



ETL Process with Stored Procedure

Efficient Data Transfer and Loading

Hijir Della Wirasti

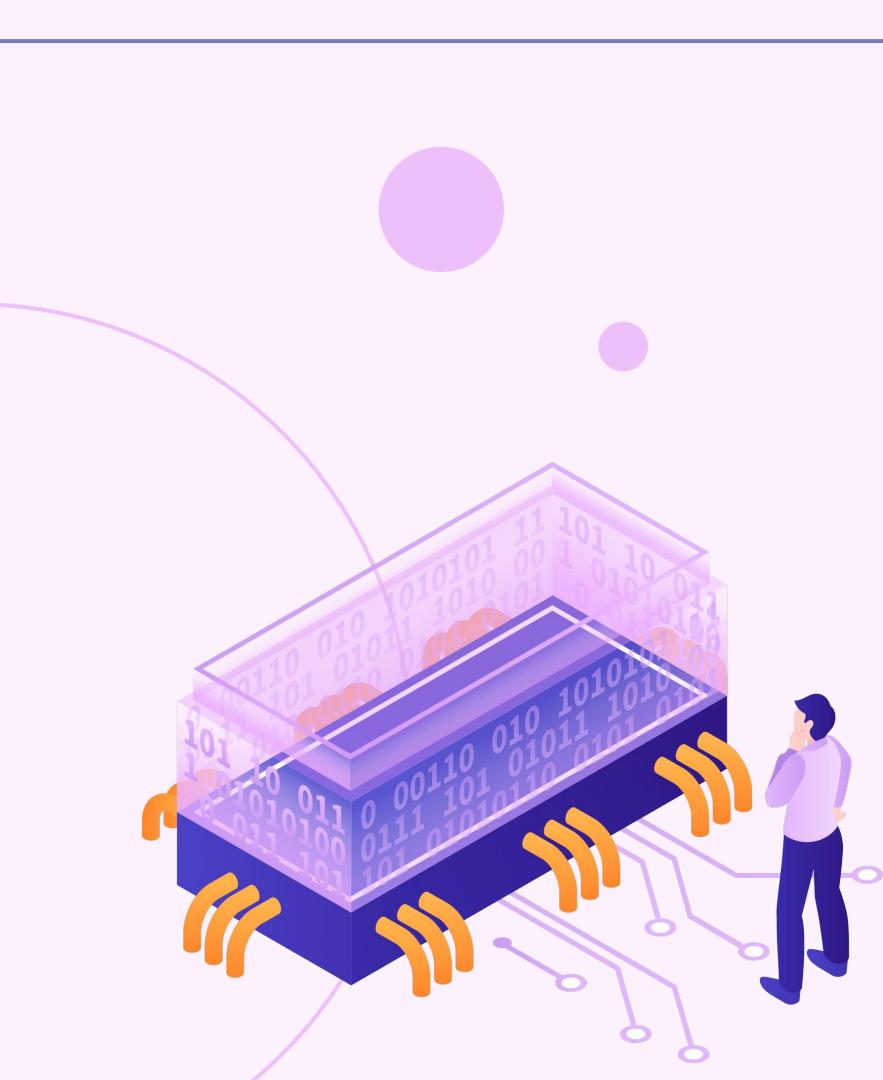
Business Intelligence
Dibimbing Batch 13

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

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

Table of contents

- | | | |
|---|---|--|
| 01
Title Slide | 02
Introduction | 03
Task Overview |
| 04
Designing the Stored Procedure | 05
Step 1: Transfer to Staging Tables | 06
Step 2: Load Dimension Tables |
| 07
Step 3: Load Fact Tables | 08
Documentation | 09
Benefits of Stored Procedure |
| 10
Execution and Testing | 11
Conclusion | |



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:

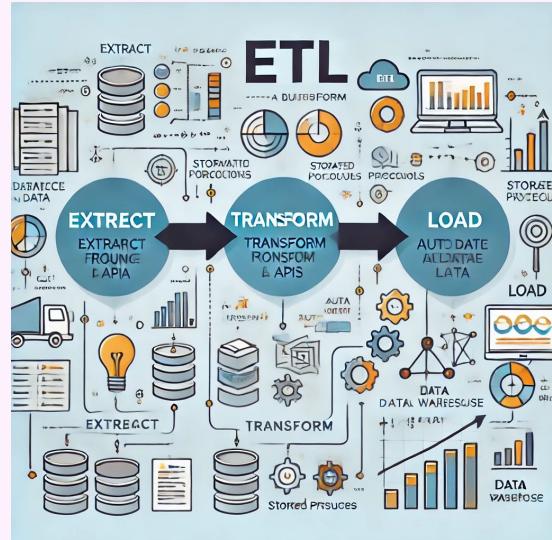
- To ensure data consistency, accuracy, and availability for decision-making.
- To integrate disparate data sources into a unified system for better insights.

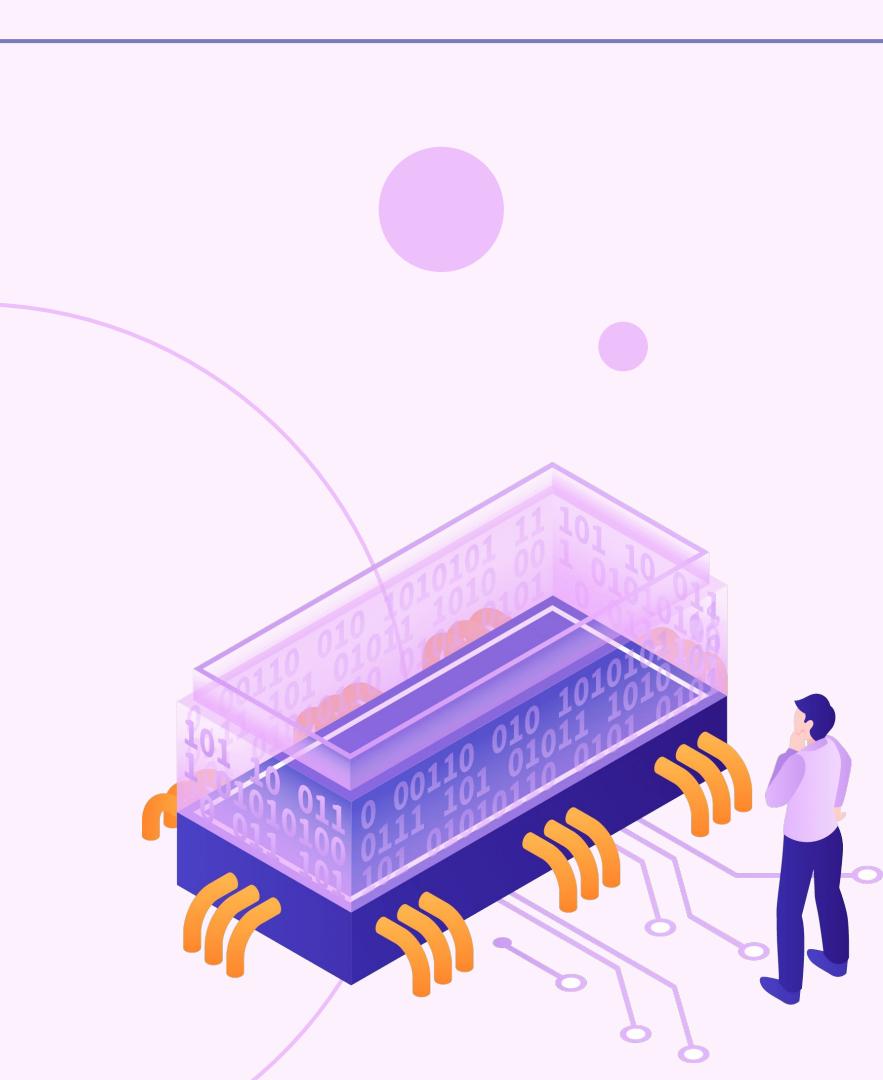
The goal

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

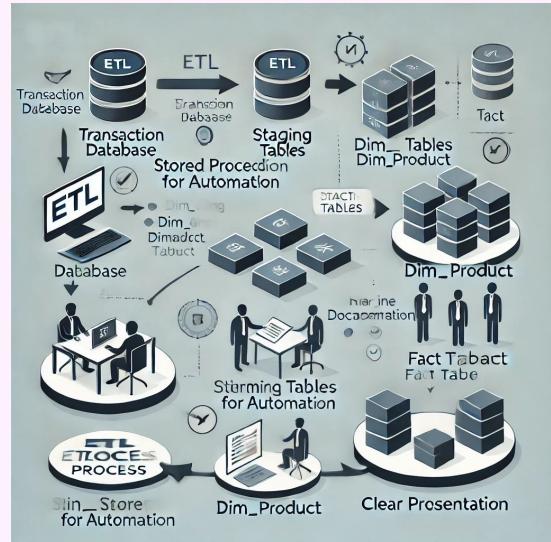
- **Objective:**
 - Design a stored procedure named `dwh.generate_sales()` to automate ETL processes.
- **Scope:**
 - Move data from a transactional database to staging tables.
 - Load data from staging tables into dimensional tables in the data warehouse.
 - Populate fact tables with processed data 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:**
 - Extract data from transaction tables and load it into staging tables.
 - Transform and organize the data into dimension tables (`dim_store`, `dim_product`, etc.).
 - Aggregate and load transformed data into a fact table (`fact_sales_transaction`).
- **Stored Procedure:**
 - Name: `dwh.generate_sales()`
 - Automate the ETL steps using SQL in PostgreSQL.
 - Include clear in-line documentation within the procedure to explain each step.
- **Presentation:**
 - Create a simple presentation that explains the steps you took to design the procedure.
 - 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

