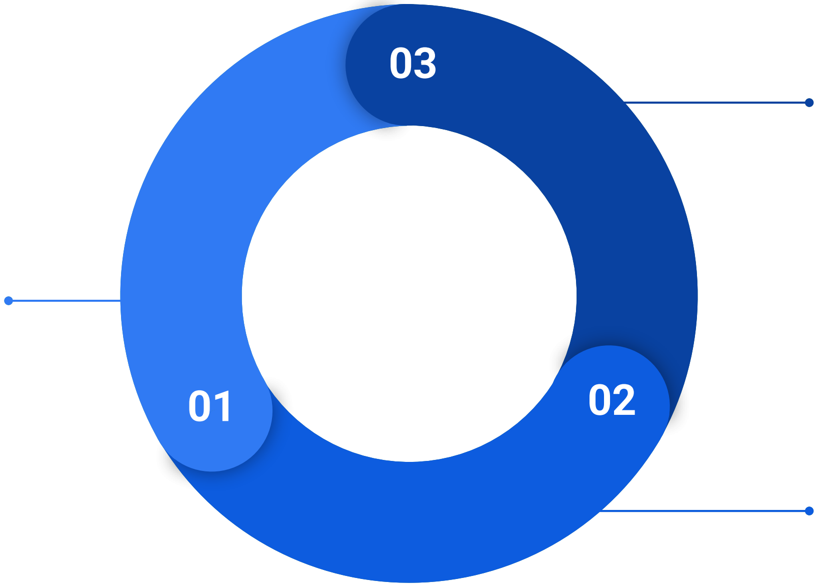
MID_SIZED RETAIL COMPANY DIMENSIONAL MODELLING REPORT

ACSP - Senior Data Analyst - Stage Eight



INTRODUCTION

Tools employed
Relevant links

Assignment Requirements

1. Data Profiling & Inference
2. Dimensional Modeling Strategy
3. Slowly Changing Dimensions (SCD)
4. ETL/ELT Strategy
5. Advanced Analysis & Scalability

Conclusion and Relevance

INTRODUCTION

- The dimensional modelling was carried out per client instructions given on dorian-twister-cea.notion.site

- The Dimension was done was carried out in PGAdmin using SQL script. Relevant screenshot are attached to the report.

- PDadmin was used to create database function to create a database called storemanagement

- Link to the dimension Model

- Link to the SQL codes for the database creation and Triggers and SCD.

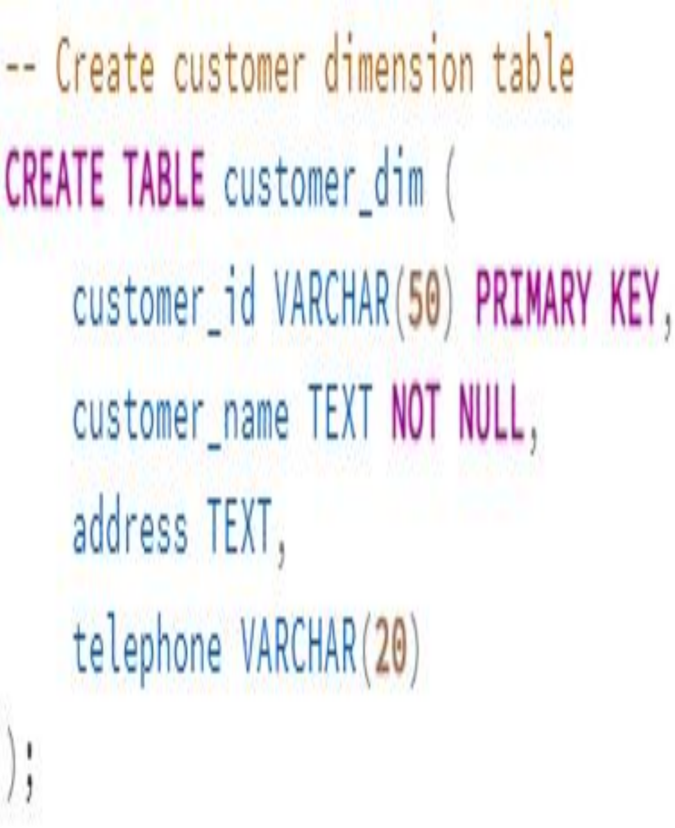
1. Data Profiling and Inference

Identifying Dimensions in the Invoice Dataset

- **Overview**
  - Dimensions provide descriptive context to transactional data
  - They help categorize, filter, and analyze key business entities.

Identified Dimension Tables: Four dimensions were identified and they are given below

- Customer Dimension (customer_dim):
  - Captures details of customers involved in transactions.
  - Attributes:
    - customer_id (Primary Key)
    - Customer_name
    - Address
    - Telephone
  - The code in the screenshot was used to load this dimension to the postgre database using the query tool.

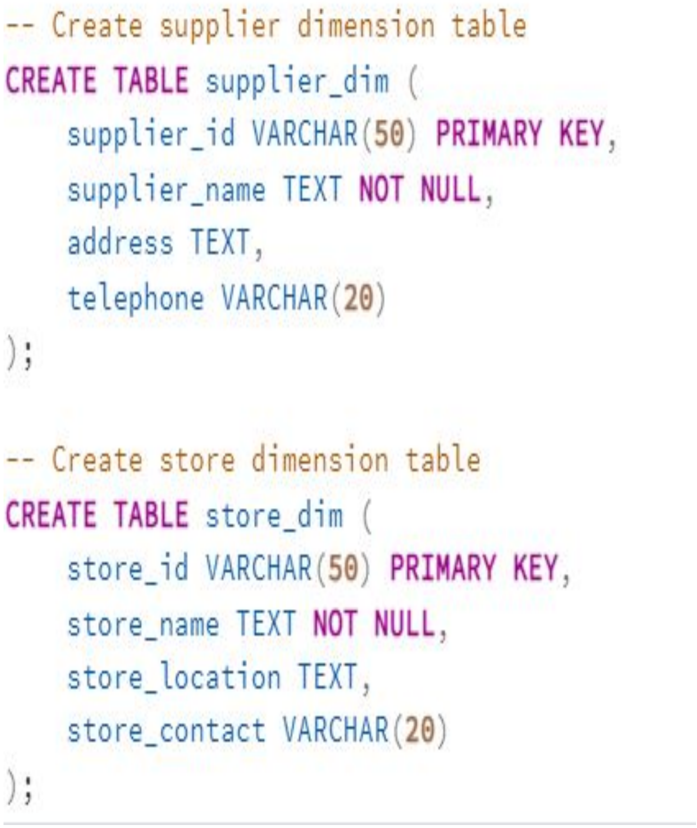


```
-- Create customer dimension table
CREATE TABLE customer_dim (
    customer_id VARCHAR(50) PRIMARY KEY,
    customer_name TEXT NOT NULL,
    address TEXT,
    telephone VARCHAR(20)
);
```

