

# **Project Report**

## **Group 10:**

Ankur Pandey - +1 (857) 398 5940

pandey.anku@northeastern.edu

Giridhar Babu - +1 (857) 398-5730 babu.gi@northeastern.edu

Percentage of Effort Contributed by Ankur Pandey: 50%

Percentage of Effort Contributed by Giridhar Babu: 50%

Signature of Student 1: Ankur Pandey

Signature of Student 2: Giridhar Babu

Submission Date: 12/07/ 2024

## Table of Contents

<b>Milestone 1 .....</b>	<b>3</b>
<b>Milestone 2 .....</b>	<b>4</b>
ER Diagram .....	4
Relationships: .....	6
Dimensions:.....	8
Measures.....	9
<b>Milestone 3 .....</b>	<b>12</b>
Conceptual Model:.....	14
Logical Model: .....	15
OLAP Operations:.....	21
<b>Milestone 4 .....</b>	<b>23</b>
<b>Milestone 5 .....</b>	<b>39</b>
<b>Milestone 6 .....</b>	<b>40</b>
<b>Milestone 7 .....</b>	<b>55</b>
Dashboard 1 .....	55
Dashboard 2 .....	58
Dashboard 3 .....	60

# Milestone 1

## Problem Statement

Sail Foods needs to design and implement a comprehensive database system to manage the culinary operations for 5,000+ sailors on the USS Theodore Roosevelt, serving over 15,000 meals per day. The system must efficiently handle inventory management, meal planning, food preparation, serving logistics, and waste management while ensuring food safety, nutritional requirements, and operational efficiency in the unique environment of a Nimitz-class aircraft carrier.

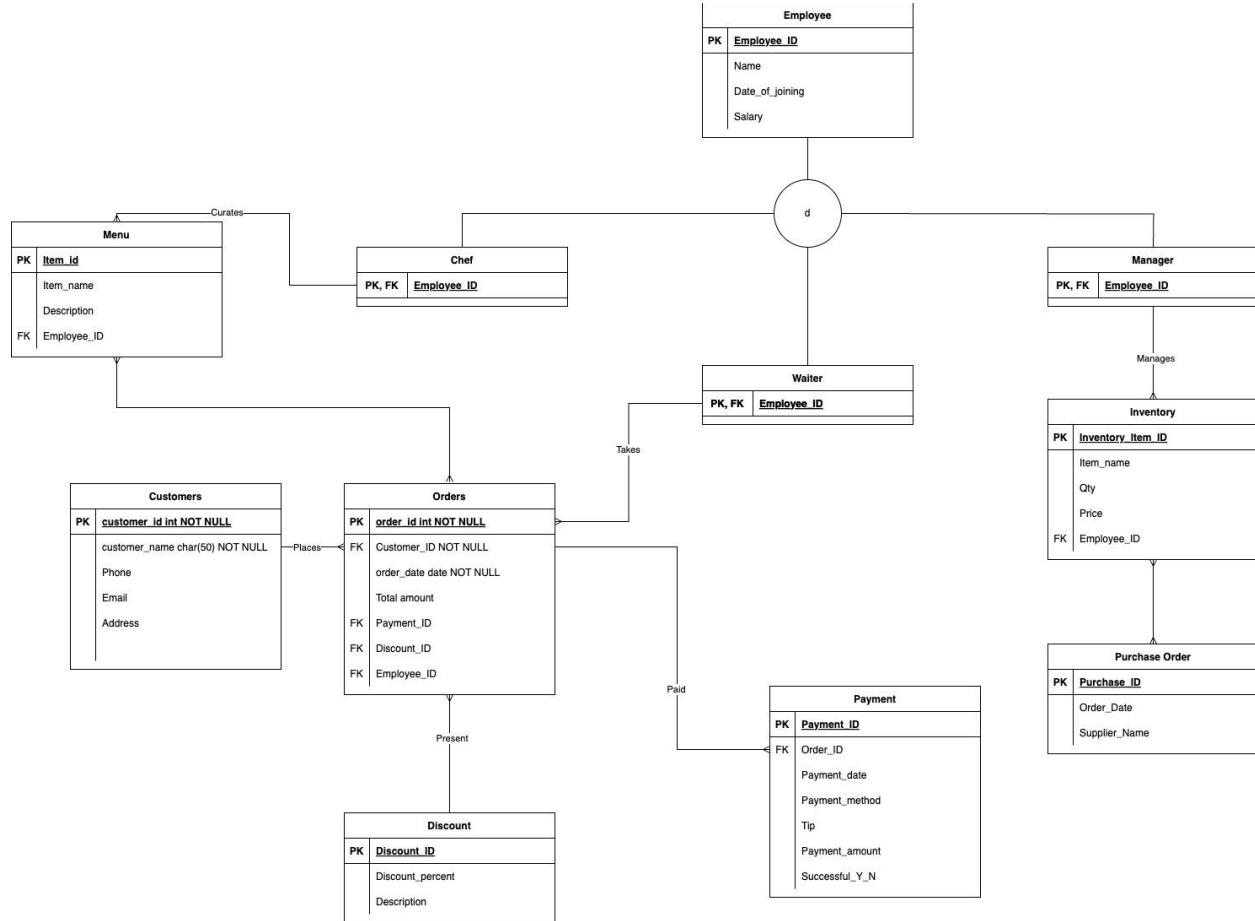
## Database Definition

The database system for Sail Foods' operation on the USS Theodore Roosevelt should encompass the following key components:

1. Inventory Management:
  - Track all food items, ingredients, and supplies
  - Monitor stock levels and expiration dates
  - Manage procurement and restocking processes
2. Menu Planning:
  - Store recipes and nutritional information
  - Create and manage meal plans
  - Accommodate dietary restrictions and preferences
3. Food Preparation:
  - Schedule food preparation tasks
  - Track cooking equipment usage and maintenance
  - Manage food safety and quality control processes
4. Serving Logistics:
  - Manage meal serving schedules
  - Track meal consumption and popularity
  - Handle special meal requests and events
5. Personnel Management:
  - Store information on culinary staff and their schedules
  - Track staff certifications and training
6. Waste Management:
  - Monitor food waste
  - Manage disposal processes
7. Reporting and Analytics:
  - Generate reports on various aspects of the operation
  - Provide data for performance analysis and optimization

# Milestone 2

## ER Diagram



## **Entities:**

1. **Customer:** Represents customers who place orders.
  - o Attributes: customer\_id (PK), customer\_name, phone, email, address
2. **Order:** Represents food orders made by customers.
  - o Attributes: order\_id (PK), order\_date, total\_amount, customer\_id (FK), payment\_id, employee\_id (FK), discount\_id (FK)
3. **Payment:** Represents payments made for orders.
  - o Attributes: payment\_id (PK), order\_id (FK), payment\_date, payment\_type, amount
4. **Discount:** Represents discounts available for orders.
  - o Attributes: discount\_id (PK), discount\_percentage, description
5. **Menu:** Contains all food items available.
  - o Attributes: item\_id (PK), item\_name, price, employee\_id (FK)
6. **Employee:** Represents employees, which include Waitrs, Managers, and Chefs.
  - o Attributes: employee\_id (PK), name, Date\_of\_joining, salary
7. **Waiter** (Specializes Employee):
  - o Attributes: employee\_id (PK, FK)
8. **Chef** (Specializes Employee):
  - o Attributes: employee\_id (PK, FK), specialization, years\_of\_experience
9. **Manager** (Specializes Employee):
  - o Attributes: employee\_id (PK, FK)
10. **Inventory:** Represents the food inventory.
  - o Attributes: inventory\_item\_id (PK), item\_name, qty, unit\_price, Employee\_id
11. **Purchase Order:** Represents purchase orders for inventory.
  - o Attributes: purchase\_id (PK), inventory\_id (FK), order\_date, quantity\_ordered, supplier\_name

## Relationships:

1. Customer to Order:
  - A customer can place multiple orders, but each order is placed by a single customer.
  - Relationship: Customer (1) → Order (N)
  - Type: One-to-Many (1)
  - Foreign Key: customer\_id in the Order table refers to customer\_id in the Customer table.
2. Order to Payment:
  - Each order can have one or more payments, but each payment is for a specific order.
  - Relationship: Order (1) → Payment (N)
  - Type: One-to-Many (1)
  - Foreign Key: order\_id in the Payment table refers to order\_id in the Order table.
3. Order to Discount:
  - An order can apply a discount, but a discount can be used for multiple orders.
  - Relationship: Order (N) → Discount (1)
  - Type: Many-to-One (N:1)
  - Foreign Key: discount\_id in the Order table refers to discount\_id in the Discount table.
4. Order to Menu (via Junction Table Order\_Menu):
  - An order can contain multiple menu items, and a menu item can be part of multiple orders (many-to-many relationship).
  - Relationship: Order (N) → Order\_Menu ← Menu (N)
  - Type: Many-to-Many (N)
  - Junction Table: Order\_Menu with order\_id (FK) and menu\_id (FK) links Order and Menu.
5. Employee to Order (for Waiters):
  - A waiter can serve multiple orders, but an order is served by a single employee (waiter).
  - Relationship: Employee (Waiter) (1) → Order (N)
  - Type: One-to-Many (1)
  - Foreign Key: employee\_id in the Order table refers to employee\_id in the Employee table for Waiters.
6. Employee to Menu (for Chefs):
  - A chef can create multiple menu items, but a menu item is associated with a single chef.
  - Relationship: Employee (Chef) (1) → Menu (N)
  - Type: One-to-Many (1)
  - Foreign Key: employee\_id in the Menu table refers to employee\_id in the Employee table for Chefs.

**7. Employee to Inventory (for Managers):**

- A manager can oversee multiple inventory items, but each inventory item is associated with one manager.
- Relationship: Employee (Manager) (1) → Inventory (N)
- Type: One-to-Many (1)
- Foreign Key: employee\_id in the Inventory table refers to employee\_id in the Employee table for Managers.

**8. Purchase Order to Inventory:**

- Each purchase order is made for a single inventory item, but each inventory item can have multiple purchase orders over time.
- Relationship: Inventory (1) → Purchase Order (N)
- Type: One-to-Many (1)
- Foreign Key: inventory\_item\_id in the Purchase Order table refers to inventory\_item\_id in the Inventory table.

**9. Employee Specialization (Inheritance):**

- Employee generalizes Waiter, Chef, and Manager. Each employee can be specialized into one of these roles.
- Type: One-to-One (1:1) Inheritance
- Foreign Key: employee\_id is shared across Employee, Waiter, Chef, and Manager.

**Diagram Structure:**

- Customer → Order → Payment, Discount
- Order → Menu (N relationship)
- Menu → Category
- Order → Waitress (Handled by waitress from the Employee table)
- Manager → Employee
- Inventory → Purchase Order

## Dimensions:

### 1. Customer Dimension

- **Attributes:** customer\_id, customer\_name, phone, email, address
- **Hierarchy:** None

### 2. Order Dimension

- **Attributes:** order\_id, order\_date, total\_amount, customer\_id, payment\_id, employee\_id, discount\_id
- **Hierarchy:** order\_date (can be split into year > month > day for time-based analysis)

### 3. Employee Dimension

- **Attributes:** employee\_id, name, date\_of\_joining, salary
- **Hierarchy:** None

### 4. Menu Dimension

- **Attributes:** item\_id, item\_name, price, employee\_id
- **Hierarchy:** Menu Type (Veg/Non-Veg) > Item Name

### 5. Payment Dimension

- **Attributes:** payment\_id, order\_id, payment\_date, payment\_type, amount
- **Hierarchy:** payment\_date (can be split into year > month > day for time-based analysis)

### 6. Discount Dimension

- **Attributes:** discount\_id, discount\_percentage, description
- **Hierarchy:** None

### 7. Inventory Dimension

- **Attributes:** inventory\_item\_id, item\_name, qty, unit\_price, employee\_id
- **Hierarchy:** None

### 8. Purchase Order Dimension

- **Attributes:** purchase\_id, inventory\_item\_id, order\_date, quantity\_ordered, supplier\_name
- **Hierarchy:** order\_date (can be split into year > month > day for time-based analysis)

## Measures

### 1. Total Bill Amount

- **Source:** Order.total\_amount
- **Type:** Additive
- **Aggregation:** SUM
- **Description:** Represents the total value of orders placed by customers. This can be aggregated across various dimensions such as time, employee (waiter/chef), customer, and menu items.

### 2. Total Payment Amount

- **Source:** Payment.amount
- **Type:** Additive
- **Aggregation:** SUM
- **Description:** Represents the total payments made by customers. This measure provides insights into revenue collected across dimensions like time, payment type, and order.

### 3. Discount Amount

- **Source:** Discount.discount\_percentage \* Order.total\_amount
- **Type:** Calculated
- **Aggregation:** SUM
- **Description:** Represents the total discount applied to orders. This helps analyze the impact of discounts on revenue.

### 4. Total Inventory Cost

- **Source:** Inventory.unit\_price \* Inventory.qty
- **Type:** Additive
- **Aggregation:** SUM
- **Description:** Represents the total cost of inventory items. It can be aggregated by item or supplier to understand inventory costs.

### 5. Total Items Ordered

- **Source:** Count of Order\_items (assuming a table links each order to items)
- **Type:** Additive
- **Aggregation:** COUNT
- **Description:** This measure represents the total number of items ordered. It can be analyzed across time, customers, and menu items to understand purchasing trends.

### 6. Number of Orders

- **Source:** Order.order\_id
- **Type:** Count
- **Aggregation:** COUNT
- **Description:** Represents the total number of orders placed. It can be broken down by time, customer, or employee (waiter/manager).

## SQL Query Execution Order Summary:

1. **Employee**
2. **Customer**
3. **Discount**
4. **Menu** (references **Employee**)
5. **Inventory** (references **Employee**)
6. **Purchase\_Order** (references **Inventory**)
7. **Order** (references **Customer**, **Employee**, and **Discount**)
8. **Payment** (references **Order**)
9. **Add payment\_id foreign key constraint to Order**
10. **Waiter** (specializes **Employee**)
11. **Chef** (specializes **Employee**)
12. **Manager** (specializes **Employee**)
13. **Order\_Menu** (junction table between **Order** and **Menu**)

## Implementation in SQL:

## 1. List of all the tables being created

DBeaver 24.2.1 - <postgres> Script-2

```
CREATE TABLE Employee (
    employee_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    date_of_joining DATE NOT NULL,
    salary DECIMAL(10, 2) NOT NULL
);

CREATE TABLE Customer (
    customer_id SERIAL PRIMARY KEY,
    customer_name VARCHAR(100) NOT NULL,
    phone VARCHAR(15),
    email VARCHAR(100),
    address TEXT
);
```

pg\_tables 1

schemaname	tablename	tableowner	tablespace	hasrules	hastriggers	rowsecurity
public	purchase_order	[NULL]	[v]	[]	[v]	[]
public	employee	[NULL]	[v]	[]	[v]	[]
public	menu	[NULL]	[v]	[]	[v]	[]
public	inventory	[NULL]	[v]	[]	[v]	[]
public	customer	[NULL]	[v]	[]	[v]	[]
public	discount	[NULL]	[v]	[]	[v]	[]
public	Order	[NULL]	[v]	[]	[v]	[]
public	payment	[NULL]	[v]	[]	[v]	[]
public	water	[NULL]	[v]	[]	[v]	[]
public	Foreign Tables	[NULL]	[v]	[]	[v]	[]
public	Views	[NULL]	[v]	[]	[v]	[]
public	Materialized Views	[NULL]	[v]	[]	[v]	[]
public	Indexes	[NULL]	[v]	[]	[v]	[]
public	Functions	[NULL]	[v]	[]	[v]	[]
public	Sequences	[NULL]	[v]	[]	[v]	[]
public	Data types	[NULL]	[v]	[]	[v]	[]

Project - General

Record

EST en\_US Writable Smart Insert 199 : 5894 Sel: 0 | 0 12 row(s) fetched - 0.003s, on 2024-10-04 at 21:07:08

## 2. List of values being populated in the Employee table

DBeaver 24.2.1 - <postgres> Script-2

```
INSERT INTO Manager (employee_id) VALUES
(5); -- Manager Sarah Brown

• INSERT INTO Order_Menu (order_id, menu_id, quantity) VALUES
(1, 1, 2), -- 2 Veg Burgers in order 1
(1, 3, 1), -- 1 French Fries in order 1
(2, 2, 1), -- 1 Chicken Sandwich in order 2
(2, 5, 1), -- 1 Pasta in order 2
(3, 4, 2), -- 2 Grilled Salmon in order 3
(4, 6, 1); -- 1 Garden Salad in order 4

SELECT * FROM pg_catalog.pg_tables WHERE schemaname='public';

select * from employee e ;
```

employee 1

employee_id	name	date_of_joining	salary
1	John Smith	2023-01-10	40,000
2	Jane Doe	2023-07-16	55,000
3	Mark Lee	2023-11-30	60,000
4	Emily Davis	2020-05-22	45,000
5	Sarah Brown	2022-02-17	48,000

Project - General

Record

EST en\_US Writable Smart Insert 199 : 27 [26] Sel: 26 | 1 5 row(s) fetched - 0.007s (0.001s fetch), on 2024-10-04 at 21:09:29

## 3. List of values being populated in the Purchase\_order table

purchase_order1					
	purchase_id	inventory_item_id	order_date	quantity_ordered	supplier_name
1	1	1	2024-01-15	100	Fresh Farm Suppliers
2	2	2	2024-01-16	200	Green Leaf Distributors
3	3	3	2024-01-17	50	Meat Packers Inc.
4	4	4	2024-01-18	150	Local Farmers Market
5	5	5	2024-01-19	30	Seafood World
6	6	6	2024-01-20	80	Gourmet Supplies Co.

Similarly, we have populated all our tables. For now, we have inserted only 5 records in each table, but we aim to add up to 1,000 records by the next milestone. I'm attaching the SQL script in a .txt file format below.

[https://drive.google.com/file/d/1GDqnbHKxYPCRflZLQZI4awitcco\\_hprH/view?usp=sharing](https://drive.google.com/file/d/1GDqnbHKxYPCRflZLQZI4awitcco_hprH/view?usp=sharing)

## Data Population Methodology:

We generated our data in batches using various LLMs and first populated the static data, such as Employee, Customer, and Items. Using this static data, we then generated the transactional data, such as Amount paid, etc.

# Milestone 3

## 1. Fact Tables

**Order:** This fact table captures detailed information about customer orders. Key measures in this table include:

**Order Date:** The date the order was placed.

**Total Sales Amount:** The total value of the order.

**Total Items Ordered:** The number of items in the order.

**Number of Orders:** The number of orders made by a customer.

**Payment:** This fact table captures details related to payments made for orders. Key measures include:

**Payment Date:** The date the payment was made.

**Payment Type:** The method of payment (e.g., cash, credit card).

**Amount:** The amount of money paid for the order.

**Inventory Purchase:** This fact table captures details about inventory purchases. Key measures include:

**Order Date:** The date when the inventory was purchased.

**Quantity Ordered:** The quantity of inventory items ordered.

## 2. Dimension Tables

Dimension tables provide descriptive data for the facts. In this model, the following dimensions are present:

**Employee:** Contains information about the employees involved in the business.

Attributes include:

Employee\_ID, Name, Date\_of\_Joining, Salary, and Employee\_Type (whether they are a Waiter, Chef, or Manager).

The specialization of employees (Chef, Waiter, Manager) is represented by different roles under the Employee\_Type dimension.

**Customer:** Contains details about customers, such as:

Customer\_ID, Customer\_Name, Phone, and Address.

**Menu:** Represents the items on the menu, capturing details such as:

Item\_ID, Item\_Name, Description, and Price.

The menu is also linked to the chef responsible for specific items.

**Discount:** Contains information about discounts applied to orders, including:

Discount\_ID, Discount\_Percent, and Description.

**Inventory:** Stores details about inventory items, including:

Inventory\_Item\_ID, Item\_Name, Qty (Quantity), Unit\_Price, and Supplier\_Name.

**Date:** This dimension captures the time-related data and is linked to multiple fact tables.

The date hierarchy is represented by:

Date, Month, and Year levels, allowing for time-based aggregation and analysis.

