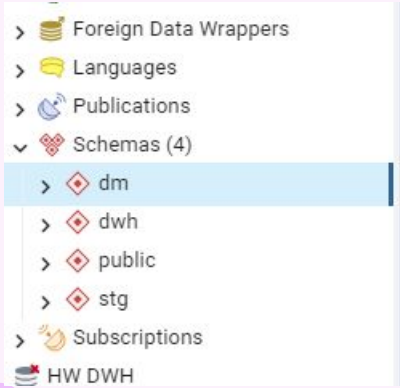
Schema – dwh




```sql
DO
$$
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM information_schema.schemata
        WHERE schema_name = 'dwh'
    ) THEN
        CREATE SCHEMA dwh;
    END IF;
END
$$;
```

Create dwh schema

Schema – dm




```
28  DO
29  $$
30  BEGIN
31      IF NOT EXISTS (
32          SELECT 1
33          FROM information_schema.schemata
34          WHERE schema_name = 'dm'
35      ) THEN
36          CREATE SCHEMA dm;
37      END IF;
38  END
39  $$;
```

Membuat skema ‘dm’ (data mart)

Schema – public

public scheme is default

Table Creation Flow

1. Public

sales_transaction

2. STG

stg_sales_transaction

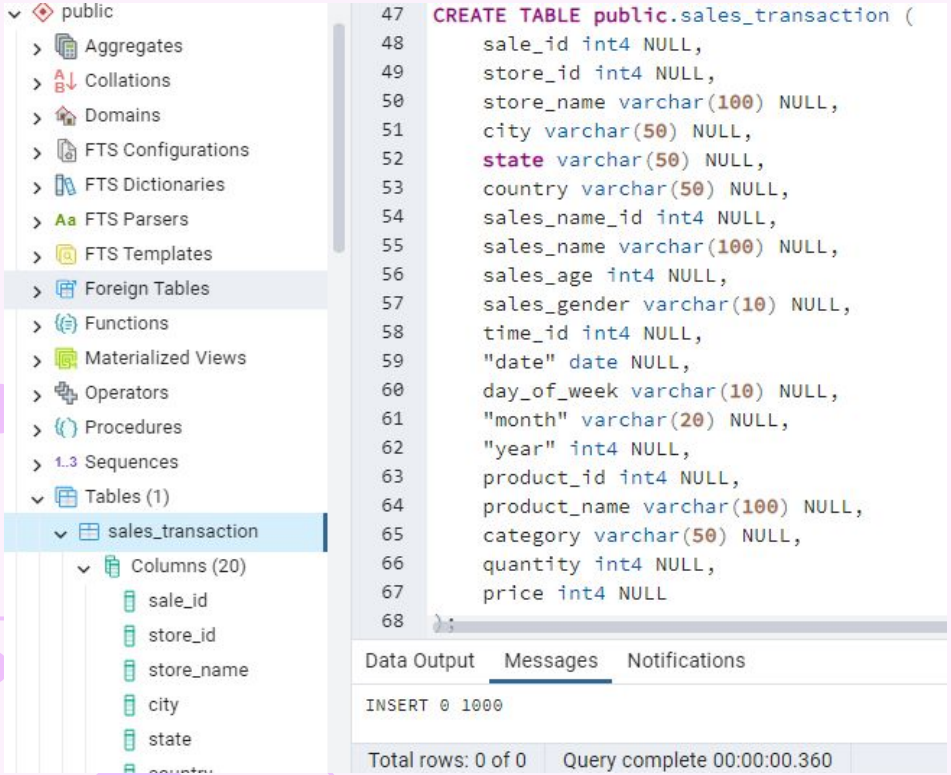
3. DWH

dim_product
dim_store
dim_time
dim_sales_name
fact_sales_transaction

4. DM

dm_sales_by_store
dm_sales_by_product
dm_sales_by_time
Dm_sales_by_salesperson
dm_sales_summary

Schema – public



Create a sales_transaction table in public

Functions as transaction data / OLTP is a public scheme

Schema – public

```
1  SELECT * FROM public.sales_transaction
```

Data Output | Messages | Notifications

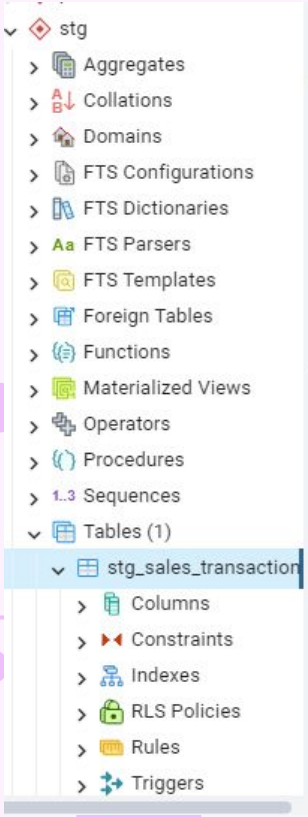
	sale_id integer	store_id integer	store_name character varying (100)	city character varying (50)	state character varying (50)	country character varying (50)	sales_name_id integer	sales_name character varying (100)	sales_age integer	sales_ge characte
1	13	3	Starbucks Corner	Chicago	IL	USA	2	Jane Smith	28	Female
2	12	2	Downtown Starbucks	Los Angeles	CA	USA	7	David Martinez	38	Male
3	1	2	Downtown Starbucks	Los Angeles	CA	USA	4	Emily Brown	32	Female
4	2	3	Starbucks Corner	Chicago	IL	USA	9	James Anderson	41	Male
5	3	1	Starbucks Central	New York	NY	USA	9	James Anderson	41	Male
6	4	1	Starbucks Central	New York	NY	USA	10	Olivia Taylor	34	Female
7	5	2	Downtown Starbucks	Los Angeles	CA	USA	2	Jane Smith	28	Female
8	6	2	Downtown Starbucks	Los Angeles	CA	USA	3	Michael Johnson	45	Male
9	7	2	Downtown Starbucks	Los Angeles	CA	USA	1	John Doe	35	Male
10	8	1	Starbucks Central	New York	NY	USA	2	Jane Smith	28	Female
11	9	2	Downtown Starbucks	Los Angeles	CA	USA	7	David Martinez	38	Male
12	10	1	Starbucks Central	New York	NY	USA	1	John Doe	35	Male
13	11	1	Starbucks Central	New York	NY	USA	10	Olivia Taylor	34	Female
14	14	3	Starbucks Corner	Chicago	IL	USA	3	Michael Johnson	45	Male
15	15	1	Starbucks Central	New York	NY	USA	10	Olivia Taylor	34	Female

Total rows: 1000 of 1000 | Query complete 00:00:00.641 | Ln 1, Col 39

Table sales_transactions

Create table dan isi tabel berhasil dilakukan
Ada 1000 row dan 39 column

Schema – stg




```sql
-- DROP TABLE stg.stg_sales_transaction;

CREATE TABLE stg.stg_sales_transaction (
    sale_id int4 NULL,
    store_id int4 NULL,
    store_name varchar(100) NULL,
    city varchar(50) NULL,
    state varchar(50) NULL,
    country varchar(50) NULL,
    sales_name_id int4 NULL,
    sales_name varchar(100) NULL,
    sales_age int4 NULL,
    sales_gender varchar(10) NULL,
    time_id int4 NULL,
    "date" date NULL,
    day_of_week varchar(10) NULL,
    "month" varchar(20) NULL,
    "year" int4 NULL,
    product_id int4 NULL,
    product_name varchar(100) NULL,
    category varchar(50) NULL,
    quantity int4 NULL,
    price int4 NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Total rows: 0 of 0 | Query complete 00:00:00.189

Create Table
stg_sales_transactions

Schema – stg

```
INSERT INTO stg.stg_sales_transaction (sale_id,store_id,store_name,city,state,country,sales_name_id,sales_name,sales_age,sales_gender,tim
    (13,3,'Starbucks Corner','Chicago','IL','USA',2,'Jane Smith',28,'Female',7,'2024-01-07','Sunday','January',2024,10,'Mocha','Coffee',
    (12,2,'Downtown Starbucks','Los Angeles','CA','USA',7,'David Martinez',38,'Male',31,'2024-01-31','Wednesday','January',2024,5,'Crois
    (1,2,'Downtown Starbucks','Los Angeles','CA','USA',4,'Emily Brown',32,'Female',4,'2024-01-04','Thursday','January',2024,3,'Iced Gree
    (2,3,'Starbucks Corner','Chicago','IL','USA',9,'James Anderson',41,'Male',22,'2024-01-22','Monday','January',2024,7,'Blueberry Muffi
    (3,1,'Starbucks Central','New York','NY','USA',9,'James Anderson',41,'Male',30,'2024-01-30','Tuesday','January',2024,9,'Flat White',
    (4,1,'Starbucks Central','New York','NY','USA',10,'Olivia Taylor',34,'Female',18,'2024-01-18','Thursday','January',2024,6,'Pumpkin S
    (5,2,'Downtown Starbucks','Los Angeles','CA','USA',2,'Jane Smith',28,'Female',27,'2024-01-27','Saturday','January',2024,6,'Pumpkin S
    (6,2,'Downtown Starbucks','Los Angeles','CA','USA',3,'Michael Johnson',45,'Male',22,'2024-01-22','Monday','January',2024,8,'Chai Tea
    (7,2,'Downtown Starbucks','Los Angeles','CA','USA',1,'John Doe',35,'Male',25,'2024-01-25','Thursday','January',2024,8,'Chai Tea Latt
    (8,1,'Starbucks Central','New York','NY','USA',2,'Jane Smith',28,'Female',27,'2024-01-27','Saturday','January',2024,2,'Caramel Macch
    (9,2,'Downtown Starbucks','Los Angeles','CA','USA',7,'David Martinez',38,'Male',5,'2024-01-05','Friday','January',2024,10,'Mocha','C
```

Query | Query History