1. Data Profiling and Inference

Identified Dimension Tables

- **Supplier Dimension (supplier_dim)**
  - When goods are out of stock or in the re-order levels, retailers reach out to their suppliers to replenish stocks.
  - This stores information about suppliers providing goods.
  - **Attributes:**
    - supplier_id (Primary Key)
    - Supplier_name
    - Address
    - Telephone
- Store Dimension (store_dim)
  - This dimension store information of physical or online stores where sales occur
  - Attributes:
    - `store_id` (Primary Key)



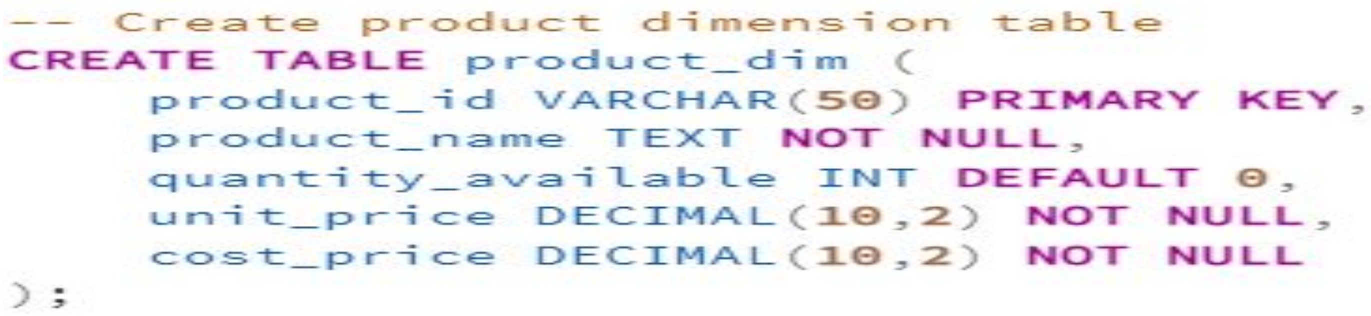
```
-- Create supplier dimension table
CREATE TABLE supplier_dim (
    supplier_id VARCHAR(50) PRIMARY KEY,
    supplier_name TEXT NOT NULL,
    address TEXT,
    telephone VARCHAR(20)
);

-- Create store dimension table
CREATE TABLE store_dim (
    store_id VARCHAR(50) PRIMARY KEY,
    store_name TEXT NOT NULL,
    store_location TEXT,
    store_contact VARCHAR(20)
);
```

1. Data Profiling and Inference

Identified Dimension Tables

- Store_name
- Store_location
- Store_contact

- **Product Dimension (product_dim)**
  - It contains details about products sold in transactions
  - Attributes:
    - product_id (Primary Key)
    - product_name
    - quantity_available
    - Unit_price
    - Cost_price
- The data types of the attributes of each dimensions are given in each code snippet



```
-- Create product dimension table
CREATE TABLE product_dim (
    product_id VARCHAR(50) PRIMARY KEY,
    product_name TEXT NOT NULL,
    quantity_available INT DEFAULT 0,
    unit_price DECIMAL(10,2) NOT NULL,
    cost_price DECIMAL(10,2) NOT NULL
);
```

1. Data Profiling and Inference

Determining Fact Table Granularity

- **Overview**
  - Granularity defines the level of detail stored in the fact table.
  - It impacts reporting, performance, and the depth of analysis.

Established Fact Table Granularity: Two Fact table were established and they are given below

- **Purchase Invoice Fact Table (purchase_invoice_fact)**:
  - **Grain**: Each record represents a single product purchase per invoice line
  - **Justification:**
    - Captures itemized purchases for cost tracking.
    - Supports supplier performance analysis.
    - Enables procurement trend analysis.

- **Sales Invoice Fact Table (sales_invoice_fact):**
  - **Grain:** Each record represents a single product sale per invoice line.

1. Data Profiling and Inference

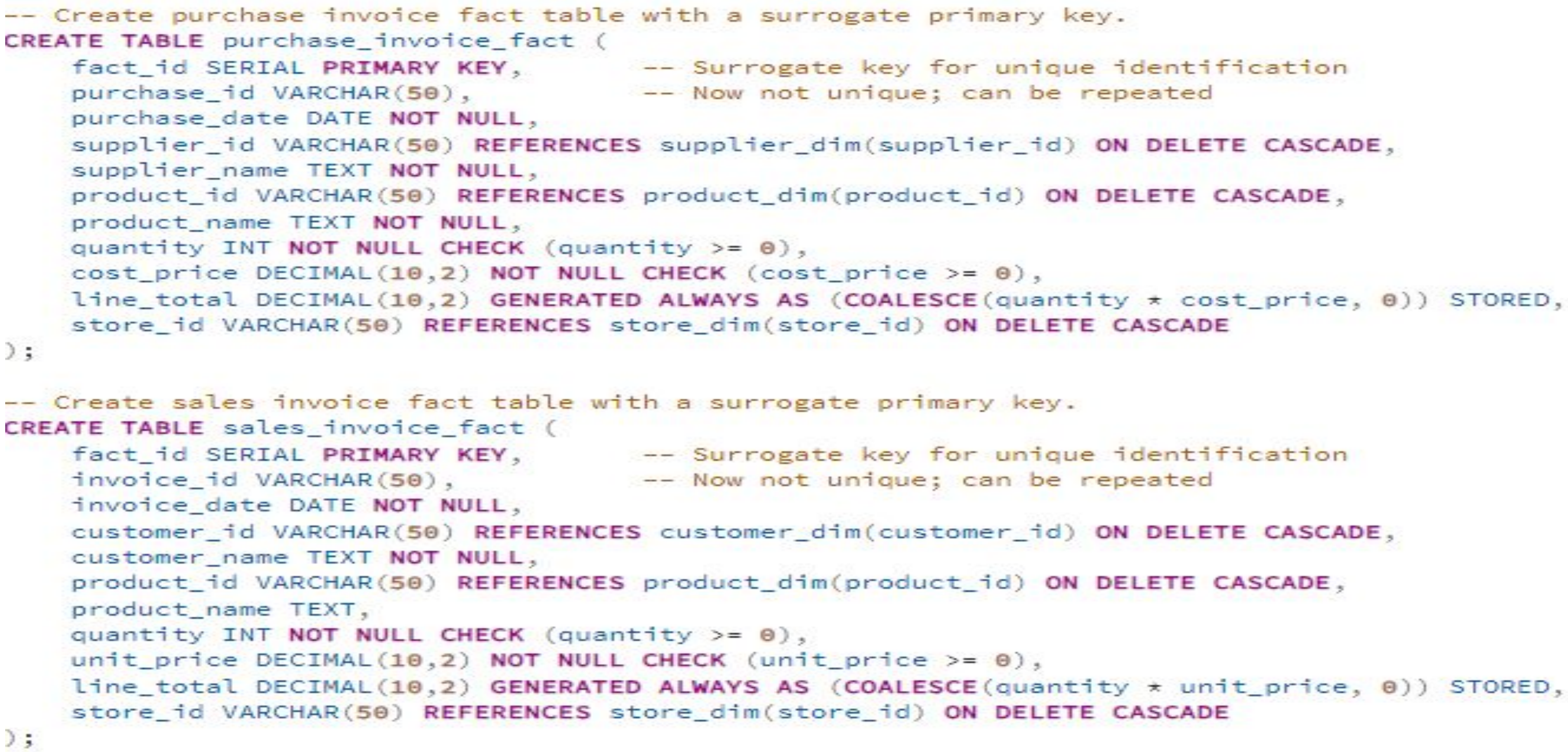
Established Fact Table Granularity

- **Sales Invoice Fact Table (sales_invoice_fact):**
  - Allows detailed sales trend analysis.
  - Supports customer purchase behavior insights.
  - Enables store-wise performance evaluation.