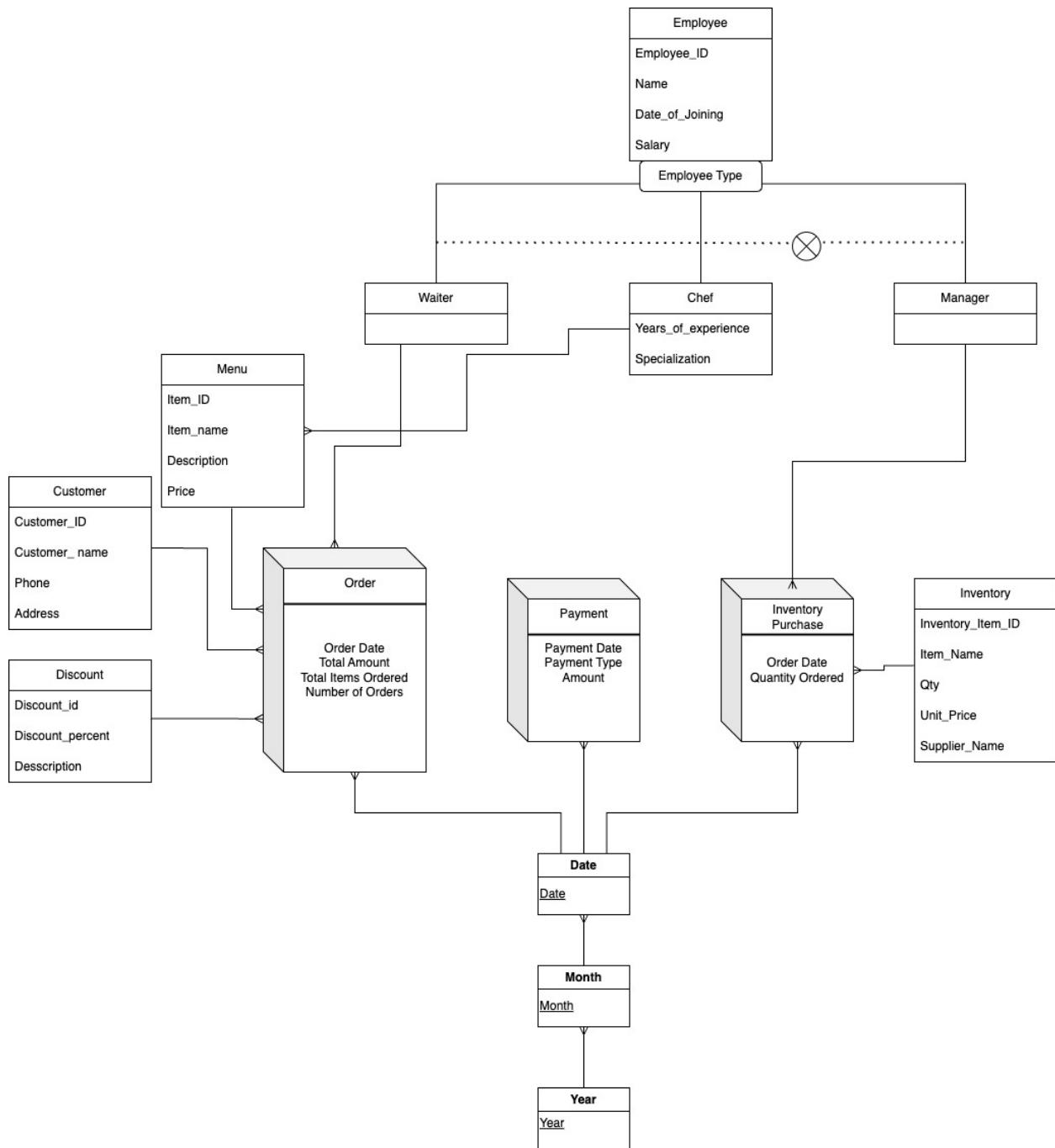
### 3. Hierarchies

**Date Dimension Hierarchy:** The Date dimension contains a hierarchy that breaks down into Year, Month, and Day, which supports time-based aggregation (e.g., total sales per year, total payments per month).

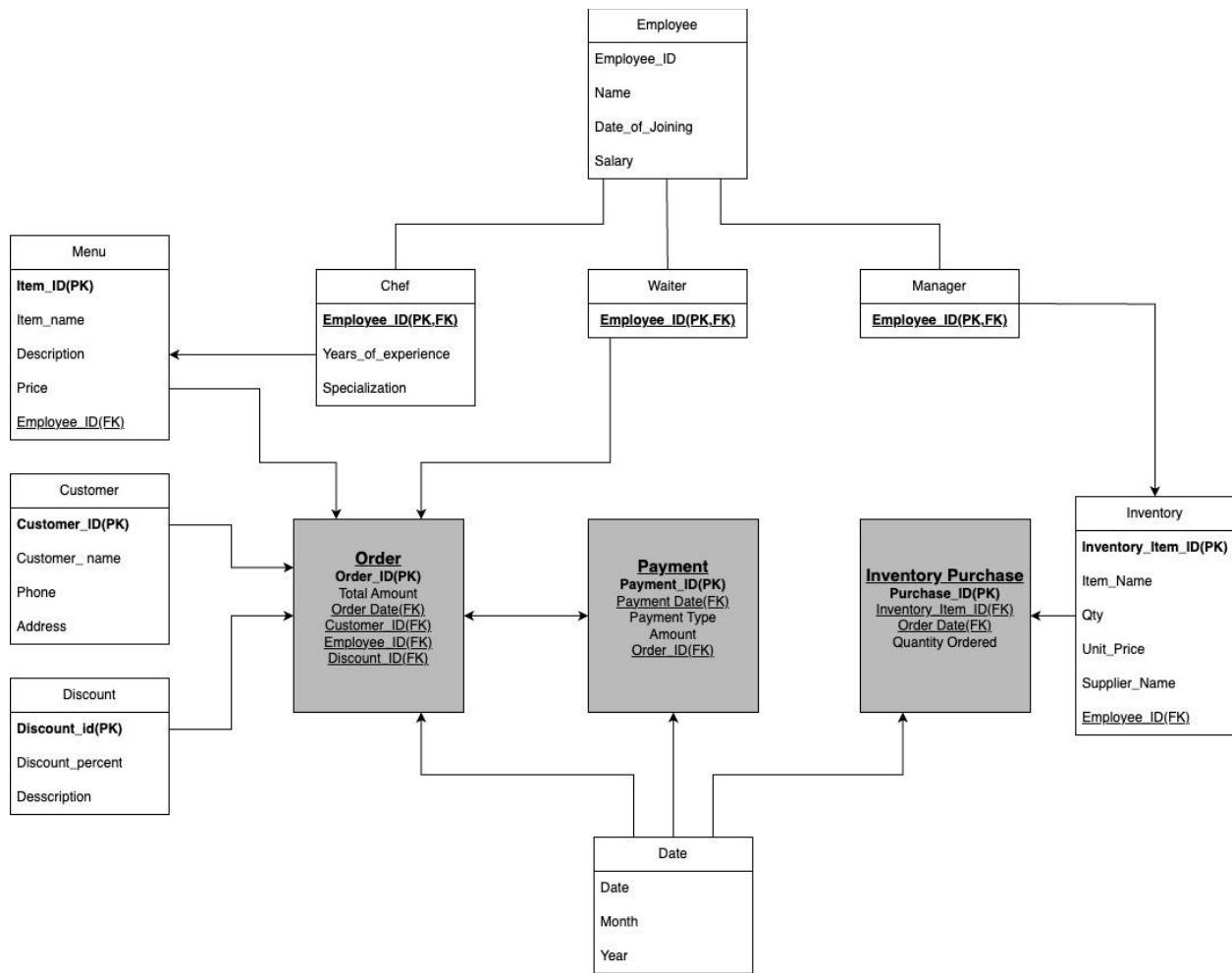
**Employee Role Hierarchy:**

The Employee dimension includes a hierarchy for employee types, categorized as Waiter, Chef, and Manager.

Conceptual Model:



Logical Model:



**Created the Schema for the Warehouse in PostgresSQL:**

- The tables created:

The screenshot shows the DBeaver interface with the following tree structure:

- Enter a part of object name here
- > DBeaver Sample Database (SQLite)
- > postgres localhost:5432
- < postgres 2 localhost:5432
  - ↳ Databases
    - CoffeeStore
    - DW\_Milestone3
  - ↳ Schemas
    - ↳ public
      - Tables
        - Order 24K
        - chef 24K
        - customer 32K
        - date\_dim 24K
        - discount 32K
        - employee 24K
        - inventory 24K
        - inventory\_purchase 24K
        - manager 24K
        - menu 32K
        - payment 24K
        - waiter 24K
      - Foreign Tables
      - Views
      - Materialized Views
      - Indexes
      - Functions
      - Sequences
      - Data types

- The schema for Employee table, I have added dummy data.

The screenshot shows the DBBeaver interface with the 'Milestone3.sql' script open. The 'employee' table is selected, displaying the following data:

	employee_id	first_name	last_name	date_of_joining	salary
1	1	John	Doe	2020-01-10	50,000
2	2	Sarah	Connor	2019-03-14	60,000
3	3	Alice	Johnson	2021-05-21	60,000
4	4	Bob	Smith	2019-07-15	52,000
5	5	Charlie	Davis	2020-08-25	57,000
6	6	Eve	White	2017-11-30	62,000
7	7	James	Bond	2016-01-19	75,000

- The fact table Order, which has the order details:

The screenshot shows the DBBeaver interface with the 'Milestone3.sql' script open. The 'Order' table is selected, displaying the following data:

	order_id	total_amount	order_date	customer_id	employee_id	discount_id
1	4,001	45.97	101	1,001	2	500
2	4,002	35.98	102	1,002	2	501
3	4,003	50.99	103	1,003	4	[NULL]
4	4,004	20.99	104	1,004	6	502
5	4,005	60.97	105	1,005	4	500

- The fact table Payment, which has the payment details:

Screenshot of DBBeaver 24.2.2 showing a PostgreSQL database connection.

The Database Navigator pane shows the following structure:

- Projects
- Database Navigator (selected)
- public@O\_W\_Milestone3
- postgres 2
- postgres localhost:5432
- Databases (selected)
- CoffeeStore
- Milestone3
- Schemas (selected)
- public
  - Tables (selected)
  - Order
  - chef
  - customer
  - date\_dim
  - discount
  - employee
  - inventory
  - inventory\_purchase
  - manager
  - menu
  - payment
  - walter
- Foreign Tables
- Views
- Materialized Views
- Indexes
- Functions
- Sequences
- Data types
- Aggregate functions
- Event Triggers
- Extensions
- Storage
- System Info

The SQL Editor pane contains the following SQL code:

```
(5002, 102, 'Cash', 35.98, 4002),
(5003, 103, 'Debit Card', 50.99, 4003),
(5004, 104, 'Cash', 20.99, 4004),
(5005, 105, 'Credit Card', 60.97, 4005);

-- Insert into Inventory_Purchase table
INSERT INTO Inventory_Purchase (Purchase_ID, Inventory_Item_ID, Order_Date, Quantity_Ordered)
VALUES
(6001, 3001, 101, 50),
(6002, 3002, 102, 25),
(6003, 3003, 103, 30),
(6004, 3004, 104, 100),
(6005, 3005, 105, 10);

select * from payment p;
```

The Results pane displays the data from the payment table:

payment_id	order_id	payment_date	payment_type	amount
5,001	101	2024-10-18	Credit Card	45.97
5,002	102	2024-10-18	Cash	35.98
5,003	103	2024-10-18	Debit Card	50.99
5,004	104	2024-10-18	Cash	20.99
5,005	105	2024-10-18	Credit Card	60.97

Bottom status bar: EST en\_US Writable, Smart Insert, 215 : 1 [26], Sel: 26 | 1, 5 row(s) fetched - 0.005s, on 2024-10-18 at 16:30:48.

## **Primary Events (Fact Tables):**

### **1. Order Event (Order table):**

- **Description:** This event occurs when a customer places an order. It captures information such as the total amount of the order, the customer who placed it, the waiter or manager who handled the order, and any discount applied.
- **Key Attributes:**
  - Order\_ID: The unique identifier for the order.
  - Total\_Amount: The total value of the order.
  - Order\_Date: The date the order was placed.
  - Customer\_ID: The customer who placed the order.
  - Employee\_ID: The waiter or manager who handled the order.
  - Discount\_ID: Any discount applied to the order.

### **2. Payment Event (Payment table):**

- **Description:** This event occurs when a customer makes a payment for an order. It captures details such as the payment type, the amount paid, and the corresponding order.
- **Key Attributes:**
  - Payment\_ID: The unique identifier for the payment.
  - Payment\_Date: The date the payment was made.
  - Payment\_Type: The method of payment (credit card, cash, etc.).
  - Amount: The amount paid.
  - Order\_ID: The order the payment corresponds to.

### **3. Inventory Purchase Event (Inventory\_Purchase table):**

- **Description:** This event occurs when the restaurant or business purchases inventory items. It captures details like the item purchased, the quantity ordered, and the date of the purchase.
- **Key Attributes:**
  - Purchase\_ID: The unique identifier for the purchase.
  - Inventory\_Item\_ID: The inventory item being purchased.
  - Order\_Date: The date of the inventory purchase.
  - Quantity\_Ordered: The quantity of the inventory item purchased.

## OLAP Operations:

- **Query 1:** Total Sales Amount by Customer, Year, and Month

ROLLUP\*(Order, Customer → Customer\_Name, Date → Year, Date → Month,  
SUM(Total\_Amount))

Description: Aggregates sales handled by each employee.

- **Query 2:** Total Sales by Employee (Waiters)

ROLLUP\*(Order, Employee → Employee\_Name, SUM(Total\_Amount))

Description: Aggregates sales handled by each employee.

- **Query 3:** Total Payments by Payment Type

AGGREGATE(Payment, Payment\_Type, SUM(Amount))

Description: Aggregates total payments received by payment type.

- **Query 4:** Drill Down Sales from Year to Month to Day

DRILLDOWN(Order, Date → Year, Date → Month, Date → Day, SUM(Total\_Amount))

Description: Drills down from yearly sales to daily sales.

- **Query 5:** Slice: Sales in the Year 2023

SLICE(Order, Date.Year = 2023, SUM(Total\_Amount))

Description: Slices the sales data to show only the year 2023.

- **Query 6:** Inventory Purchases by Supplier and Item

ROLLUP\*(Inventory\_Purchase, Supplier → Supplier\_Name, Inventory → Item\_Name,  
SUM(Quantity\_Ordered))

Description: Summarizes inventory purchases by supplier and item.

- **Query 7:** Payments by Date and Employee

ROLLUP\*(Payment, Date → Year, Employee → Employee\_Name, SUM(Amount))

Description: Aggregates payments received by each employee per year.

- **Query 8:** Top 5 Best-Selling Menu Items

ROLLUP\*(Menu, Item\_Name, SUM(Total\_Amount))

ORDER BY SUM(Total\_Amount) DESC LIMIT 5

Description: Finds the top 5 menu items based on total sales

- **Query 9:** Total Discount Given Per Customer by Year

ROLLUP\*(Order, Customer → Customer\_Name, Date → Year, Discount → Discount\_percent,  
SUM(Total\_Amount))

Description: Aggregates total discount per customer by year.

- **Query 10:** Inventory Purchases in 2023

SLICE(Inventory\_Purchase, Date.Year = 2023, SUM(Quantity\_Ordered))

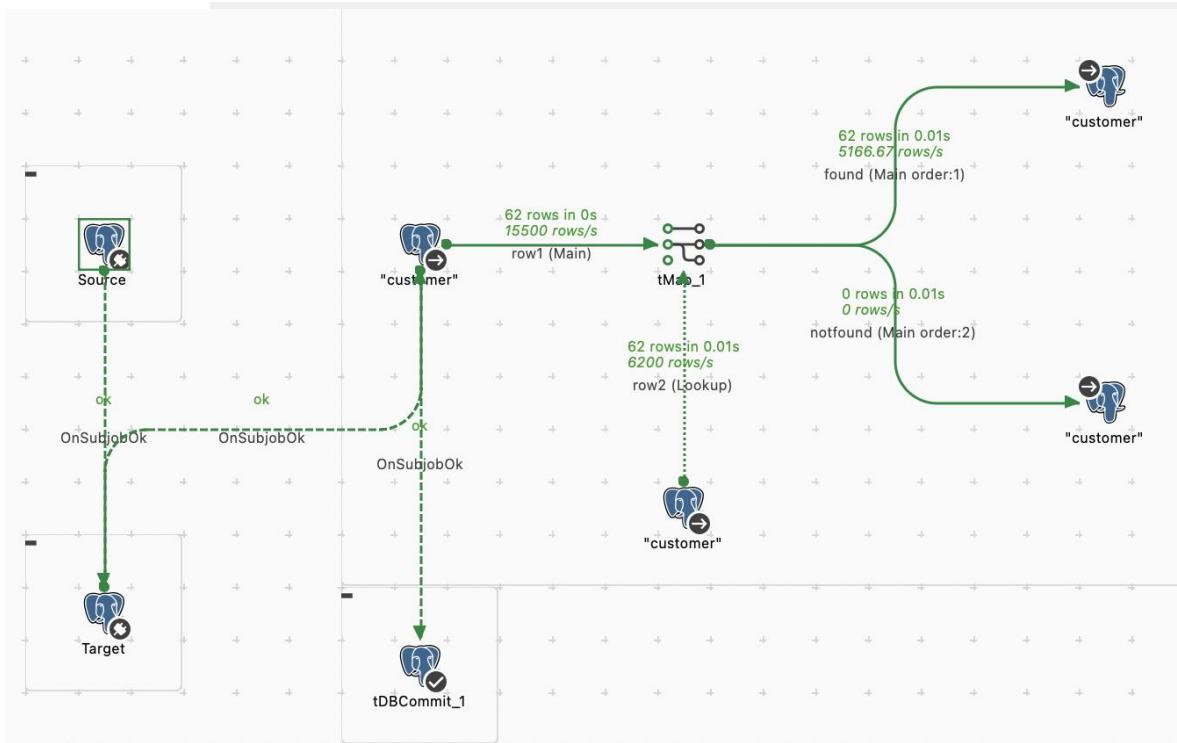
Description: Slices the inventory purchases to show data for the year 2023.