```
SELECT * FROM stg.stg_sales_transaction
```

Data Output | Messages | Notifications

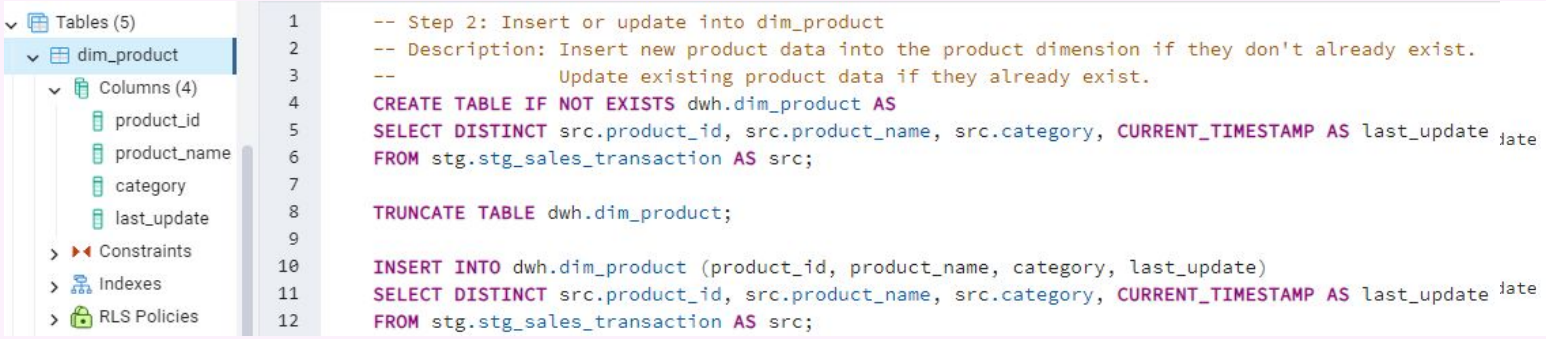
	sale_id integer	store_id integer	store_name character varying (100)	city character varying (50)	state character varying (50)	country character varying (50)	sales_name_id integer	sales_name character varying (100)	sales_age integer	sales_gend character v
1	13	3	Starbucks Corner	Chicago	IL	USA	2	Jane Smith	28	Female
2	12	2	Downtown Starbucks	Los Angeles	CA	USA	7	David Martinez	38	Male
3	1	2	Downtown Starbucks	Los Angeles	CA	USA	4	Emily Brown	32	Female
4	2	3	Starbucks Corner	Chicago	IL	USA	9	James Anderson	41	Male
5	3	1	Starbucks Central	New York	NY	USA	9	James Anderson	41	Male
6	4	1	Starbucks Central	New York	NY	USA	10	Olivia Taylor	34	Female
7	5	2	Downtown Starbucks	Los Angeles	CA	USA	2	Jane Smith	28	Female
8	6	2	Downtown Starbucks	Los Angeles	CA	USA	3	Michael Johnson	45	Male
9	7	2	Downtown Starbucks	Los Angeles	CA	USA	1	John Doe	35	Male
10	8	1	Starbucks Central	New York	NY	USA	2	Jane Smith	28	Female
11	9	2	Downtown Starbucks	Los Angeles	CA	USA	7	David Martinez	38	Male
12	10	1	Starbucks Central	New York	NY	USA	1	John Doe	35	Male
13	11	1	Starbucks Central	New York	NY	USA	10	Olivia Taylor	34	Female
14	14	3	Starbucks Corner	Chicago	IL	USA	3	Michael Johnson	45	Male

Total rows: 1000 of 1000 | Query complete 00:00:00.959 | Ln 1, Col 1

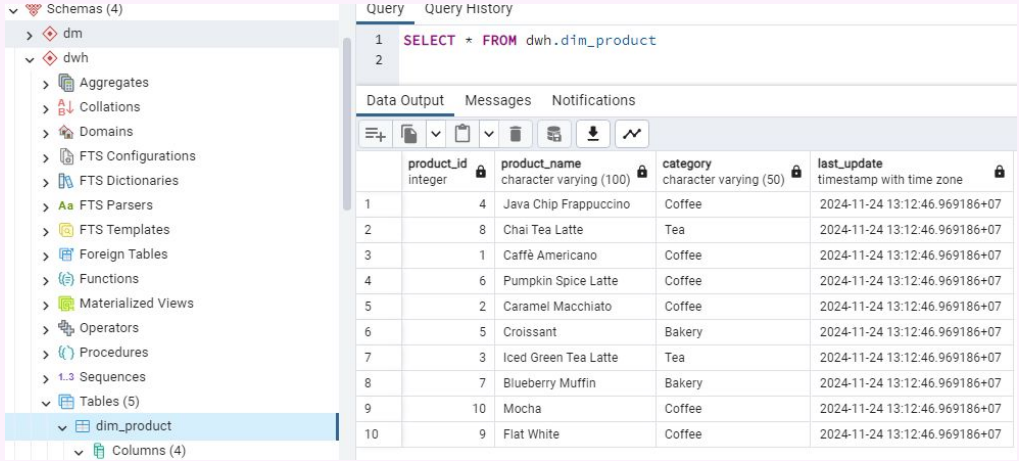
Table stg_sales_transactions

Create table dan isi tabel berhasil dilakukan
Ada 1000 row

Schema – dwh – dim_product

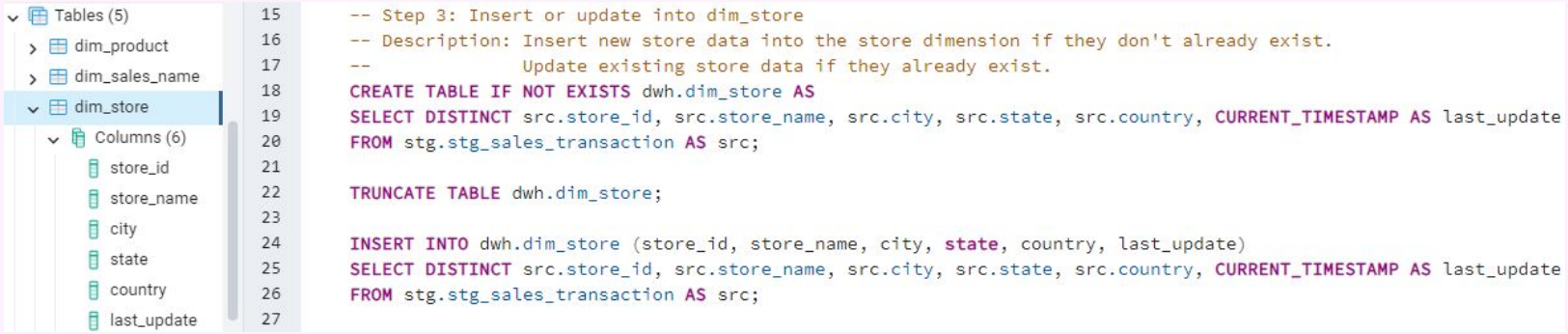


Create table dim_product berhasil

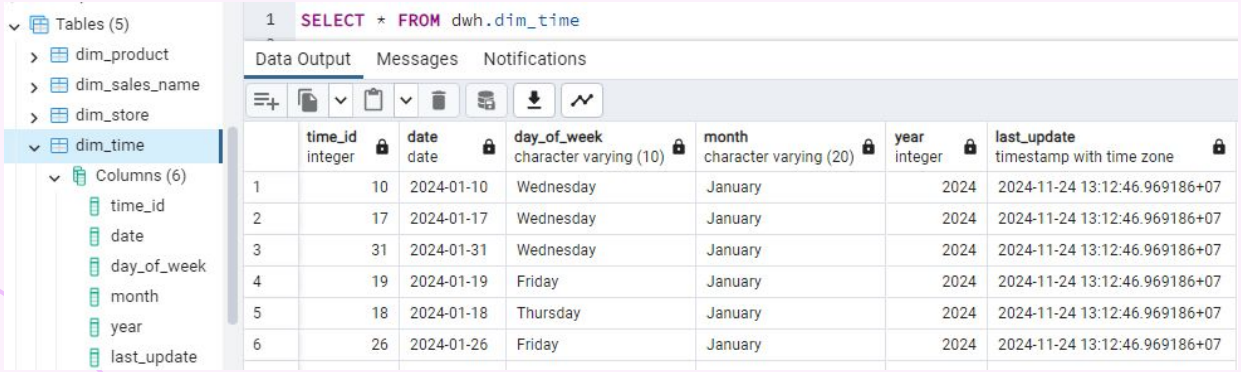


	product_id integer	product_name character varying (100)	category character varying (50)	last_update timestamp with time zone
1	4	Java Chip Frappuccino	Coffee	2024-11-24 13:12:46.969186+07
2	8	Chai Tea Latte	Tea	2024-11-24 13:12:46.969186+07
3	1	Caffè Americano	Coffee	2024-11-24 13:12:46.969186+07
4	6	Pumpkin Spice Latte	Coffee	2024-11-24 13:12:46.969186+07
5	2	Caramel Macchiato	Coffee	2024-11-24 13:12:46.969186+07
6	5	Croissant	Bakery	2024-11-24 13:12:46.969186+07
7	3	Iced Green Tea Latte	Tea	2024-11-24 13:12:46.969186+07
8	7	Blueberry Muffin	Bakery	2024-11-24 13:12:46.969186+07
9	10	Mocha	Coffee	2024-11-24 13:12:46.969186+07
10	9	Flat White	Coffee	2024-11-24 13:12:46.969186+07

Schema – dwh – dim_store

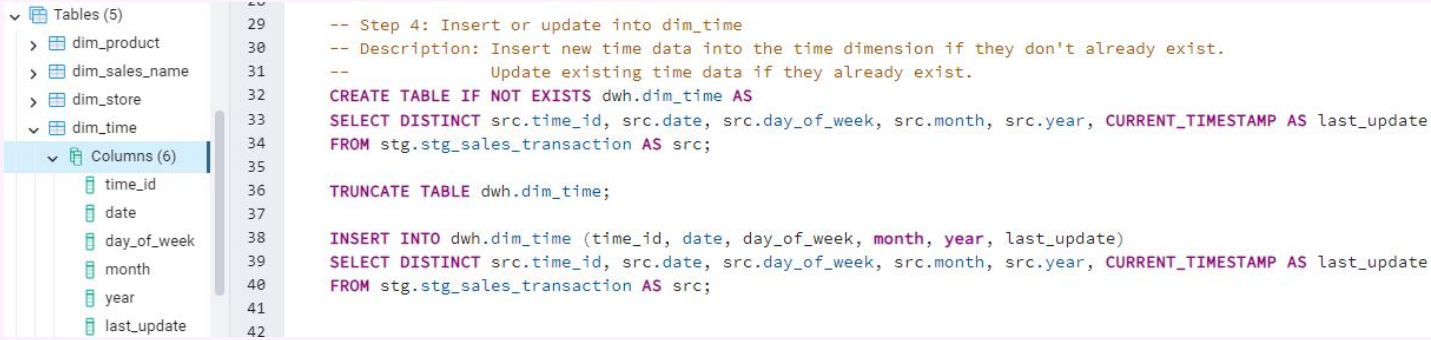


Create table dim_store berhasil

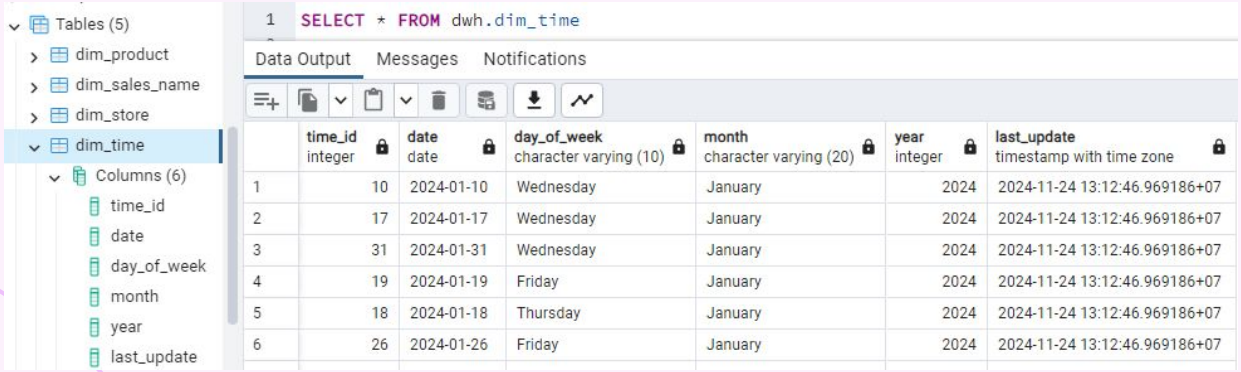


	time_id integer	date date	day_of_week character varying (10)	month character varying (20)	year integer	last_update timestamp with time zone
1	10	2024-01-10	Wednesday	January	2024	2024-11-24 13:12:46.969186+07
2	17	2024-01-17	Wednesday	January	2024	2024-11-24 13:12:46.969186+07
3	31	2024-01-31	Wednesday	January	2024	2024-11-24 13:12:46.969186+07
4	19	2024-01-19	Friday	January	2024	2024-11-24 13:12:46.969186+07
5	18	2024-01-18	Thursday	January	2024	2024-11-24 13:12:46.969186+07
6	26	2024-01-26	Friday	January	2024	2024-11-24 13:12:46.969186+07

Schema – dwh – dim_time

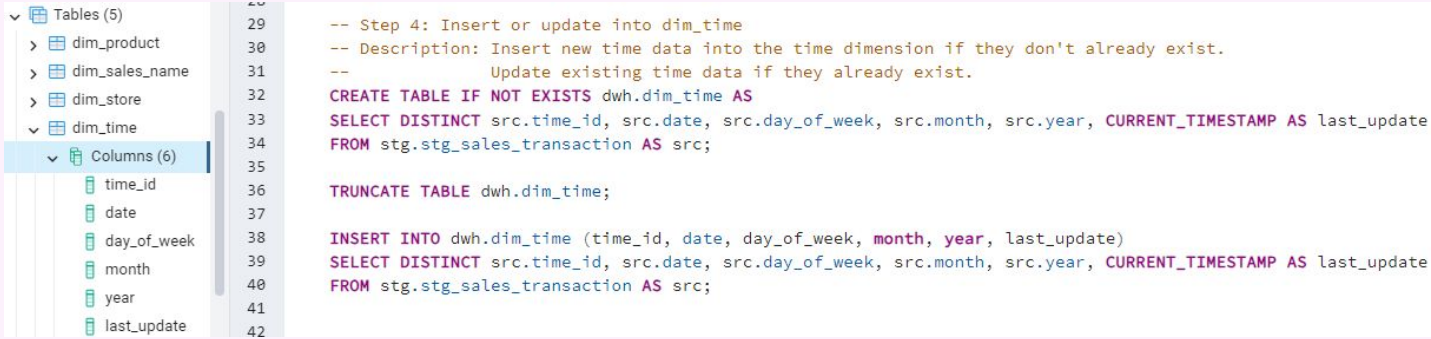


Create table dim_store berhasil

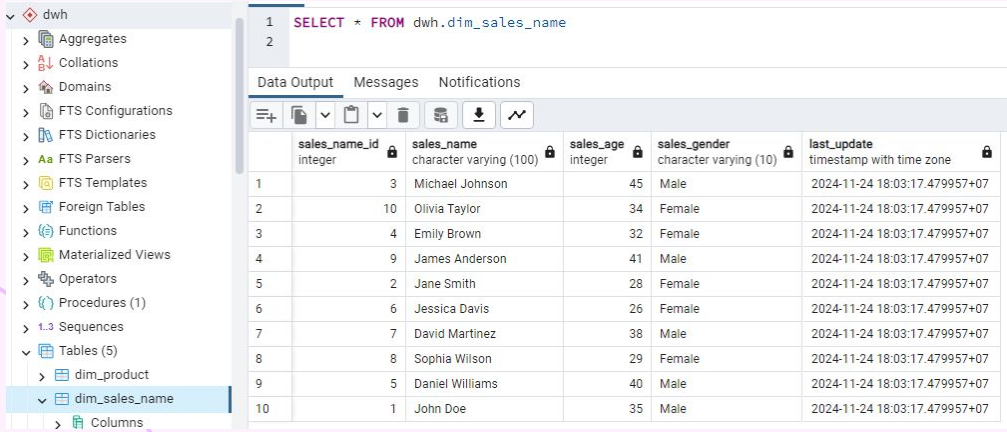


	time_id integer	date date	day_of_week character varying (10)	month character varying (20)	year integer	last_update timestamp with time zone