A screenshot of a database schema browser interface. On the left, there's a tree view of database objects. Under the 'Schemas' node, which is expanded, there are four entries: 'dm', 'dwh', 'public', and 'stg'. The 'stg' entry is highlighted with a light blue background. Other nodes like 'Languages', 'Publications', and 'Subscriptions' are also visible but not expanded.

```
1 -- Check if the schema exists, if not, create it
2 DO
3 $$
4 BEGIN
5 IF NOT EXISTS (
6     SELECT 1
7     FROM information_schema.schemata
8     WHERE schema_name = 'stg'
9 ) THEN
10     CREATE SCHEMA stg;
11 END IF;
12 END
13 $$;
```

Create stg schema

Schema - dwh

A screenshot of a database schema browser interface. On the left, there's a tree view of database objects. Under the 'Schemas' node, which is expanded, there are four entries: 'dm', 'dwh', 'public', and 'stg'. The 'dwh' entry is highlighted with a light blue background. Other nodes like 'Foreign Data Wrappers', 'Languages', and 'Subscriptions' are also visible but not expanded.

```
15 DO
16 $$
17 BEGIN
18 IF NOT EXISTS (
19     SELECT 1
20     FROM information_schema.schemata
21     WHERE schema_name = 'dwh'
22 ) THEN
23     CREATE SCHEMA dwh;
24 END IF;
25 END
26 $$;
```

Create dwh schema

Schema - dm

- > Foreign Data Wrappers
- > Languages
- > Publications
- > Schemas (4)
 - > dm
 - > dwh
 - > public
 - > stg
- > Subscriptions
- HW DWH

```
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

The screenshot shows the pgAdmin interface for a PostgreSQL database. On the left, a tree view lists the schema structure under 'public'. The 'Tables (1)' node is expanded, and 'sales_transaction' is selected. Under 'Columns (20)', several columns are listed with icons: sale_id, store_id, store_name, city, state, and country. To the right, a code editor displays the SQL command to create the table:

```
47 CREATE TABLE public.sales_transaction (
48     sale_id int4 NULL,
49     store_id int4 NULL,
50     store_name varchar(100) NULL,
51     city varchar(50) NULL,
52     state varchar(50) NULL,
53     country varchar(50) NULL,
54     sales_name_id int4 NULL,
55     sales_name varchar(100) NULL,
56     sales_age int4 NULL,
57     sales_gender varchar(10) NULL,
58     time_id int4 NULL,
59     "date" date NULL,
60     day_of_week varchar(10) NULL,
61     "month" varchar(20) NULL,
62     "year" int4 NULL,
63     product_id int4 NULL,
64     product_name varchar(100) NULL,
65     category varchar(50) NULL,
66     quantity int4 NULL,
67     price int4 NULL
68 );
```

Below the code editor, there are tabs for 'Data Output', 'Messages' (which is selected), and 'Notifications'. A status bar at the bottom shows 'INSERT 0 1000' and 'Total rows: 0 of 0 Query complete 00:00:00.360'.

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

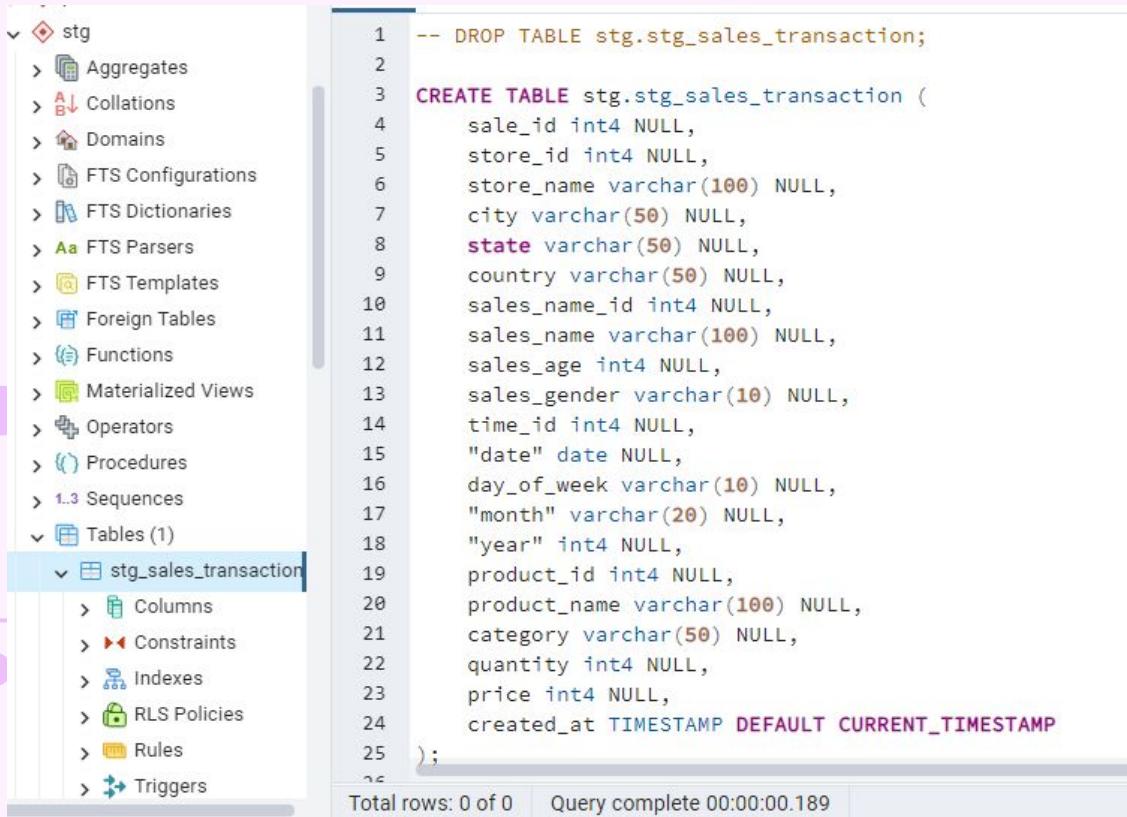
	sale_id	store_id	store_name	city	state	country	sales_name_id	sales_name	sales_age	sales_g
	integer	integer	character varying (100)	character varying (50)	character varying (50)	character varying (50)	integer	character varying (100)	integer	character
1		13	3 Starbucks Corner	Chicago	IL	USA	2	Jane Smith	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



The screenshot shows a database schema browser interface. On the left, a tree view lists the schema 'stg' with various objects: Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (1). Under 'Tables (1)', the table 'stg_sales_transaction' is selected, revealing its columns, constraints, indexes, RLS Policies, Rules, and Triggers.

```
-- 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

```
30 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  
31 (13,3,'Starbucks Corner','Chicago','IL','USA',2,'Jane Smith',28,'Female',7,'2024-01-07','Sunday','January',2024,10,'Mocha','Coffee',  
32 (12,2,'Downtown Starbucks','Los Angeles','CA','USA',7,'David Martinez',38,'Male',31,'2024-01-31','Wednesday','January',2024,5,'Crois  
33 (1,2,'Downtown Starbucks','Los Angeles','CA','USA',4,'Emily Brown',32,'Female',4,'2024-01-04','Thursday','January',2024,3,'Iced Gree  
34 (2,3,'Starbucks Corner','Chicago','IL','USA',9,'James Anderson',41,'Male',22,'2024-01-22','Monday','January',2024,7,'Blueberry Muffi  
35 (3,1,'Starbucks Central','New York','NY','USA',9,'James Anderson',41,'Male',30,'2024-01-30','Tuesday','January',2024,9,'Flat White',  
36 (4,1,'Starbucks Central','New York','NY','USA',10,'Olivia Taylor',34,'Female',18,'2024-01-18','Thursday','January',2024,6,'Pumpkin S  
37 (5,2,'Downtown Starbucks','Los Angeles','CA','USA',28,'Jane Smith',28,'Female',27,'2024-01-27','Saturday','January',2024,6,'Pumpkin S  
38 (6,2,'Downtown Starbucks','Los Angeles','CA','USA',3,'Michael Johnson',45,'Male',22,'2024-01-22','Monday','January',2024,8,'Chai Tea  
39 (7,2,'Downtown Starbucks','Los Angeles','CA','USA',1,'John Doe',35,'Male',25,'2024-01-25','Thursday','January',2024,8,'Chai Tea Latt  
40 (8,1,'Starbucks Central','New York','NY','USA',2,'Jane Smith',28,'Female',27,'2024-01-27','Saturday','January',2024,2,'Caramel Macch  
41 (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

1 SELECT * FROM stg.stg_sales_transaction

2

Data Output Messages Notifications

sale_id store_id store_name city state country sales_name_id sales_name sales_age sales_gend
1 13 Starbucks Corner Chicago IL USA 2 Jane Smith 28 Female
2 12 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

Table stg_sales_transactions

Create table dan isi tabel
berhasil dilakukan
Ada 1000 row

Schema - dwh - dim_product

The screenshot shows a database schema editor interface. On the left, a tree view displays the schema structure:

- Tables (5)
 - dim_product
 - Columns (4)
 - product_id
 - product_name
 - category
 - last_update

On the right, the SQL code for creating the table is shown:

```
-- Step 2: Insert or update into dim_product
-- Description: Insert new product data into the product dimension if they don't already exist.
--             Update existing product data if they already exist.
CREATE TABLE IF NOT EXISTS dwh.dim_product AS
SELECT DISTINCT src.product_id, src.product_name, src.category, CURRENT_TIMESTAMP AS last_update_date
FROM stg.stg_sales_transaction AS src;

TRUNCATE TABLE dwh.dim_product;

INSERT INTO dwh.dim_product (product_id, product_name, category, last_update)
SELECT DISTINCT src.product_id, src.product_name, src.category, CURRENT_TIMESTAMP AS last_update_date
FROM stg.stg_sales_transaction AS src;
```

Create table
dim_product berhasil

The screenshot shows a database query tool interface. On the left, a tree view displays the schema structure:

- Schemas (4)
 - dm
 - dwh
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Operators
 - Procedures
 - Sequences
 - Tables (5)
 - dim_product
 - Columns (4)

In the center, a query window shows the following SQL statement:

```
1 SELECT * FROM dwh.dim_product
```

Below the query window, a data grid displays the contents of the dim_product table:

	product_id	product_name	category	last_update
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

The screenshot shows a database schema browser with the following details:

- Tables (5):** dim_product, dim_sales_name, dim_store.
- dim_store Columns (6):** store_id, store_name, city, state, country, last_update.
- Script:** A SQL script for creating the dim_store table and inserting data from a source table (stg_stg_sales_transaction). It includes a step for inserting or updating existing data.

```
-- 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;
```

Create table
dim_store
berhasil

The screenshot shows a database interface with the following details:

- Tables (5):** dim_product, dim_sales_name, dim_store, dim_time.
- dim_time Columns (6):** time_id, date, day_of_week, month, year, last_update.
- Data Output:** A table showing the data for the dim_time table.

	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

The screenshot shows a database schema editor with the following details:

- Tables (5):** dim_product, dim_sales_name, dim_store, dim_time.
- dim_time Columns (6):** time_id, date, day_of_week, month, year, last_update.
- SQL Script:**

```
-- Step 4: Insert or update into dim_time
-- Description: Insert new time data into the time dimension if they don't already exist.
--             Update existing time data if they already exist.
CREATE TABLE IF NOT EXISTS dwh.dim_time AS
SELECT DISTINCT src.time_id, src.date, src.day_of_week, src.month, src.year, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;

TRUNCATE TABLE dwh.dim_time;

INSERT INTO dwh.dim_time (time_id, date, day_of_week, month, year, last_update)
SELECT DISTINCT src.time_id, src.date, src.day_of_week, src.month, src.year, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;
```

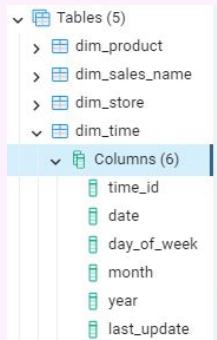
Create table
dim_store
berhasil

The screenshot shows a database query tool with the following details:

- Tables (5):** dim_product, dim_sales_name, dim_store, dim_time.
- dim_time Columns (6):** time_id, date, day_of_week, month, year, last_update.
- Query Result:**

	time_id	date	day_of_week	month	year	last_update
	integer	date	character varying (10)	character varying (20)	integer	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

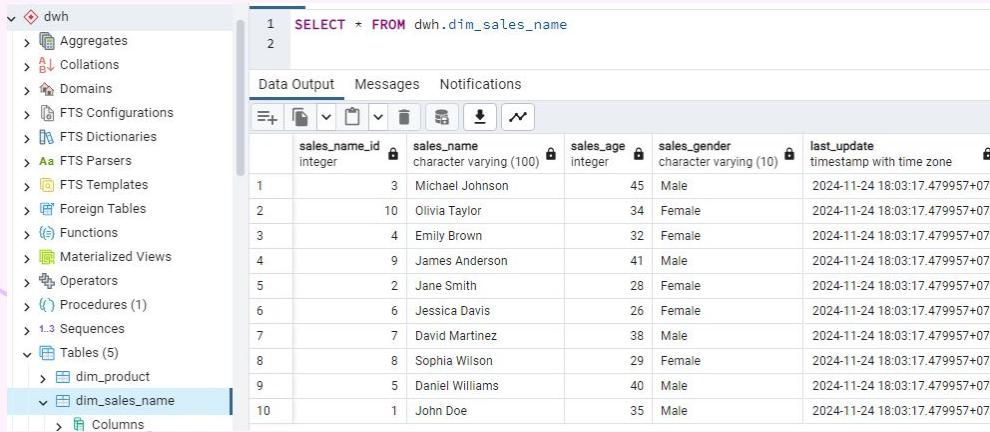


```
-- Step 4: Insert or update into dim_time
-- Description: Insert new time data into the time dimension if they don't already exist.
--             Update existing time data if they already exist.
CREATE TABLE IF NOT EXISTS dwh.dim_time AS
SELECT DISTINCT src.time_id, src.date, src.day_of_week, src.month, src.year, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;

TRUNCATE TABLE dwh.dim_time;

INSERT INTO dwh.dim_time (time_id, date, day_of_week, month, year, last_update)
SELECT DISTINCT src.time_id, src.date, src.day_of_week, src.month, src.year, CURRENT_TIMESTAMP AS last_update
FROM stg.stg_sales_transaction AS src;
```

Create table
dim_store
berhasil



1 SELECT * FROM dwh.dim_sales_name

2

Data Output Messages Notifications

	sales_name_id	sales_name	sales_age	sales_gender	last_update
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

The screenshot shows the schema for the `dm_sales_by_store` table. It has 8 columns: `store_id`, `store_name`, `city`, `state`, `country`, `total_quantity`, `total_sales_amount`, and `created_at`. The creation script for the table is as follows:

```
1 --1. Tabel dm.dm_sales_by_store
2 --Tabel ini berisi total penjualan berdasarkan toko untuk analisis performa toko.
3 CREATE TABLE IF NOT EXISTS dm.dm_sales_by_store AS
4 SELECT
5     s.store_id,
6     s.store_name,
7     s.city,
8     s.state,
9     s.country,
10    SUM(f.quantity) AS total_quantity,
11    SUM(f.total_sales_amount) AS total_sales_amount,
12    CURRENT_TIMESTAMP AS created_at
13   FROM dwh.fact_sales_transaction f
14  JOIN dwh.dim_store s ON f.store_id = s.store_id
15 GROUP BY s.store_id, s.store_name, s.city, s.state, s.country;
```

Create table
dm_sales_by_
store

The screenshot shows the data output for the `dm_sales_by_store` table. The results are as follows:

	store_id	store_name	city	state	country	total_quantity	total_sales_amount	crea
1	1	Starbucks Central	New York	NY	USA	952	9413	202
2	3	Starbucks Corner	Chicago	IL	USA	1080	10474	202
3	2	Downtown Starbucks	Los Angeles	CA	USA	969	9767	202

Schema - dm - dm_sales_by_product

Tables (5)

dm_sales_by_product

Columns (6)

- product_id
- product_name
- category
- total_quantity
- total_sales_amount
- created_at

Constraints

Indexes

RLS Policies