# Milestone 4

## Implemented:

- Insert/update data with surrogate key
- Control flow
- Implemented Type 3 SCD
- Implemented 2 calculated measures:
  - o purchase\_amount → inventorypurchase in Target
  - o net\_amount → order\_details in Target
- Used 2 sources of data:
  - o csv file for date
  - o PostgresDB for the rest
- Performed transformations:
  - o Date format
  - o Manipulated the type of data for calculations

## Insert/Update flow for Customer:



## tMap for Customer:

Talend Real-time Big Data Platform - tMap - tMap\_1

**row1**

Column	customer_id	customer_name	phone	email	address
customer_id					
customer_name					
phone					
email					
address					

**row2**

Property	Value
Lookup Model	Load once
Match Model	Unique match
Join Model	Inner Join
Store temp data	false

**Expr. key**

row1.customer\_id

**Column**

row2.ckey

row2.customer\_id

row1.customer\_name

row1.phone

row1.email

row1.address

**found**

Property	Value
Catch output reject	false
Catch lookup inner join reject	false
Schema Type	Built-In

**Expression**

row2.ckey

row2.customer\_id

row1.customer\_name

row1.phone

row1.email

row1.address

**Column**

row2.ckey

customer\_id

customer\_name

phone

email

address

**notfound**

Property	Value
Catch output reject	false
Catch lookup inner join reject	true
Schema Type	Built-In

**Expression**

row1.customer\_id

row1.customer\_name

row1.phone

row1.email

row1.address

**Column**

row1.ckey

customer\_id

customer\_name

phone

email

address

**Schema editor**

**row1**

Column	Key	Type	Nullab	Date Pattern (Ctrl+! Length)	Precision	Default	Comment
customer_id		int		10	0		nextval('c
customer_name		String		255	0		
phone		String		50	0		
email		String		255	0		
address		String		255	0		

**row2**

Column	Key	Type	Nullab	Date Pattern (Ctrl+! Length)	Precision	Default	Comment
customer_id		int		10	0		
customer_name		String		255	0		
phone		String		50	0		
email		String		255	0		
address		String		255	0		2147483647:0

**found**

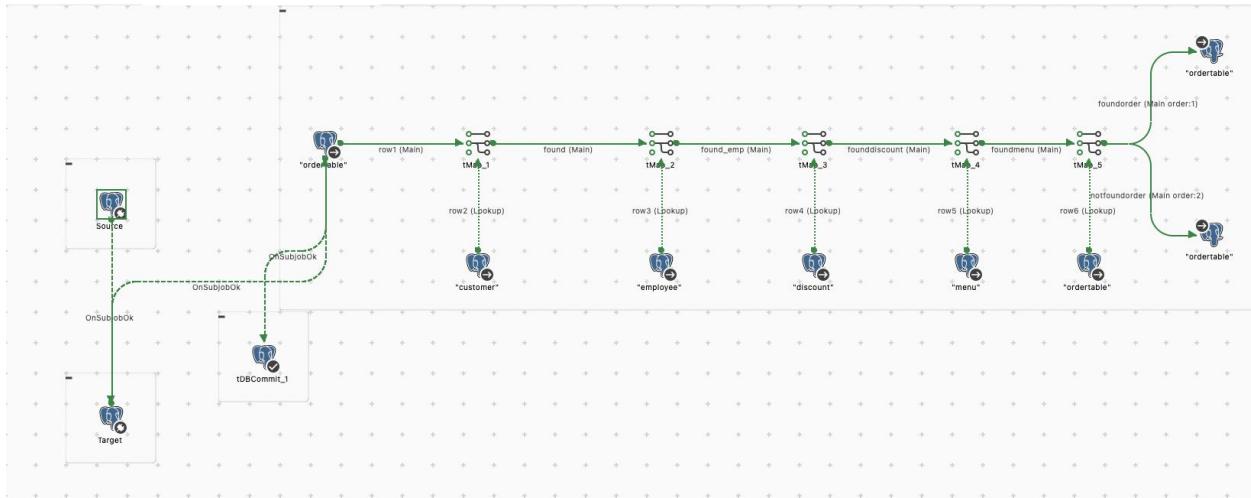
Column	Key	Type	Nullab	Date Pattern (Ctrl+! Length)	Precision	Default	Comment
customer_id		int		10	0		
customer_name		String		255	0		
phone		String		50	0		
email		String		255	0		
address		String		255	0		

**notfound**

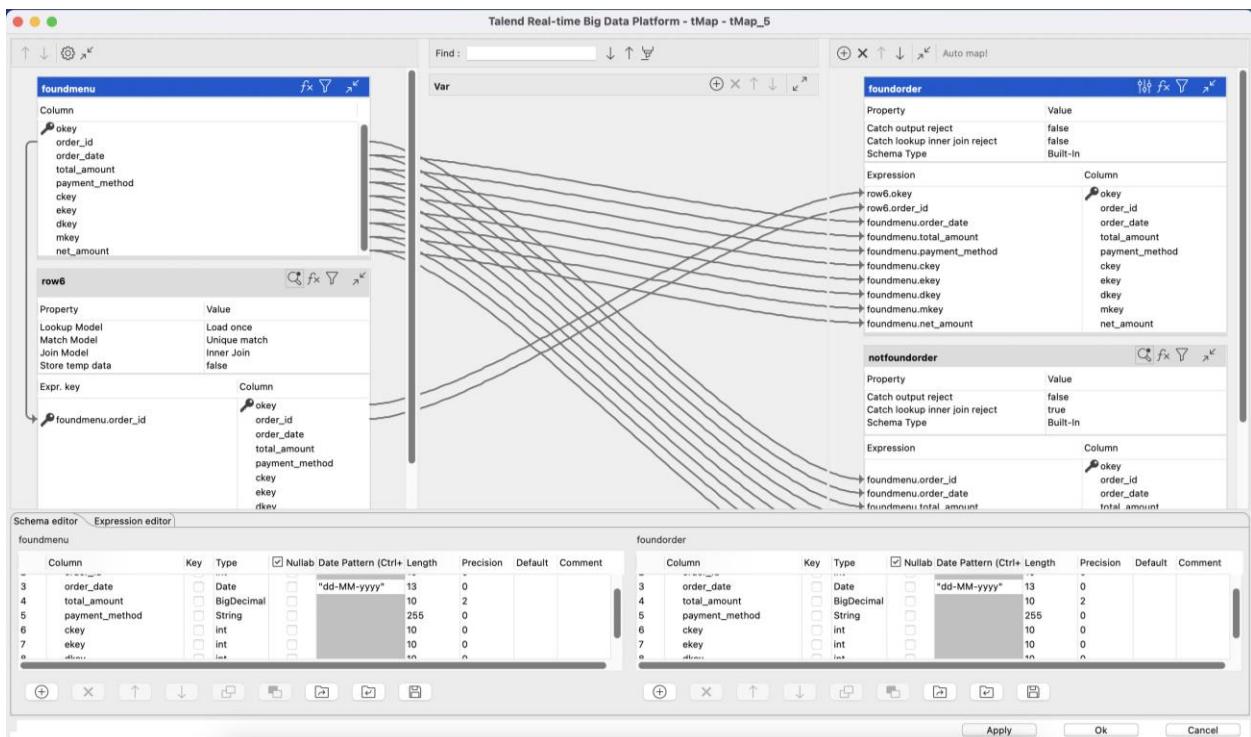
Column	Key	Type	Nullab	Date Pattern (Ctrl+! Length)	Precision	Default	Comment
customer_id		int		10	0		
customer_name		String		255	0		
phone		String		50	0		
email		String		255	0		
address		String		255	0		

Apply Ok Cancel

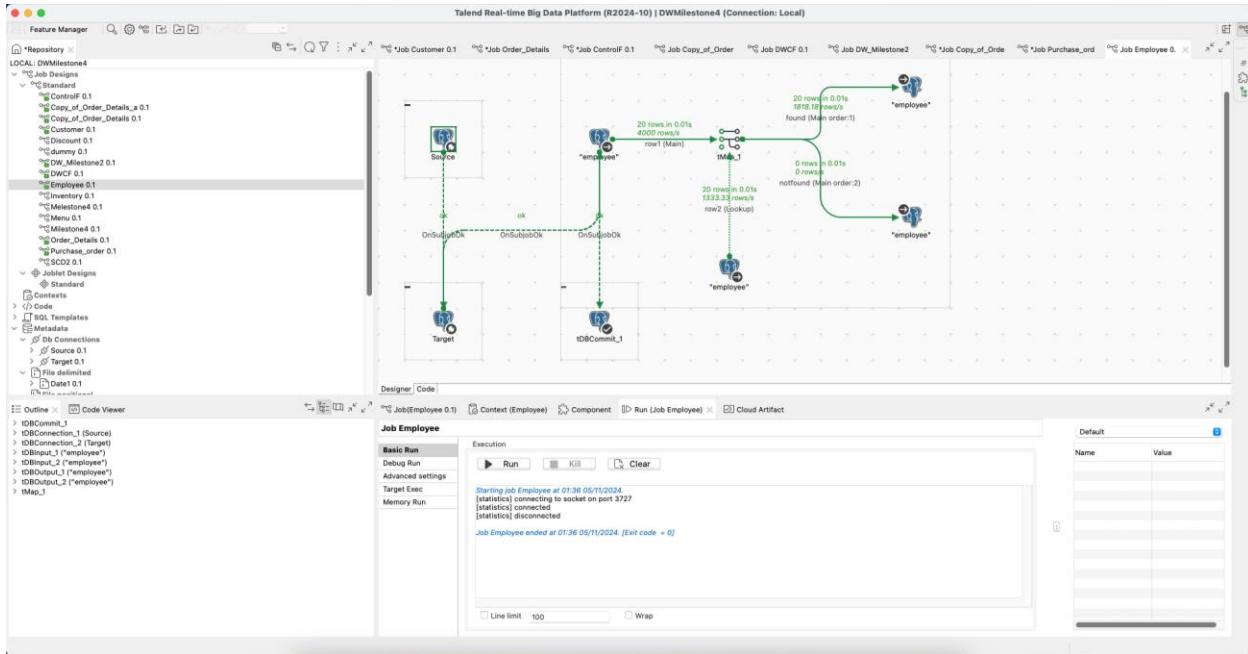
## Insert/Update flow for Order\_details:



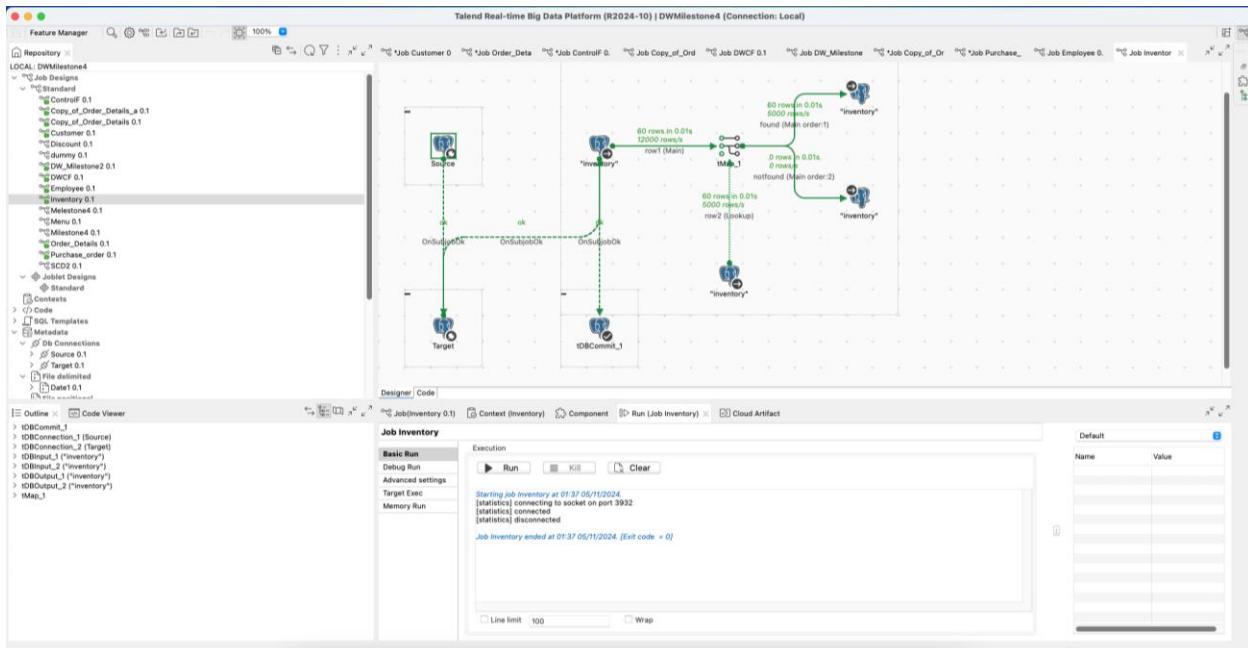
## tMap for Order\_details:



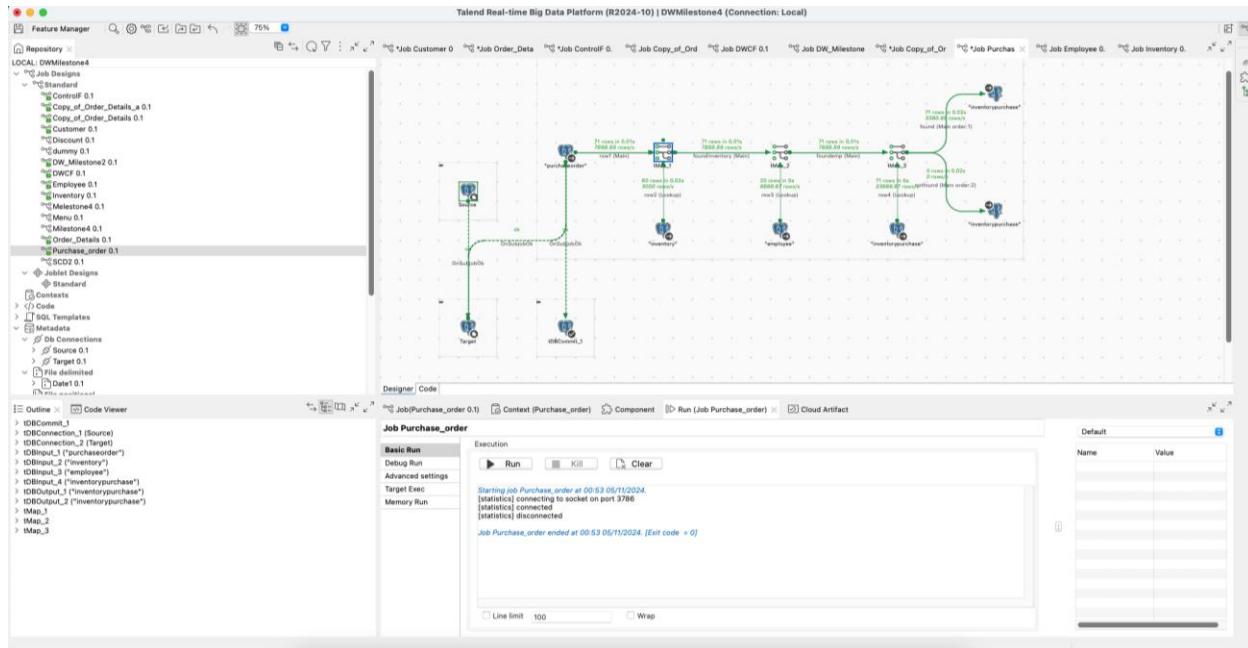
## Insert and Update flow for Employee:



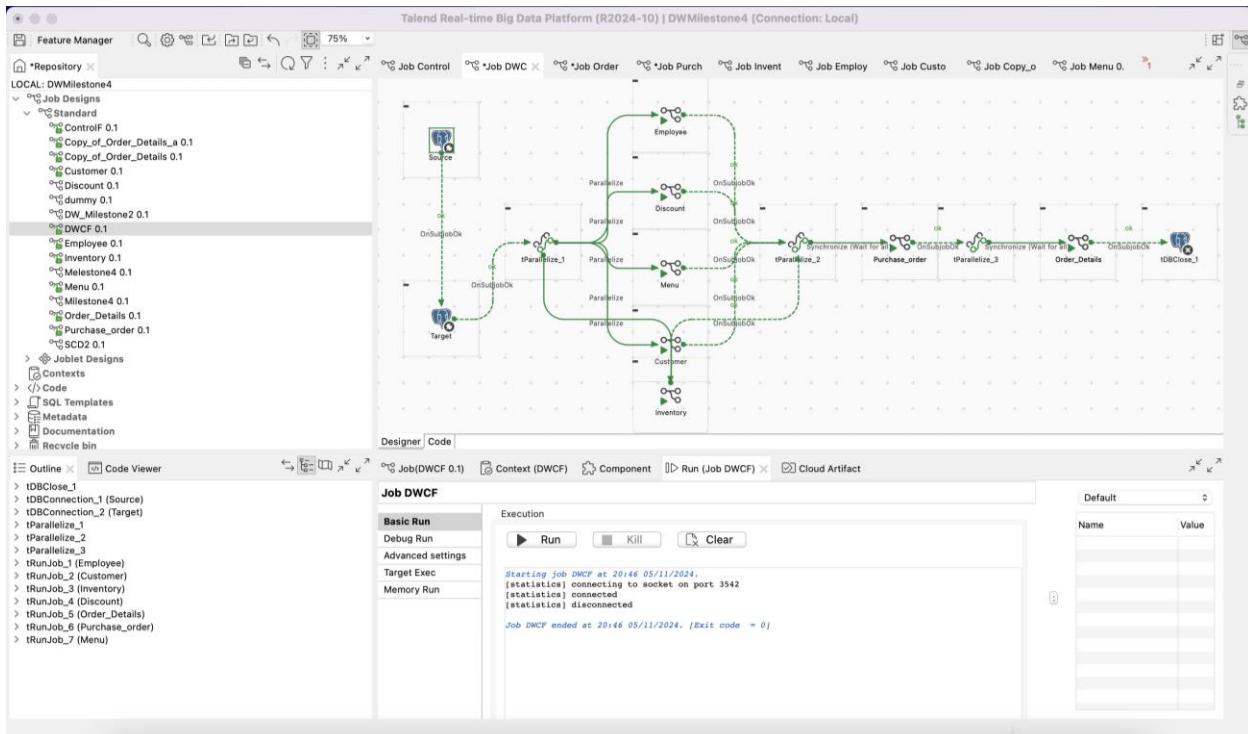
## Insert and Update flow for Inventory:



## Insert and Update flow for Purchase Order (Fact Table):



## Control Flow:



## Data populated in the Data Warehouse with the Surrogate Keys:

- Customer: Surrogate Key → ckey

DBeaver 24.2.2 - customer

The screenshot shows the DBeaver interface with the 'customer' table selected in the central grid. The table has columns: ckey, customer\_id, customer\_name, phone, email, and address. The data consists of 46 rows, each with a unique surrogate key (1 to 46) and real-world values for the other columns. The primary key is ckey.

ckey	customer_id	customer_name	phone	email	address
1	301	John Doe	555-0101	john.doe@example.com	101 Green Lane
2	302	Jane Smith	555-0102	jane.smith@example.com	102 Oak Drive
3	303	Alice Johnson	555-0103	alice.johnson@example.com	103 Pine Street
4	304	Carlos Brown	555-0104	carlos.brown@example.com	104 Maple Avenue
5	305	Diana Harris	555-0105	diana.harris@example.com	105 Elm Road
6	306	Ethan Wright	555-0106	ethan.wright@example.com	106 Cedar Blvd
7	307	Fiona Glen	555-0107	fiona.glen@example.com	107 Birch Path
8	308	George King	555-0108	george.king@example.com	108 Walnut Street
9	309	Hanna Ziegler	555-0109	hanna.ziegler@example.com	109 Cherry Lane
10	310	Ian Fleming	555-0110	ian.fleming@example.com	110 Chestnut Road
11	311	Jane Austin	555-0111	jane.austin@example.com	111 Willowwood
12	312	Kyle Mendez	555-0112	kyle.mendez@example.com	112 Grape Street
13	313	Liam Neeson	555-0113	liam.neeson@example.com	113 Berry Road
14	314	Mia Wong	555-0114	mia.wong@example.com	114 Lime Path
15	315	Nina Dobrev	555-0115	nina.dobrev@example.com	115 Coconut Lane
16	316	Oscar Wilde	555-0116	oscar.wilde@example.com	116 Mango Drive
17	317	Pia Guerra	555-0117	pia.guerra@example.com	117 Banana Avenue
18	318	Quinn Fabray	555-0118	quinn.fabray@example.com	118 Orange Street
19	319	Rosa Diaz	555-0119	rosa.diaz@example.com	119 Melon Blvd
20	320	Steven Tyler	555-0120	steven.tyler@example.com	120 Watermelon Path
21	363	Harry Potter	555-0124	john.doe@example.com	44 Parker Hill
22	321	Tina Fey	555-0121	tina.fey@example.com	121 Kiwi Lane
23	322	Ben Stiller	555-0122	ben.stiller@example.com	122 Lemon Drive
24	323	Chris Jones	555-0123	chris.jones@example.com	123 Pine Street
25	324	David Guetta	555-0124	david.guetta@example.com	124 Apricot Avenue
26	325	Emma Watson	555-0125	emma.watson@example.com	125 Pear Blvd
27	326	Frank Ocean	555-0126	frank.ocean@example.com	126 Avocado Road
28	327	Grace Hopper	555-0127	grace.hopper@example.com	127 Fig Path
29	328	Harry Styles	555-0128	harry.styles@example.com	128 Date Lane
30	329	Isabella Knight	555-0129	isabella.knight@example.com	129 Guava Drive
31	330	Jason Momoa	555-0130	jason.momoa@example.com	130 Papaya Avenue
32	331	Katy Perry	555-0131	katy.perry@example.com	131 Curant Street
33	332	Leo DiCaprio	555-0132	leo.dicaprio@example.com	132 Raspberry Blvd
34	333	Molly Ringwald	555-0133	molly.ringwald@example.com	133 Blackberry Path
35	334	Nate Archibald	555-0134	nate.archibald@example.com	134 Elderberry Lane
36	335	Olivia Pope	555-0135	olivia.pope@example.com	135 gooseberry Drive
37	336	Pete Wentz	555-0136	pete.wentz@example.com	136 Cranberry Avenue
38	337	Quentin Tarantino	555-0137	quentin.tarantino@example.com	137 Damson Street
39	338	Ryan Gosling	555-0138	ryan.gosling@example.com	138 Jujube Blvd
40	339	Scarlett Johansson	555-0139	scarlett.johansson@example.com	139 Larches Path