- **Key Business Insights Enabled:**
  - **Profitability Analysis**: Compare unit prices vs. cost prices at an item level.
  - **Inventory Management**: Track product movement across stores.
  - **Customer & Supplier Performance:** Identify top buyers and best-performing suppliers.
  - **Trend Forecasting:** Analyze demand fluctuations over time

- The code snippet containing the dtypes, attributes and other relevant information of the fact table are given in the next slide

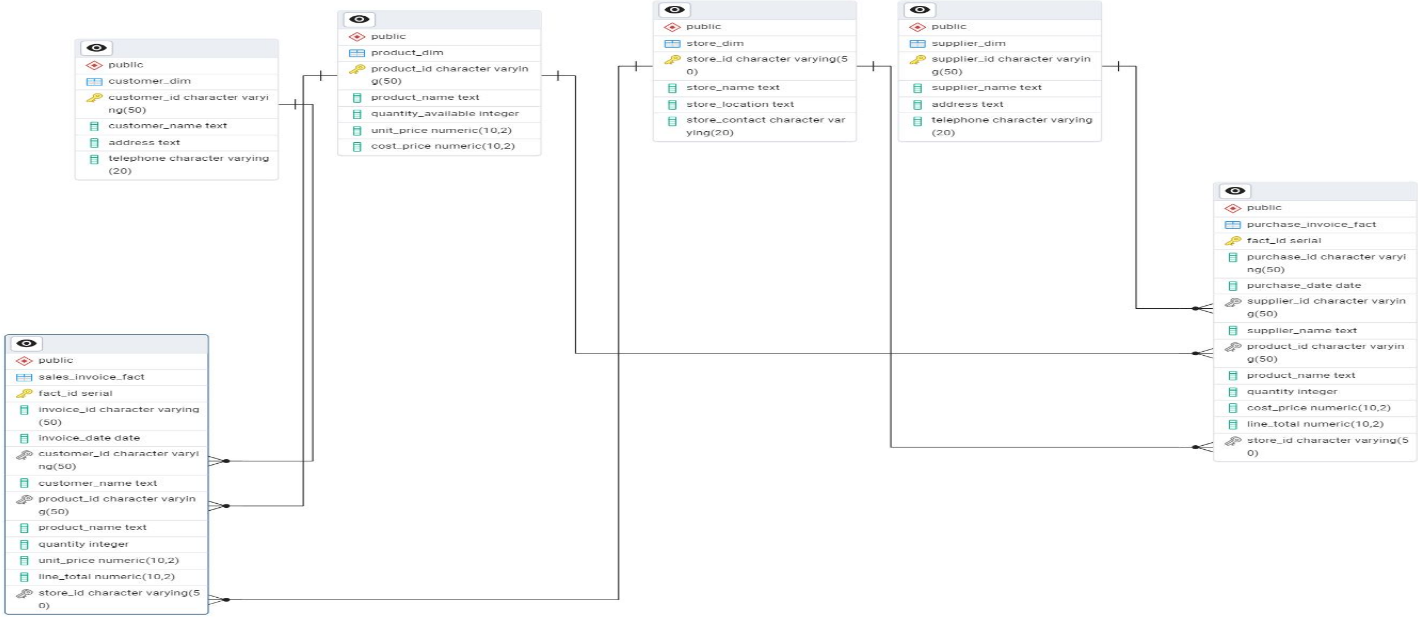
1. Data Profiling and Inference



```sql
-- Create purchase invoice fact table with a surrogate primary key.
CREATE TABLE purchase_invoice_fact (
    fact_id SERIAL PRIMARY KEY,        -- Surrogate key for unique identification
    purchase_id VARCHAR(50),           -- Now not unique; can be repeated
    purchase_date DATE NOT NULL,
    supplier_id VARCHAR(50) REFERENCES supplier_dim(supplier_id) ON DELETE CASCADE,
    supplier_name TEXT NOT NULL,
    product_id VARCHAR(50) REFERENCES product_dim(product_id) ON DELETE CASCADE,
    product_name TEXT NOT NULL,
    quantity INT NOT NULL CHECK (quantity >= 0),
    cost_price DECIMAL(10,2) NOT NULL CHECK (cost_price >= 0),
    line_total DECIMAL(10,2) GENERATED ALWAYS AS (COALESCE(quantity * cost_price, 0)) STORED,
    store_id VARCHAR(50) REFERENCES store_dim(store_id) ON DELETE CASCADE
);

-- Create sales invoice fact table with a surrogate primary key.
CREATE TABLE sales_invoice_fact (
    fact_id SERIAL PRIMARY KEY,        -- Surrogate key for unique identification
    invoice_id VARCHAR(50),            -- Now not unique; can be repeated
    invoice_date DATE NOT NULL,
    customer_id VARCHAR(50) REFERENCES customer_dim(customer_id) ON DELETE CASCADE,
    customer_name TEXT NOT NULL,
    product_id VARCHAR(50) REFERENCES product_dim(product_id) ON DELETE CASCADE,
    product_name TEXT,
    quantity INT NOT NULL CHECK (quantity >= 0),
    unit_price DECIMAL(10,2) NOT NULL CHECK (unit_price >= 0),
    line_total DECIMAL(10,2) GENERATED ALWAYS AS (COALESCE(quantity * unit_price, 0)) STORED,
    store_id VARCHAR(50) REFERENCES store_dim(store_id) ON DELETE CASCADE
);
```

2. Dimensional Modeling Strategy

❖ Star Schema design



public
customer_dim
customer_id character varying(50)
customer_name text
address text
telephone character varying(20)

public
product_dim
product_id character varying(50)
product_name text
quantity_available integer
unit_price numeric(10,2)
cost_price numeric(10,2)

public
store_dim
store_id character varying(50)
store_name text
store_location text
store_contact character varying(20)

public
supplier_dim
supplier_id character varying(50)
supplier_name text
address text
telephone character varying(20)

public
purchase_invoice_fact
fact_id serial
purchase_id character varying(50)
purchase_date date
supplier_id character varying(50)
supplier_name text
product_id character varying(50)
product_name text
quantity integer
cost_price numeric(10,2)
line_total numeric(10,2)
store_id character varying(50)

public
sales_invoice_fact
fact_id serial
invoice_id character varying(50)
invoice_date date
customer_id character varying(50)
customer_name text
product_id character varying(50)
product_name text
quantity integer
unit_price numeric(10,2)
line_total numeric(10,2)
store_id character varying(50)

2. Dimensional Modeling Strategy

- **Key Benefits of The Star Schema design**
  - Fast Query Performance – Optimized for aggregations and analytics
  - Simplified Reporting – Enhances sales, inventory, and supplier insights.
  - Scalability – Easily extends by adding new dimensions or measures.
  - Efficient Storage – Reduces redundancy while maintaining data integrity.