1	10	2024-01-10	Wednesday	January	2024	2024-11-24 13:12:46.969186+07
2	17	2024-01-17	Wednesday	January	2024	2024-11-24 13:12:46.969186+07
3	31	2024-01-31	Wednesday	January	2024	2024-11-24 13:12:46.969186+07
4	19	2024-01-19	Friday	January	2024	2024-11-24 13:12:46.969186+07
5	18	2024-01-18	Thursday	January	2024	2024-11-24 13:12:46.969186+07
6	26	2024-01-26	Friday	January	2024	2024-11-24 13:12:46.969186+07

Schema – dwh – dim_sales_name

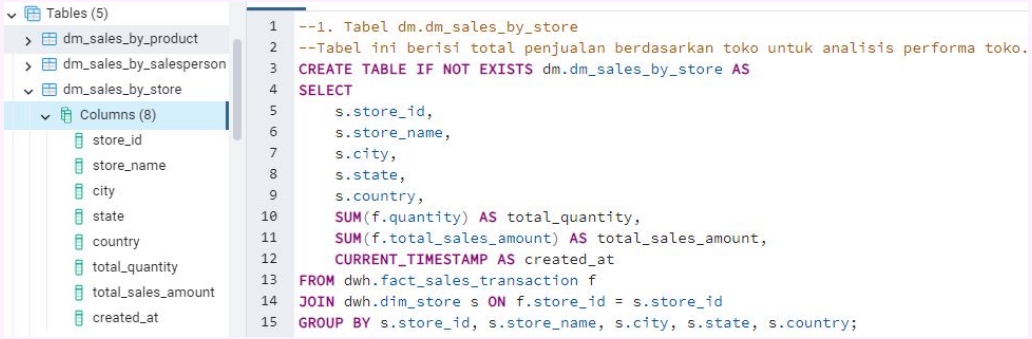


Create table dim_store berhasil

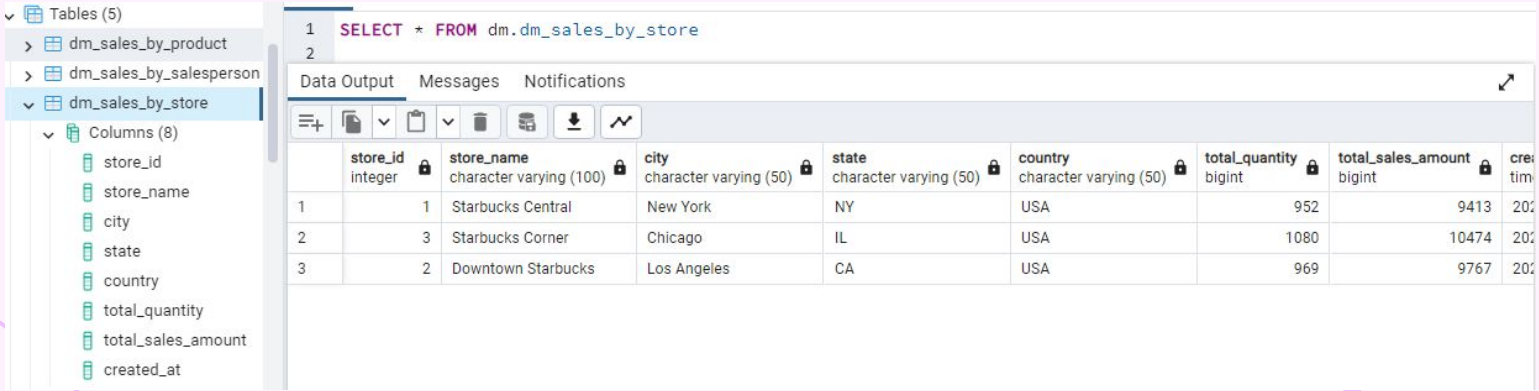


	sales_name_id integer	sales_name character varying (100)	sales_age integer	sales_gender character varying (10)	last_update timestamp with time zone
1	3	Michael Johnson	45	Male	2024-11-24 18:03:17.479957+07
2	10	Olivia Taylor	34	Female	2024-11-24 18:03:17.479957+07
3	4	Emily Brown	32	Female	2024-11-24 18:03:17.479957+07
4	9	James Anderson	41	Male	2024-11-24 18:03:17.479957+07
5	2	Jane Smith	28	Female	2024-11-24 18:03:17.479957+07
6	6	Jessica Davis	26	Female	2024-11-24 18:03:17.479957+07
7	7	David Martinez	38	Male	2024-11-24 18:03:17.479957+07
8	8	Sophia Wilson	29	Female	2024-11-24 18:03:17.479957+07
9	5	Daniel Williams	40	Male	2024-11-24 18:03:17.479957+07
10	1	John Doe	35	Male	2024-11-24 18:03:17.479957+07

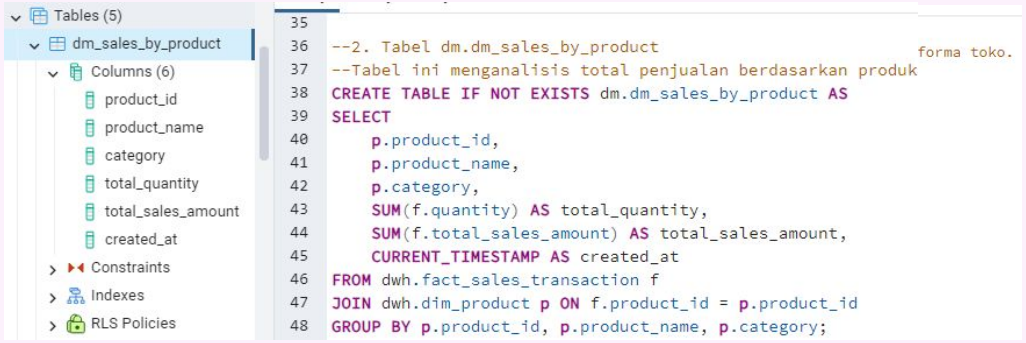
Schema – dm – dm_sales_by_store



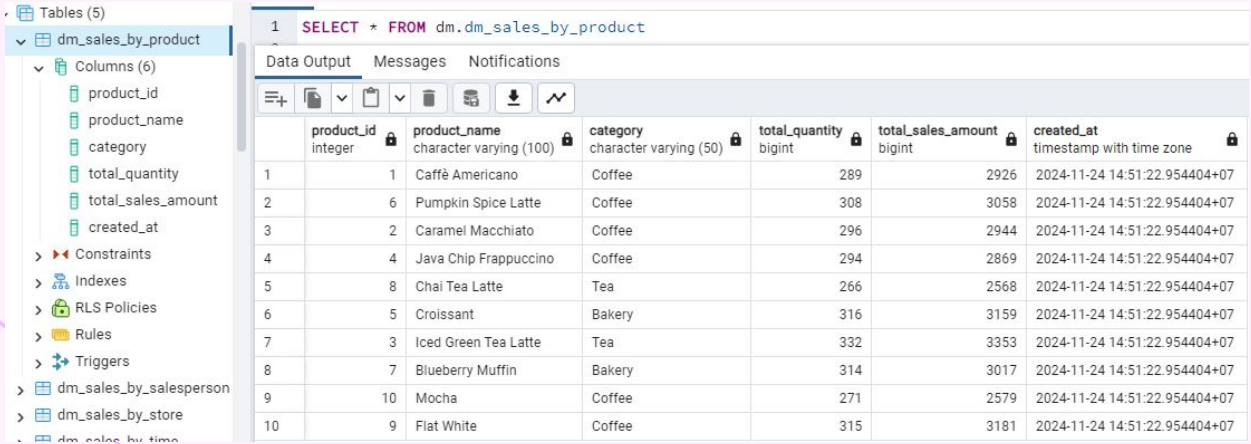
Create table dm_sales_by_store



Schema – dm – dm_sales_by_product

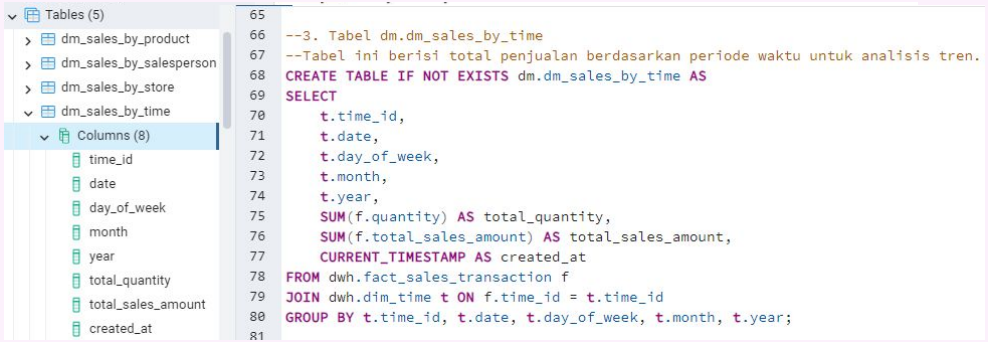


Create table dm_sales_by_product

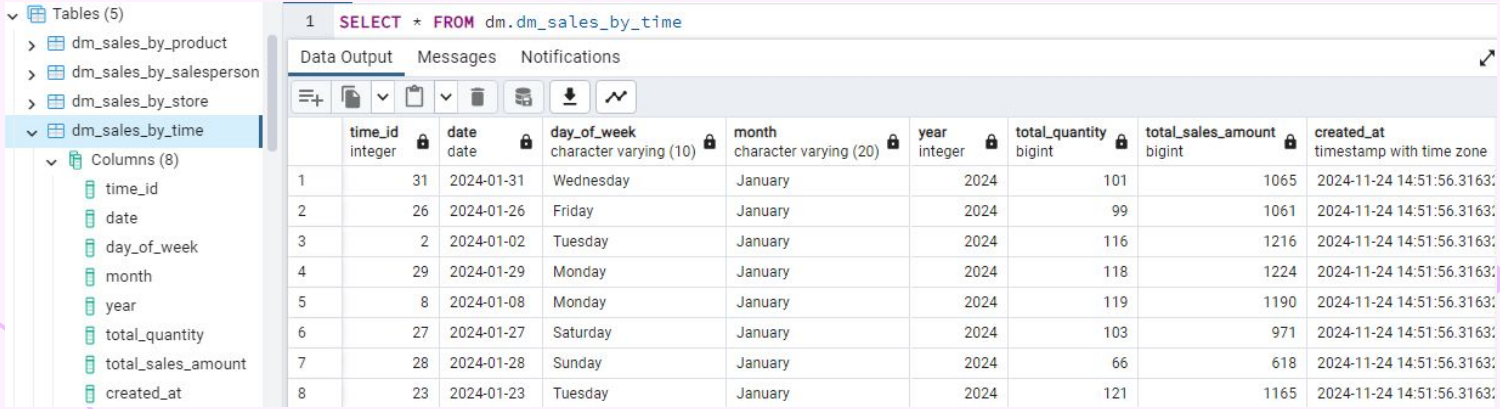


	product_id integer	product_name character varying (100)	category character varying (50)	total_quantity bigint	total_sales_amount bigint	created_at timestamp with time zone
1	1	Caffè Americano	Coffee	289	2926	2024-11-24 14:51:22.954404+07
2	6	Pumpkin Spice Latte	Coffee	308	3058	2024-11-24 14:51:22.954404+07
3	2	Caramel Macchiato	Coffee	296	2944	2024-11-24 14:51:22.954404+07
4	4	Java Chip Frappuccino	Coffee	294	2869	2024-11-24 14:51:22.954404+07
5	8	Chai Tea Latte	Tea	266	2568	2024-11-24 14:51:22.954404+07
6	5	Croissant	Bakery	316	3159	2024-11-24 14:51:22.954404+07
7	3	Iced Green Tea Latte	Tea	332	3353	2024-11-24 14:51:22.954404+07
8	7	Blueberry Muffin	Bakery	314	3017	2024-11-24 14:51:22.954404+07
9	10	Mocha	Coffee	271	2579	2024-11-24 14:51:22.954404+07
10	9	Flat White	Coffee	315	3181	2024-11-24 14:51:22.954404+07

Schema – dm – dm_sales_by_time

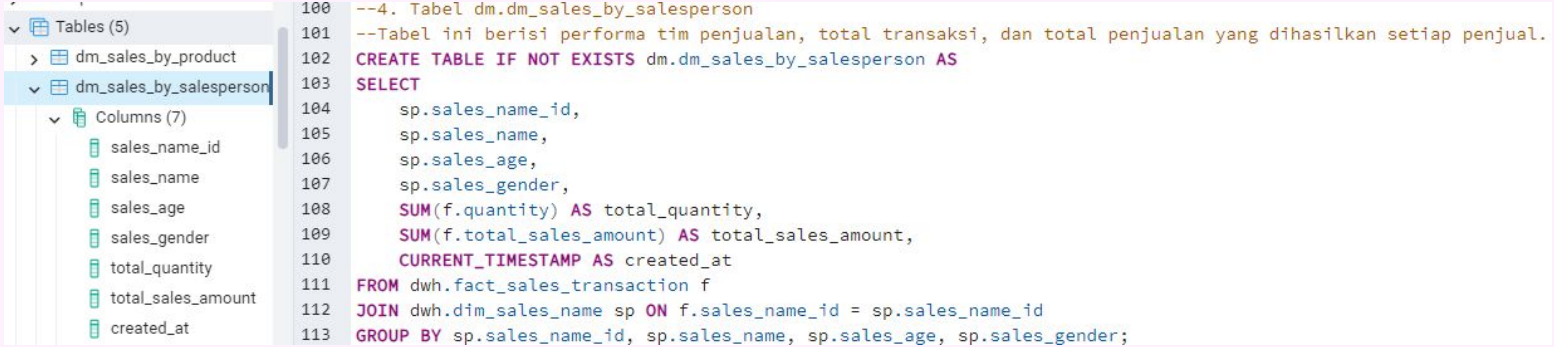


Create table dm_sales_by_time



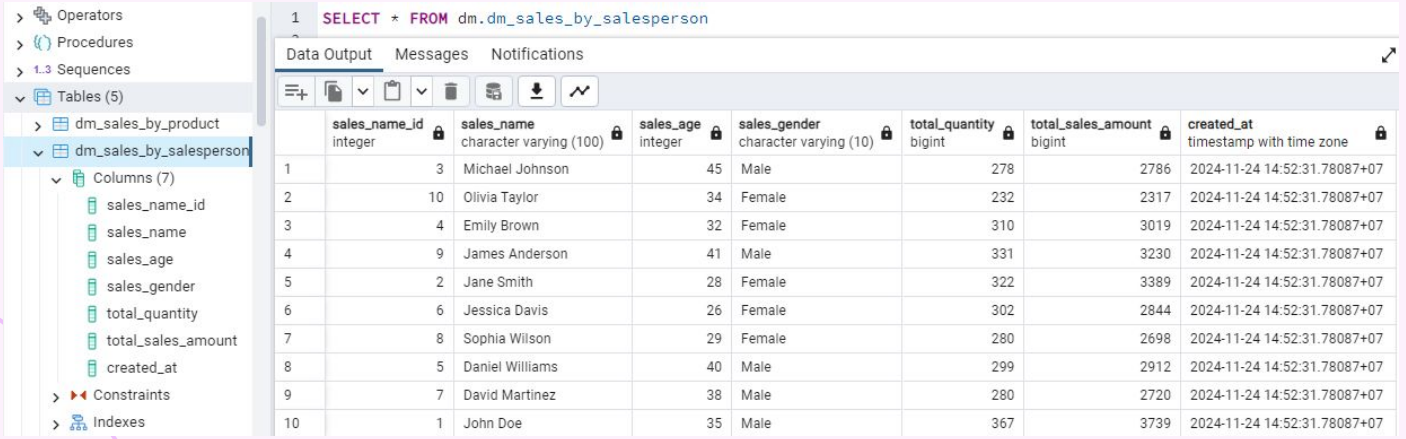
	time_id integer	date date	day_of_week character varying (10)	month character varying (20)	year integer	total_quantity bigint	total_sales_amount bigint	created_at timestamp with time zone
1	31	2024-01-31	Wednesday	January	2024	101	1065	2024-11-24 14:51:56.3163
2	26	2024-01-26	Friday	January	2024	99	1061	2024-11-24 14:51:56.3163
3	2	2024-01-02	Tuesday	January	2024	116	1216	2024-11-24 14:51:56.3163
4	29	2024-01-29	Monday	January	2024	118	1224	2024-11-24 14:51:56.3163
5	8	2024-01-08	Monday	January	2024	119	1190	2024-11-24 14:51:56.3163
6	27	2024-01-27	Saturday	January	2024	103	971	2024-11-24 14:51:56.3163
7	28	2024-01-28	Sunday	January	2024	66	618	2024-11-24 14:51:56.3163
8	23	2024-01-23	Tuesday	January	2024	121	1165	2024-11-24 14:51:56.3163

Schema – dm – dm_sales_by_salesperson



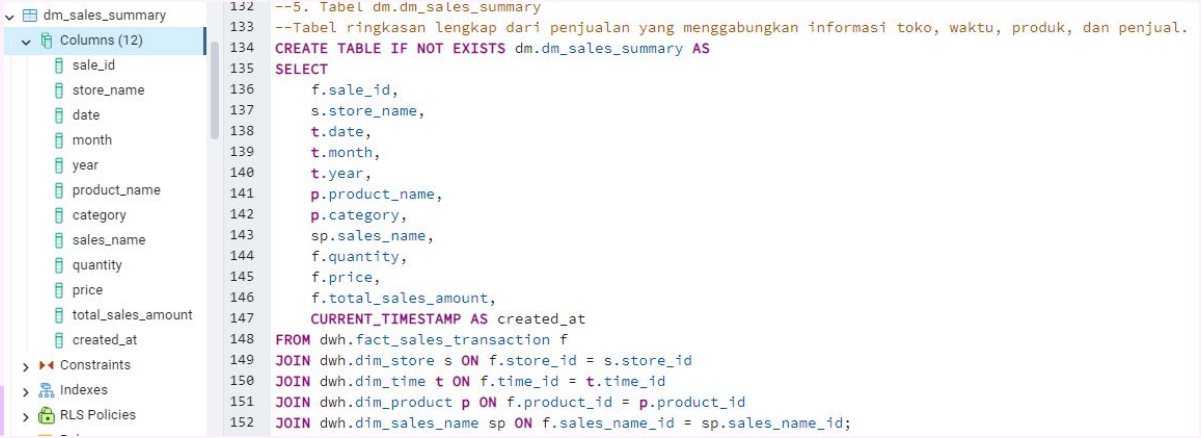