EST en\_US      62 row(s) fetched - 0.016s, on 2024-11-05 at 01:15:45

- Discount: Surrogate Key → dkey

DBeaver 24.2.2 - discount

The screenshot shows the DBeaver interface with the 'discount' table selected in the central grid. The table has columns: dkey, discount\_id, discount\_percent, and description. The data consists of 2 rows, each with a unique surrogate key (1 and 2) and real-world values for the other columns. The primary key is dkey.

dkey	discount_id	discount_percent	description
1	1	50	50% Off - Major Discount
2	2	0	No Discount Applied

EST en\_US      2 row(s) fetched - 0.000s, on 2024-11-05 at 01:17:07

- Employee: Surrogate Key → ekey

DBeaver 24.2.2 - employee

Grid

	ekey	employee_id	name	date_of_joining	salary	employee_type
1	21	1	Allison Johnson	2019-01-10	55,000	Manager
2	22	2	Mark Chen	2019-02-15	50,000	Analyst
3	23	3	Carolin Diaz	2019-03-20	48,000	Walter
4	24	4	Diana Reyes	2019-04-25	51,000	Chef
5	25	5	Ethan Hawke	2019-05-30	53,000	Manager
6	26	6	Fiona Glen	2019-06-05	49,500	Walter
7	27	7	George King	2019-07-10	47,000	Walter
8	28	8	Hanna Ziegler	2019-08-15	48,500	Chef
9	29	9	Ian Fleming	2019-09-20	52,000	Manager
10	30	10	Judy Austin	2019-10-25	54,000	Chef
11	31	11	Kyle Mendez	2019-11-30	56,000	Manager
12	32	12	Liam Neeson	2019-12-05	45,500	Walter
13	33	13	Mia Wong	2020-01-10	57,500	Manager
14	34	14	Nina Dobrev	2020-02-15	46,500	Walter
15	35	15	Olivia Wilde	2020-03-20	48,000	Analyst
16	36	16	Pia Guerra	2020-04-25	50,500	Chef
17	37	17	Quinn Fakray	2020-05-30	51,500	Walter
18	38	18	Rosa Diaz	2020-06-05	52,500	Manager
19	39	19	Steven Tyler	2020-07-10	49,500	Walter
20	40	20	Tina Fey	2020-08-15	55,000	Manager

- Inventory: Surrogate Key → ikey

DBeaver 24.2.2 - inventory

Grid

	ikey	inventory_id	item_name	qty	unit_price
1	62	1	All-purpose Flour	100	0.95
2	63	2	Granulated Sugar	150	0.85
3	64	3	Sea Salt	200	0.5
4	65	4	Black Pepper	120	3
5	66	5	Olive Oil	80	8.5
6	67	6	Balsamic Vinegar	60	7.25
7	68	7	Rice	300	1.1
8	69	8	Pasta	250	1.3
9	70	9	Tomato Sauce	180	2.2
10	71	10	Cornel Beans	200	0.99
11	72	11	Tuna Cans	100	2.5
12	73	12	Corn Flakes	90	3.1
13	74	13	Coffee Beans	70	16
14	75	14	Green Tea	85	6
15	76	15	Baking Powder	75	1.4
16	77	16	Yeast	50	4
17	78	17	Butter	130	2.5
18	79	18	Milk	200	0.9
19	80	19	Eggs	400	0.25
20	81	20	Cheddar Cheese	110	5
21	82	21	Mozzarella Cheese	120	4.75
22	83	22	Apples	160	1
23	84	23	Oranges	180	0.6
24	85	24	Oranges	170	1.2
25	86	25	Broccoli	90	2
26	87	26	Chicken Breasts	150	3.5
27	88	27	Beef Steaks	130	5.2
28	89	28	Pork Chops	140	4
29	90	29	Salmon Fillets	100	10
30	91	30	Shrimp	80	12
31	92	31	Lobster Tails	50	20
32	93	32	Scallops	60	18
33	94	33	Cod Fillets	90	8
34	95	34	Tilapia	110	6.5
35	96	35	Haddock	120	7
36	97	36	Shrimps	150	8
37	98	37	Mussels	160	8.5
38	99	38	Oysters	140	11
39	100	39	Crab Meat	70	15
40	101	40	Smurfs	RR	19

- Inventory purchase: Surrogate Key → pkey

DBeaver 24.2.2 - inventorypurchase

Grid

	pkey	purchase_id	order_date	quantity_ordered	supplier_name	key	purchase_amount
1	261	8	2024-02-07	375	Bulk Basics LLC	69	28
2	268	9	2024-02-07	270	Organic Farms Co.	70	29
3	269	10	2024-02-08	300	Culinary Essentials Inc.	71	30
4	270	11	2024-02-05	280	Global Supplies Inc.	72	31
5	271	12	2024-02-06	260	Quality Foods Ltd.	73	32
6	272	13	2024-02-07	240	Bulk Basics LLC	74	33
7	273	14	2024-02-08	220	Organic Farms Co.	75	34
8	274	15	2024-02-09	200	Culinary Essentials Inc.	76	35
9	275	16	2024-02-10	180	Global Supplies Inc.	77	36
10	276	17	2024-02-11	160	Quality Foods Ltd.	78	37
11	277	18	2024-02-05	140	Bulk Basics LLC	79	38
12	278	19	2024-02-06	130	Organic Farms Co.	80	39
13	279	20	2024-02-07	100	Culinary Essentials Inc.	81	40
14	280	21	2024-02-08	170	Global Supplies Inc.	82	21
15	281	22	2024-02-09	190	Quality Foods Ltd.	83	22
16	282	23	2024-02-10	210	Bulk Basics LLC	84	23
17	283	24	2024-02-11	230	Organic Farms Co.	85	24
18	284	25	2024-02-05	250	Culinary Essentials Inc.	86	25
19	285	26	2024-02-06	270	Global Supplies Inc.	87	26
20	286	27	2024-02-07	290	Quality Foods Ltd.	88	27
21	287	28	2024-02-08	310	Bulk Basics LLC	89	28
22	288	29	2024-02-09	330	Organic Farms Co.	90	29
23	289	30	2024-02-10	350	Culinary Essentials Inc.	91	30
24	290	31	2024-02-08	200	Global Supplies Inc.	91	31
25	291	32	2024-02-09	190	Quality Foods Ltd.	92	32
26	292	33	2024-02-09	210	Culinary Essentials Inc.	93	33
27	293	34	2024-02-09	180	Organic Farms Co.	94	34
28	294	35	2024-02-09	160	Global Supplies Inc.	95	35
29	295	36	2024-02-10	170	Bulk Basics LLC	96	36
30	296	37	2024-02-10	140	Culinary Essentials Inc.	97	37
31	297	38	2024-02-10	220	Organic Farms Co.	98	38
32	298	39	2024-02-11	130	Global Supplies Inc.	99	39
33	299	40	2024-02-11	240	Quality Foods Ltd.	100	40
34	300	41	2024-02-11	190	Bulk Basics LLC	101	21
35	301	42	2024-02-11	175	Culinary Essentials Inc.	102	22
36	302	43	2024-02-11	180	Organic Farms Co.	103	23
37	303	44	2024-02-11	150	Global Supplies Inc.	104	24
38	304	45	2024-02-11	210	Quality Foods Ltd.	105	25
39	305	46	2024-02-11	160	Bulk Basics LLC	106	30
40	306	47	2024-02-11	165	Culinary Essentials Inc.	107	37

71 row(s) fetched - 0.006s, on 2024-11-05 at 01:15:29

- menu: Surrogate Key → mkey

DBeaver 24.2.2 - menu

Grid

	mkey	item_id	item_name
1	84	34	Kung Pao Chicken
2	85	35	Fish Tacos
3	86	36	Quesadilla
4	87	37	Vegan Burger
5	88	38	Beef Tacos
6	89	39	Spicy Noodle Soup
7	90	40	Pulled Pork Sandwich
8	91	41	Crispy Tofu Bowl
9	92	42	Salmon Sushi
10	93	43	Spaghetti Bolognese
11	94	44	Chicken Caesar Wrap
12	95	45	Pho
13	96	46	Jambalaya
14	97	47	Stuffed Bell Peppers
15	98	48	Crab Cakes
16	99	49	Gyro Plate
17	1	50	Spaghetti Carbonara
18	52	51	Fettuccine Alfredo
19	53	52	Garlic Shrimp
20	54	53	Beef Wellington
21	55	54	Chicken Parmesan
22	56	55	Peking Duck
23	57	56	Margarita Pizza
24	58	57	Port Belly Ramen
25	59	58	Fish and Chips
26	60	59	Crispy Chicken Sandwich
27	61	60	Grilled Salmon
28	62	61	Barbecue Ribs
29	63	62	Tuna Tartare
30	64	63	Steak Frites
31	65	64	Sushi Platter
32	66	65	Lamb Curry
33	67	66	Beef Pho
34	68	67	Roast Chicken
35	69	68	Vegetarian Lasagna
36	70	69	Duck a Orange
37	71	70	Lobster Bisque
38	72	71	Mushroom Risotto
39	73	72	Tom Yum Goong
40	74	73	Falafel Wrap

50 row(s) fetched - 0.012s, on 2024-11-05 at 01:18:51

- order: Surrogate Key → okey

DBeaver 24.2.2 - orderable

Database Navigator > Projects

Enter a part of object name here

45.79.206.143 45.79.206.143:3306  
DBeaver Sample Database (SQLite)  
postgres 2 localhost:5432

Databases  
Customer  
DW\_Milestone2  
DW\_Milestone3  
Norwind  
Project\_DW  
Source  
Target  
Schemas  
public  
Tables  
customer  
customer1  
customer2  
date  
discount  
employee  
inventory  
menu  
orderable  
orderTable  
Tables  
customer  
customer1  
customer2  
date  
discount  
employee  
inventory  
inventorypurchase  
menu  
orderable  
Design Tables  
Views  
Materialized Views  
Indexes  
Functions  
Sequences  
Data types  
Aggregate functions

Project - General < DataSources < Record

Name  
Bookmarks  
Dashboards  
Diagrams  
Scripts

Grid  
Text  
ER Diagram

orderTable | Enter a SQL expression to filter results (use Ctrl+Space)

okey order\_id order\_date total\_amount ckey okey diey mkey net\_amount

	okey	order_id	order_date	total_amount	ckey	okey	diey	mkey	net_amount
1	660	62	2024-02-11	64.28	debit_card	302	31	37	62
2	661	63	2024-02-11	45.67	zelle	303	32	38	63
3	662	64	2024-02-05	38.22	cash	304	36	37	64
4	663	65	2024-02-06	90.43	credit_card	305	32	38	65
5	664	66	2024-02-07	91.77	debit_card	306	33	37	66
6	665	67	2024-02-08	56.11	zelle	307	35	38	67
7	666	68	2024-02-09	18.24	cash	308	40	37	68
8	667	69	2024-02-10	28.55	credit_card	309	40	38	69
9	668	70	2024-02-11	67.78	debit_card	310	31	37	70
10	669	71	2024-02-05	50.32	zelle	311	39	37	71
11	670	72	2024-02-06	88.19	cash	312	33	38	72
12	671	73	2024-02-07	21.85	credit_card	313	32	37	73
13	672	74	2024-02-07	26.88	zelle	314	37	38	74
14	673	75	2024-02-08	60.39	credit_card	315	39	37	75
15	674	76	2024-02-09	86.79	debit_card	316	37	38	76
16	675	77	2024-02-10	60.12	zelle	317	32	37	77
17	676	78	2024-02-11	31.55	cash	318	32	38	78
18	677	79	2024-02-05	98.76	credit_card	319	34	37	79
19	678	170	2024-02-06	44.23	debit_card	320	30	38	80
20	782	184	2024-02-06	49.57	cash	321	31	37	81
21	783	185	2024-02-07	10.84	credit_card	322	37	38	82
22	784	186	2024-02-08	71.34	debit_card	323	33	37	83
23	785	187	2024-02-09	28.45	zelle	324	32	38	84
24	822	224	2024-02-11	6.6	cash	325	34	38	85
25	823	225	2024-02-05	49.99	credit_card	326	34	37	86
26	824	226	2024-02-06	89.12	debit_card	327	39	38	87
27	825	227	2024-02-07	77.2	zelle	328	33	37	88
28	826	228	2024-02-08	65.55	cash	329	33	38	89
29	827	229	2024-02-09	40.4	credit_card	330	38	37	90
30	828	230	2024-02-10	25.25	debit_card	331	36	38	91
31	829	231	2024-02-11	10.1	zelle	332	32	37	92
32	830	232	2024-02-05	99.99	cash	333	35	38	93
33	831	233	2024-02-06	88.88	credit_card	334	40	37	94
34	832	234	2024-02-07	77.77	debit_card	335	34	38	95
35	833	235	2024-02-08	66.66	zelle	336	39	37	96
36	834	236	2024-02-09	57.3	cash	337	35	38	97
37	835	237	2024-02-10	24.6	credit_card	338	32	37	98
38	836	238	2024-02-11	38.95	debit_card	339	31	38	99
39	837	239	2024-02-05	60.75	cash	340	39	37	100
40	948	949	2024-02-04	75	cash	341	40	38	101

Refresh | Save | Cancel | Export data | 200 | 200+ | 200 row(s) fetched - 0.001s, on 2024-11-05 at 01:19:07 | EST en\_US

## Implemented SCD type 3

The screenshot illustrates the implementation of SCD Type 3 using Talend Data Integration and PostgreSQL.

**Talend Expression Builder:**

The Expression Builder shows the logic for determining the current address for a customer. The expression is:

```
row1.address.equals(row2.address) ? null : row2.address
```

The context variables are:

- row1.customer\_id
- row1.customer\_name
- row1.phone
- row1.email
- row1.address
- row2
- row2.customer\_id
- row2.customer\_name
- row2.phone
- row2.email
- row2.address
- row2.previous\_address

**Schema Editor:**

The schema editor shows the table structure with columns:

Column	Type
customer_id	int
customer_name	String
phone	String
email	String
address	String

**Database Navigator in DBeaver:**

The database navigator shows the PostgreSQL connection and the target schema 'public'. The target table 'customer' has been updated with new addresses for specific customers:

```

UPDATE customer
SET address = '231 Huntington'
WHERE customer_name = 'Harry Potter';

UPDATE customer
SET address = '221 Park Drive'
WHERE customer_name = 'Hermoinie';

```

The table 'customer2' is also visible, showing the previous addresses for each customer:

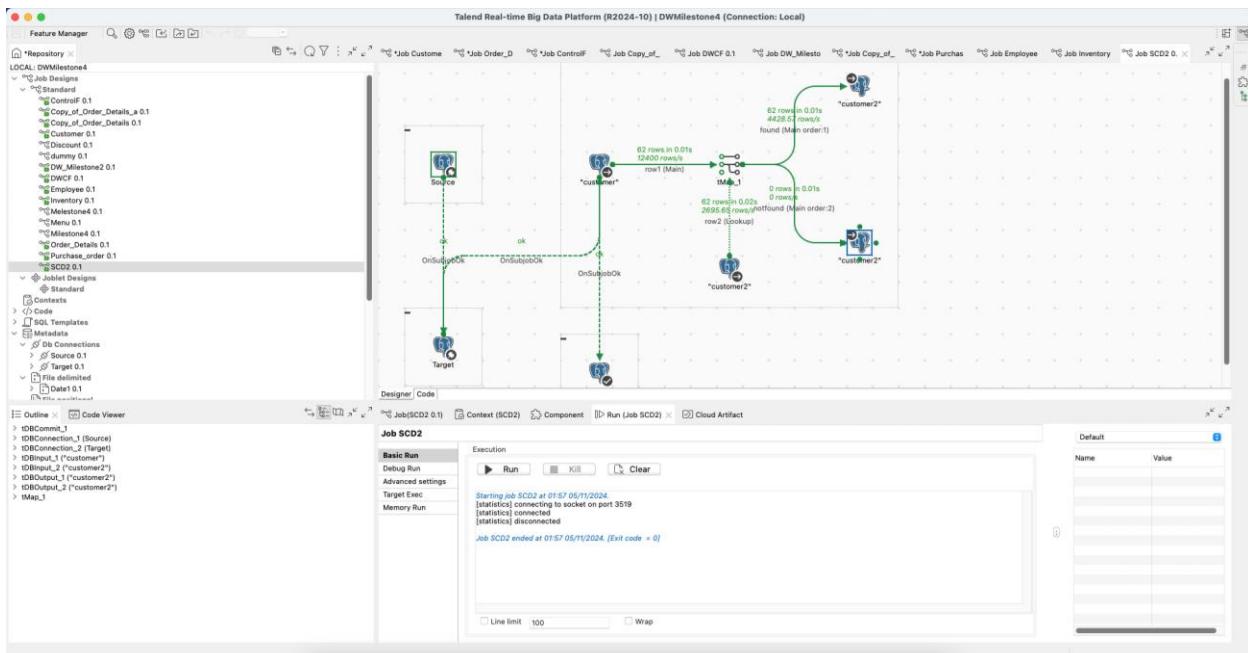
```

select * from customer2
where ckey in (61,62);

```

The grid view shows the following data for customers 61 and 62:

customer_id	customer_name	phone	email	address	previous_address
61	Harry Potter	555-1234	john.doe@example.com	44 Parker Hill	[NULL]
62	Hermoine	555-5678	jane.smith@example.com	333 Centre Street	[NULL]



ckey	customer_id	customer_name	phone	email	address	previous_address
61	61	Harry Potter	555-1234	john.doe@example.com	44 Parker Hill	231 Huntington
62	62	Hermoinie	555-5678	jane.smith@example.com	333 Centre Street	221 Park Drive

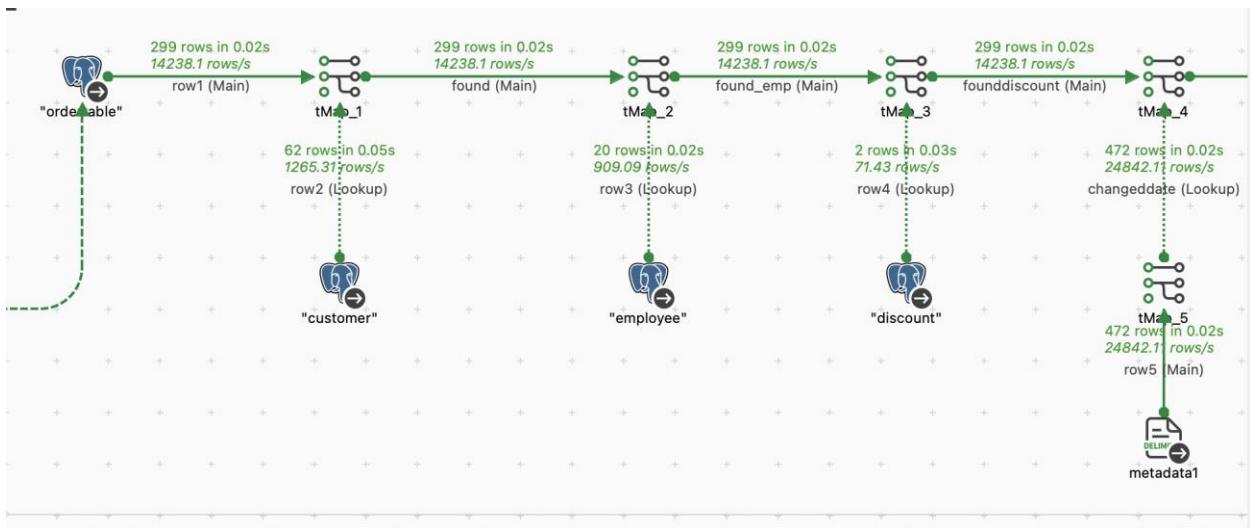
After updating the Address of customer and running the customer job, the **previous\_address** is being updated in the Data Warehouse.