Handling Degenerate Dimensions

- Degenerate dimensions are identifiers that exist in fact tables but have no corresponding dimension table.

Identified Degenerate Dimensions

- **purchase_id (in purchase_invoice_fact)**
  - Represents a unique purchase transaction.
  - No additional descriptive attributes justify a separate dimension table
- **invoice_id (in sales_invoice_fact)**
  - Represents a unique sales transaction.
  - Acts as a reference for order-level aggregation and tracking.
- **Management Strategy**
  - **Stored in the Fact Tables** – Since they have no descriptive attributes, they remain in the respective fact tables.
  - **Optimized for Query Performance** – Indexed for efficient filtering and lookups.

2. Dimensional Modeling Strategy

- **Management Strategy**
  - **Stored in the Fact Tables** – Since they have no descriptive attributes, they remain in the respective fact tables.
  - **Optimized for Query Performance** – Indexed for efficient filtering and lookups.
  - **Facilitates Drill-Down Analysis** – Enables tracking of individual transactions.
  - **Supports Data Integrity** – Ensures accurate reconciliation between transactions.

- **Business Insights Enabled**
  - **Sales Trend Analysis** – Group sales data by invoice_id to analyze transaction patterns.
  - **Purchase Order Tracking** – Aggregate purchases using purchase_id for procurement efficiency.
  - **Operational Auditing** – Helps track specific transactions for validation and troubleshooting.

3. Slowly Changing Dimensions (SCD)

- **Understanding SCD**
  - SCD refers to handling historical changes in dimension attributes over time.
  - Common SCD types: Type 1 (Overwrite), Type 2 (Versioning), Type 3 (Limited History).

- **Chosen Approach: SCD Type 2**
  - Tracks Historical Changes – Maintains multiple versions of a record.
  - Adds Surrogate Keys – Ensures unique historical records.
  - Uses Effective & Expiration Dates – Identifies active vs. archived records.
  - Current Flag Indicator – Marks the latest version of the record.

- **Implementation Details**
  - Customer Dimension (customer_dim_scd)
    - Tracks customer details like name, address, and telephone over time.
    - Trigger (trg_scd_customer_dim_update) archives old records before an update.
  - Product Dimension (product_dim_scd)
    - Maintains historical product attributes like price and quantity available.
    - Trigger (trg_capture_product_history) updates expiration dates of old records.

3. Slowly Changing Dimensions (SCD)

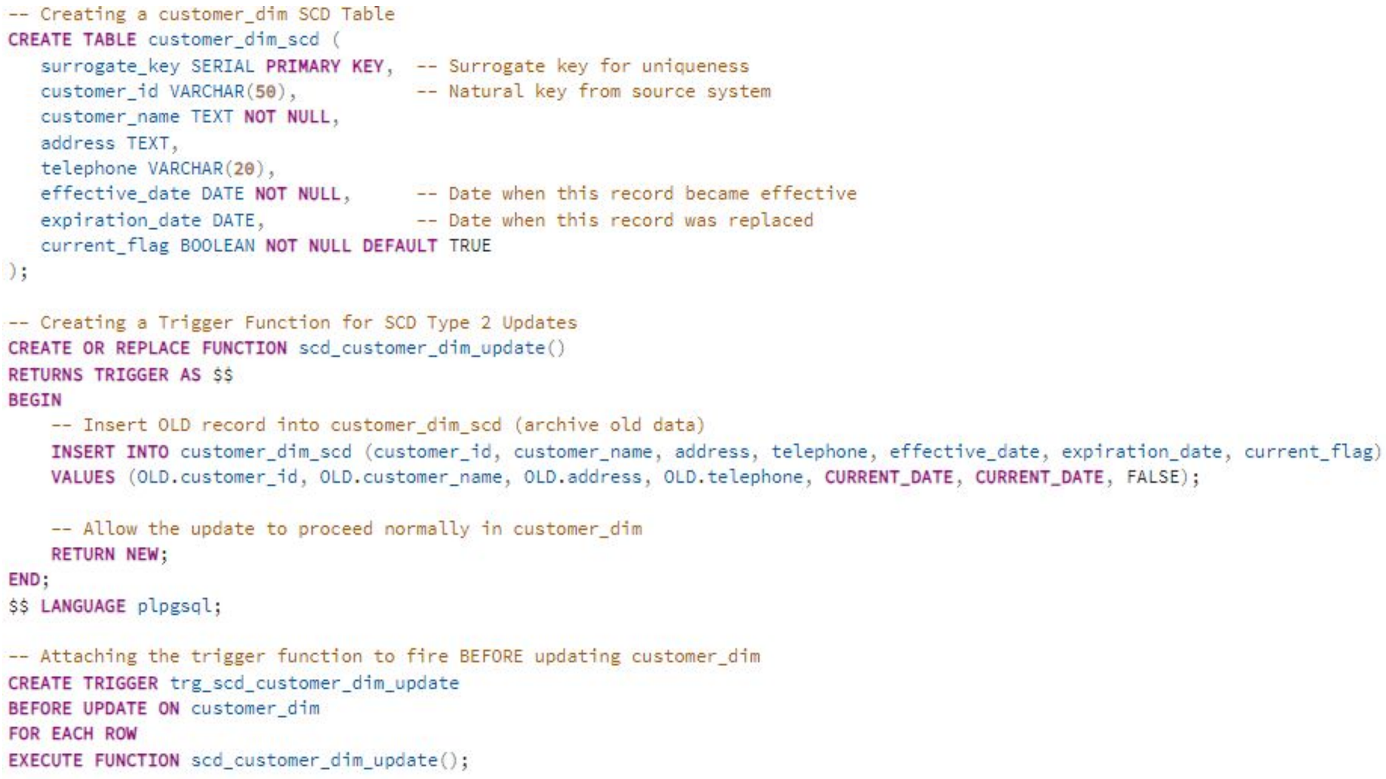
- **Trade-offs & Considerations**
  - Pros
    - Enables historical trend analysis and auditing.
    - Preserves a full history of changes for business intelligence.
  - Cons
    - Increased storage due to multiple versions.
    - More complex queries needed to retrieve current vs. historical data.

- **Business Benefits**
  - **Customer Analysis** – Track customer address or phone number changes for targeted marketing.
  - **Price & Inventory Insights** – Analyze historical product pricing and stock variations.
  - **Sales Trends** – Evaluate past product performance across different pricing strategies.