```
35
36 --2. Tabel dm.dm_sales_by_product
37 --Tabel ini menganalisis total penjualan berdasarkan produk
38
39 CREATE TABLE IF NOT EXISTS dm.dm_sales_by_product AS
40
41     SELECT
42         p.product_id,
43         p.product_name,
44         p.category,
45         SUM(f.quantity) AS total_quantity,
46         SUM(f.total_sales_amount) AS total_sales_amount,
47         CURRENT_TIMESTAMP AS created_at
48     FROM dwh.fact_sales_transaction f
49     JOIN dwh.dim_product p ON f.product_id = p.product_id
50     GROUP BY p.product_id, p.product_name, p.category;
```

Create table
dm_sales_by_
product

Tables (5)

dm_sales_by_product

Columns (6)

- product_id
- product_name
- category
- total_quantity
- total_sales_amount
- created_at

Constraints

Indexes

RLS Policies

Rules

Triggers

dm_sales_by_salesperson

dm_sales_by_store

dm_sales_by_time

1 SELECT * FROM dm.dm_sales_by_product

Data Output Messages Notifications

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

Schema - dm - dm_sales_by_time

Tables (5)

- dm_sales_by_product
- dm_sales_by_salesperson
- dm_sales_by_store
- dm_sales_by_time

Columns (8)

time_id	date	day_of_week	month	year	total_quantity	total_sales_amount	created_at
1	2024-01-31	Wednesday	January	2024	101	1065	2024-11-24 14:51:56.31632
2	2024-01-26	Friday	January	2024	99	1061	2024-11-24 14:51:56.31632
3	2024-01-02	Tuesday	January	2024	116	1216	2024-11-24 14:51:56.31632
4	2024-01-29	Monday	January	2024	118	1224	2024-11-24 14:51:56.31632
5	2024-01-08	Monday	January	2024	119	1190	2024-11-24 14:51:56.31632
6	2024-01-27	Saturday	January	2024	103	971	2024-11-24 14:51:56.31632
7	2024-01-28	Sunday	January	2024	66	618	2024-11-24 14:51:56.31632
8	2024-01-23	Tuesday	January	2024	121	1165	2024-11-24 14:51:56.31632

```
65
66 --3. Tabel dm.dm_sales_by_time
67 --Tabel ini berisi total penjualan berdasarkan periode waktu untuk analisis tren.
68 CREATE TABLE IF NOT EXISTS dm.dm_sales_by_time AS
69
70     t.time_id,
71     t.date,
72     t.day_of_week,
73     t.month,
74     t.year,
75     SUM(f.quantity) AS total_quantity,
76     SUM(f.total_sales_amount) AS total_sales_amount,
77     CURRENT_TIMESTAMP AS created_at
78
79 FROM dwh.fact_sales_transaction f
80 JOIN dwh.dim_time t ON f.time_id = t.time_id
81 GROUP BY t.time_id, t.date, t.day_of_week, t.month, t.year;
```

Create table
dm_sales_by_ time

Tables (5)

- dm_sales_by_product
- dm_sales_by_salesperson
- dm_sales_by_store
- dm_sales_by_time

Columns (8)

time_id	date	day_of_week	month	year	total_quantity	total_sales_amount	created_at
1	2024-01-31	Wednesday	January	2024	101	1065	2024-11-24 14:51:56.31632
2	2024-01-26	Friday	January	2024	99	1061	2024-11-24 14:51:56.31632
3	2024-01-02	Tuesday	January	2024	116	1216	2024-11-24 14:51:56.31632
4	2024-01-29	Monday	January	2024	118	1224	2024-11-24 14:51:56.31632
5	2024-01-08	Monday	January	2024	119	1190	2024-11-24 14:51:56.31632
6	2024-01-27	Saturday	January	2024	103	971	2024-11-24 14:51:56.31632
7	2024-01-28	Sunday	January	2024	66	618	2024-11-24 14:51:56.31632
8	2024-01-23	Tuesday	January	2024	121	1165	2024-11-24 14:51:56.31632

1 SELECT * FROM dm.dm_sales_by_time

Data Output Messages Notifications

Schema - dm - dm_sales_by_salesperson

The screenshot shows a database interface with the following details:

- Tables (5)**:
 - dm_sales_by_product**
 - dm_sales_by_salesperson** (selected):
 - Columns (7)**: sales_name_id, sales_name, sales_age, sales_gender, total_quantity, total_sales_amount, created_at

```
100 --4. Tabel dm.dm_sales_by_salesperson
101 --Tabel ini berisi performa tim penjualan, total transaksi, dan total penjualan yang dihasilkan setiap penjual.
102 CREATE TABLE IF NOT EXISTS dm.dm_sales_by_salesperson AS
103 SELECT
104     sp.sales_name_id,
105     sp.sales_name,
106     sp.sales_age,
107     sp.sales_gender,
108     SUM(f.quantity) AS total_quantity,
109     SUM(f.total_sales_amount) AS total_sales_amount,
110     CURRENT_TIMESTAMP AS created_at
111 FROM dwh.fact_sales_transaction f
112 JOIN dwh.dim_sales_name sp ON f.sales_name_id = sp.sales_name_id
113 GROUP BY sp.sales_name_id, sp.sales_name, sp.sales_age, sp.sales_gender;
```

Create table
dm_sales_by_
salesperson

The screenshot shows a database interface with the following details:

- Tables (5)**:
 - dm_sales_by_product**
 - dm_sales_by_salesperson** (selected):
 - Columns (7)**: sales_name_id, sales_name, sales_age, sales_gender, total_quantity, total_sales_amount, created_at

1 SELECT * FROM dm.dm_sales_by_salesperson

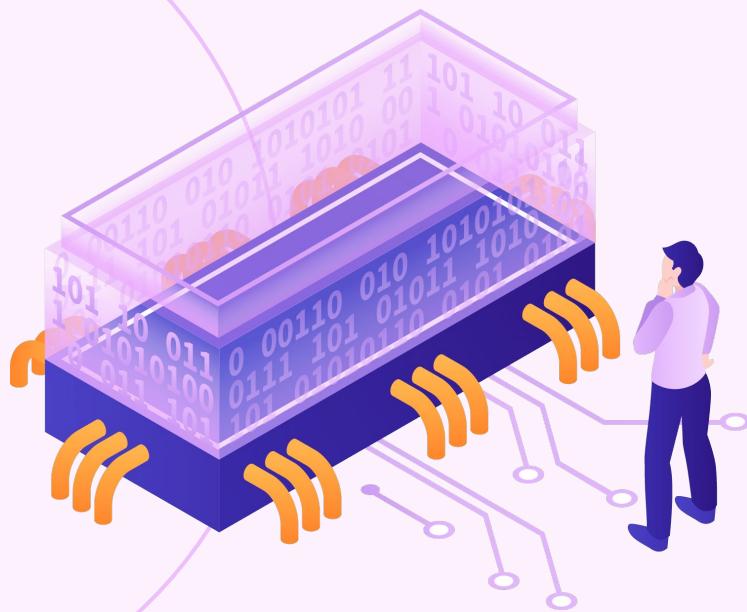
	sales_name_id	sales_name	sales_age	sales_gender	total_quantity	total_sales_amount	created_at
1	3	Michael Johnson	45	Male	278	2786	2024-11-24 14:52:31.78087+07
2	10	Olivia Taylor	34	Female	232	2317	2024-11-24 14:52:31.78087+07
3	4	Emily Brown	32	Female	310	3019	2024-11-24 14:52:31.78087+07
4	9	James Anderson	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

```
132 --5. Tabel dm.dim_sales_summary
133 --Tabel ringkasan lengkap dari penjualan yang menggabungkan informasi toko, waktu, produk, dan penjual.
134 CREATE TABLE IF NOT EXISTS dm.dim_sales_summary AS
135 SELECT
136     f.sale_id,
137     s.store_name,
138     t.date,
139     t.month,
140     t.year,
141     p.product_name,
142     p.category,
143     sp.sales_name,
144     f.quantity,
145     f.price,
146     f.total_sales_amount,
147     CURRENT_TIMESTAMP AS created_at
148 FROM dwh.fact_sales_transaction f
149 JOIN dwh.dim_store s ON f.store_id = s.store_id
150 JOIN dwh.dim_time t ON f.time_id = t.time_id
151 JOIN dwh.dim_product p ON f.product_id = p.product_id
152 JOIN dwh.dim_sales_name sp ON f.sales_name_id = sp.sales_name_id;
```

Create table dm_sales_summary

1 SELECT * FROM dm.dim_sales_summary										
Data Output Messages Notifications										
	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 John		
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

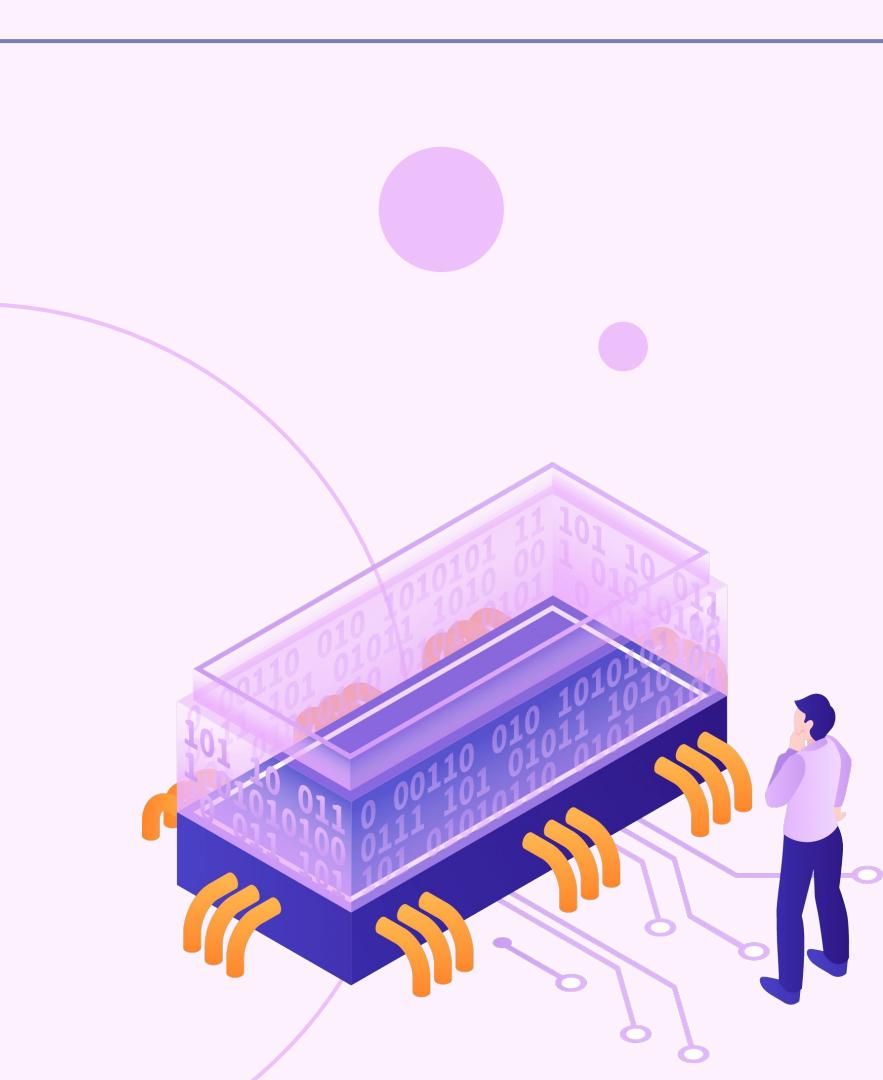
Untuk menyimpan data yang telah diproses, diorganisasi, dan disusun untuk keperluan analisis jangka panjang.

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