## Used Multiple source of Data:

Metadata

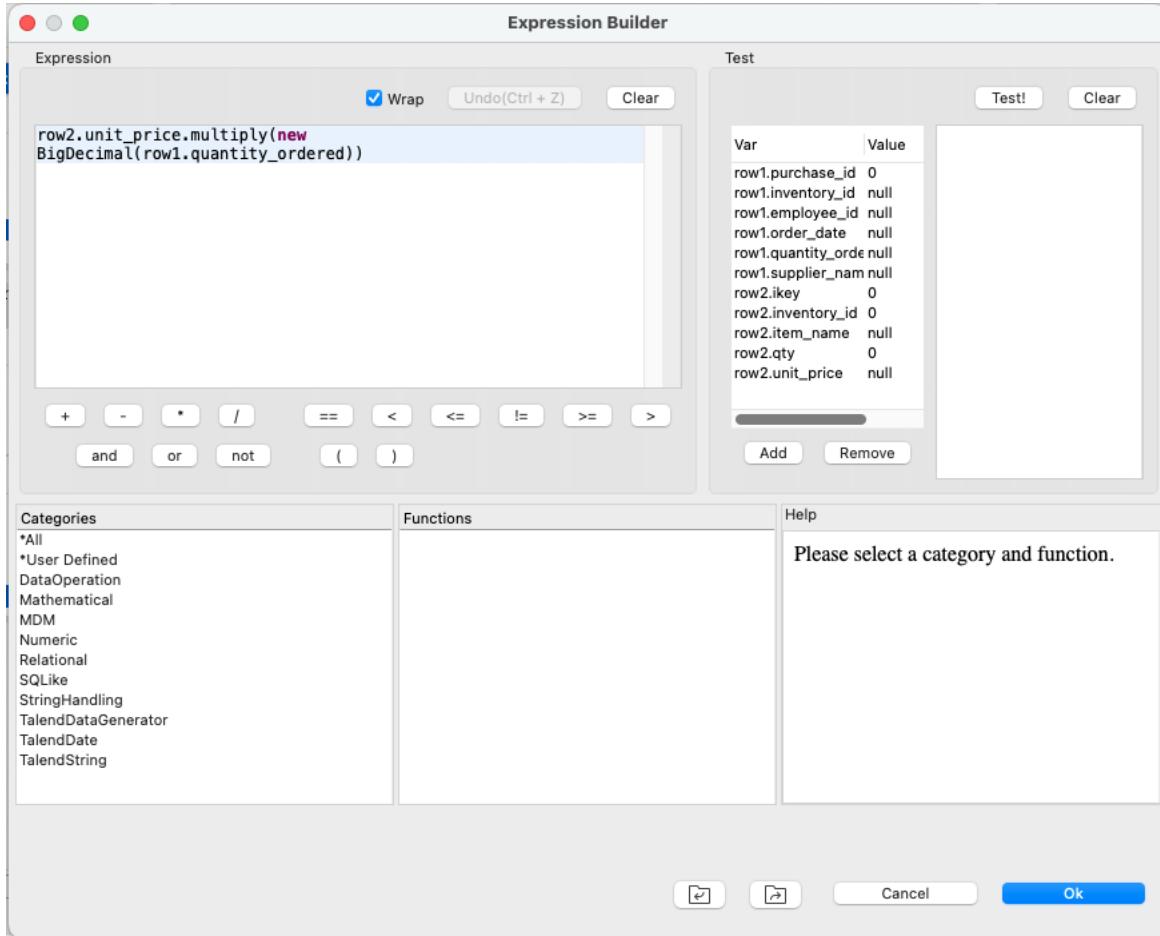
- Db Connections
  - Source 0.1
  - Target 0.1
- File delimited
  - Date1 0.1

A	B
1	Date
2	1/17/24
3	1/18/24
4	1/19/24
5	1/20/24
6	1/21/24
7	1/22/24
8	1/23/24
9	1/24/24
10	1/25/24
11	1/26/24
12	1/27/24
13	1/28/24
14	1/29/24
15	1/30/24
16	1/31/24
17	2/1/24
18	2/2/24

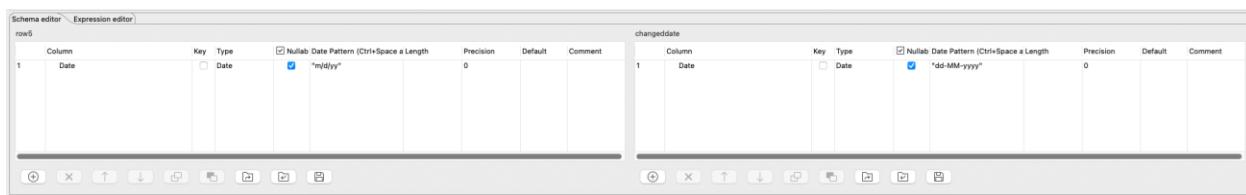


## Transformations:

- While creating a calculated measure, type casted quantity ordered from integer to BigInteger to perform appropriate calculation



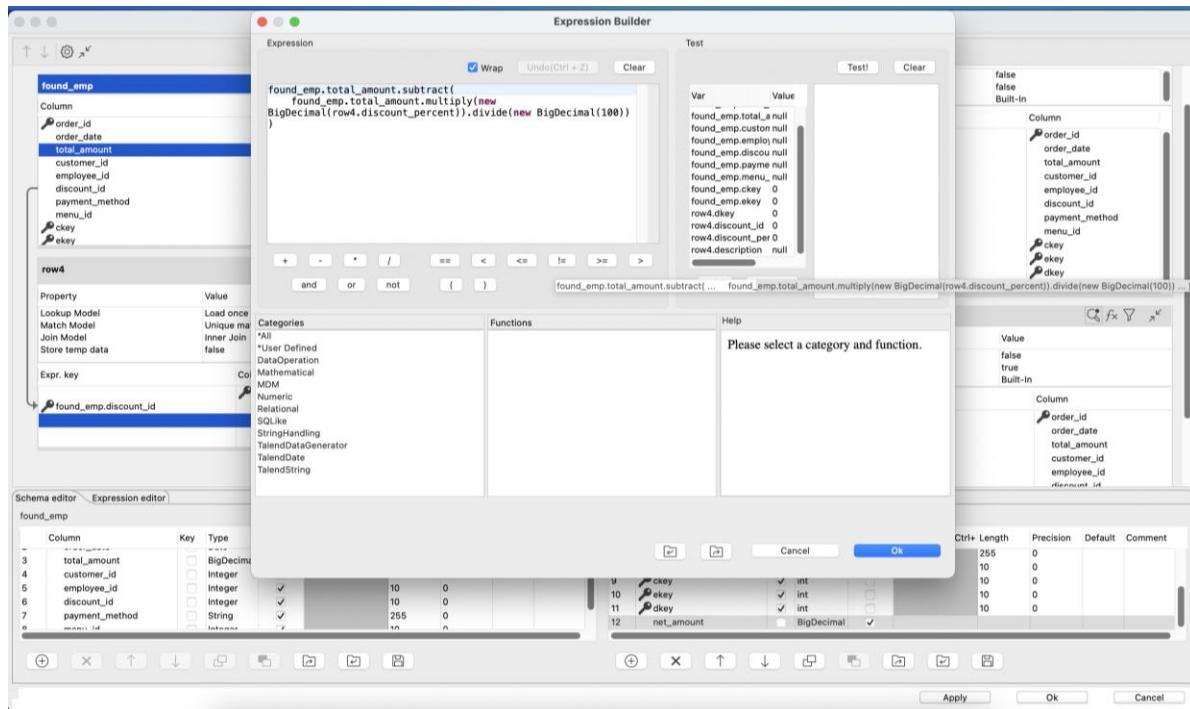
- Changed the format of date for lookup:



## Calculated Measures:

### - net\_amount

This value is being calculated in the order\_detail fact table. It is used to calculate the net bill amount once the discount is applied.

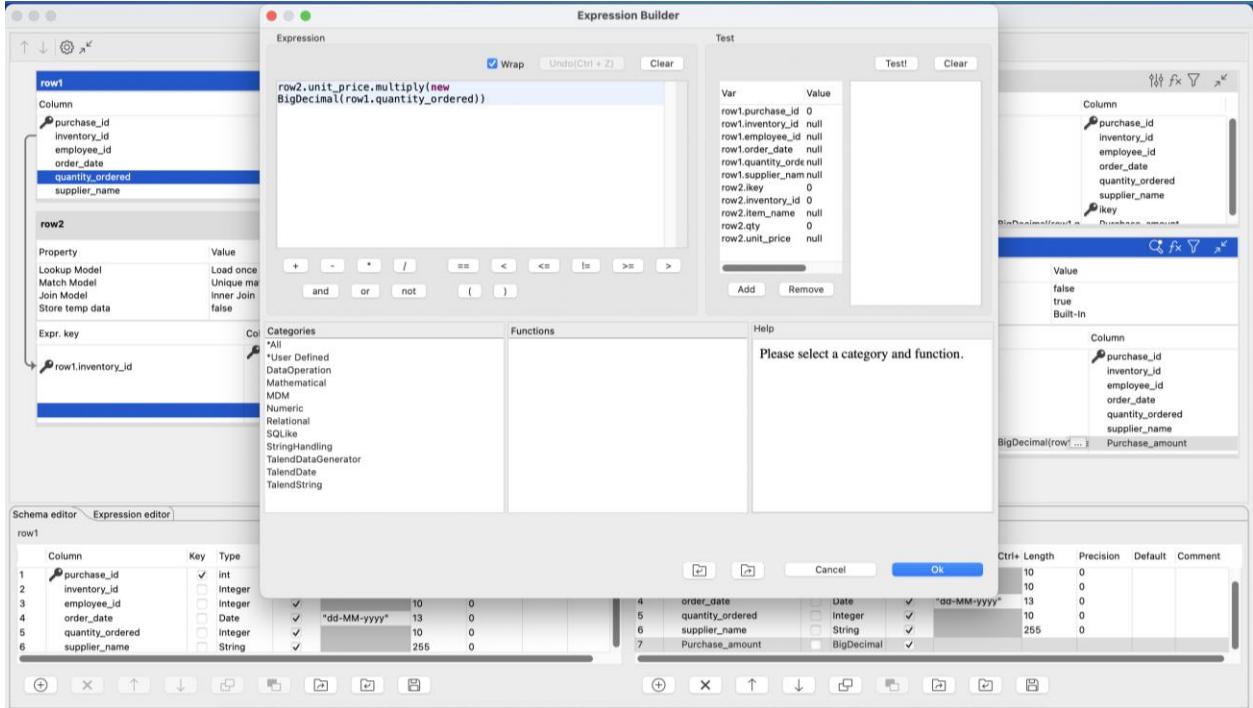


Net\_amount being added to the order\_details(Target Fact Table):

DBeaver 24.2.2 - ordertable												
Enter a SQL expression to filter results (use Ctrl+Space)												
	123 key	123 order_id	123 order_date	123 total_amount	123 payment_method	123 key	123 key	123 key	123 mkey	123 net_amount		
1	660	62	2024-02-10	64.28	debit_card	302	31	37	62	32.14		
2	661	63	2024-02-11	45.67	zelle	303	32	38	63	45.67		
3	662	64	2024-02-05	38.22	cash	304	36	37	64	19.11		
4	663	65	2024-02-06	99.43	credit_card	305	32	38	65	99.43		
5	664	66	2024-02-07	91.77	debit_card	306	33	37	66	45.89		
6	665	67	2024-02-08	56.11	zelle	307	35	38	67	56.11		
7	666	68	2024-02-09	18.24	cash	308	40	37	68	9.12		
8	667	69	2024-02-10	28.55	credit_card	309	40	38	69	28.55		
9	668	70	2024-02-11	67.78	debit_card	310	31	37	70	33.89		
10	669	71	2024-02-05	50.32	zelle	311	39	37	71	25.16		
11	670	72	2024-02-06	88.16	cash	312	33	38	72	88.16		
12	671	73	2024-02-07	21.34	credit_card	313	32	37	73	10.67		
13	762	164	2024-02-07	26.98	cash	344	37	38	64	26.98		
14	763	165	2024-02-08	50.39	debit_card	345	39	37	65	25.2		
15	764	166	2024-02-09	85.73	debit_card	346	37	38	66	85.73		
16	765	167	2024-02-10	60.12	zelle	347	32	37	67	30.06		
17	766	168	2024-02-11	31.55	cash	348	32	38	68	31.55		
18	767	169	2024-02-05	98.76	credit_card	349	34	37	69	49.38		
19	768	170	2024-02-06	44.23	debit_card	350	30	38	70	44.23		
20	782	184	2024-02-06	49.67	cash	304	31	37	84	24.79		
21	783	185	2024-02-07	10.84	credit_card	305	37	38	85	10.84		
22	784	186	2024-02-08	71.34	debit_card	306	33	37	86	35.67		
23	785	187	2024-02-09	28.45	zelle	307	32	38	87	28.45		
24	822	224	2024-02-11	66	cash	344	34	38	74	66		
25	823	225	2024-02-05	49.99	credit_card	345	34	37	75	25		
26	824	226	2024-02-06	88.75	debit_card	346	39	38	76	88.75		
27	825	227	2024-02-07	77.2	zelle	347	33	37	77	38.6		
28	826	228	2024-02-08	55.55	cash	348	33	38	78	55.55		
29	827	229	2024-02-09	40.4	credit_card	349	38	37	79	20.2		
30	828	230	2024-02-10	25.25	debit_card	350	36	38	80	25.25		
31	829	231	2024-02-11	10.1	zelle	351	32	37	81	5.05		
32	830	232	2024-02-05	99.99	cash	352	35	38	82	99.99		
33	831	233	2024-02-06	88.88	credit_card	353	40	37	83	44.44		

- **purchase\_amount**

The purchase amount is being calculated in the inventory purchase table. This measure is being calculated using the unit price being referred from the inventory table and quantity ordered referred from the fact table. This measure signifies the amount being spent on each inventory order.



- **purchase\_amount being added to the purchase inventory table:**

Grid	123 pkey	123 purchase_id	123 order_date	123 quantity_ordered	123 supplier_name	123 key	123 ekey	123 purchase_amount
1	260	1	2024-02-05	150	Global Supplies Inc.	62	21	142.5
2	261	2	2024-02-05	225	Quality Foods Ltd.	63	22	191.25
3	262	3	2024-02-05	300	Bulk Basics LLC	64	23	150
4	263	4	2024-02-06	180	Organic Farms Co.	65	24	540
5	264	5	2024-02-06	120	Culinary Essentials Inc.	66	25	1,020
6	265	6	2024-02-06	90	Global Supplies Inc.	67	26	652.5
7	266	7	2024-02-07	450	Quality Foods Ltd.	68	27	495
8	267	8	2024-02-07	375	Bulk Basics LLC	69	28	487.5
9	268	9	2024-02-07	270	Organic Farms Co.	70	29	594
10	269	10	2024-02-08	300	Culinary Essentials Inc.	71	30	297
11	270	11	2024-02-05	280	Global Supplies Inc.	72	31	700
12	271	12	2024-02-06	260	Quality Foods Ltd.	73	32	806
13	272	13	2024-02-07	240	Bulk Basics LLC	74	33	3,600
14	273	14	2024-02-08	220	Organic Farms Co.	75	34	1,320
15	274	15	2024-02-09	200	Culinary Essentials Inc.	76	35	280
16	275	16	2024-02-10	180	Global Supplies Inc.	77	36	720
17	276	17	2024-02-11	160	Quality Foods Ltd.	78	37	400
18	277	18	2024-02-05	140	Bulk Basics LLC	79	38	126
19	278	19	2024-02-06	130	Organic Farms Co.	80	39	32.5
20	279	20	2024-02-07	150	Culinary Essentials Inc.	81	40	750
21	280	21	2024-02-08	170	Global Supplies Inc.	82	21	807.5
22	281	22	2024-02-09	190	Quality Foods Ltd.	83	22	190
23	282	23	2024-02-10	210	Bulk Basics LLC	84	23	126
24	283	24	2024-02-11	230	Organic Farms Co.	85	24	276
25	284	25	2024-02-05	250	Culinary Essentials Inc.	86	25	500
26	285	26	2024-02-06	270	Global Supplies Inc.	87	26	945
27	286	27	2024-02-07	290	Quality Foods Ltd.	88	27	1,508
28	287	28	2024-02-08	310	Bulk Basics LLC	89	28	1,240
29	288	29	2024-02-09	330	Organic Farms Co.	90	29	3,300
30	289	30	2024-02-10	350	Culinary Essentials Inc.	91	30	4,200
31	290	31	2024-02-08	200	Global Supplies Inc.	91	31	2,400
32	291	32	2024-02-09	100	Quality Foods Ltd.	92	32	980

# Milestone 5

## Problem Statement

Sail Foods needs to design and implement a comprehensive database system to manage the culinary operations for 5,000+ sailors on the USS Theodore Roosevelt, serving over 15,000 meals per day. The system must efficiently handle inventory management, meal planning, food preparation, serving logistics, and waste management while ensuring food safety, nutritional requirements, and operational efficiency in the unique environment of a Nimitz-class aircraft carrier.

### Database Definition:

12. **Customer:** Represents customers who place orders.
  - Attributes: customer\_id (PK), customer\_name, phone, email, address
13. **Order:** Represents food orders made by customers.
  - Attributes: order\_id (PK), order\_date, total\_amount, customer\_id (FK), payment\_id, employee\_id (FK), discount\_id (FK)
14. **Payment:** Represents payments made for orders.
  - Attributes: payment\_id (PK), order\_id (FK), payment\_date, payment\_type, amount
15. **Discount:** Represents discounts available for orders.
  - Attributes: discount\_id (PK), discount\_percentage, description
16. **Menu:** Contains all food items available.
  - Attributes: item\_id (PK), item\_name, price, employee\_id (FK)
17. **Employee:** Represents employees, which include Waitrs, Managers, and Chefs.
  - Attributes: employee\_id (PK), name, Date\_of\_joining, salary
18. **Inventory:** Represents the food inventory.
  - Attributes: inventory\_item\_id (PK), item\_name, qty, unit\_price, Employee\_id
19. **Purchase Order:** Represents purchase orders for inventory.
  - Attributes: purchase\_id (PK), inventory\_id (FK), order\_date, quantity\_ordered, supplier\_name

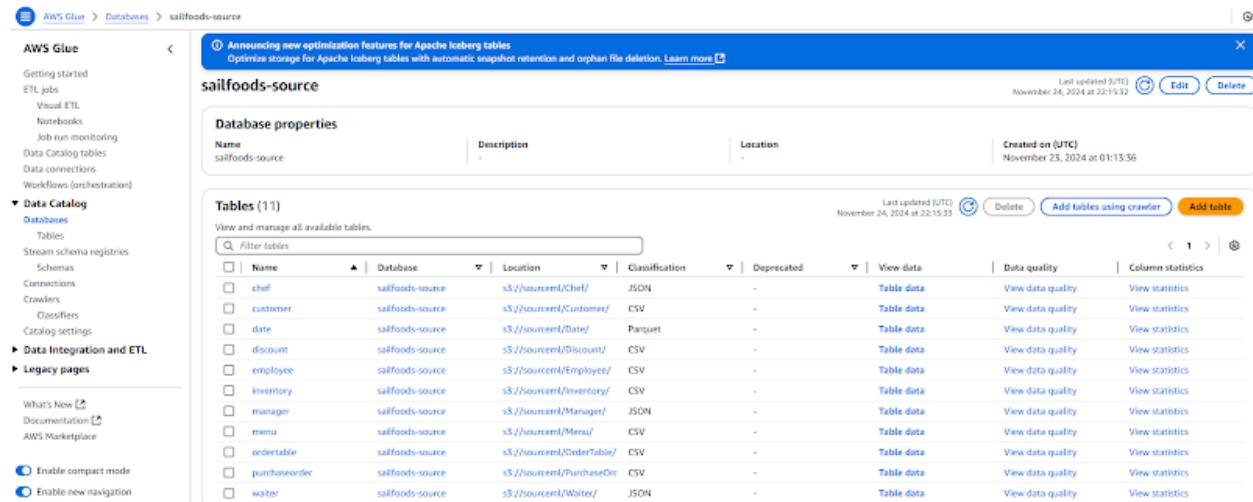
### Data Population Methodology:

We generated our data in batches using various LLMs and first populated the static data, such as Employee, Customer, and Items. Using this static data, we then generated the transactional data, such as Amount paid, etc.