- The code snippet of the SCD implementation and the star schema design after the implementation are displayed below

3. Slowly Changing Dimensions (SCD)

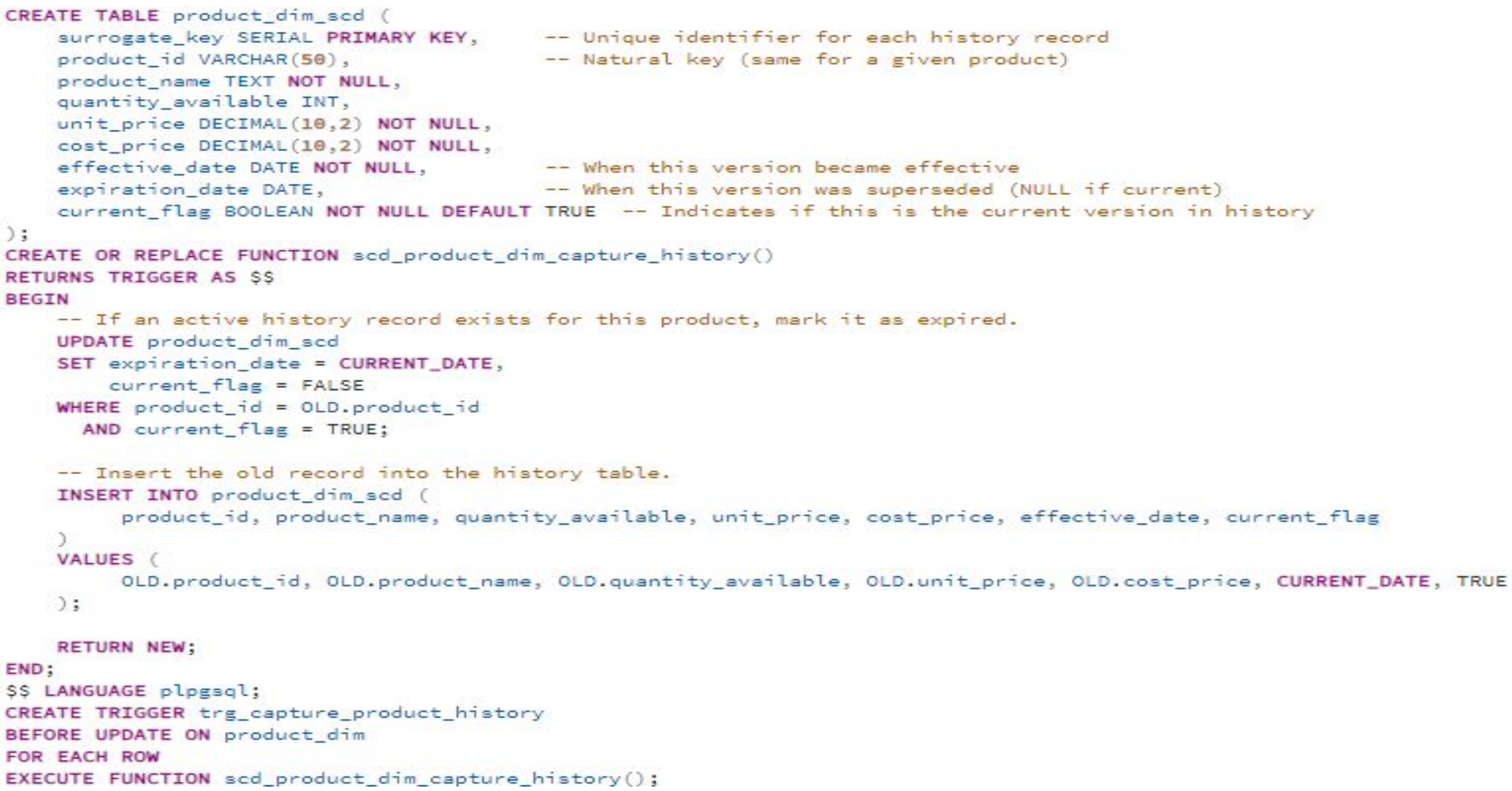
- **Customer Dimension (customer_dim_scd)**



```
-- Creating a customer_dim SCD Table
CREATE TABLE customer_dim_scd (
   surrogate_key SERIAL PRIMARY KEY,  -- Surrogate key for uniqueness
   customer_id VARCHAR(50),           -- Natural key from source system
   customer_name TEXT NOT NULL,
   address TEXT,
   telephone VARCHAR(20),
   effective_date DATE NOT NULL,      -- Date when this record became effective
   expiration_date DATE,              -- Date when this record was replaced
   current_flag BOOLEAN NOT NULL DEFAULT TRUE
);

-- Creating a Trigger Function for SCD Type 2 Updates
CREATE OR REPLACE FUNCTION scd_customer_dim_update()
RETURNS TRIGGER AS $$
BEGIN
    -- Insert OLD record into customer_dim_scd (archive old data)
    INSERT INTO customer_dim_scd (customer_id, customer_name, address, telephone, effective_date, expiration_date, current_flag)
    VALUES (OLD.customer_id, OLD.customer_name, OLD.address, OLD.telephone, CURRENT_DATE, CURRENT_DATE, FALSE);

    -- Allow the update to proceed normally in customer_dim
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Attaching the trigger function to fire BEFORE updating customer_dim
CREATE TRIGGER trg_scd_customer_dim_update
BEFORE UPDATE ON customer_dim
FOR EACH ROW
EXECUTE FUNCTION scd_customer_dim_update();
```