```
1 -- Sample Store Procedure
2 CREATE OR REPLACE PROCEDURE dwh.generate_sales()
3 LANGUAGE plpgsql
4 AS $procedure$
5 BEGIN
6     -- Step 1: Truncate and Insert into stg SCD Type 1
7     -- Description: Clear the staging table and populate it with data from the source table.
8     TRUNCATE TABLE stg.stg_sales_transaction;
9     INSERT INTO stg.stg_sales_transaction
10    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

```
26 -- Step 3: Insert or update into dim_store
27 -- Description: Insert new store data into the store dimension if they don't already exist.
28 --           Update existing store data if they already exist.
29 CREATE TABLE IF NOT EXISTS dwh.dim_store AS
30 SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
31 FROM stg.stg_sales_transaction AS src;
32
33 TRUNCATE TABLE dwh.dim_store;
34
35 INSERT INTO dwh.dim_store (store_id, store_name, city, state, country, last_update)
36 SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
37 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

```
26 -- Step 3: Insert or update into dim_store
27 -- Description: Insert new store data into the store dimension if they don't already exist.
28 --           Update existing store data if they already exist.
29 CREATE TABLE IF NOT EXISTS dwh.dim_store AS
30 SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
31 FROM stg.stg_sales_transaction AS src;
32
33 TRUNCATE TABLE dwh.dim_store;
34
35 INSERT INTO dwh.dim_store (store_id, store_name, city, state, country, last_update)
36 SELECT DISTINCT src.store_id, src.store_name, src.city, src.state, src.country, CURRENT_TIMESTAMP AS last_update
37 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.

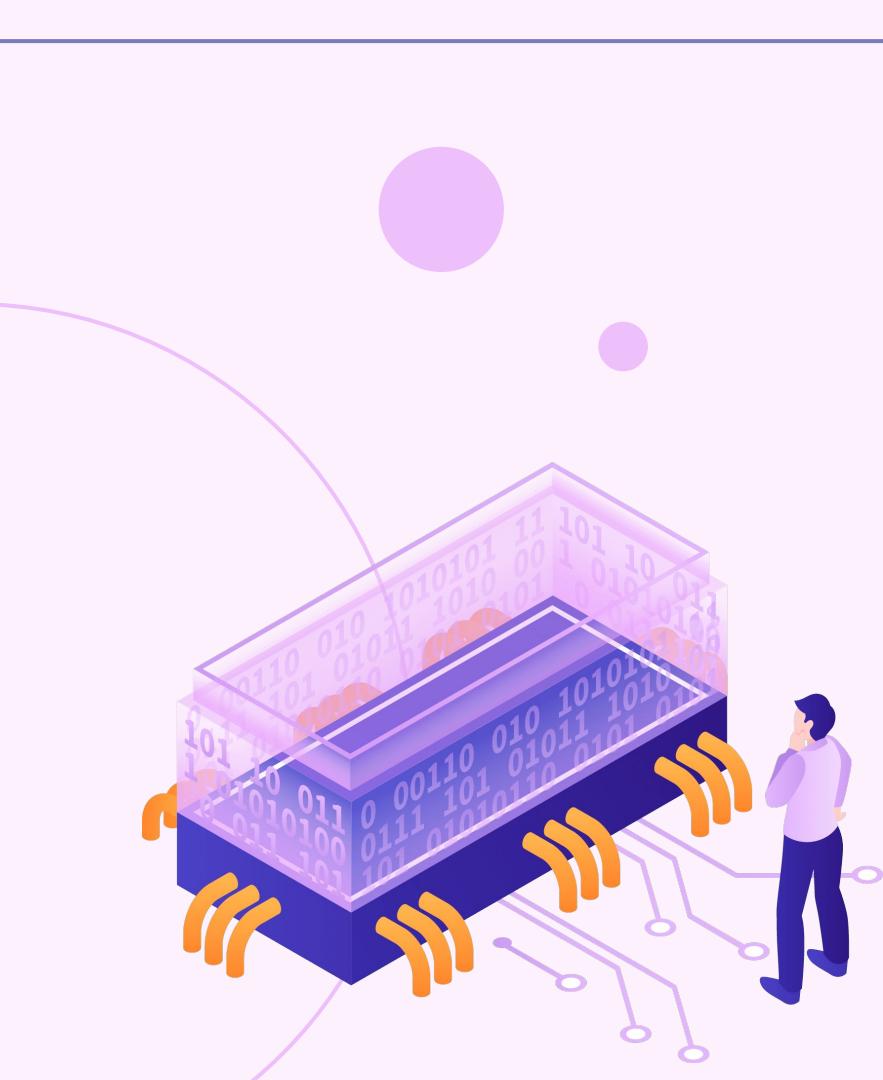
Store Procedure

```
54 -- Step 5: Insert or update into dim_sales_name
55 -- Description: Insert new sales name data into the sales name dimension if they don't already exist.
56 --           Update existing sales name data if they already exist.
57 CREATE TABLE IF NOT EXISTS dwh.dim_sales_name AS
58 SELECT DISTINCT src.sales_name_id, src.sales_name, src.sales_age, src.sales_gender, CURRENT_TIMESTAMP AS last_update
59 FROM stg.stg_sales_transaction AS src;
60
61 TRUNCATE TABLE dwh.dim_sales_name;
62
63 INSERT INTO dwh.dim_sales_name (sales_name_id, sales_name, sales_age, sales_gender, last_update)
64 SELECT DISTINCT src.sales_name_id, src.sales_name, src.sales_age, src.sales_gender, CURRENT_TIMESTAMP AS last_update
65 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

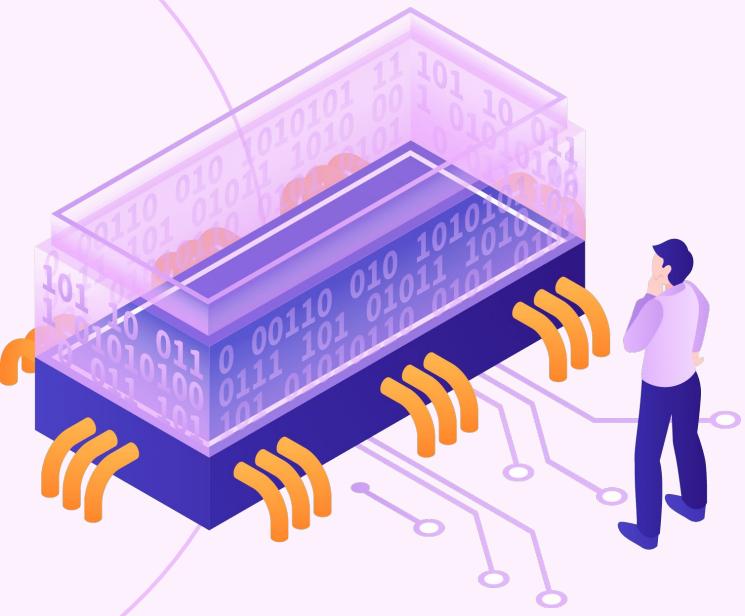
```
67  -- Step 6: Insert into fact SCD Type 2
68  -- Description: Insert new sales transactions into the fact table if they don't already exist.
69  INSERT INTO dwh.fact_sales_transaction
70    (sale_id, store_id, sales_name_id, product_id, time_id, date, quantity, price)
71  SELECT
72    src.sale_id,
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`):