# Milestone 6

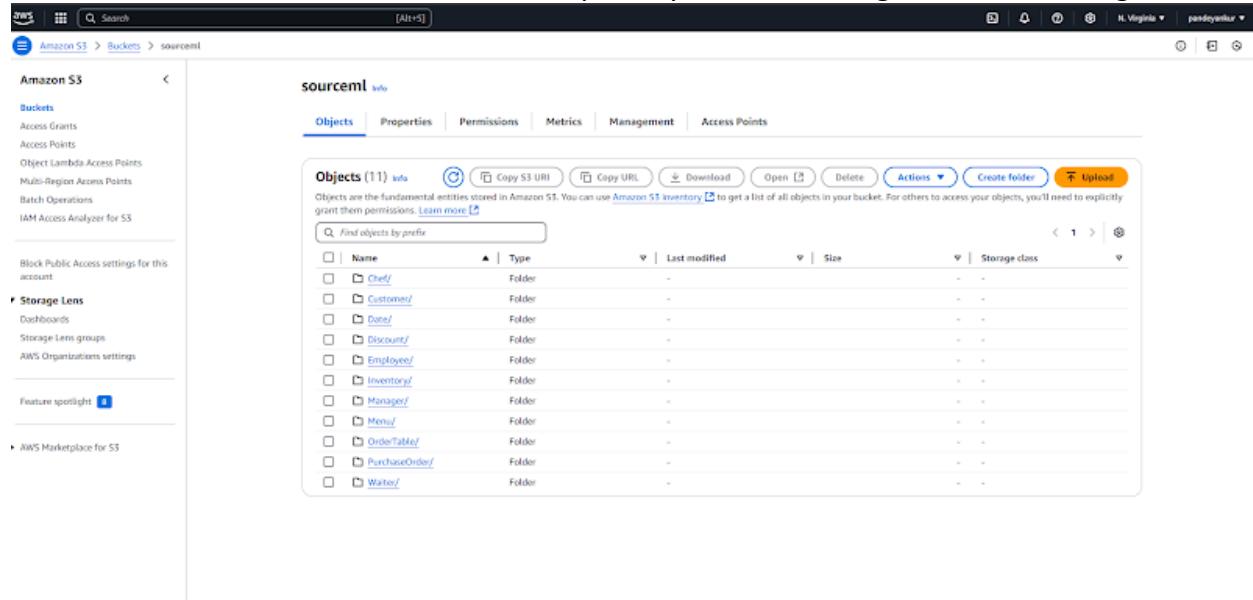
## Files and Data Sources.

We have used CSV, Parquet, and JSON files to populate our source data source on Amazon S3 via Crawler.



The screenshot shows the AWS Glue interface for managing databases. On the left, there's a sidebar with navigation links for AWS Glue, Data Catalog, Data Integration, and ETL. The main area is titled "sailfoods-source" under "Database properties". It shows a table with 11 rows, each representing a table in the database. The columns include Name, Database, Location, Classification, Deprecated, View data, Data quality, and Column statistics. The tables listed are: chef, customer, date, discount, employee, inventory, manager, menu, orderable, purchaseorder, and waiter. Each table has a "Table data" link and a "View data quality" link.

Then we used the Amazon S3 bucket as our primary source and target database storage.



The screenshot shows the Amazon S3 console. The left sidebar includes options for Buckets, Storage Lens, and AWS Marketplace. The main area shows the "source1" bucket. It has tabs for Objects, Properties, Permissions, Metrics, Management, and Access Points. Under the Objects tab, there's a table listing 11 objects, all of which are folders named after the tables from the previous screenshot: Chef/, Customer/, Date/, Discount/, Employee/, Inventory/, Manager/, Menu/, OrderTable/, PurchaseOrder/, and Waiter/. Each folder has a "Copy S3 URI" link, a "Download" link, and a "Delete" link.

Amazon S3 < datawarehouse/

**datawarehouse/**

Objects [4] info Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 Inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Name	Type	Last modified	Size	Storage class
customer/	Folder	-	-	-
Date/	Folder	-	-	-
discount/	Folder	-	-	-
employee/	Folder	-	-	-

Finally, we are using AWS glue to host our source and target database.

AWS Glue < Databases

Announcing new optimization features for Apache Iceberg tables. Optimize storage for Apache Iceberg tables with automatic snapshot retention and orphan file deletion. Learn more

Databases [3]

A database is a set of associated table definitions, organized into a logical group.

Name	Description	Location URI	Created on (UTC)
default	-	-	November 24, 2024 at 22:15:32
saffoods-dw	-	-	November 25, 2024 at 01:49:15
saffoods-source	-	-	November 23, 2024 at 02:06:31

## Pipeline Tools:

We are using the tools below to create, host, and orchestrate our data pipelines:

**Amazon S3:** We have used S3 buckets for our source and target database storage as well as storing file data, which we used to populate our source database.

The screenshot shows the Amazon S3 Buckets page. On the left, there's a sidebar with links for Amazon S3 services like Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, and IAM Access Analyzer for S3. Below that are sections for Block Public Access settings and Storage Lens dashboards. A Feature spotlight box is also present.

The main content area has tabs for "General purpose buckets" (selected) and "Directory buckets". An account snapshot is displayed at the top: "Account snapshot - updated every 24 hours" with "All AWS Regions". A link to "View Storage Lens dashboard" is available.

The "General purpose buckets" table lists six buckets:

Name	AWS Region	IAM Access Analyzer	Creation date
aws-glue-assets-491376990606-us-east-1	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	November 22, 2024, 21:01:28 (UTC-05:00)
dbnr	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	November 23, 2024, 23:19:33 (UTC-05:00)
dw-group10	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	November 22, 2024, 20:35:02 (UTC-05:00)
dw-group10	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	November 22, 2024, 20:01:56 (UTC-05:00)
sourcen	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	November 23, 2024, 23:15:08 (UTC-05:00)
targeted	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	November 23, 2024, 23:37:07 (UTC-05:00)

**Amazon Glue:** we are using Glue to host our databases and Glue Crawler to populate the database with CSV and JSON files. We are also, using glue to write ETL scripts and perform transformation over our source data, then store it in our target warehouse.

The screenshot shows the AWS Glue Studio page. The left sidebar includes links for Getting started, ETL jobs (Visual ETL, Notebooks, Job run monitoring), Data Catalog tables, Data connections, Workflows (orchestration), Data Catalog, Data Integration and ETL, and Legacy pages. It also features "What's New", "Documentation", "AWS Marketplace", and navigation options for compact mode and new navigation.

The main content area is titled "AWS Glue Studio" and contains sections for "Create job" (with options for Visual ETL, Notebook, or Script editor), "Example jobs", and "Your jobs [5]".

The "Your jobs [5]" table lists five jobs:

Job name	Type	Created by	Last modified	AWS Glue version
test	Glue ETL	Notebook	24/11/2024, 2:40:45 pm	4.0
Menu	Glue ETL	Script	24/11/2024, 2:37:49 pm	4.0
DateFormat	Glue ETL	Script	24/11/2024, 4:57:35 am	4.0
Employee	Glue ETL	Script	24/11/2024, 3:53:00 am	4.0
Customer	Glue ETL	Script	24/11/2024, 1:46:02 am	4.0

**Amazon Athena:** we have used Athena to create and modify our Database tables.

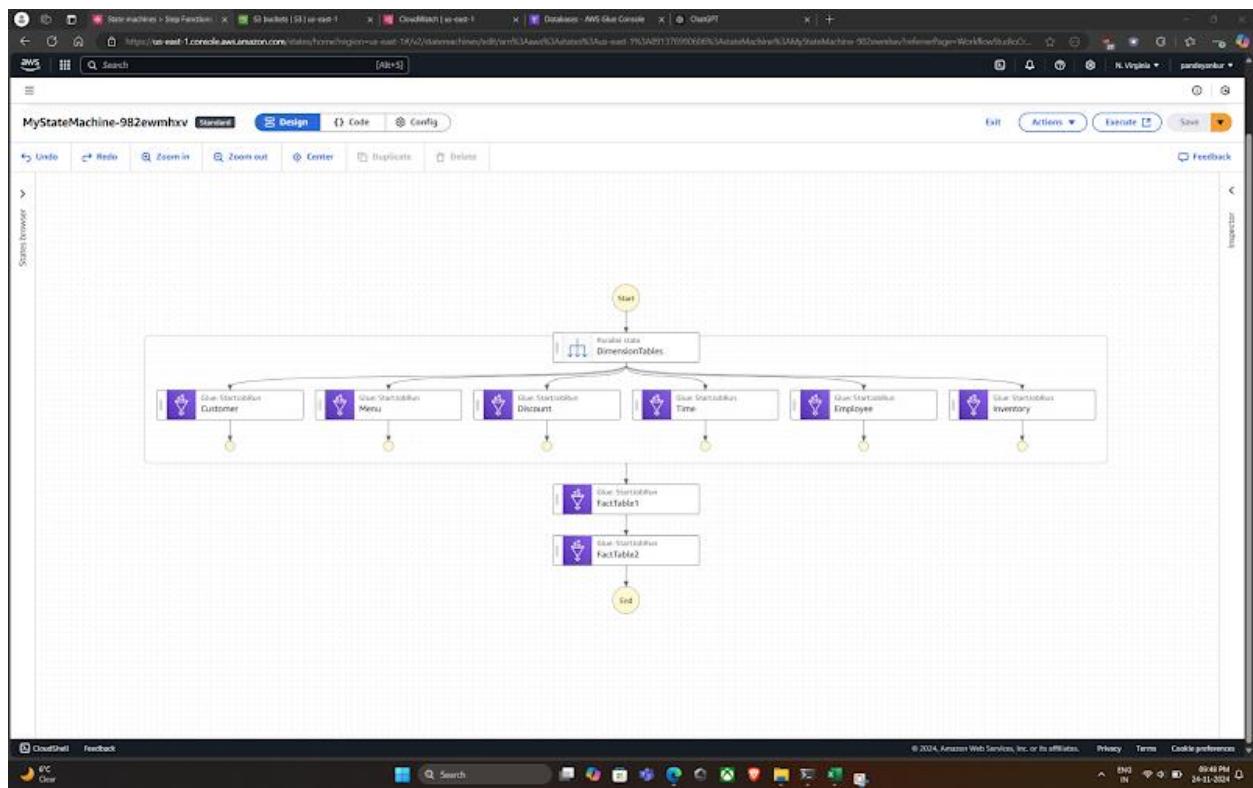
The screenshot shows the Amazon Athena Query Editor interface. On the left, there's a sidebar titled 'Data' with sections for 'Data source' (AwsDataCatalog), 'Database' (sailfoods-dw), and 'Tables and views' (customer, date, discount, employee, menu, orderable). The main area has a query editor with a SQL query: 'SELECT \* FROM "sailfoods-dw"."customer" limit 10;'. Below it is a 'Query results' section showing a table with 10 rows of customer data. The table includes columns: #, customer\_id, customer\_name, phone, email, address, and last\_updated. The results show two entries: Ethan Wright and Quentin Tarantino.

#	customer_id	customer_name	phone	email	address	last_updated
1	6	Ethan Wright	555-0106	ethan.wright@example.com	106 Cedar Blvd	2024-11-24 06:47:17.137
2	37	Quentin Tarantino	555-0157	quentin.tarantino@example.com	137 Damon Street	2024-11-24 06:47:17.137

**Amazon Cloudwatch:** we are utilizing CloudWatch to monitor database, jobs, and query logs, to debug and troubleshoot issues.

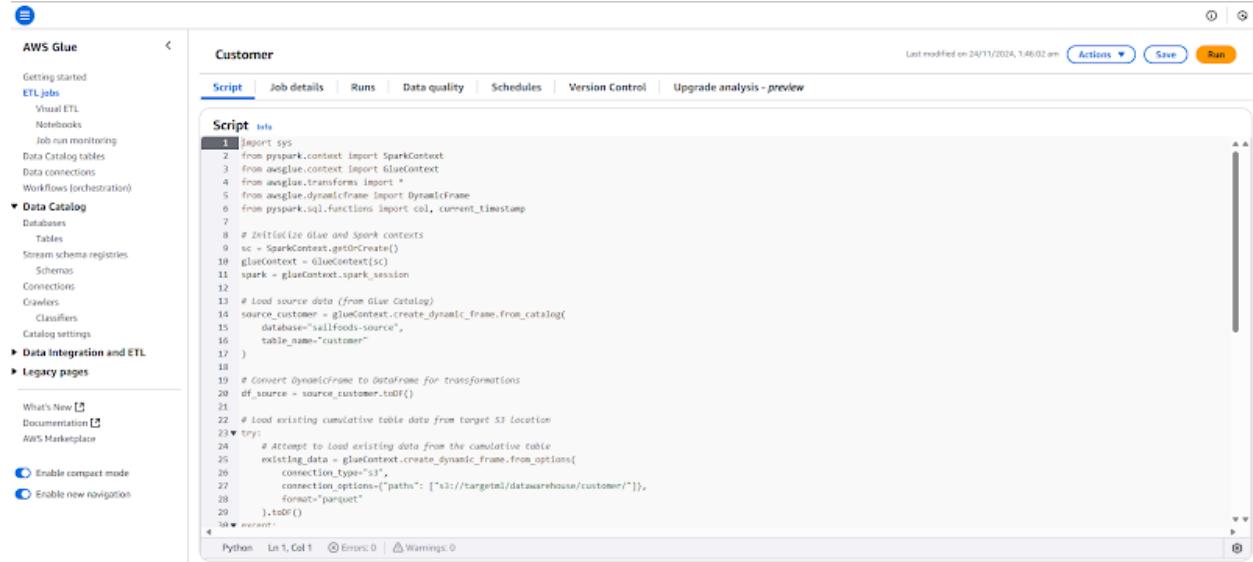
The screenshot shows the AWS CloudWatch Log Groups interface. The left sidebar lists various services like CloudWatch Metrics, CloudWatch Logs, CloudWatch Metrics Insights, CloudWatch Metrics Explorer, and CloudWatch Metrics Streams. The main area shows a log group named '/aws-glue/crawlers'. It displays 'Log group details' including ARN, Creation time (2 days ago), Retention (Never expire), and Storage bytes (1.17 KB). Below this is a 'Log streams' section listing four log streams: 'Log stream', 'targetcustomer', 'source-crawler', and 'targetemployee'. Each log stream has a timestamp of '2024-11-24 04:42:08 (UTC-05:00)'.

**Amazon Step Function:** we are using the Amazon step function to orchestrate our job workflow.



**Programming Choice:**

We are leveraging **Python** to write ETL scripts, which will read the data from our source tables, perform necessary transformations, and then populate the data onto our target data warehouse.

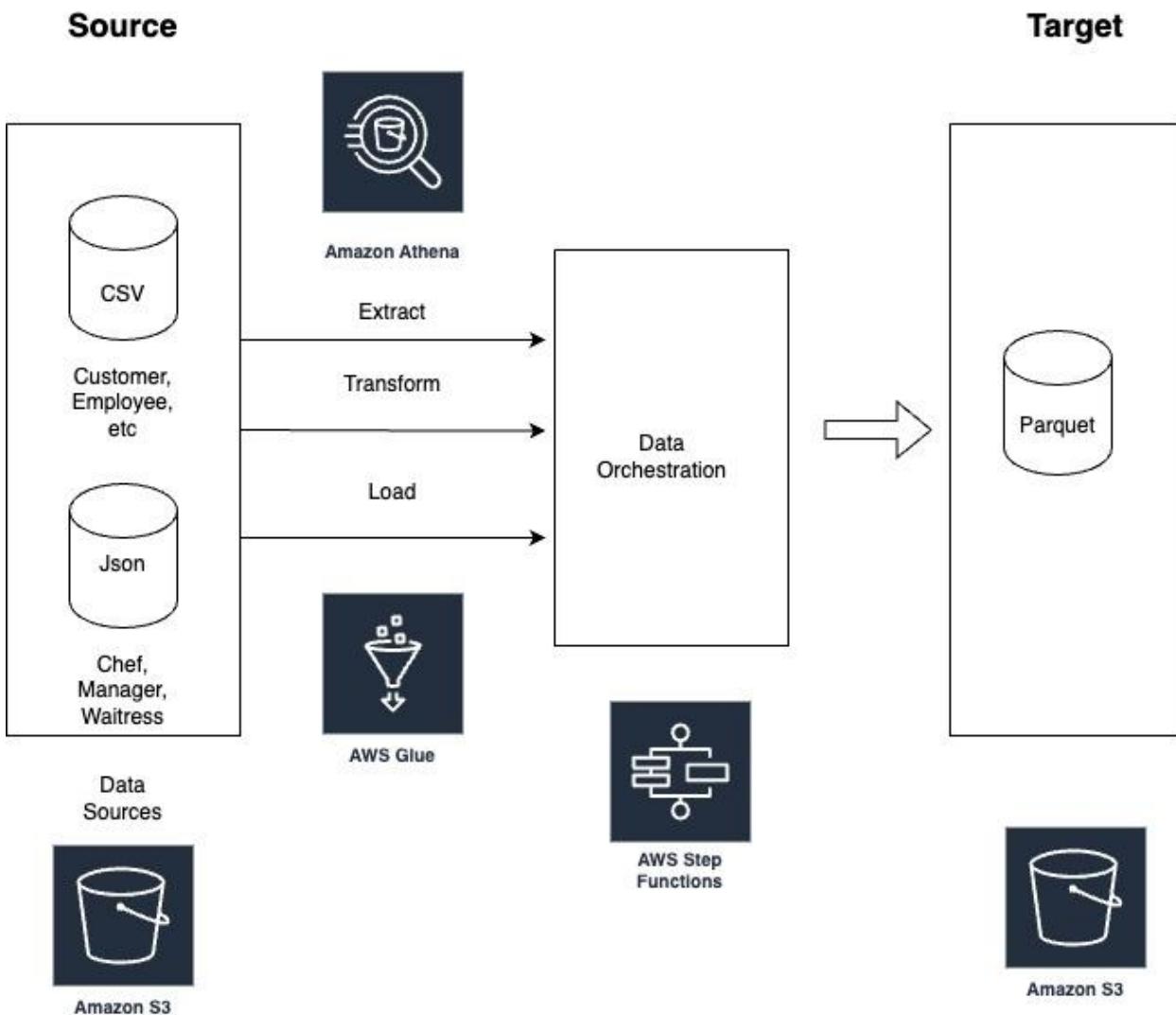


The screenshot shows the AWS Glue ETL job editor interface. On the left, there's a sidebar with navigation links for AWS Glue, ETL jobs, Data Catalog, Data Integration and ETL, and Legacy pages. The main area is titled 'Customer' and shows a 'Script' tab with the following Python code:

```
1 import sys
2 from pyspark.context import SparkContext
3 from awsglue.context import GlueContext
4 from awsglue.transforms import *
5 from awsglue.dynamicframe import DynamicFrame
6 from pyspark.sql.functions import col, current_timestamp
7
8 # Initialize glue and spark contexts
9 sc = SparkContext.getOrCreate()
10 glueContext = glueContext(sc)
11 spark = glueContext.spark_session
12
13 # Load source data (from Glue Catalog)
14 source_customer = glueContext.create_dynamic_frame.from_catalog(
15     database="safefoods-source",
16     table_name="customer"
17 )
18
19 # Convert DynamicFrame to DataFrame for transformations
20 df_source = source_customer.toDF()
21
22 # Load existing cumulative table data from target S3 location
23 try:
24     # Attempt to load existing data from the cumulative table
25     existing_data = glueContext.create_dynamic_frame.from_options(
26         connection_type="s3",
27         connection_options={"paths": ["s3://targetml/datawarehouse/customer/"]},
28         format="parquet"
29     ).toDF()
30 except:
31     pass
```

At the bottom, it says 'Python' and 'Ln 1, Col 1'. There are also 'Actions', 'Save', and 'Run' buttons at the top right.