```sql
--4. Tabel dm.dm_sales_by_salesperson
--Tabel ini berisi performa tim penjualan, total transaksi, dan total penjualan yang dihasilkan setiap penjual.
CREATE TABLE IF NOT EXISTS dm.dm_sales_by_salesperson AS
SELECT
    sp.sales_name_id,
    sp.sales_name,
    sp.sales_age,
    sp.sales_gender,
    SUM(f.quantity) AS total_quantity,
    SUM(f.total_sales_amount) AS total_sales_amount,
    CURRENT_TIMESTAMP AS created_at
FROM dwh.fact_sales_transaction f
JOIN dwh.dim_sales_name sp ON f.sales_name_id = sp.sales_name_id
GROUP BY sp.sales_name_id, sp.sales_name, sp.sales_age, sp.sales_gender;
```

Create table dm_sales_by_salesperson

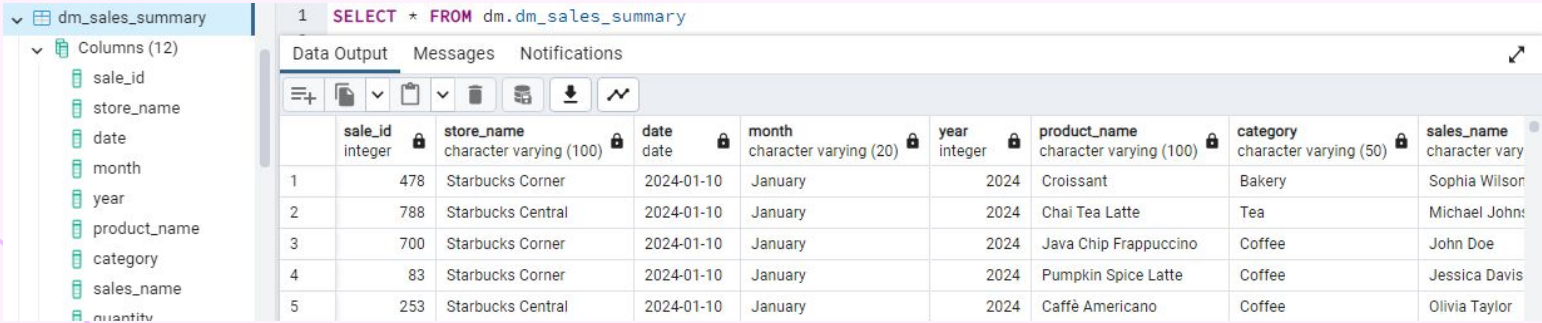


	sales_name_id integer	sales_name character varying (100)	sales_age integer	sales_gender character varying (10)	total_quantity bigint	total_sales_amount bigint	created_at timestamp with time zone
1	3	Michael Johnson	45	Male	278	2786	2024-11-24 14:52:31.78087+07
2	10	Olivia Taylor	34	Female	232	2317	2024-11-24 14:52:31.78087+07
3	4	Emily Brown	32	Female	310	3019	2024-11-24 14:52:31.78087+07
4	9	James Anderson	41	Male	331	3230	2024-11-24 14:52:31.78087+07
5	2	Jane Smith	28	Female	322	3389	2024-11-24 14:52:31.78087+07
6	6	Jessica Davis	26	Female	302	2844	2024-11-24 14:52:31.78087+07
7	8	Sophia Wilson	29	Female	280	2698	2024-11-24 14:52:31.78087+07
8	5	Daniel Williams	40	Male	299	2912	2024-11-24 14:52:31.78087+07
9	7	David Martinez	38	Male	280	2720	2024-11-24 14:52:31.78087+07
10	1	John Doe	35	Male	367	3739	2024-11-24 14:52:31.78087+07

Schema – dm – dm_sales_summary



Create table dm_sales_summary



	sale_id integer	store_name character varying (100)	date date	month character varying (20)	year integer	product_name character varying (100)	category character varying (50)	sales_name character vary
1	478	Starbucks Corner	2024-01-10	January	2024	Croissant	Bakery	Sophia Wilson
2	788	Starbucks Central	2024-01-10	January	2024	Chai Tea Latte	Tea	Michael Johns
3	700	Starbucks Corner	2024-01-10	January	2024	Java Chip Frappuccino	Coffee	John Doe
4	83	Starbucks Corner	2024-01-10	January	2024	Pumpkin Spice Latte	Coffee	Jessica Davis
5	253	Starbucks Central	2024-01-10	January	2024	Caffè Americano	Coffee	Olivia Taylor

04

Designing Store Procedure

Store Procedure Creation Flow

Staging

Area sementara untuk menyimpan data yang diekstraksi dari berbagai sumber sebelum diproses lebih lanjut.

Data Warehouse

Untuk menyimpan data yang telah diproses, diorganisasi, dan disusun untuk keperluan analisis jangka panjang.

Data Mart

Subset dari Data Warehouse yang berfokus pada kebutuhan analisis spesifik, seperti penjualan, pemasaran, atau keuangan.

Public

Skema default dalam PostgreSQL yang biasanya digunakan untuk menyimpan data umum atau tabel yang bersifat publik

05

Step 1: Transfer to Staging Tables

Store Procedure