3. Slowly Changing Dimensions (SCD)

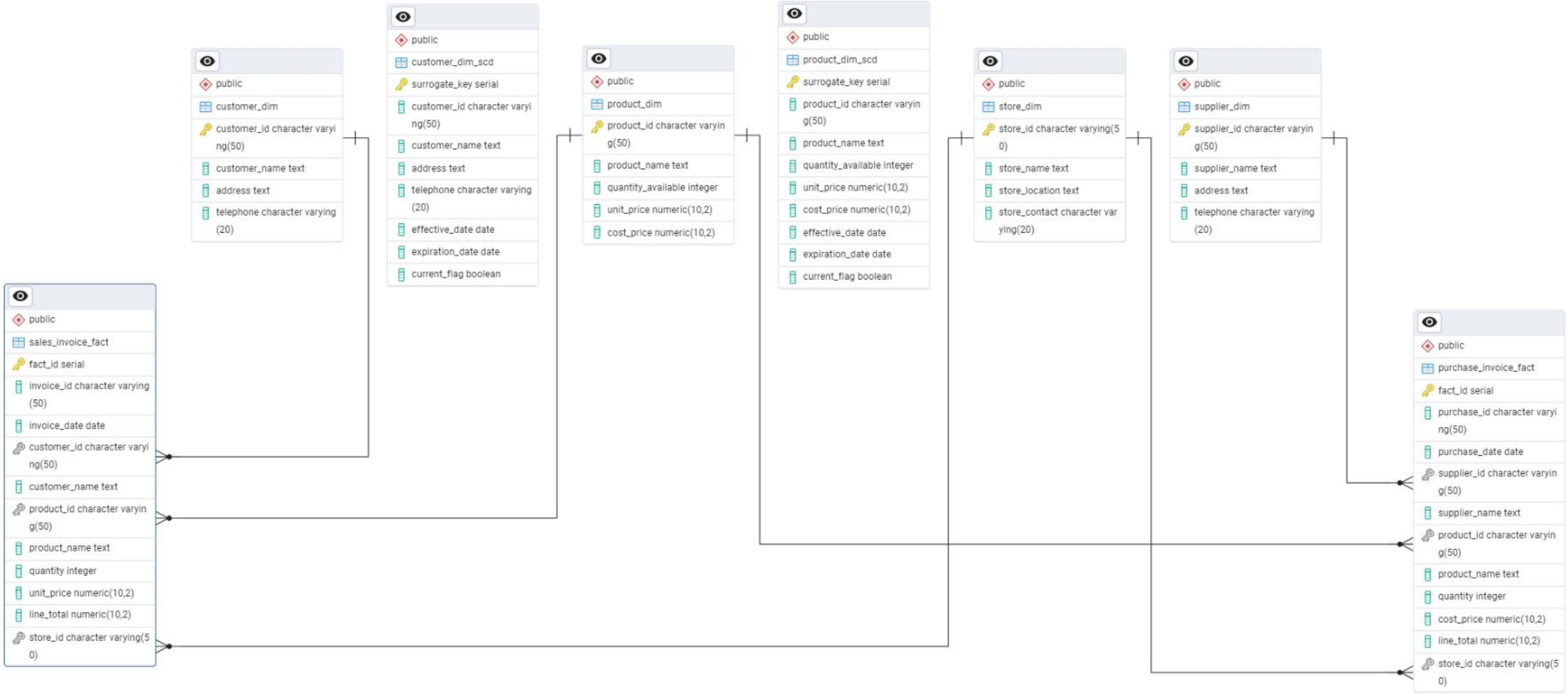
- **Product Dimension (product_dim_scd)**



```sql
CREATE TABLE product_dim_scd (
    surrogate_key SERIAL PRIMARY KEY,     -- Unique identifier for each history record
    product_id VARCHAR(50),               -- Natural key (same for a given product)
    product_name TEXT NOT NULL,
    quantity_available INT,
    unit_price DECIMAL(10,2) NOT NULL,
    cost_price DECIMAL(10,2) NOT NULL,
    effective_date DATE NOT NULL,         -- When this version became effective
    expiration_date DATE,                 -- When this version was superseded (NULL if current)
    current_flag BOOLEAN NOT NULL DEFAULT TRUE  -- Indicates if this is the current version in history
);
CREATE OR REPLACE FUNCTION scd_product_dim_capture_history()
RETURNS TRIGGER AS $$
BEGIN
    -- If an active history record exists for this product, mark it as expired.
    UPDATE product_dim_scd
    SET expiration_date = CURRENT_DATE,
        current_flag = FALSE
    WHERE product_id = OLD.product_id
      AND current_flag = TRUE;

    -- Insert the old record into the history table.
    INSERT INTO product_dim_scd (
         product_id, product_name, quantity_available, unit_price, cost_price, effective_date, current_flag
    )
    VALUES (
         OLD.product_id, OLD.product_name, OLD.quantity_available, OLD.unit_price, OLD.cost_price, CURRENT_DATE, TRUE
    );

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_capture_product_history
BEFORE UPDATE ON product_dim
FOR EACH ROW
EXECUTE FUNCTION scd_product_dim_capture_history();
```

3. Slowly Changing Dimensions (SCD)

- **Star schema design after SCD implementation**



public
customer_dim
customer_id character varying(50)
customer_name text
address text
telephone character varying(20)

public
customer_dim_scd
surrogate_key serial
customer_id character varying(50)
customer_name text
address text
telephone character varying(20)
effective_date date
expiration_date date
current_flag boolean

public
product_dim
product_id character varying(50)
product_name text
quantity_available integer
unit_price numeric(10,2)
cost_price numeric(10,2)

public
product_dim_scd
surrogate_key serial
product_id character varying(50)
product_name text
quantity_available integer
unit_price numeric(10,2)
cost_price numeric(10,2)
effective_date date
expiration_date date
current_flag boolean

public
store_dim
store_id character varying(50)
store_name text
store_location text
store_contact character varying(20)

public
supplier_dim
supplier_id character varying(50)
supplier_name text
address text
telephone character varying(20)

public
sales_invoice_fact
fact_id serial
invoice_id character varying(50)
invoice_date date
customer_id character varying(50)
customer_name text
product_id character varying(50)
product_name text
quantity integer
unit_price numeric(10,2)
line_total numeric(10,2)
store_id character varying(50)

public
purchase_invoice_fact
fact_id serial
purchase_id character varying(50)
purchase_date date
supplier_id character varying(50)
supplier_name text
product_id character varying(50)
product_name text
quantity integer
cost_price numeric(10,2)
line_total numeric(10,2)
store_id character varying(50)

4. ETL/ELT Strategy

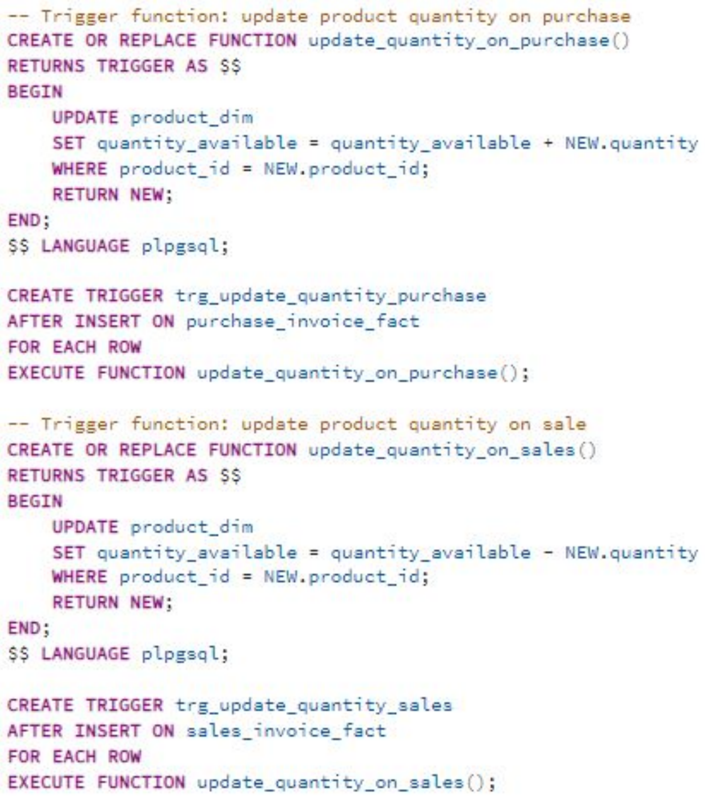
- **Data Integration Approach**
  - **Fact and Dimension Table Relationships** – Ensures proper linkage between sales, purchases, and product/customer/supplier details.
  - **Trigger-Based Data Updates** – Automates data population and updates for real-time consistency.
  - **Incremental Data Updates** – Uses AFTER INSERT and BEFORE INSERT triggers for efficient data changes.
  - **Hybrid ETL/ELT Approach**:
    - ETL (Extract, Transform, Load) – Ensures data transformation before insertion.
    - ELT (Extract, Load, Transform) – Uses database triggers to transform after loading.

- **Data Population & Automation**
  - **Product Quantity Management**
    - **On Purchase** → Increases quantity_available in product_dim
    - **On Sale** → Decreases quantity_available in product_dim.
  - **Automated Data Enrichment**
    - **Sales Invoice** → Populates product_name and customer_name from respective dimensions.
    - **Purchase Invoice** → Populates product_name and supplier_name from respective dimensions.