## Pipeline Architecture:



**InventoryOrder Job:**

The screenshot shows the AWS Glue Studio interface for the 'InventoryOrder' job. The left sidebar contains navigation links for AWS Glue, ETL jobs, Data Catalog, Data Integration and ETL, and Legacy pages. The main content area displays the 'Script' tab of the 'InventoryOrder' job, which was last modified on 11/24/2024 at 8:39:02 PM. The script code is written in Python and reads data from the 'soilfoods-source' database:

```
1 import sys
2 from pyspark.context import SparkContext
3 from awsglue.context import GlueContext
4 from awsglue.dynamicframe import DynamicFrame
5 from pyspark.sql.functions import col, when
6
7 # Initialize Glue and Spark contexts
8 sc = SparkContext.getOrCreate()
9 glueContext = GlueContext(sc)
10 spark = glueContext.spark_session
11
12 # Load purchaseorder table
13 purchaseorder_table = glueContext.create_dynamic_frame.from_catalog(
14     database="soilfoods-source",
15     table_name="purchaseorder"
16 ).toDF()
17
18 # Load related tables (inventory and employee)
19 inventory_table = glueContext.create_dynamic_frame.from_catalog(
20     database="soilfoods-source",
21     table_name="inventory"
22 ).toDF()
23
24 # Join the tables
25 joined_df = purchaseorder_table.join(inventory_table, "id")
26
27 # Write the output to S3
28 joined_df.write.parquet("s3://my-bucket/purchaseorder.parquet")
```

## OrderTable:

The screenshot shows the AWS Glue Studio interface for the 'OrderTable' job. The left sidebar contains navigation links for AWS Glue, ETL jobs, Data Catalog, Data Integration and ETL, and Legacy pages. The main content area displays the 'Script' tab of the 'OrderTable' job, which was last modified on 11/24/2024 at 8:39:02 PM. The script code is identical to the one in the 'InventoryOrder' job, reading data from the 'soilfoods-source' database:

```
1 import sys
2 from pyspark.context import SparkContext
3 from awsglue.context import GlueContext
4 from awsglue.dynamicframe import DynamicFrame
5 from pyspark.sql.functions import col, when
6
7 # Initialize Glue and Spark contexts
8 sc = SparkContext.getOrCreate()
9 glueContext = GlueContext(sc)
10 spark = glueContext.spark_session
11
12 # Load purchaseorder table
13 purchaseorder_table = glueContext.create_dynamic_frame.from_catalog(
14     database="soilfoods-source",
15     table_name="purchaseorder"
16 ).toDF()
17
18 # Load related tables (inventory and employee)
19 inventory_table = glueContext.create_dynamic_frame.from_catalog(
20     database="soilfoods-source",
21     table_name="inventory"
22 ).toDF()
23
24 # Join the tables
25 joined_df = purchaseorder_table.join(inventory_table, "id")
26
27 # Write the output to S3
28 joined_df.write.parquet("s3://my-bucket/ordertable.parquet")
```

## Inventory Job:

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
from awsglue.dynamicframe import DynamicFrame
from pypark.sql.functions import col, current_timestamp

# Initialize Glue and Spark contexts
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Load source inventory data (from Glue Catalog)
source_inventory = glueContext.create_dynamic_frame.from_catalog(
    database="solfoods-source", # Source database
    table_name="inventory" # Source table
)

# Convert DynamicFrame to DataFrame for transformations
df_inventory = source_inventory.toDF()

```

## Discount Job:

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
from awsglue.dynamicframe import DynamicFrame
from pypark.sql.functions import col, current_timestamp

# Initialize Glue and Spark contexts
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Load source inventory data (from Glue Catalog)
source_inventory = glueContext.create_dynamic_frame.from_catalog(
    database="solfoods-source", # Source database
    table_name="inventory" # Source table
)

# Convert DynamicFrame to DataFrame for transformations
df_inventory = source_inventory.toDF()

```

## Menu Job:

The screenshot shows the AWS Glue Studio interface for editing a job. The left sidebar navigation includes 'AWS Glue' (selected), 'Getting started', 'ETL jobs', 'Data Catalog' (expanded), 'Data Integration and ETL', and 'Legacy pages'. The main content area is titled 'Menu' and shows a Python script. The script code is as follows:

```

4  from awsglue.transforms import *
5  from awsglue.dynamicframe import DynamicFrame
6  from pyspark.sql.functions import col, current_timestamp
7
8  # Initialize Glue and Spark contexts
9  sc = SparkContext.getOrCreate()
10 glueContext = GlueContext(sc)
11 spark = glueContext.spark_session
12
13 # Load source menu data (from Glue Catalog)
14 source_menu = glueContext.create_dynamic_frame.from_catalog(
15     database="soilfoods-source",
16     table_name="menu"
17 )
18
19 # Convert DynamicFrame to DataFrame for transformations
20 df_menu = source_menu.toDF()
21
22 # Example transformation (if needed): Add a timestamp column for auditing purposes
23 df_menu = df_menu.withColumn("last_updated", current_timestamp())

```

The script is run in Python, Line 1, Col 1. There are 0 errors and 0 warnings.

## Date Job:

The screenshot shows the AWS Glue Studio interface for editing a job. The left sidebar navigation includes 'AWS Glue' (selected), 'Getting started', 'ETL jobs', 'Data Catalog' (expanded), 'Data Integration and ETL', and 'Legacy pages'. The main content area is titled 'Menu' and shows a Python script. The script code is identical to the one in the previous screenshot:

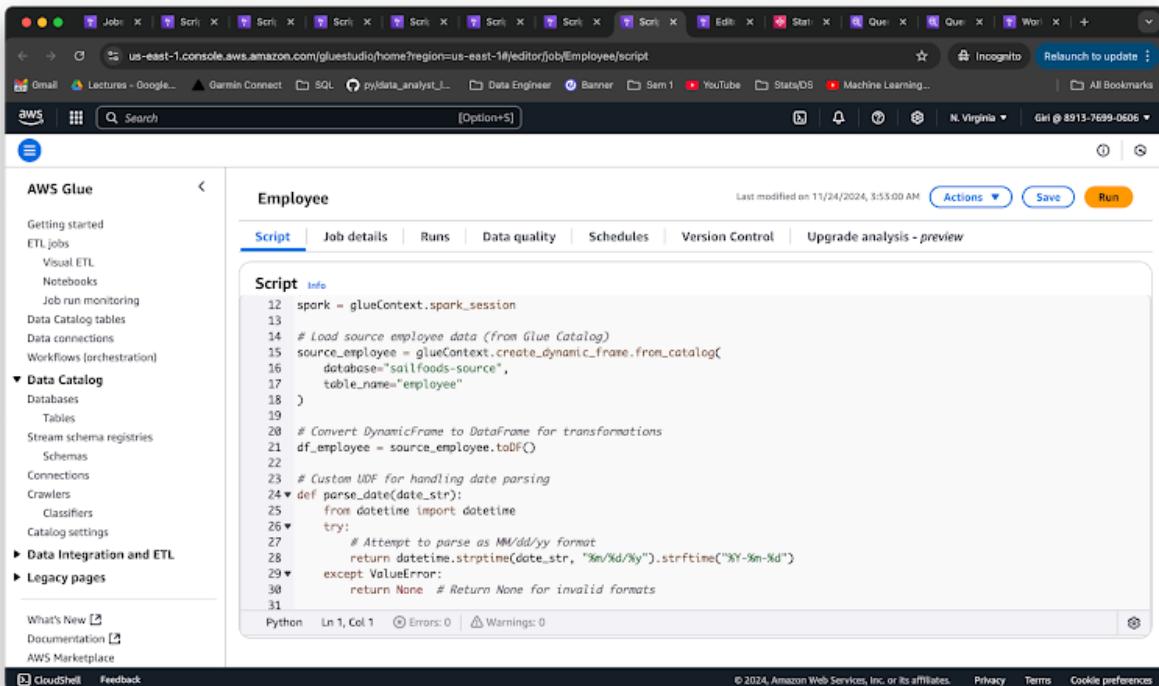
```

4  from awsglue.transforms import *
5  from awsglue.dynamicframe import DynamicFrame
6  from pyspark.sql.functions import col, current_timestamp
7
8  # Initialize Glue and Spark contexts
9  sc = SparkContext.getOrCreate()
10 glueContext = GlueContext(sc)
11 spark = glueContext.spark_session
12
13 # Load source menu data (from Glue Catalog)
14 source_menu = glueContext.create_dynamic_frame.from_catalog(
15     database="soilfoods-source",
16     table_name="menu"
17 )
18
19 # Convert DynamicFrame to DataFrame for transformations
20 df_menu = source_menu.toDF()
21
22 # Example transformation (if needed): Add a timestamp column for auditing purposes
23 df_menu = df_menu.withColumn("last_updated", current_timestamp())

```

The script is run in Python, Line 1, Col 1. There are 0 errors and 0 warnings.

## Employee Job:



```

Employee
Last modified on 11/24/2024, 3:55:00 AM Actions Save Run

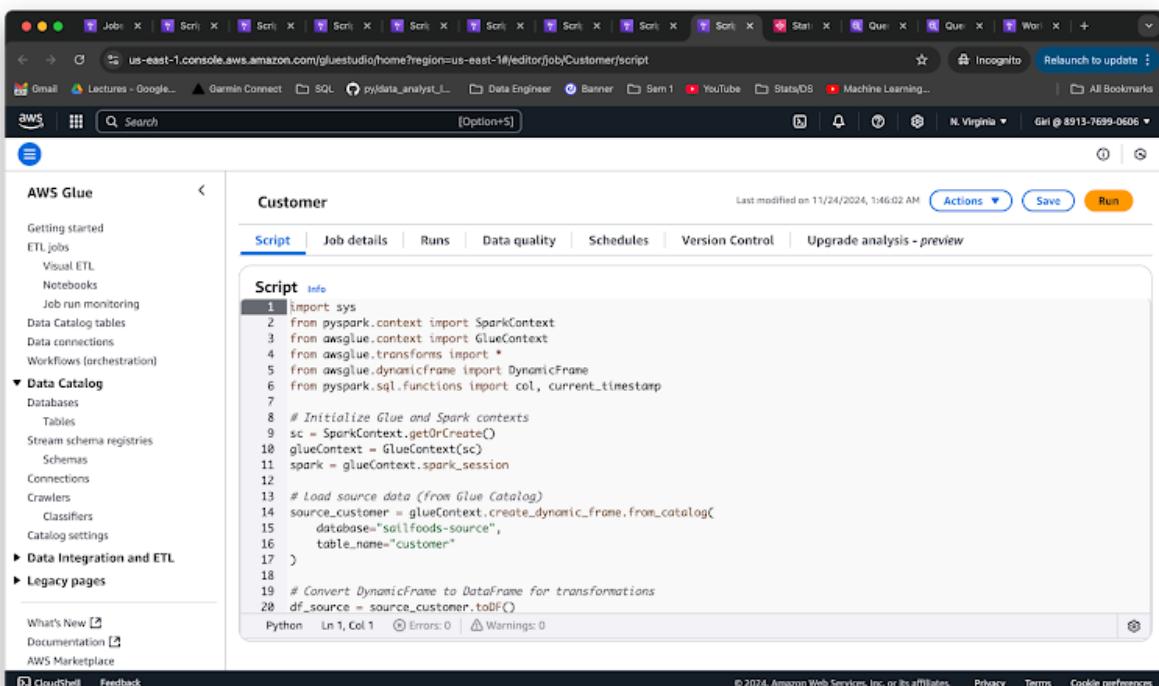
Script Info
12 spark = glueContext.spark_session
13
14 # Load source employee data (from Glue Catalog)
15 source_employee = glueContext.create_dynamic_frame.from_catalog(
16     database="sailfoods-source",
17     table_name="employee"
18 )
19
20 # Convert DynamicFrame to DataFrame for transformations
21 df_employee = source_employee.toDF()
22
23 # Custom UDF for handling date parsing
24 def parse_date(date_str):
25     from datetime import datetime
26     try:
27         # Attempt to parse as MM/dd/yy format
28         return datetime.strptime(date_str, "%m/%d/%y").strftime("%Y-%m-%d")
29     except ValueError:
30         return None # Return None for invalid formats
31

```

Python | Line 1, Col 1 | Errors: 0 | Warnings: 0

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

## Customer Job:



```

Customer
Last modified on 11/24/2024, 1:46:02 AM Actions Save Run

Script Info
1 import sys
2 from pyspark.context import SparkContext
3 from awsglue.context import GlueContext
4 from awsglue.transforms import *
5 from awsglue.dynamicframe import DynamicFrame
6 from pyspark.sql.functions import col, current_timestamp
7
8 # Initialize Glue and Spark contexts
9 sc = SparkContext.getOrCreate()
10 glueContext = GlueContext(sc)
11 spark = glueContext.spark_session
12
13 # Load source data (from Glue Catalog)
14 source_customer = glueContext.create_dynamic_frame.from_catalog(
15     database="sailfoods-source",
16     table_name="customer"
17 )
18
19 # Convert DynamicFrame to DataFrame for transformations
20 df_source = source_customer.toDF()

```

Python | Line 1, Col 1 | Errors: 0 | Warnings: 0

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

## Jobs executed successfully:

The screenshot shows the AWS Glue Monitoring interface. On the left, a sidebar navigation includes sections for AWS Glue (Getting started, ETL jobs, Visual ETL, Notebooks, Job run monitoring, Data Catalog tables, Data connections, Workflows (orchestration)), Data Catalog (Databases, Tables, Stream schema registries, Schema, Connections, Crawlers, Classifiers, Catalog settings), Data Integration and ETL (ETL jobs, Visual ETL, Notebooks, Job run monitoring, Interactive Sessions, Data classification tools, Sensitive data detection, Record Matching, Triggers, Workflows (orchestration), Blueprints, Security configurations, Cost management, New), and Legacy pages (What's New, Documentation, AWS Marketplace). The main area is titled 'Monitoring' and displays a 'Job runs summary' with metrics: Total runs (56), Running (0), Canceled (0), Successful runs (33), Failed runs (23), Run success rate (59%), and CPU hours (13). Below this is a table titled 'Job runs (56)' with columns for Job name, Run status, Type, Start time (Local), End time (Local), Run time, Capacity, Worker type, and CPU hours. The table lists numerous successful Glue ETL jobs, each with a unique name like 'InventoryOrder', 'OrderTable', etc., and their respective execution details.

## Data Populated in the target data warehouse:

The screenshot shows the AWS Glue Databases interface. The sidebar navigation is identical to the one in the previous screenshot. The main area shows the 'sailfoods-dw' database properties: Name (sailfoods-dw), Description (empty), Location (empty), and Created on (UTC) (November 23, 2024 at 02:06:31). Below this is a table titled 'Tables (8)' with columns for Name, Database, Location, Classification, Deprecated, View data, Data quality, and Column statistics. The table lists eight tables: customer, date, discount, employee, inventory, inventoryorder, menu, and ordertable\_e0fc31, all located in the sailfoods-dw database and using Parquet format.

Transformation implemented in the target employee table:

Employee

Last modified on 11/24/2024, 3:53:00 AM Actions Save Run

**Script** Job details Runs Data quality Schedules Version Control Upgrade analysis - preview

**Script Info**

```

59 # Convert DynamicFrames to DataFrames for transformations
60 df_chef = source_chef.toDF()
61 df_manager = source_manager.toDF()
62 df_waiter = source_waiter.toDF()
63
64 # Add a specialization column to employees based on their presence in the specialization tables
65 df_employee_with_type = (
66     df_employee
67     .withColumn(
68         "Employee_Type",
69         when(col("employee_id").isin(df_chef.select("employee_id").rdd.flatMap(lambda x: x).collect()), "Chef")
70         .when(col("employee_id").isin(df_manager.select("employee_id").rdd.flatMap(lambda x: x).collect()), "Manager")
71         .when(col("employee_id").isin(df_waiter.select("employee_id").rdd.flatMap(lambda x: x).collect()), "Waiter")
72         .otherwise("General") # Default to "General" if no specialization found
73     )
74 )
75
76 # Load existing cumulative employee table data from target S3 location
77 try:
78     # Attempt to load existing data from the cumulative table

```

Python Ln 1, Col 1 Errors: 0 | Warnings: 0

The screenshot shows the AWS Athena Query Editor interface. The left sidebar lists tables: employee, inventory, inventoryorder, menu, and ordertable\_e0fc3146b036e28670c8007\_a8788760e. The main area displays the 'Query results' tab for a completed query. The results table has columns: #, employee\_id, name, date\_of\_joining, salary, employee\_type, and last\_updated. The results are as follows:

#	employee_id	name	date_of_joining	salary	employee_type	last_updated
1	16	Pia Guerra	2020-04-25	50500	Chef	2024-11-24 08:54:46.137
2	12	Liam Neeson	2019-12-05	45500	Waiter	2024-11-24 08:54:46.137
3	13	Mia Wong	2020-01-10	57500	Manager	2024-11-24 08:54:46.137
4	19	Steven Tyler	2020-07-10	49500	Waiter	2024-11-24 08:54:46.137
5	11	Kyle Mendez	2019-11-30	56000	Manager	2024-11-24 08:54:46.137
6	4	Diana Reyes	2019-04-25	51000	Chef	2024-11-24 08:54:46.137
7	5	Ethan Hawke	2019-05-30	53000	Manager	2024-11-24 08:54:46.137
8	8	Hanna Ziegler	2019-08-15	48500	Chef	2024-11-24 08:54:46.137
9	15	Oscar Wilde	2020-03-20	49000	Chef	2024-11-24 08:54:46.137
10	2	Bob Brown	2019-02-15	50000	Chef	2024-11-24 08:54:46.137

Measures implemented in the order table:

The screenshot shows the AWS Glue Script Editor interface. On the left, a sidebar navigation includes 'AWS Glue', 'Getting started', 'ETL jobs', 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. The main content area is titled 'Orderable' and shows a Python script:

```

46 # Join with discount on discount_id
47 joined_data = joined_data.join(discount_table, "discount_id", "left")
48
49 # Join with menu table: Match menu.item_id to orderable.menu_id
50 joined_data = joined_data.join(menu_table, orderable["menu_id"] == menu_table["item_id"], "left")
51
52 # Apply discount logic: If discount_id == 1, apply 50% off total_amount
53 joined_data = joined_data.withColumn(
54     "final_amount",
55     when(col("discount_id") == 1, col("total_amount") * 0.5).otherwise(col("total_amount"))
56 )
57
58 # Optional: Drop unnecessary columns or rename for clarity
59 output_data = joined_data.select(
60     "order_id",
61     "order_date",
62     "customer_id",
63     "employee_id",
64     "menu_id",

```

Below the code editor, there are tabs for 'Script' (selected), 'Job details', 'Runs', 'Data quality', 'Schedules', 'Version Control', and 'Upgrade analysis - preview'. A status bar at the bottom indicates 'Python' and 'Line 1, Col 1'. The footer includes links for 'CloudShell', 'Feedback', and copyright information.