```sql
-- Sample Store Procedure
CREATE OR REPLACE PROCEDURE dwh.generate_sales()
LANGUAGE plpgsql
AS $procedure$
BEGIN
    -- Step 1: Truncate and Insert into stg SCD Type 1
    -- Description: Clear the staging table and populate it with data from the source table.
    TRUNCATE TABLE stg.stg_sales_transaction;
    INSERT INTO stg.stg_sales_transaction
    SELECT * FROM public.sales_transaction;
```

Step 1: Transfer to Staging Tables

This query is the first step in the ETL process and is designed to:

1. **Clear the staging table** to ensure no duplicate or outdated data exists.
2. **Populate the staging table with fresh data** extracted from the source (`public.sales_transaction`), setting up for the next steps of transformation and loading.

06

Step : Load Dimension Tables

Store Procedure

```
12    -- Step 2: Insert or update into dim_product
13    -- Description: Insert new product data into the product dimension if they don't already exist.
14    --              Update existing product data if they already exist.
15    CREATE TABLE IF NOT EXISTS dwh.dim_product AS
16    SELECT DISTINCT src.product_id, src.product_name, src.category, CURRENT_TIMESTAMP AS last_update
17    FROM stg.stg_sales_transaction AS src;
18
19    TRUNCATE TABLE dwh.dim_product;
20
21    INSERT INTO dwh.dim_product (product_id, product_name, category, last_update)
22    SELECT DISTINCT src.product_id, src.product_name, src.category, CURRENT_TIMESTAMP AS last_update
23    FROM stg.stg_sales_transaction AS src;
```

2. Load Dimension Tables

This query updates the **product dimension** (`dim_product`):

1. **Create Table**: Ensures `dwh.dim_product` exists and initializes it with distinct product data.
2. **Clear Old Data**: Removes outdated entries using `TRUNCATE`.
3. **Insert Fresh Data**: Populates `dim_product` with unique product details from `stg.stg_sales_transaction` and updates the `last_update` timestamp.

Store Procedure

```
-- Step 3: Insert or update into dim_store
-- Description: Insert new store data into the store dimension if they don't already exist.
--              Update existing store data if they already exist.
CREATE TABLE IF NOT EXISTS dwh.dim_store AS
SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;

TRUNCATE TABLE dwh.dim_store;

INSERT INTO dwh.dim_store (store_id, store_name, city, state, country, last_update)
SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;
```

2. Load Dimension Tables

This query updates the **store dimension** (`dim_store`):

1. **Create Table**: Ensures `dwh.dim_store` exists and initializes it with distinct store data.
2. **Clear Old Data**: Removes outdated entries using `TRUNCATE`.
3. **Insert Fresh Data**: Populates `dim_store` with unique store details (`store_id`, `store_name`, `city`, `state`, `country`) from `stg.stg_sales_transaction` and updates the `last_update` timestamp.

Store Procedure

```
-- Step 3: Insert or update into dim_store
-- Description: Insert new store data into the store dimension if they don't already exist.
--              Update existing store data if they already exist.
CREATE TABLE IF NOT EXISTS dwh.dim_store AS
SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;

TRUNCATE TABLE dwh.dim_store;

INSERT INTO dwh.dim_store (store_id, store_name, city, state, country, last_update)
SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;
```

2. Load Dimension Tables

This query updates the **store dimension** (`dim_store`):

1. **Create Table**: Ensures `dwh.dim_store` exists and initializes it with distinct store data.
2. **Clear Old Data**: Removes outdated entries using `TRUNCATE`.
3. **Insert Fresh Data**: Populates `dim_store` with unique store details (`store_id`, `store_name`, `city`, `state`, `country`) from `stg.stg_sales_transaction` and updates the `last_update` timestamp.

Store Procedure

```
40  -- Step 4: Insert or update into dim_time
41  -- Description: Insert new time data into the time dimension if they don't already exist.
42  --              Update existing time data if they already exist.
43  CREATE TABLE IF NOT EXISTS dwh.dim_time AS
44  SELECT DISTINCT src.time_id, src.date, src.day_of_week, src.month, src.year, CURRENT_TIMESTAMP AS last_update
45  FROM stg.stg_sales_transaction AS src;
46
47  TRUNCATE TABLE dwh.dim_time;
48
49  INSERT INTO dwh.dim_time (time_id, date, day_of_week, month, year, last_update)
50  SELECT DISTINCT src.time_id, src.date, src.day_of_week, src.month, src.year, CURRENT_TIMESTAMP AS last_update
51  FROM stg.stg_sales_transaction AS src;
```

2. Load Dimension Tables

This query updates the **time dimension** (`dim_time`):

1. **Create Table**: Ensures `dwh.dim_time` exists and initializes it with distinct time data.
2. **Clear Old Data**: Removes outdated entries using `TRUNCATE`.
3. **Insert Fresh Data**: Populates `dim_time` with unique time details (`time_id`, `date`, `day_of_week`, `month`, `year`) from `stg.stg_sales_transaction` and updates the `last_update` timestamp.

Store Procedure

```sql
-- Step 5: Insert or update into  dim_sales_name
-- Description: Insert new sales name data into the sales name dimension if they don't already exist.
--              Update existing sales name data if they already exist.
CREATE TABLE IF NOT EXISTS dwh.dim_sales_name AS
SELECT DISTINCT src.sales_name_id, src.sales_name, src.sales_age, src.sales_gender, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;

TRUNCATE TABLE dwh.dim_sales_name;

INSERT INTO dwh.dim_sales_name (sales_name_id, sales_name, sales_age, sales_gender, last_update)
SELECT DISTINCT src.sales_name_id, src.sales_name, src.sales_age, src.sales_gender, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;
```

2. Load Dimension Tables

This query updates the **sales name dimension** (`dim_sales_name`):

1. **Create Table**: Ensures `dwh.dim_sales_name` exists and initializes it with distinct sales name data.
2. **Clear Old Data**: Removes outdated entries using `TRUNCATE`.
3. **Insert Fresh Data**: Populates `dim_sales_name` with unique sales details (`sales_name_id`, `sales_name`, `sales_age`, `sales_gender`) from `stg.stg_sales_transaction` and updates the `last_update` timestamp.

07

Step 3: Load Fact Tables

Store Procedure

```
67      -- Step 6: Insert into fact SCD Type 2
68      -- Description: Insert new sales transactions into the fact table if they don't already exist.
69      INSERT INTO dwh.fact_sales_transaction
70          (sale_id, store_id, sales_name_id, product_id, time_id, date, quantity, price)
71      SELECT
72          src.sale_id,
73          src.store_id,
74          src.sales_name_id,
75          src.product_id,
76          src.time_id,
77          src.date,
78          src.quantity,
79          src.price
80      FROM
81          stg.stg_sales_transaction AS src
82      WHERE NOT EXISTS (
83          SELECT 1
84          FROM dwh.fact_sales_transaction AS fact
85          WHERE
86              COALESCE(fact.sale_id, 0) = COALESCE(src.sale_id, 0) AND
87              COALESCE(fact.store_id, 0) = COALESCE(src.store_id, 0) AND
88              COALESCE(fact.sales_name_id, 0) = COALESCE(src.sales_name_id, 0) AND
89              COALESCE(fact.product_id, 0) = COALESCE(src.product_id, 0) AND
90              COALESCE(fact.time_id, 0) = COALESCE(src.time_id, 0) AND
91              COALESCE(fact.date, CURRENT_DATE) = COALESCE(src.date, CURRENT_DATE) AND
92              COALESCE(fact.quantity, 0) = COALESCE(src.quantity, 0) AND
93              COALESCE(fact.price, 0) = COALESCE(src.price, 0)
94      );
```

Step 3: Load Fact Tables

This query updates the **fact table** (`fact_sales_transaction`):

1. **Insert New Records**: Adds unique sales data from `stg.stg_sales_transaction`.
2. **Avoid Duplicates**: Ensures no duplicate entries using `WHERE NOT EXISTS`.
3. **Field Matching**: Uses `COALESCE` to handle null values during comparison.

Load dm Tables

Store Procedure

```
98   -- Step 5: Truncate and Insert into dm SCD Type 1
99   --Langkah Pemrosesan (ETL)
100  -- 1. Membersihkan data lama dm_sales_by_store
101  TRUNCATE TABLE dm.dm_sales_by_store;
102
103
104  -- Memasukkan data terbaru
105  INSERT INTO dm.dm_sales_by_store
106  SELECT
107      s.store_id,
108      s.store_name,
109      s.city,
110      s.state,
111      s.country,
112      SUM(f.quantity) AS total_quantity,
113      SUM(f.quantity * f.price) AS total_sales_amount, -- Perhitungan langsung
114      CURRENT_TIMESTAMP AS created_at
115  FROM dwh.fact_sales_transaction f
116  JOIN dwh.dim_store s ON f.store_id = s.store_id
117  WHERE f.sale_id IN (SELECT DISTINCT sale_id FROM stg.stg_sales_transaction)
118  GROUP BY s.store_id, s.store_name, s.city, s.state, s.country;
```

Load dm Tables

This query updates the **data mart table** (`dm_sales_by_store`):

1. **Clear Old Data**: Empties `dm_sales_by_store` using `TRUNCATE` to prepare for fresh data.
2. **Insert New Data**: Populates the table with aggregated sales data (`total_quantity` and `total_sales_amount`) grouped by store attributes (`store_id`, `store_name`, `city`, `state`, `country`).
3. **Direct Calculation**: Computes `total_sales_amount` as `SUM(f.quantity * f.price)` during the insertion.

Store Procedure

```
121    -- 2. Membersihkan data lama dm_sales_by_product
122
123    -- Membersihkan data lama
124    TRUNCATE TABLE dm.dm_sales_by_product;
125
126    -- Memasukkan data terbaru
127    INSERT INTO dm.dm_sales_by_product
128    SELECT
129        p.product_id,
130        p.product_name,
131        p.category,
132        SUM(f.quantity) AS total_quantity,
133        SUM(f.quantity * f.price) AS total_sales_amount, -- Perhitungan langsung
134        CURRENT_TIMESTAMP AS created_at
135    FROM dwh.fact_sales_transaction f
136    JOIN dwh.dim_product p ON f.product_id = p.product_id
137    WHERE f.sale_id IN (SELECT DISTINCT sale_id FROM stg.stg_sales_transaction)
138    GROUP BY p.product_id, p.product_name, p.category;
139
```

Load dm Tables

This query updates the **data mart table** (`dm_sales_by_product`):

1. **Clear Old Data**: Removes existing data from `dm_sales_by_product` using `TRUNCATE`.
2. **Insert New Data**: Populates the table with aggregated sales data (`total_quantity` and `total_sales_amount`) grouped by product attributes (`product_id`, `product_name`, `category`).
3. **Direct Calculation**: Computes `total_sales_amount` as `SUM(f.quantity * f.price)` during the insertion.

Store Procedure