4. ETL/ELT Strategy

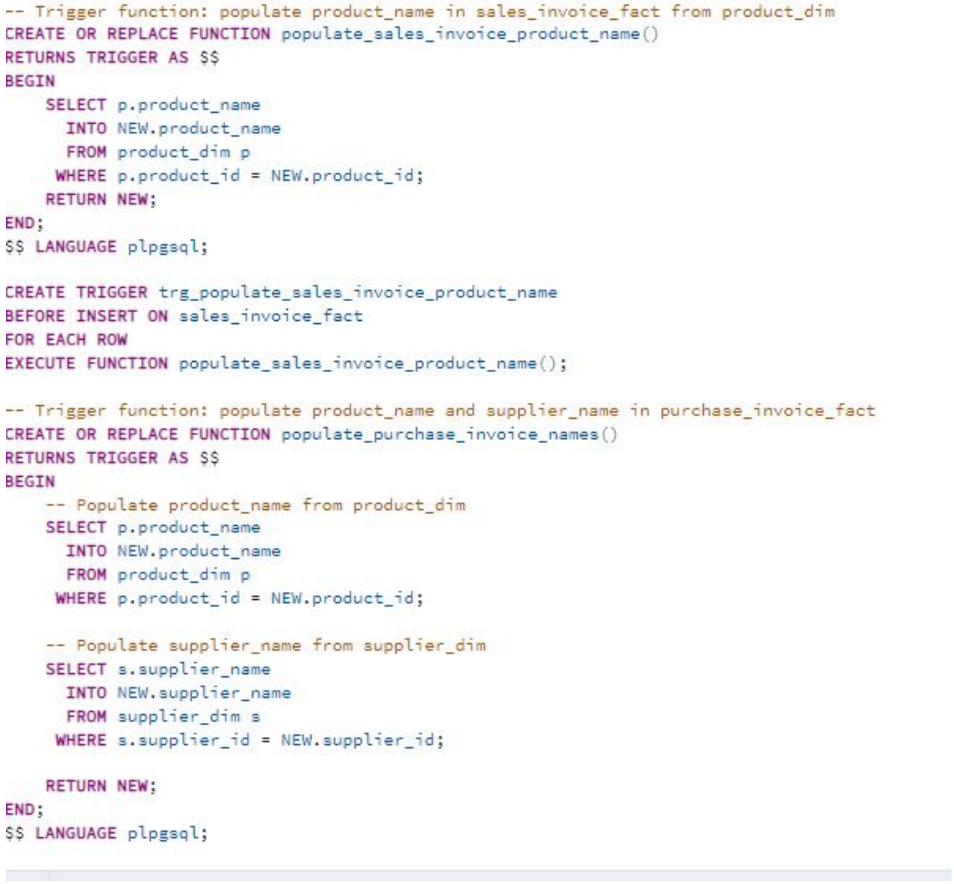
- **Data Quality & Consistency**
  - **Referential Integrity –** Ensures product_id, customer_id, and supplier_id exist before insertion.
  - **Trigger-Based Consistency** – Ensures real-time updates across related tables.
  - **Error Prevention** – Avoids missing or incorrect data by enforcing automatic lookups.
  - **Timely Updates** – Data changes are reflected immediately after transactions.

- **Performance Considerations**
  - Trigger Optimization – Prevents performance bottlenecks by limiting the scope of updates.
  - Indexing Key Columns – Improves query efficiency on product_id, customer_id, and supplier_id.
  - Batch Processing for Large Volumes – Reduces overhead when handling bulk inserts.

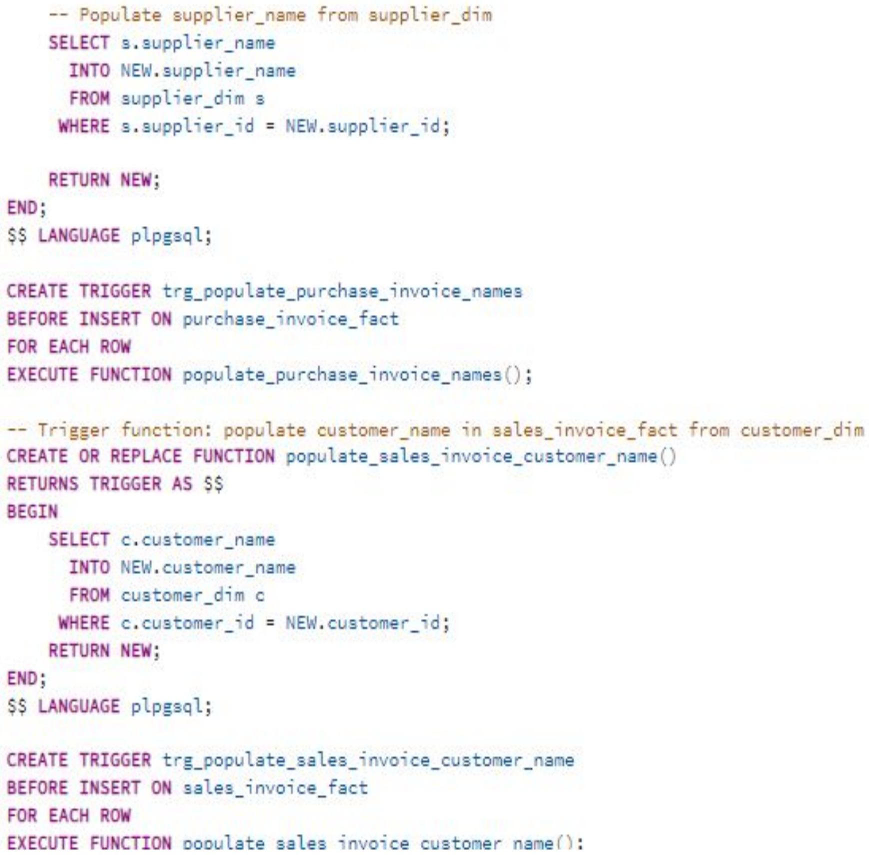


```sql
-- Trigger function: update product quantity on purchase
CREATE OR REPLACE FUNCTION update_quantity_on_purchase()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE product_dim
    SET quantity_available = quantity_available + NEW.quantity
    WHERE product_id = NEW.product_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_quantity_purchase
AFTER INSERT ON purchase_invoice_fact
FOR EACH ROW
EXECUTE FUNCTION update_quantity_on_purchase();

-- Trigger function: update product quantity on sale
CREATE OR REPLACE FUNCTION update_quantity_on_sales()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE product_dim
    SET quantity_available = quantity_available - NEW.quantity
    WHERE product_id = NEW.product_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_quantity_sales
AFTER INSERT ON sales_invoice_fact
FOR EACH ROW
EXECUTE FUNCTION update_quantity_on_sales();
```