- Insert New Records:** Adds unique sales data from `stg.stg_sales_transaction`.
- Avoid Duplicates:** Ensures no duplicate entries using `WHERE NOT EXISTS`.
- 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.dim_sales_by_salesperson;
170
171    -- Memasukkan data terbaru
172    INSERT INTO dm.dim_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

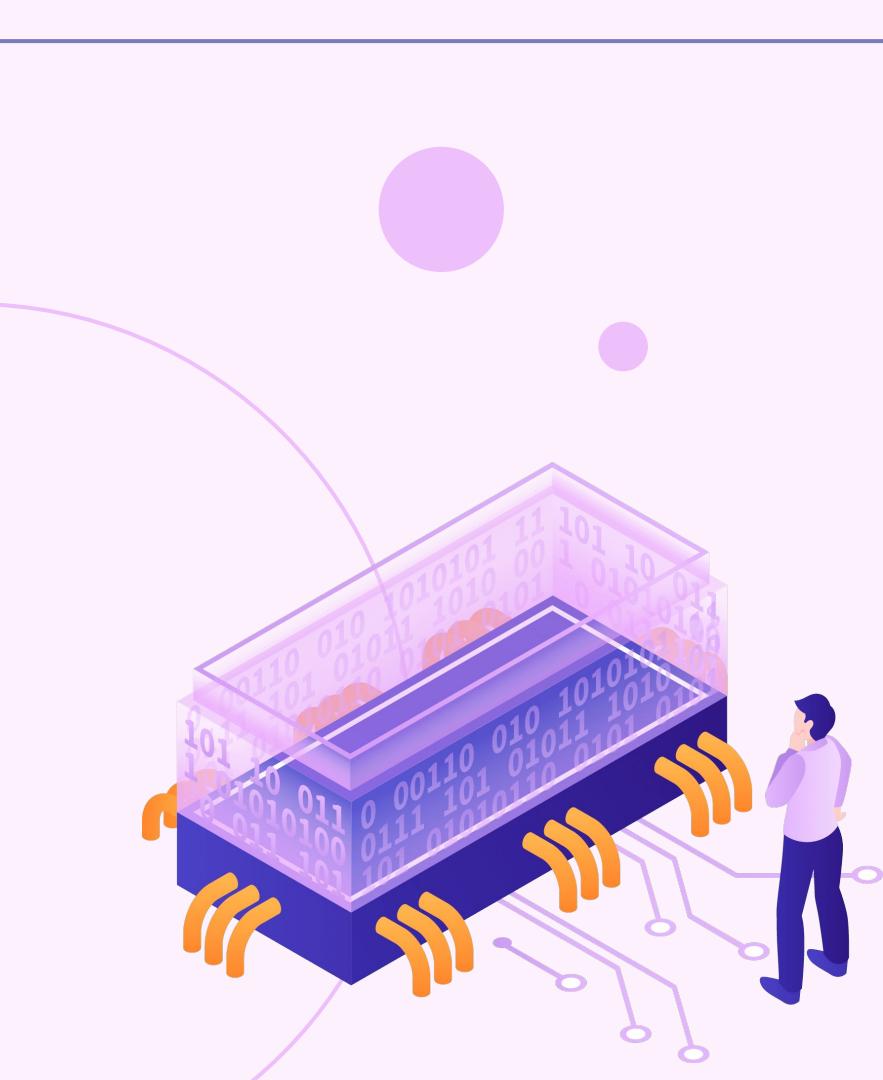
```
188    -- Membersihkan data lama
189    TRUNCATE TABLE dm.dim_sales_summary;
190
191    -- Memasukkan data terbaru
192    INSERT INTO dm.dim_sales_summary
193        SELECT
194            f.sale_id,
195            s.store_name,
196            t.date,
197            t.month,
198            t.year,
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

The screenshot shows the pgAdmin 4 interface. The left pane is the Object Explorer, displaying a tree structure of database objects under the 'dwh' schema, including Procedures (1) named 'generate_sales'. The right pane is the SQL editor, showing a query window titled 'HW BI Store Procedure/postgres@PostgreSQL 16'. The query itself is a CREATE OR REPLACE PROCEDURE statement:

```
1 -- Sample Store Procedure
2 CREATE OR REPLACE PROCEDURE dwh.generate_sales()
3 LANGUAGE plpgsql
4 AS $procedure$
5 BEGIN
6     -- Step 1: Truncate and Insert into stg SCD Type 1
7     -- Description: Clear the staging table and populate it with data from the source table.
8     TRUNCATE TABLE stg.stg_sales_transaction;
9     INSERT INTO stg.stg_sales_transaction
10    SELECT * FROM public.sales_transaction;
11
```

Below the query, the status bar indicates 'CREATE PROCEDURE'. A message box at the bottom right states 'Store Procedures created successfully'.

Store Procedure

pgAdmin 4

File Object Tools Help

Object Explorer

dwh

- Aggregates
- Collations
- Domains
- FTS Configurations
- FTS Dictionaries
- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Operators
- Procedures (1)
- generate_sales
- Sequences
- Tables
- Trigger Functions
- Types
- Views

public

- Aggregates
- Collations
- Domains
- FTS Configurations
- FTS Dictionaries

Properties SQL stg.sql dm.sql dwh-dim-fact.sql Store Procedure.sql

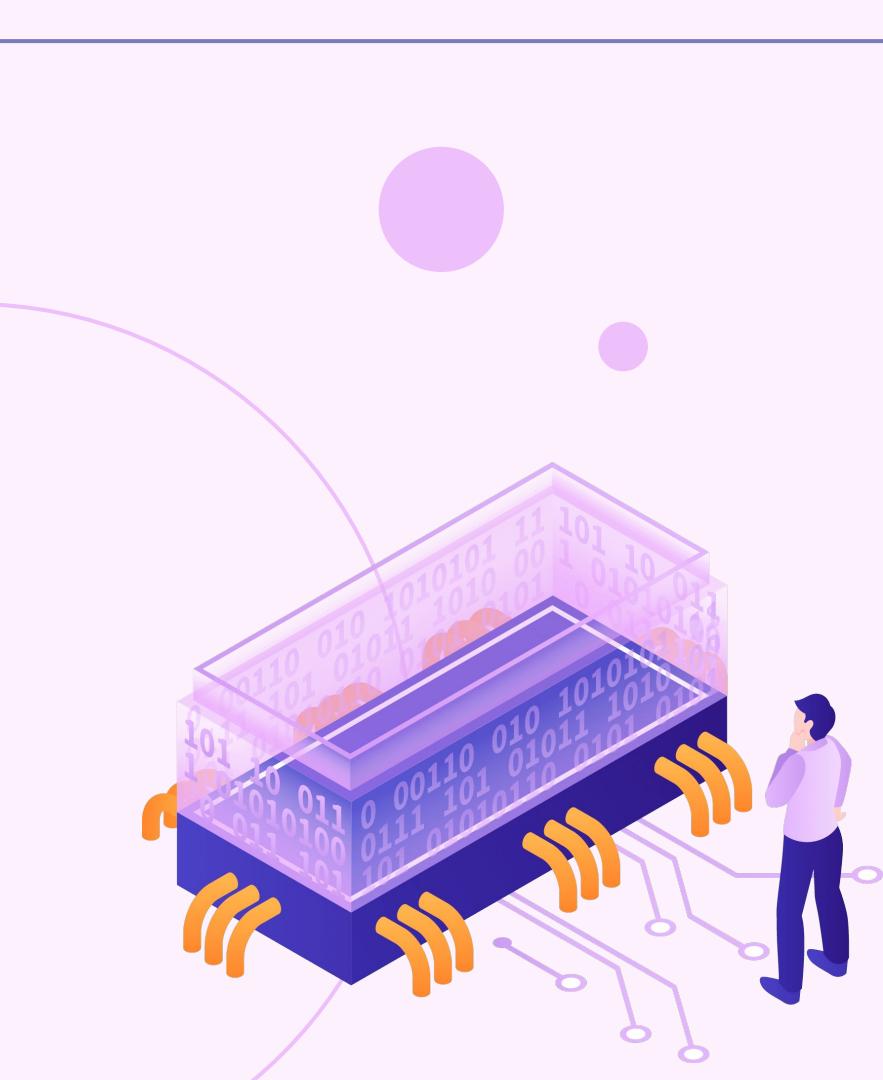
generate_sales

General Definition Code Options Parameters Security SQL