```
141     -- 3. Membersihkan data lama dm_sales_by_time
142     -- Membersihkan data lama
143     -- Membersihkan data lama
144     TRUNCATE TABLE dm.dm_sales_by_time;
145
146     -- Memasukkan data terbaru ke dm.dm_sales_by_time
147     -- Memasukkan data terbaru ke dm.dm_sales_by_time
148     INSERT INTO dm.dm_sales_by_time
149     SELECT
150         t.time_id, -- Masukkan time_id sebagai kolom pertama
151         t.date,
152         t.day_of_week,
153         t.month,
154         t.year,
155         SUM(f.quantity) AS total_quantity,
156         SUM(f.quantity * f.price) AS total_sales_amount, -- Perhitungan langsung
157         CURRENT_TIMESTAMP AS created_at
158     FROM dwh.fact_sales_transaction f
159     JOIN dwh.dim_time t ON f.time_id = t.time_id -- Gunakan time_id untuk join
160     WHERE f.sale_id IN (SELECT DISTINCT sale_id FROM stg.stg_sales_transaction)
161     GROUP BY t.time_id, t.date, t.day_of_week, t.month, t.year;
```

Load dm Tables

This query updates the **data mart table** (`dm_sales_by_time`):

1. **Clear Old Data**: Empties `dm_sales_by_time` using `TRUNCATE` to prepare for new data.
2. **Insert New Data**: Populates the table with aggregated sales data (`total_quantity` and `total_sales_amount`) grouped by time attributes (`time_id`, `date`, `day_of_week`, `month`, `year`).
3. **Direct Calculation**: Computes `total_sales_amount` as `SUM(f.quantity * f.price)` during the insertion.

Store Procedure

```
167     -- 4. Membersihkan data lama dm_sales_by_salesperson
168     -- Membersihkan data lama
169     TRUNCATE TABLE dm.dm_sales_by_salesperson;
170
171     -- Memasukkan data terbaru
172     INSERT INTO dm.dm_sales_by_salesperson
173     SELECT
174         sp.sales_name_id,
175         sp.sales_name,
176         sp.sales_age,
177         sp.sales_gender,
178         SUM(f.quantity) AS total_quantity,
179         SUM(f.quantity * f.price) AS total_sales_amount, -- Perhitungan langsung
180         CURRENT_TIMESTAMP AS created_at
181     FROM dwh.fact_sales_transaction f
182     JOIN dwh.dim_sales_name sp ON f.sales_name_id = sp.sales_name_id
183     WHERE f.sale_id IN (SELECT DISTINCT sale_id FROM stg.stg_sales_transaction)
184     GROUP BY sp.sales_name_id, sp.sales_name, sp.sales_age, sp.sales_gender;
```

Load dm Tables

This query updates the **data mart table** (`dm_sales_by_salesperson`):

1. **Clear Old Data**: Removes existing data from `dm_sales_by_salesperson` using `TRUNCATE`.
2. **Insert New Data**: Populates the table with aggregated sales data (`total_quantity` and `total_sales_amount`) grouped by salesperson attributes (`sales_name_id`, `sales_name`, `sales_age`, `sales_gender`).
3. **Direct Calculation**: Computes `total_sales_amount` as `SUM(f.quantity * f.price)` during the insertion.

Store Procedure

```
188     -- Membersihkan data lama
189     TRUNCATE TABLE dm.dm_sales_summary;
190
191     -- Memasukkan data terbaru
192     INSERT INTO dm.dm_sales_summary
193     SELECT
194         f.sale_id,
195         s.store_name,
196         t.date,
197         t.month,
198         t.year,
199         p.product_name,
200         p.category,
201         sp.sales_name,
202         f.quantity,
203         f.price,
204         f.quantity * f.price AS total_sales_amount, -- Perhitungan langsung
205         CURRENT_TIMESTAMP AS created_at
206     FROM dwh.fact_sales_transaction f
207     JOIN dwh.dim_store s ON f.store_id = s.store_id
208     JOIN dwh.dim_time t ON f.time_id = t.time_id
209     JOIN dwh.dim_product p ON f.product_id = p.product_id
210     JOIN dwh.dim_sales_name sp ON f.sales_name_id = sp.sales_name_id
211     WHERE f.sale_id IN (SELECT DISTINCT sale_id FROM stg.stg_sales_transaction);
```

```
212
213 END;
214 $procedure$;
215
```

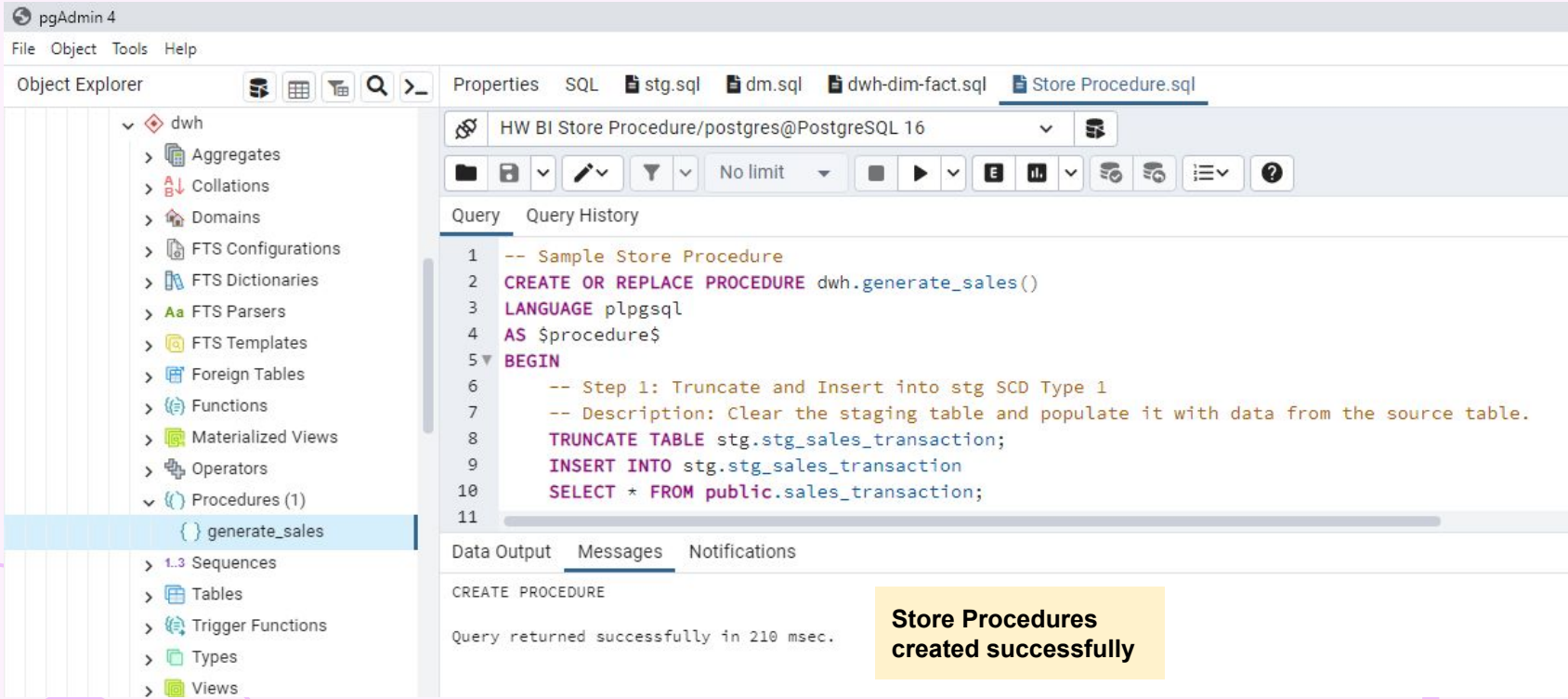
Load dm Tables

Step 1: Clear outdated data in both `dm_sales_by_salesperson` and `dm_sales_summary`.
Step 2: Insert updated and aggregated data into `dm_sales_by_salesperson`.
Step 3: Populate `dm_sales_summary` with detailed transaction-level data enriched with dimension attributes.

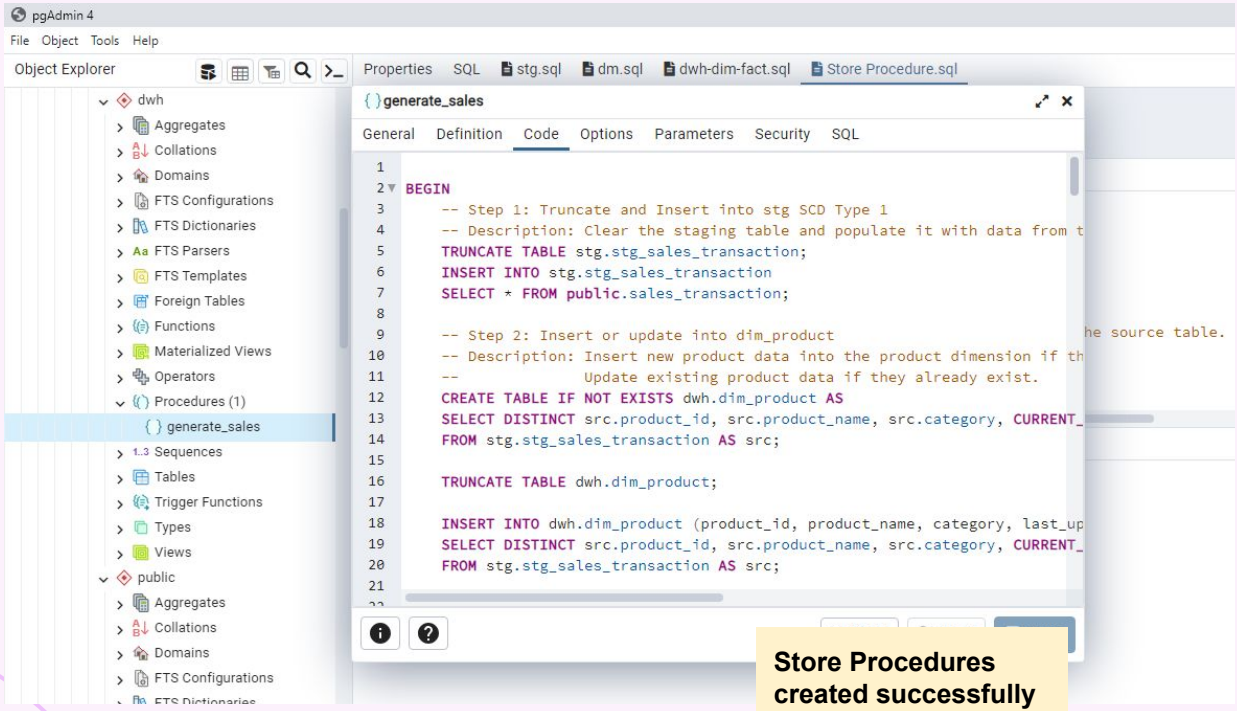
08

Documentation

Store Procedure



Store Precedure



09

Benefits of Stored Procedure

Solutions

1. Performance Optimization

Stored procedures are precompiled, which means they execute faster than dynamic SQL queries as the database engine caches their execution plan.

2. Consistency and Reusability

Reusable across multiple applications, ensuring consistent execution of database operations.

3. Enhanced Security

Permissions can be granted at the procedure level, restricting direct access to underlying tables and protecting sensitive data.

4. Maintainability

Encapsulate business logic in one place, making it easier to maintain and modify without changing application code.

5. Error Handling

Support for robust error handling mechanisms, allowing for detailed logging and recovery in case of issues.

6. Reduction of Network Traffic

Executes directly on the database server, minimizing the need to send multiple queries from the client to the server.

7. Scalability

Efficiently handles complex operations and large datasets, making it suitable for enterprise-level applications.

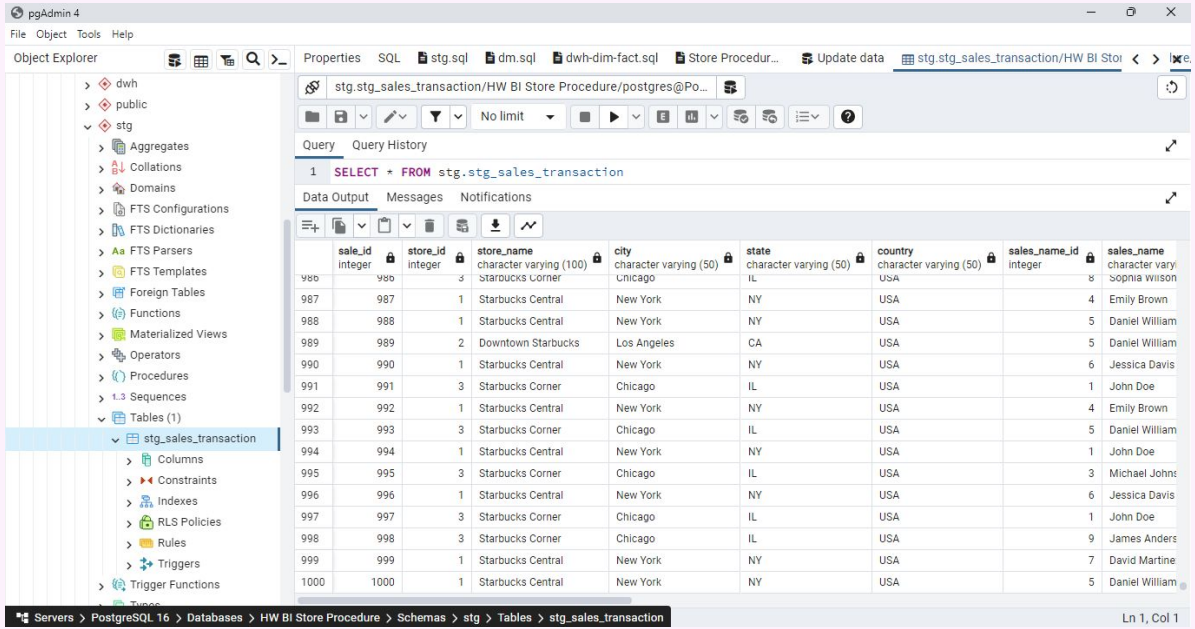
8. Improved Data Integrity

Centralized logic ensures uniformity in how data is processed, reducing the risk of inconsistencies.

10

Execution and Testing

Initial Data in public



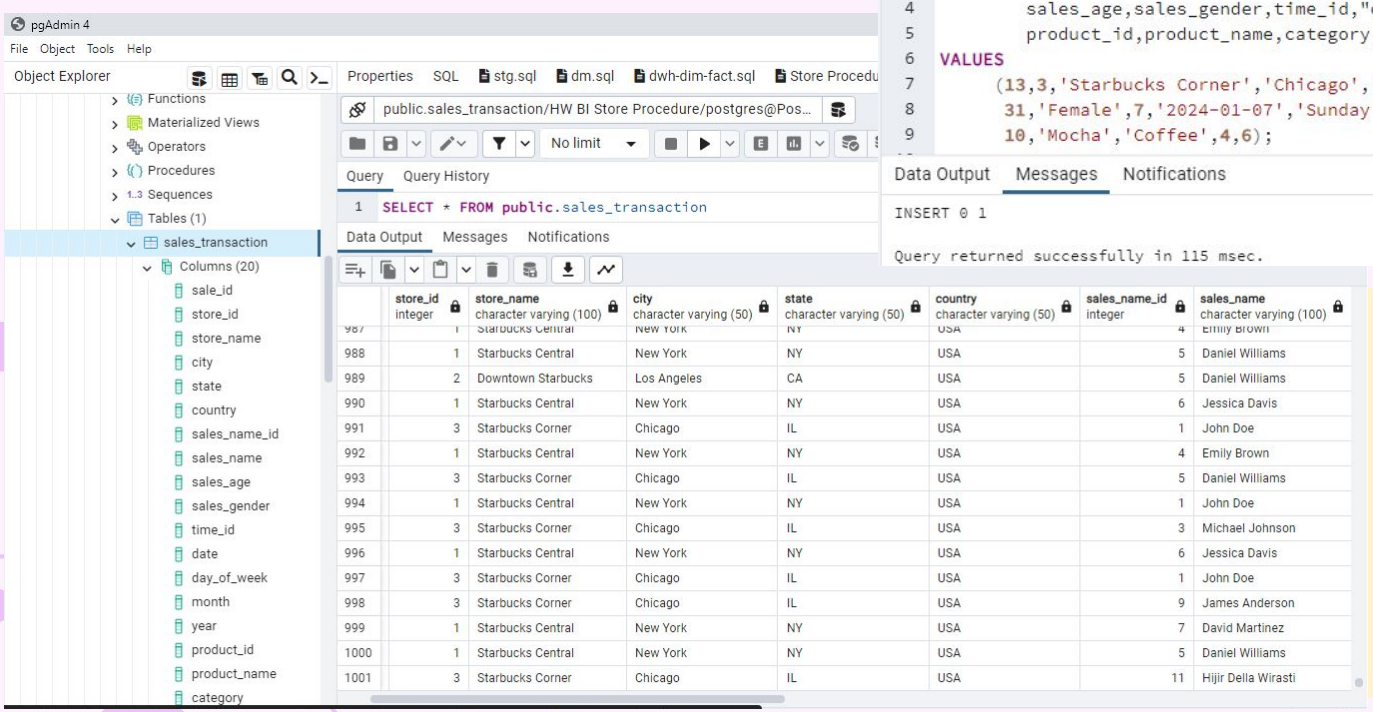
The data in the `stg_sales_transaction` table within the `public` schema initially contains 1,000 rows.

Updated Data in Public