4. ETL/ELT Strategy

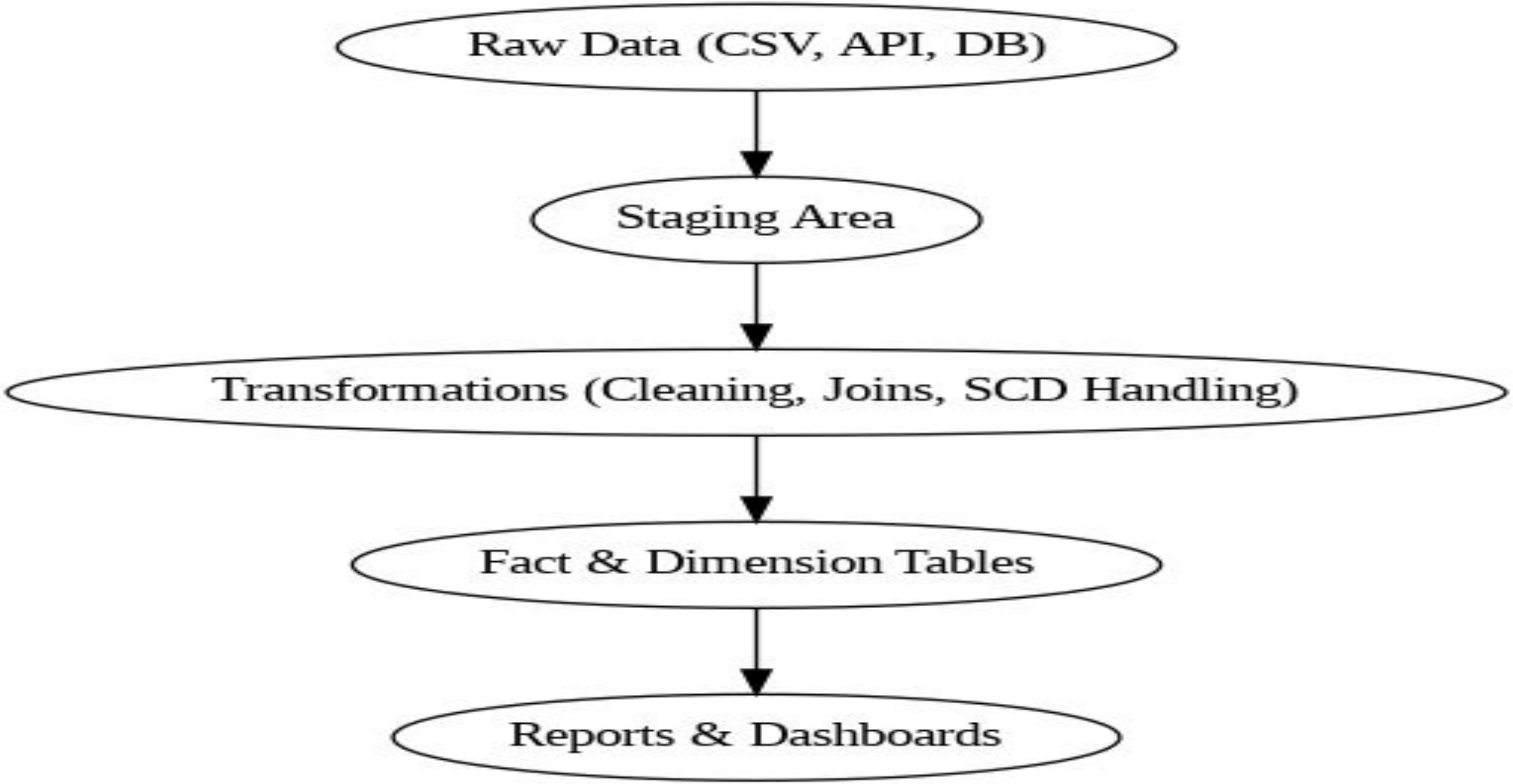


```sql
-- Trigger function: populate product_name in sales_invoice_fact from product_dim
CREATE OR REPLACE FUNCTION populate_sales_invoice_product_name()
RETURNS TRIGGER AS $$
BEGIN
    SELECT p.product_name
      INTO NEW.product_name
      FROM product_dim p
     WHERE p.product_id = NEW.product_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_populate_sales_invoice_product_name
BEFORE INSERT ON sales_invoice_fact
FOR EACH ROW
EXECUTE FUNCTION populate_sales_invoice_product_name();

-- Trigger function: populate product_name and supplier_name in purchase_invoice_fact
CREATE OR REPLACE FUNCTION populate_purchase_invoice_names()
RETURNS TRIGGER AS $$
BEGIN
    -- Populate product_name from product_dim
    SELECT p.product_name
      INTO NEW.product_name
      FROM product_dim p
     WHERE p.product_id = NEW.product_id;

    -- Populate supplier_name from supplier_dim
    SELECT s.supplier_name
      INTO NEW.supplier_name
      FROM supplier_dim s
     WHERE s.supplier_id = NEW.supplier_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



```sql
    -- Populate supplier_name from supplier_dim
    SELECT s.supplier_name
      INTO NEW.supplier_name
      FROM supplier_dim s
     WHERE s.supplier_id = NEW.supplier_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_populate_purchase_invoice_names
BEFORE INSERT ON purchase_invoice_fact
FOR EACH ROW
EXECUTE FUNCTION populate_purchase_invoice_names();

-- Trigger function: populate customer_name in sales_invoice_fact from customer_dim
CREATE OR REPLACE FUNCTION populate_sales_invoice_customer_name()
RETURNS TRIGGER AS $$
BEGIN
    SELECT c.customer_name
      INTO NEW.customer_name
      FROM customer_dim c
     WHERE c.customer_id = NEW.customer_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_populate_sales_invoice_customer_name
BEFORE INSERT ON sales_invoice_fact
FOR EACH ROW
EXECUTE FUNCTION populate_sales_invoice_customer_name();
```

4. ETL/ELT Strategy



Raw Data (CSV, API, DB)

Staging Area

Transformations (Cleaning, Joins, SCD Handling)

Fact & Dimension Tables

Reports & Dashboards

5. Advanced Analysis & Scalability

- **Hierarchical Relationships**
  - **Multi-Level Drill-Down Analysis**
    - Organize products by category, subcategory, and brand to enhance reporting.
    - Enable time-based analysis (Year → Quarter → Month → Day) for trend evaluation
  - **Parent-Child Relationships**.
    - Support roll-up and drill-down queries for regional performance tracking.
  - **Aggregated Reporting**
    - Design summary tables for quick retrieval of high-level insights.
    - Use materialized views to precompute common aggregations.

- **Performance Optimization**
  - **Database Indexing**
    - Use composite indexes for frequent filter combinations (e.g., date & product).
  - **Partitioning Strategy**
    - Partition fact tables by date range to speed up historical data retrieval.
  - Surrogate Key Implementation
    - Replace natural keys with integer surrogate keys for faster joins.
  - **Query Optimization Techniques**
    - Use caching for frequently accessed data to reduce database load.
  - Scalability & Growth Considerations
    - Design schema to accommodate future data growth with minimal restructuring

Relevance to Retailers

1. Supports Analytical Reporting

2. Enhances Drill-Down Capabilities

3. Ensures Historical Tracking

4. Improves Scalability & Performance

5. Facilitates Business Decision-Making

END OF REPORT