```
1 BEGIN
2 -- Step 1: Truncate and Insert into stg SCD Type 1
3 -- Description: Clear the staging table and populate it with data from the source table.
4 TRUNCATE TABLE stg.stg_sales_transaction;
5 INSERT INTO stg.stg_sales_transaction
6 SELECT * FROM public.sales_transaction;
7
8 -- Step 2: Insert or update into dim_product
9 -- Description: Insert new product data into the product dimension if they do not already exist.
10 -- Update existing product data if they already exist.
11 CREATE TABLE IF NOT EXISTS dwh.dim_product AS
12 SELECT DISTINCT src.product_id, src.product_name, src.category, CURRENT_DATE AS last_update
13 FROM stg.stg_sales_transaction AS src;
14
15 TRUNCATE TABLE dwh.dim_product;
16
17 INSERT INTO dwh.dim_product (product_id, product_name, category, last_update)
18 SELECT DISTINCT src.product_id, src.product_name, src.category, CURRENT_DATE AS last_update
19 FROM stg.stg_sales_transaction AS src;
20
21
22
```

i ?

Store Procedures created successfully



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.

5. Error Handling

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

2. Consistency and Reusability

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

6. Reduction of Network Traffic

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

3. Enhanced Security

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

7. Scalability

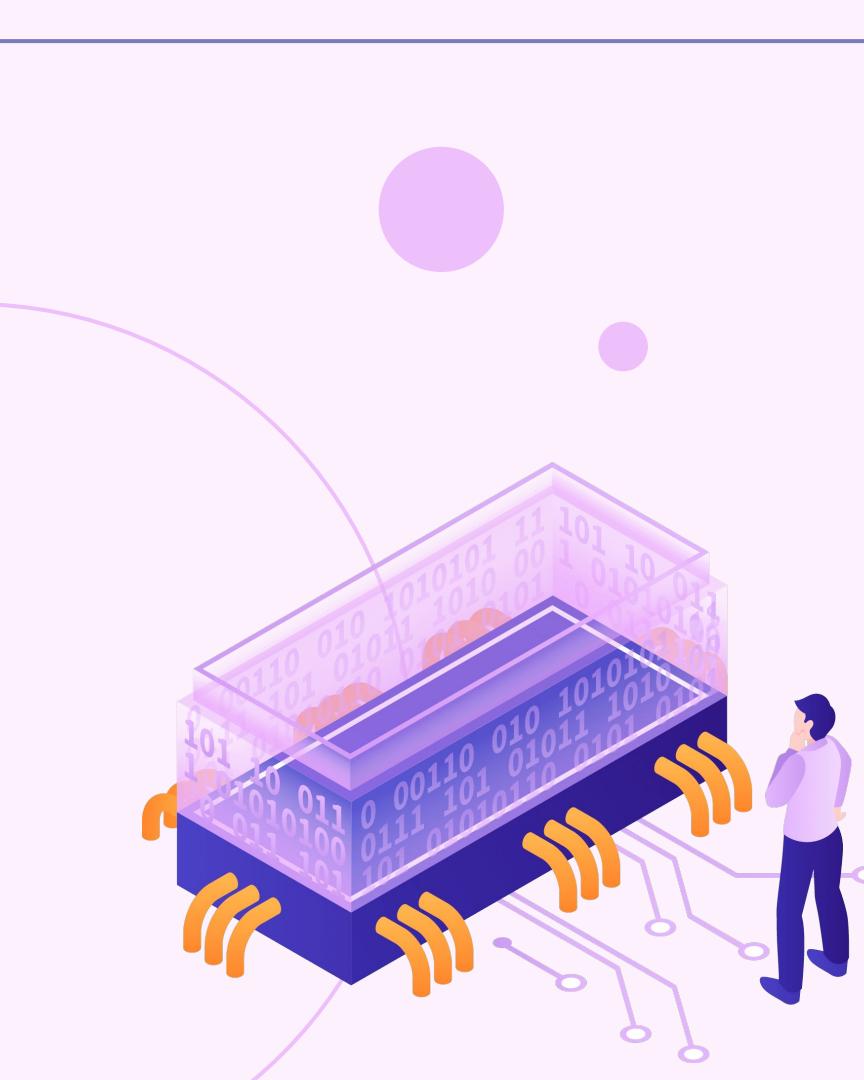
Efficiently handles complex operations and large datasets, making it suitable for enterprise-level applications.

4. Maintainability

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

8. Improved Data Integrity

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



A stylized illustration of a computer chip. The chip is purple and blue, with orange heat sinks attached to its bottom. Binary code (0s and 1s) is visible on the surface of the chip. A small figure of a person in a pink shirt and blue pants stands to the right of the chip, looking up at it with a thoughtful expression. The background is white with some abstract purple shapes and dots.

10

Execution and Testing

Initial Data in public

The screenshot shows the pgAdmin 4 interface. The left sidebar is the Object Explorer, showing databases dwh, public, and stg. Under stg, there are various objects like Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables. The Tables node is expanded, and the stg_sales_transaction table is selected, indicated by a blue border. The main pane shows the table structure with columns: sale_id, store_id, store_name, city, state, country, sales_name_id, and sales_name. Below the structure is a data grid containing 1,000 rows of transaction data. The top menu bar includes File, Object, Tools, Help, Properties, SQL, stg.sql, dm.sql, dwh-dim-fact.sql, Store Procedure..., Update data, and the current tab, stg.stg_sales_transaction/HW BI Stor.

sale_id	store_id	store_name	city	state	country	sales_name_id	sales_name
987	987	Starbucks Corner	Chicago	IL	USA	4	Emily Brown
988	988	Starbucks Central	New York	NY	USA	5	Daniel William
989	989	Downtown Starbucks	Los Angeles	CA	USA	5	Daniel William
990	990	Starbucks Central	New York	NY	USA	6	Jessica Davis
991	991	Starbucks Corner	Chicago	IL	USA	1	John Doe
992	992	Starbucks Central	New York	NY	USA	4	Emily Brown
993	993	Starbucks Corner	Chicago	IL	USA	5	Daniel William
994	994	Starbucks Central	New York	NY	USA	1	John Doe
995	995	Starbucks Corner	Chicago	IL	USA	3	Michael Johns
996	996	Starbucks Central	New York	NY	USA	6	Jessica Davis
997	997	Starbucks Corner	Chicago	IL	USA	1	John Doe
998	998	Starbucks Corner	Chicago	IL	USA	9	James Anders
999	999	Starbucks Central	New York	NY	USA	7	David Martine
1000	1000	Starbucks Central	New York	NY	USA	5	Daniel William

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

Updated Data in Public

The screenshot shows the pgAdmin 4 interface. The left sidebar is the Object Explorer, displaying various database objects like Functions, Materialized Views, Operators, Procedures, Sequences, and Tables. Under Tables, there is a expanded entry for 'sales_transaction' which contains 20 columns: 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, and quantity.

The main pane shows a query editor with the following content:

```
--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);
```

The Data Output tab shows the result of the INSERT statement:

```
INSERT 0 1
```

The message bar at the bottom indicates: "Query returned successfully in 115 msec."

sale_id	store_id	store_name	city	state	country	sales_name_id	sales_name
987	1	Starbucks Central	New York	NY	USA	4	Emily Brown
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 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

The screenshot shows the pgAdmin 4 interface. The left sidebar, titled 'Object Explorer', displays the database structure under the 'stg' schema, including tables like 'stg_sales_transaction'. The main pane shows a query editor with the following SQL query:

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

The results pane displays a table with 1,000 rows of data from the 'stg_sales_transaction' table. The columns and their data types are:

sale_id	store_id	store_name	city	state	country	sales_name_id	sales_name
987	987	Starbucks Corner	Chicago	IL	USA	4	Emily Brown
988	988	Starbucks Central	New York	NY	USA	5	Daniel William
989	989	Downtown Starbucks	Los Angeles	CA	USA	5	Daniel William
990	990	Starbucks Central	New York	NY	USA	6	Jessica Davis
991	991	Starbucks Corner	Chicago	IL	USA	1	John Doe
992	992	Starbucks Central	New York	NY	USA	4	Emily Brown
993	993	Starbucks Corner	Chicago	IL	USA	5	Daniel William
994	994	Starbucks Central	New York	NY	USA	1	John Doe
995	995	Starbucks Corner	Chicago	IL	USA	3	Michael Johns
996	996	Starbucks Central	New York	NY	USA	6	Jessica Davis
997	997	Starbucks Corner	Chicago	IL	USA	1	John Doe
998	998	Starbucks Corner	Chicago	IL	USA	9	James Anders
999	999	Starbucks Central	New York	NY	USA	7	David Martine
1000	1000	Starbucks Central	New York	NY	USA	5	Daniel William

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

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer pane lists various database objects under the schema 'stg'. In the center, the main window displays a query editor with the following content:

```
11 --Calling the stored procedure to generate the datawarehouse
12 CALL dwh.generate_sales();
13
14
```

The 'Data Output' tab is selected, showing the results of the query:

store_id	store_name	city	state	country	sales_name_id	sales_name
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

On the right, the 'Messages' tab shows the following log entries:

```
NOTICE: relation "dim_product" already exists, skipping
NOTICE: relation "dim_store" already exists, skipping
NOTICE: relation "dim_time" already exists, skipping
NOTICE: relation "dim_sales_name" already exists, skipping
CALL
```

A message at the bottom indicates: "Query returned successfully in 196 msec."

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

- ▶ Procedures (1)
 - { } generate_sales
- > 1..3 Sequences
- > Tables
- > Trigger Functions
- > Types
- > Views
- ❖ public
- > Aggregates
- > A↓ Collations

```
11 --Calling the stored procedure to generate the datawarehouse
12 CALL dwh.generate_sales();
13
```

Data Output Messages Notifications

```
NOTICE: relation "dim_product" already exists, skipping
NOTICE: relation "dim_store" already exists, skipping
NOTICE: relation "dim_time" already exists, skipping
NOTICE: relation "dim_sales_name" already exists, skipping
CALL
```

Query returned successfully in 196 msec.

Before

AFTER

	SELECT * FROM dwh.dim_sales_name				
	Data Output Messages Notifications				
	sales_name_id	sales_name	sales_age	sales_gender	last_update
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

- ❖ Schemas (4)
 - ❖ dm
 - ❖ dwh
 - > Aggregates
 - > A↓ Collations
 - > Domains
 - > FTS Configurations
 - > FTS Dictionaries
 - > A↓ FTS Parsers
 - > FTS Templates
 - > Foreign Tables
 - > Functions
 - > Materialized Views
 - > Operators
 - > Procedures (1)
 - > 1..3 Sequences
- > Tables (5)
 - > dim_product
 - > dim_sales_name
 - > Columns
 - > Constraints
 - > Columns

	SELECT * FROM dwh.dim_sales_name				
	Data Output Messages Notifications				
	sales_name_id	sales_name	sales_age	sales_gender	last_update
1	3	Michael Johnson	45	Male	2024-11-24 18:04:31.695846+07
2	10	Olivia Taylor	34	Female	2024-11-24 18:04:31.695846+07
3	4	Emily Brown	32	Female	2024-11-24 18:04:31.695846+07
4	9	James Anderson	41	Male	2024-11-24 18:04:31.695846+07
5	2	Jane Smith	28	Female	2024-11-24 18:04:31.695846+07
6	6	Jessica Davis	26	Female	2024-11-24 18:04:31.695846+07
7	7	David Martinez	38	Male	2024-11-24 18:04:31.695846+07
8	8	Sophia Wilson	29	Female	2024-11-24 18:04:31.695846+07
9	5	Daniel Williams	40	Male	2024-11-24 18:04:31.695846+07
10	11	Hijri Della Wirasti	31	Female	2024-11-24 18:04:31.695846+07
11	1	John Doe	35	Male	2024-11-24 18:04:31.695846+07

Data dwh

dim_sales_name

- ▶ Procedures (1)
 - { } generate_sales
- > 1..3 Sequences
- > Tables
- > Trigger Functions
- > Types
- > Views
- ❖ public
- > Aggregates
- > A↓ Collations

```
11 --Calling the stored procedure to generate the datawarehouse
12 CALL dwh.generate_sales();
13
```

Data Output Messages Notifications

```
NOTICE: relation "dim_product" already exists, skipping
NOTICE: relation "dim_store" already exists, skipping
NOTICE: relation "dim_time" already exists, skipping
NOTICE: relation "dim_sales_name" already exists, skipping
CALL
```

Query returned successfully in 196 msec.

Before

AFTER

	SELECT * FROM dwh.dim_sales_name				
	Data Output Messages Notifications				
	sales_name_id	sales_name	sales_age	sales_gender	last_update
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

- ❖ Schemas (4)
 - > dm
 - > dwh
 - > Aggregates
 - > A↓ Collations
 - > Domains
 - > FTS Configurations
 - > FTS Dictionaries
 - > A↓ FTS Parsers
 - > FTS Templates
 - > Foreign Tables
 - > Functions
 - > Materialized Views
 - > Operators
 - > Procedures (1)
 - > 1..3 Sequences
 - > Tables (5)
 - > dim_product
 - > dim_sales_name
 - > Columns
 - > Constraints
 - > Columns

	SELECT * FROM dwh.dim_sales_name				
	Data Output Messages Notifications				
	sales_name_id	sales_name	sales_age	sales_gender	last_update
1	3	Michael Johnson	45	Male	2024-11-24 18:04:31.695846+07
2	10	Olivia Taylor	34	Female	2024-11-24 18:04:31.695846+07
3	4	Emily Brown	32	Female	2024-11-24 18:04:31.695846+07
4	9	James Anderson	41	Male	2024-11-24 18:04:31.695846+07
5	2	Jane Smith	28	Female	2024-11-24 18:04:31.695846+07
6	6	Jessica Davis	26	Female	2024-11-24 18:04:31.695846+07
7	7	David Martinez	38	Male	2024-11-24 18:04:31.695846+07
8	8	Sophia Wilson	29	Female	2024-11-24 18:04:31.695846+07
9	5	Daniel Williams	40	Male	2024-11-24 18:04:31.695846+07
10	11	Hijri Della Wirasti	31	Female	2024-11-24 18:04:31.695846+07
11	1	John Doe	35	Male	2024-11-24 18:04:31.695846+07

Data dwh

fact_sales_transaction

Data Output										
	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		

Data dm

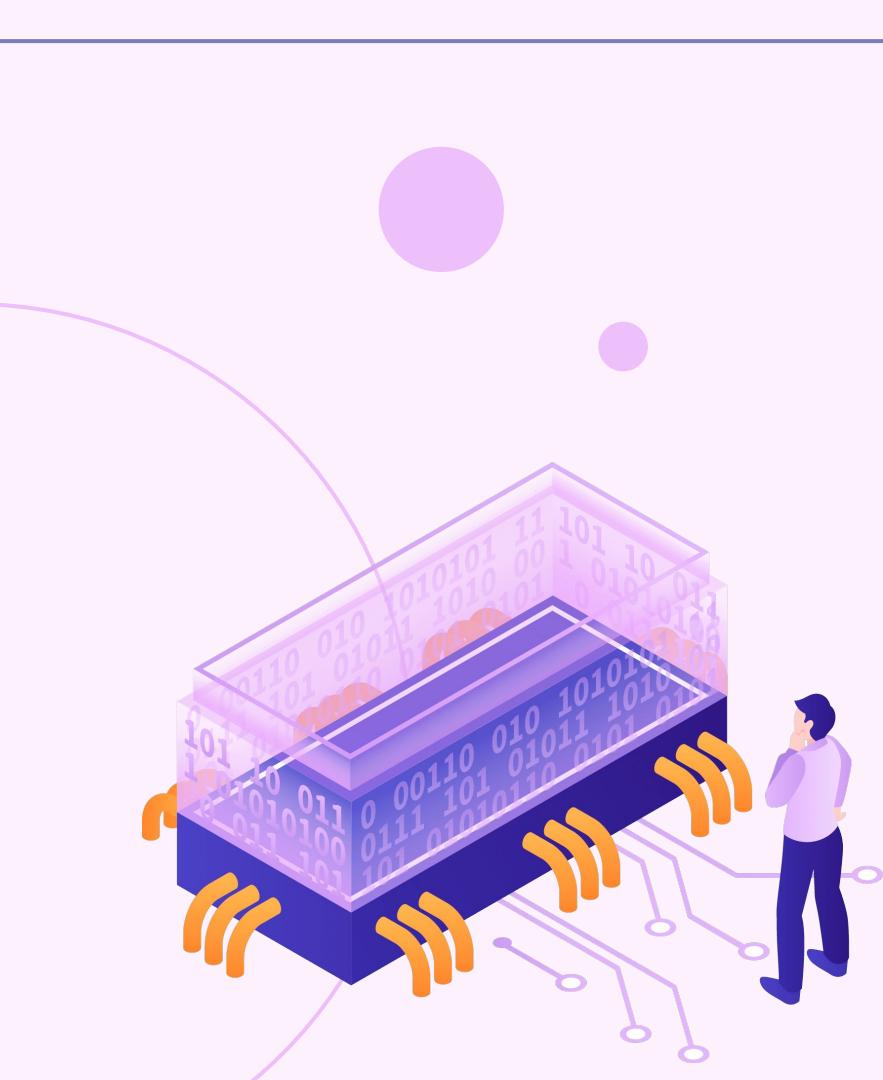
dm_sales_by_salesperson

Before

	sales_name_id	sales_name	sales_age	sales_gender	total_quantity	total_sales_amount	created_at
	integer	character varying (100)	integer	character varying (10)	bigint	bigint	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

	sales_name_id	sales_name	sales_age	sales_gender	total_quantity	total_sales_amount	created_at
	integer	character varying (100)	integer	character varying (10)	bigint	bigint	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	Hijri 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/>