```sql
--Inserting new data into public.sales_transaction table
INSERT INTO public.sales_transaction(
        sale_id,store_id,store_name,city,state,country,sales_name_id,sales_name,
        sales_age,sales_gender,time_id,"date",day_of_week,"month","year",
        product_id,product_name,category,quantity,price)
VALUES
     (13,3,'Starbucks Corner','Chicago','IL','USA',11,'Hijir Della Wirasti',
      31,'Female',7,'2024-01-07','Sunday','January',2024,
      10,'Mocha','Coffee',4,6);
```

Data Output | Messages | Notifications

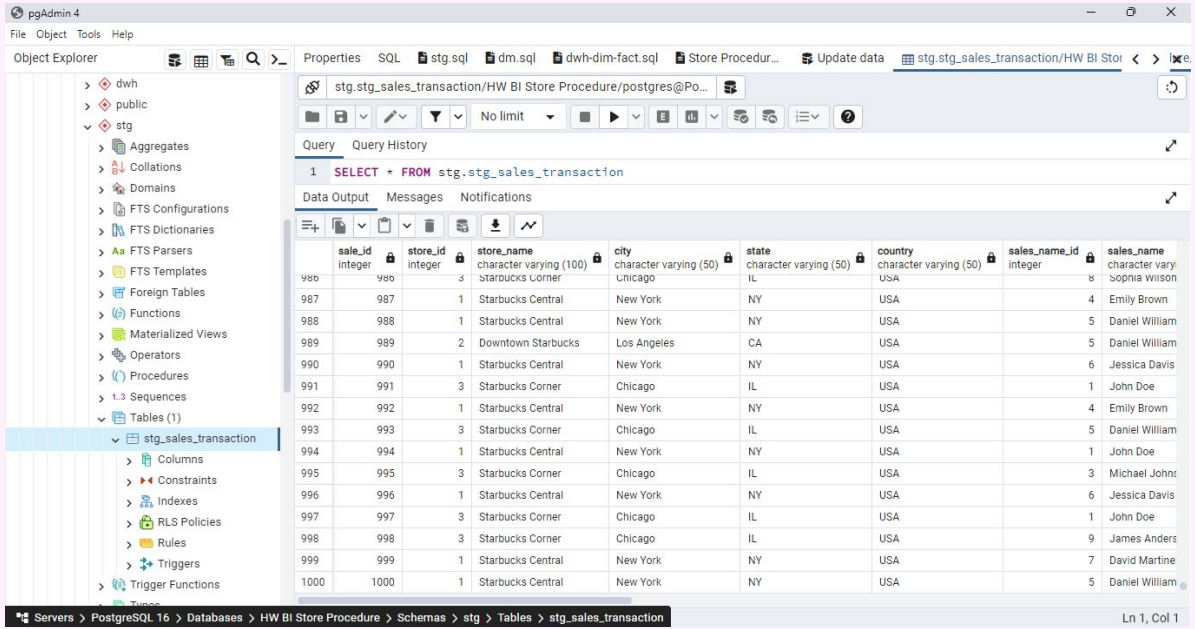
```
INSERT 0 1