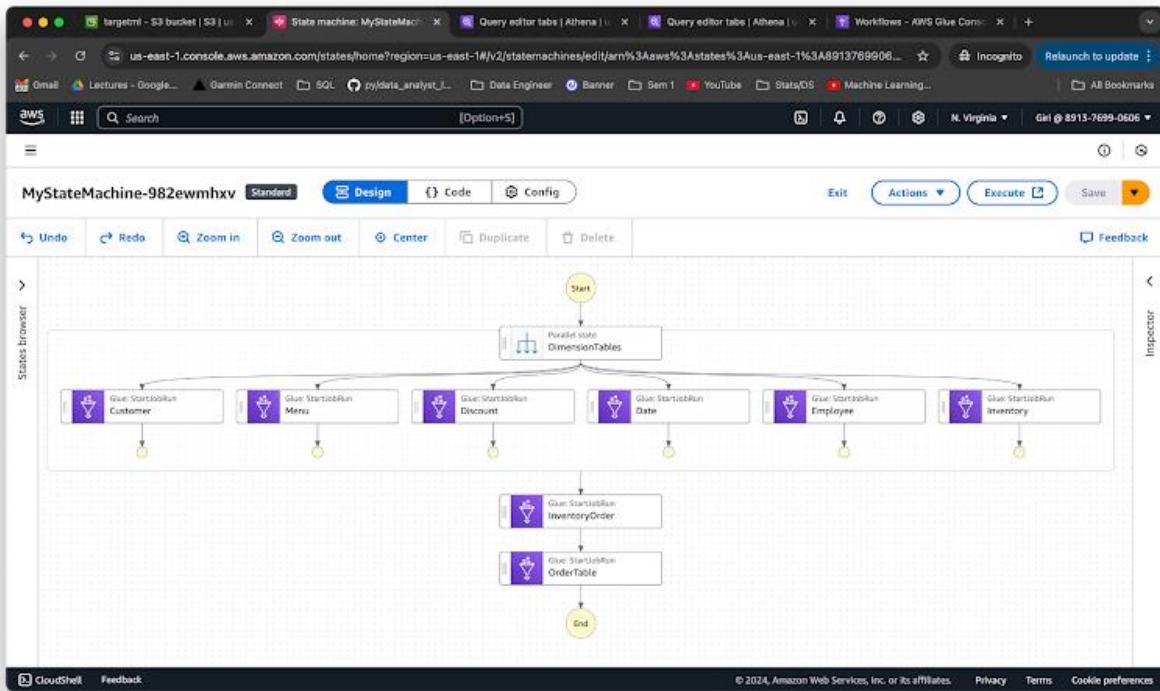
The screenshot shows the Amazon Athena Query Editor interface. On the left, a sidebar lists tables: 'inventoryorder', 'menu', 'ordertable\_e0fc3146b036e28670c8007\_a8788760e', and 'Views (0)'. The main content area is titled 'Completed' and shows a table of results:

**Results (10)**

#	order_id	order_date	customer_id	employee_id	menu_id	total_amount	discount_id	final_amount
1	54	2024-02-09	54	16	4	74.66	1	37.33
2	254	2024-02-06	14	13	4	20.39	1	10.195
3	204	2024-02-05	24	19	4	67.89	1	33.945
4	154	2024-02-11	34	17	4	29.44	2	29.44
5	4	2024-02-08	4	16	4	60.0	2	60.0
6	104	2024-02-10	44	18	4	61.34	2	61.34
7	5	2024-02-09	5	17	5	20.0	1	10.0
8	155	2024-02-05	35	12	5	92.34	1	46.17
9	105	2024-02-11	45	18	5	80.45	1	40.225
10	255	2024-02-07	15	14	5	89.44	2	89.44

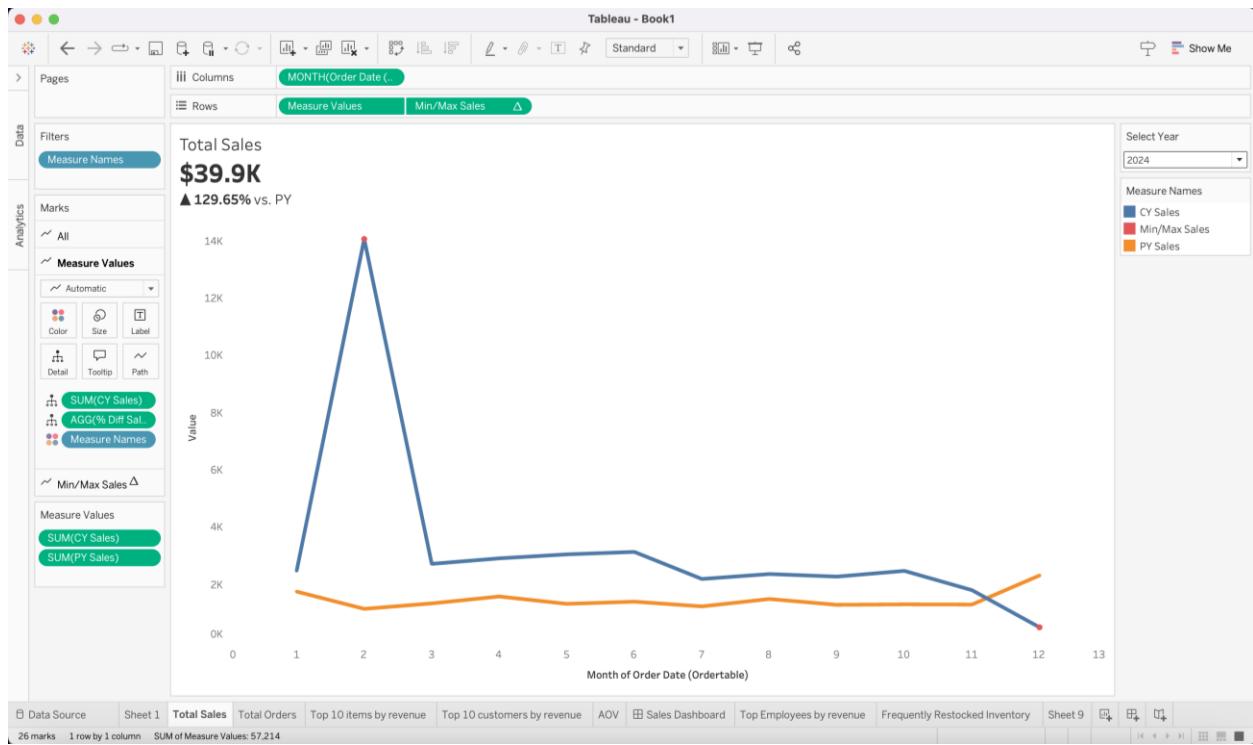
At the top right of the results table, there are buttons for 'Copy' and 'Download results'. The footer includes links for 'CloudShell', 'Feedback', and copyright information.

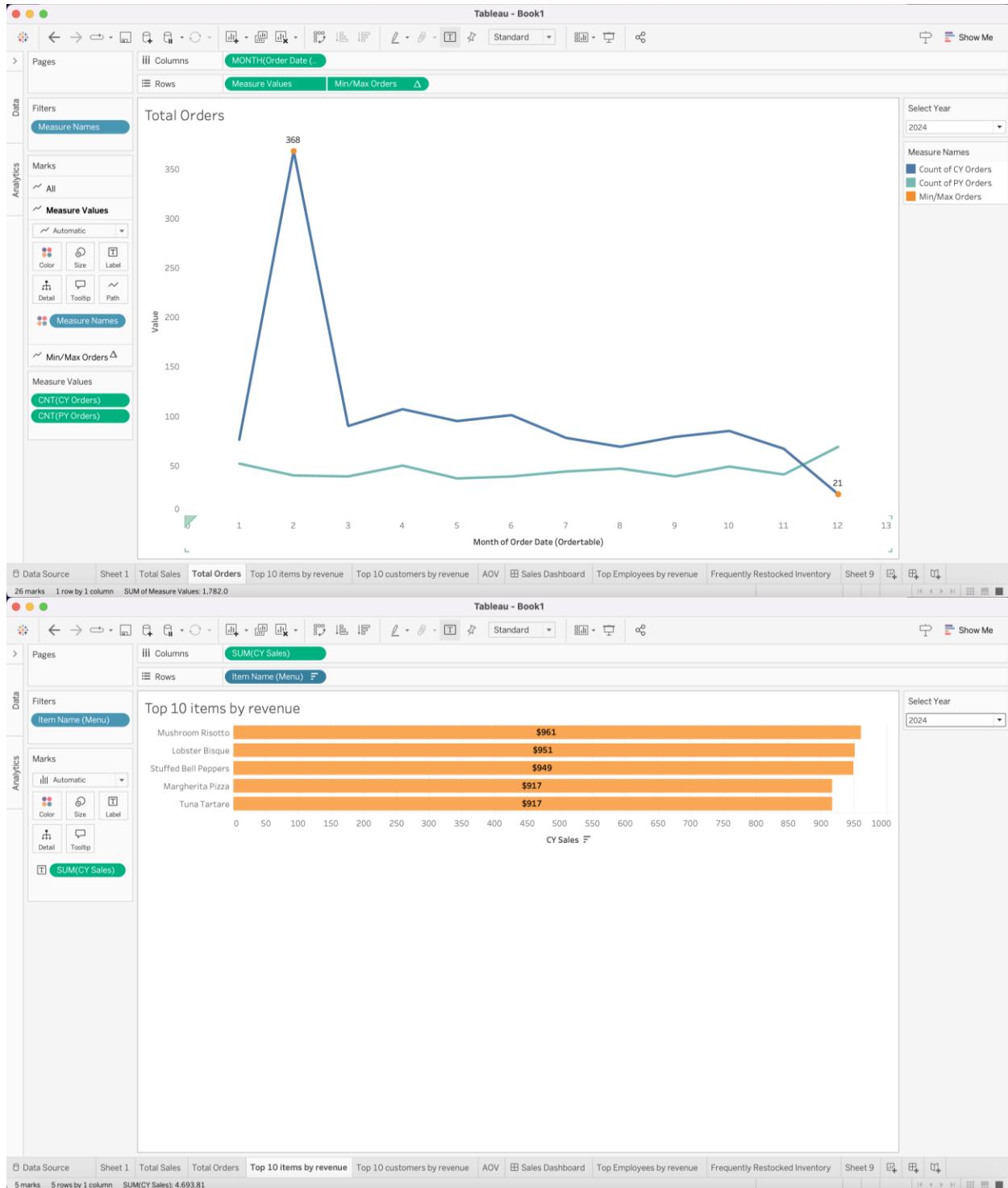
## Control Flow:

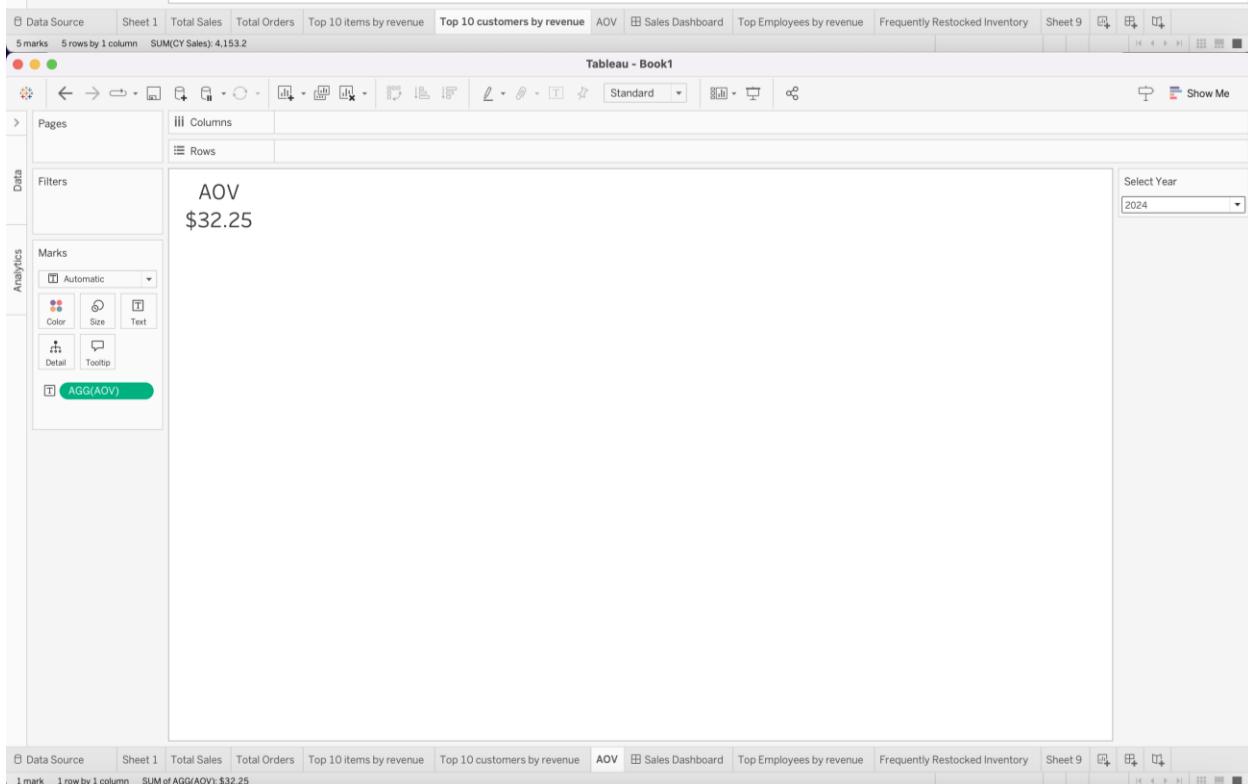
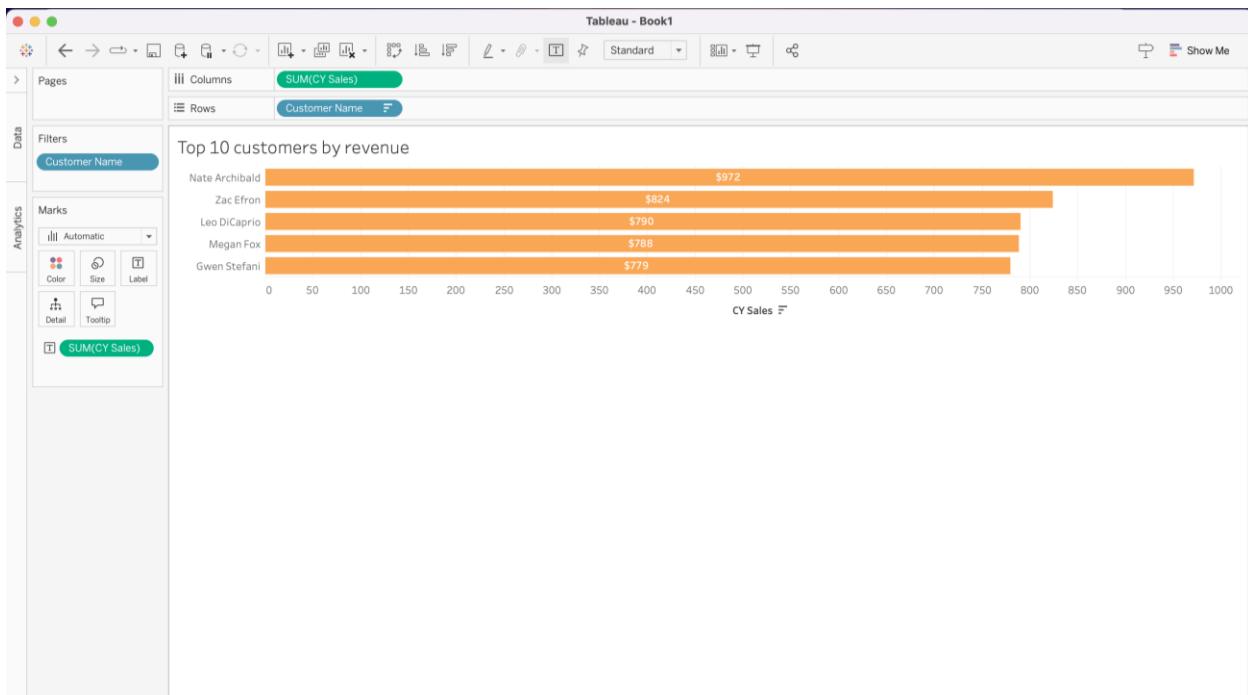


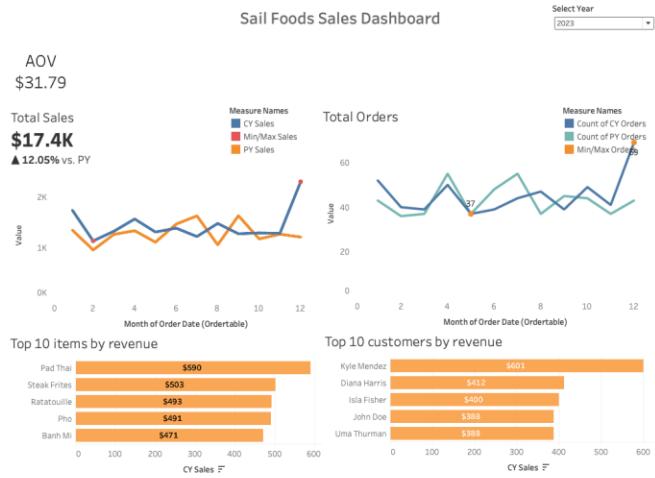
# Milestone 7

## Dashboard 1

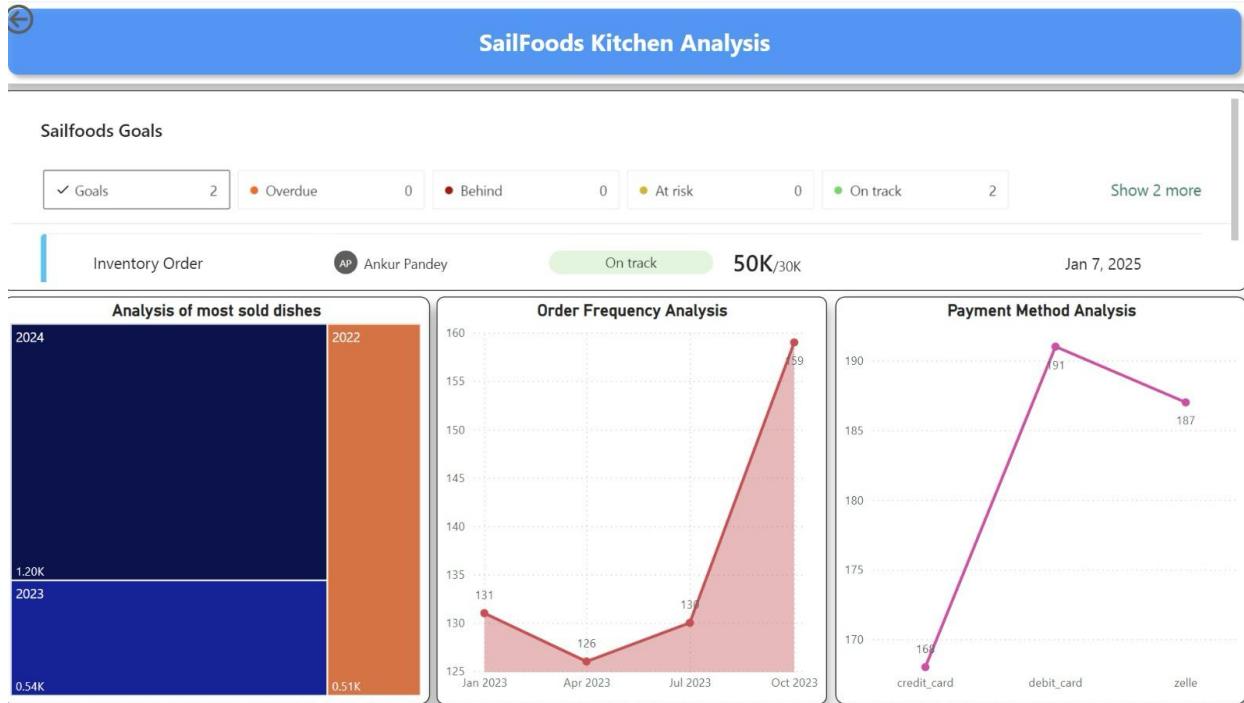








# Dashboard 2



# Dashboard 3



## Employee Metrics



AOV Sales Dashboard Frequently Restocked Inventory Frequently Restocked Inventory ... Top Employees by revenue Highest Paid Employee Order Volume per Employee No. of Employees Sheet 13 Dashboard 2