Query returned successfully in 115 msec.
```



After adding new data (sales_name: Hijir Della Wirasti), the `stg_sales_transaction` table now contains 1,001 rows, with the update reflected only in the `public` schema.

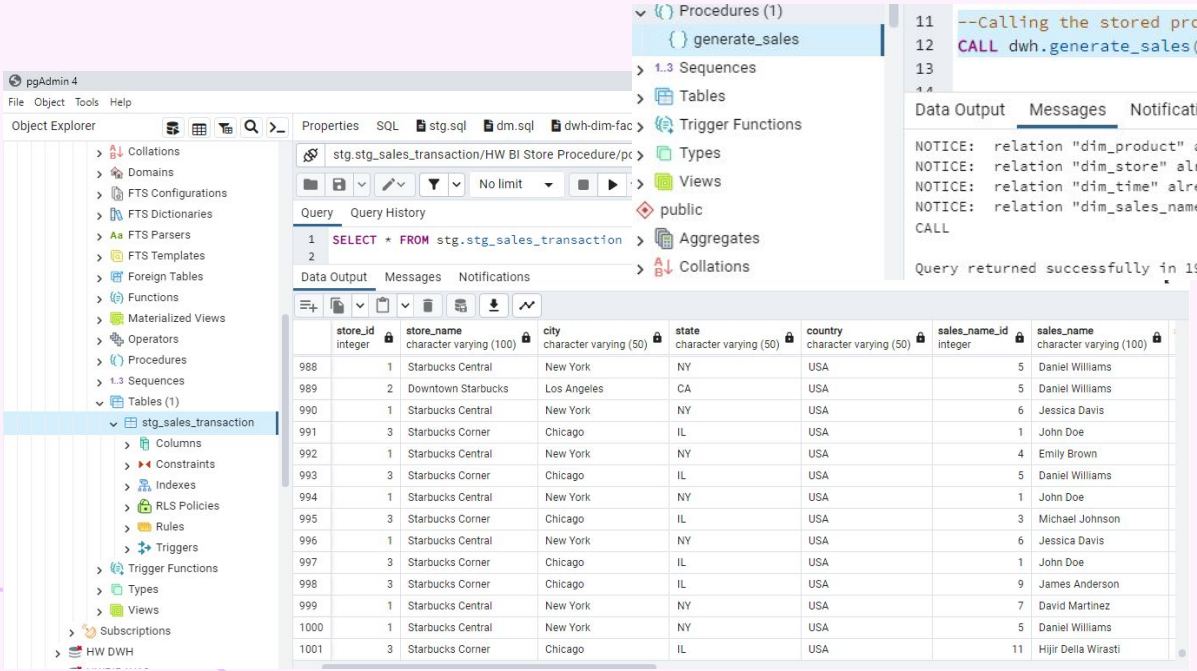
To synchronize the changes across the `stg`, `dwh`, and `dm` schemas, the `generate_sales` stored procedure needs to be executed.

Initial Data in stg



It can be seen that there are no changes yet in the `stg` schema, as it still contains 1,000 rows.

Data stg update 1001 Row

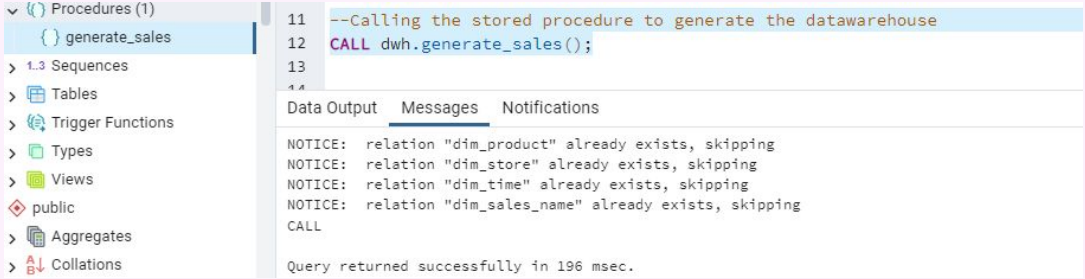


	store_id integer	store_name character varying (100)	city character varying (50)	state character varying (50)	country character varying (50)	sales_name_id integer	sales_name character varying (100)
988	1	Starbucks Central	New York	NY	USA	5	Daniel Williams
989	2	Downtown Starbucks	Los Angeles	CA	USA	5	Daniel Williams
990	1	Starbucks Central	New York	NY	USA	6	Jessica Davis
991	3	Starbucks Corner	Chicago	IL	USA	1	John Doe
992	1	Starbucks Central	New York	NY	USA	4	Emily Brown
993	3	Starbucks Corner	Chicago	IL	USA	5	Daniel Williams
994	1	Starbucks Central	New York	NY	USA	1	John Doe
995	3	Starbucks Corner	Chicago	IL	USA	3	Michael Johnson
996	1	Starbucks Central	New York	NY	USA	6	Jessica Davis
997	3	Starbucks Corner	Chicago	IL	USA	1	John Doe
998	3	Starbucks Corner	Chicago	IL	USA	9	James Anderson
999	1	Starbucks Central	New York	NY	USA	7	David Martinez
1000	1	Starbucks Central	New York	NY	USA	5	Daniel Williams
1001	3	Starbucks Corner	Chicago	IL	USA	11	Hijir Della Wirasti

After executing the `generate_sales()` stored procedure, the `stg_sales_transaction` table in the `stg` schema has increased to 1,001 rows.

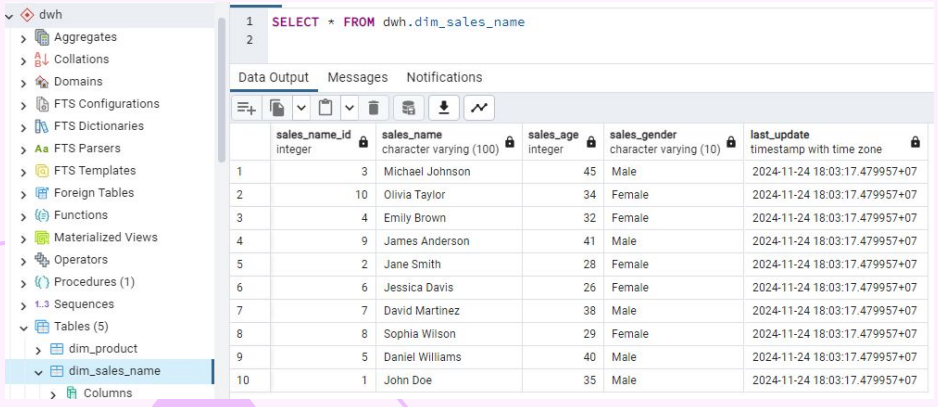
Data dwh

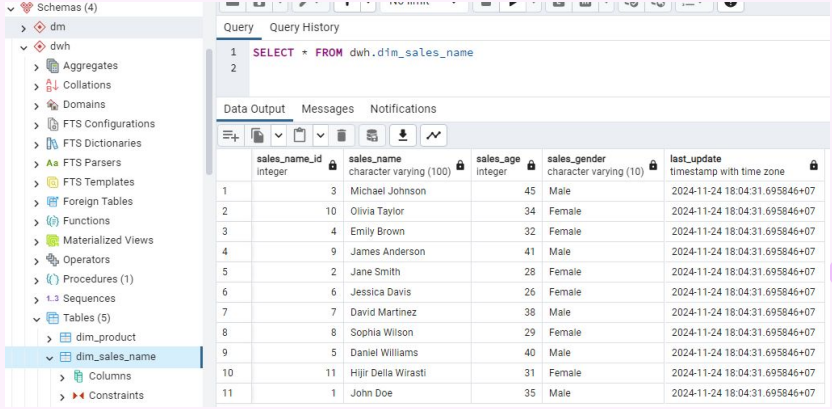
dim_sales_name



Before

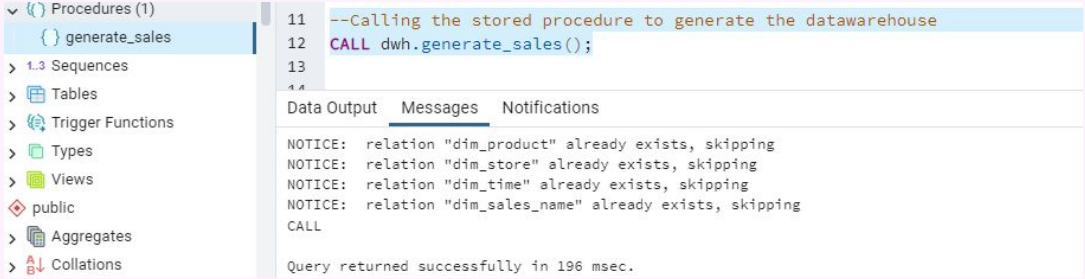
AFTER





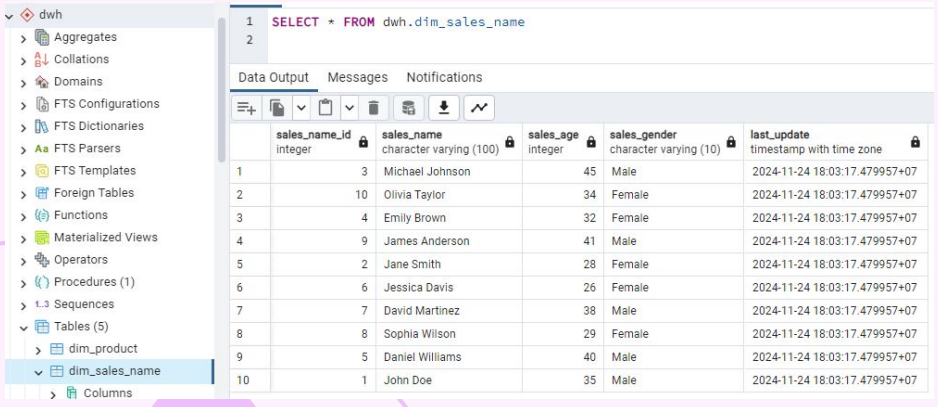
Data dwh

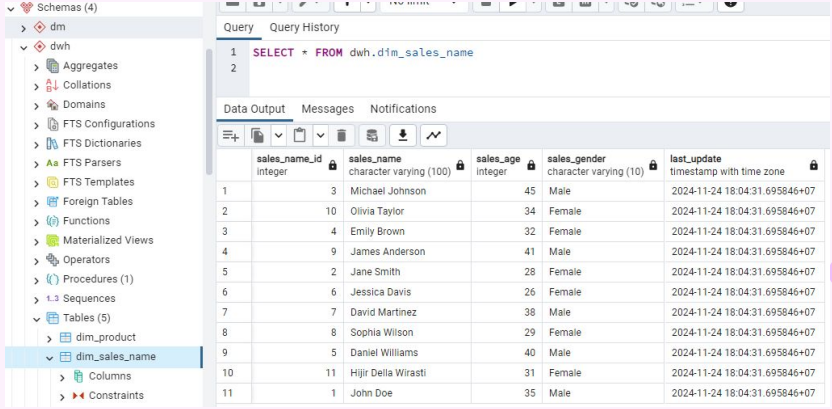
dim_sales_name



Before

AFTER





Data dwh

fact_sales_transaction

Query | Query History

```
1  SELECT * FROM dwh.fact_sales_transaction
2
```

Data Output | Messages | Notifications

	sale_id integer	store_id integer	sales_name_id integer	time_id integer	date date	product_id integer	quantity integer	price integer
988	531	1	9	7	2024-01-07	5	3	15
989	220	1	5	8	2024-01-08	10	4	12
990	125	1	6	4	2024-01-04	10	2	5
991	297	3	1	2	2024-01-02	6	2	15
992	503	3	2	10	2024-01-10	8	3	10
993	889	2	2	9	2024-01-09	5	3	8
994	335	2	7	16	2024-01-16	4	4	7
995	271	1	8	31	2024-01-31	3	3	15
996	971	2	9	14	2024-01-14	4	3	9
997	949	2	4	17	2024-01-17	9	4	14
998	101	2	8	11	2024-01-11	5	4	9
999	788	1	3	10	2024-01-10	8	2	15
1000	141	3	1	25	2024-01-25	5	3	8
1001	13	3	11	7	2024-01-07	10	4	6

Total rows: 1001 of 1001 | Query complete 00:00:00.766

Data dm

dm_sales_by_salesperson

Before

```
1  SELECT * FROM dm.dm_sales_by_salesperson
```

Data Output | Messages | Notifications

	sales_name_id integer	sales_name character varying (100)	sales_age integer	sales_gender character varying (10)	total_quantity bigint	total_sales_amount bigint	created_at timestamp with time zone
1	3	Michael Johnson	45	Male	278	2786	2024-11-24 14:52:31.78087+07
2	10	Olivia Taylor	34	Female	232	2317	2024-11-24 14:52:31.78087+07
3	4	Emily Brown	32	Female	310	3019	2024-11-24 14:52:31.78087+07
4	9	James Anderson	41	Male	331	3230	2024-11-24 14:52:31.78087+07
5	2	Jane Smith	28	Female	322	3389	2024-11-24 14:52:31.78087+07
6	6	Jessica Davis	26	Female	302	2844	2024-11-24 14:52:31.78087+07
7	8	Sophia Wilson	29	Female	280	2698	2024-11-24 14:52:31.78087+07
8	5	Daniel Williams	40	Male	299	2912	2024-11-24 14:52:31.78087+07
9	7	David Martinez	38	Male	280	2720	2024-11-24 14:52:31.78087+07
10	1	John Doe	35	Male	367	3739	2024-11-24 14:52:31.78087+07

AFTER

```
1  SELECT * FROM dm.dm_sales_by_salesperson
2
```

Data Output | Messages | Notifications

	sales_name_id integer	sales_name character varying (100)	sales_age integer	sales_gender character varying (10)	total_quantity bigint	total_sales_amount bigint	created_at timestamp with time zone
1	3	Michael Johnson	45	Male	278	2786	2024-11-24 18:04:31.695846+07
2	10	Olivia Taylor	34	Female	232	2317	2024-11-24 18:04:31.695846+07
3	4	Emily Brown	32	Female	310	3019	2024-11-24 18:04:31.695846+07
4	9	James Anderson	41	Male	331	3230	2024-11-24 18:04:31.695846+07
5	2	Jane Smith	28	Female	322	3389	2024-11-24 18:04:31.695846+07
6	6	Jessica Davis	26	Female	302	2844	2024-11-24 18:04:31.695846+07
7	8	Sophia Wilson	29	Female	280	2698	2024-11-24 18:04:31.695846+07
8	5	Daniel Williams	40	Male	299	2912	2024-11-24 18:04:31.695846+07
9	11	Hijir Della Wirasti	31	Female	4	24	2024-11-24 18:04:31.695846+07
10	7	David Martinez	38	Male	280	2720	2024-11-24 18:04:31.695846+07
11	1	John Doe	35	Male	367	3739	2024-11-24 18:04:31.695846+07

11

Conclusion

Conclusion

1. Summary of Key Points

ETL Process Workflow: Successfully demonstrated the entire ETL process, including extracting raw data from `public` schema, transforming it through the `stg` and `dwh` schemas, and loading the results into the `dm` (Data Mart) schema for analysis.
Stored Procedure Implementation: Designed and implemented the `generate_sales()` stored procedure to automate data movement, transformation, and aggregation across schemas.
Efficient Data Integration: Utilized SQL techniques like `TRUNCATE`, `INSERT`, and joins with dimension tables to ensure accurate and consistent data processing.

Data Marts for Analysis:

- Created detailed and aggregated data marts, such as `dm_sales_by_store`, `dm_sales_by_product`, `dm_sales_by_time`, `dm_sales_by_salesperson`, and `dm_sales_summary`, for easy access and insights.

2. Reinforcing the Importance of Efficient ETL Processes:

Data Accuracy: Ensures up-to-date and error-free data across all schemas (`stg`, `dwh`, `dm`).
Automation: Simplifies complex data operations, reducing manual effort and errors through a robust stored procedure.
Scalability: Supports large datasets and complex transformations, making it suitable for enterprise-level applications.
Actionable Insights: Prepares clean and structured data for analytics, enabling informed business decisions.

By implementing efficient ETL processes, businesses can maintain data integrity, reduce operational complexity, and unlock powerful insights for strategic growth.

Thanks!

Do you have any questions?

hijirdw@gmail.com
https://github.com/hijirdella
https://www.linkedin.com/in/hijirdella/