**Technical Report for Café Database System**

JSOM, University of Texas at Dallas

Punya Ira Anand

COMPREHENSIVE DATABASE IMPLEMENTATION FOR CAFE MANAGEMENT

In-Scope Work

• Project requirements documentation – Introduction, Overview, Literature review, Assumptions, Constraints

• Entity-relationship model - Design and Requirements

• DDL Scripts

• DML Scripts

• SQL Scripts

• Comprehensive Report

**Introduction**

This report is about the systematic approach that we took to develop, design and implement a relational database management system (DBMS) for a cafe chain’s operations. This report includes the complete scope of the project which is tailored for managing and optimizing the operations of this cafe. This system is focused on managing and streamlining the operations pertaining to orders, product inventory, and customer interactions and therefore increasing the efficiency and reliability of the data overall across all the branches of the cafe.

**Overview**

The cafe DBMS project addresses the need for a data management system that handles and manages extensive operational data throughout multiple locations of cafe efficiently. By collaborating data with functions – spreading from stock management to order process and customer management system. The DBMS is efficiently designed to fetch and provide comprehensive insights and analytics, enhance customer service, and improve operational decisions. At the end of this, we anticipate a robust database that will support day-day operations while providing strategic data insights through advanced data analytics

**Literature Review**

In the exploratory phase, research was conducted to understand existing DB solutions in other retail and hospitality environments. Studies such as "Efficient Data Management in Retail Operations" and "Database Solutions for Multi-location Restaurant Chains" provided analytics into the challenges and effective strategies in the hospitality industry. The literature emphasized the importance of a scalable and flexible database architecture that can adapt to the varying demands of a growing business, such as a café chain. Additionally, recent advancements in relational database technology and their application in real-time data processing were reviewed to ensure the proposed system could handle high transaction volumes and complex query requirements effectively.

**Assumptions and Constraints**

**Assumptions:**

- The database system will handle up to 10,000 transactions daily without performance degradation.

- The system will be used continuously during business hours with minimal downtime.

- All locations will have uniform product offerings and pricing to ensure consistency and integrity in reporting and inventory management.

**Constraints**

- Real-time synchronization of data across all locations is mandatory to ensure inventory and orders are up to date.

- The system must comply with all applicable data protection regulations to safeguard customer and business data.

- Scalability must be maintained, allowing for the addition of new café locations without significant system reconfigurations.

**Project Step #1: Initial Design and Requirements**

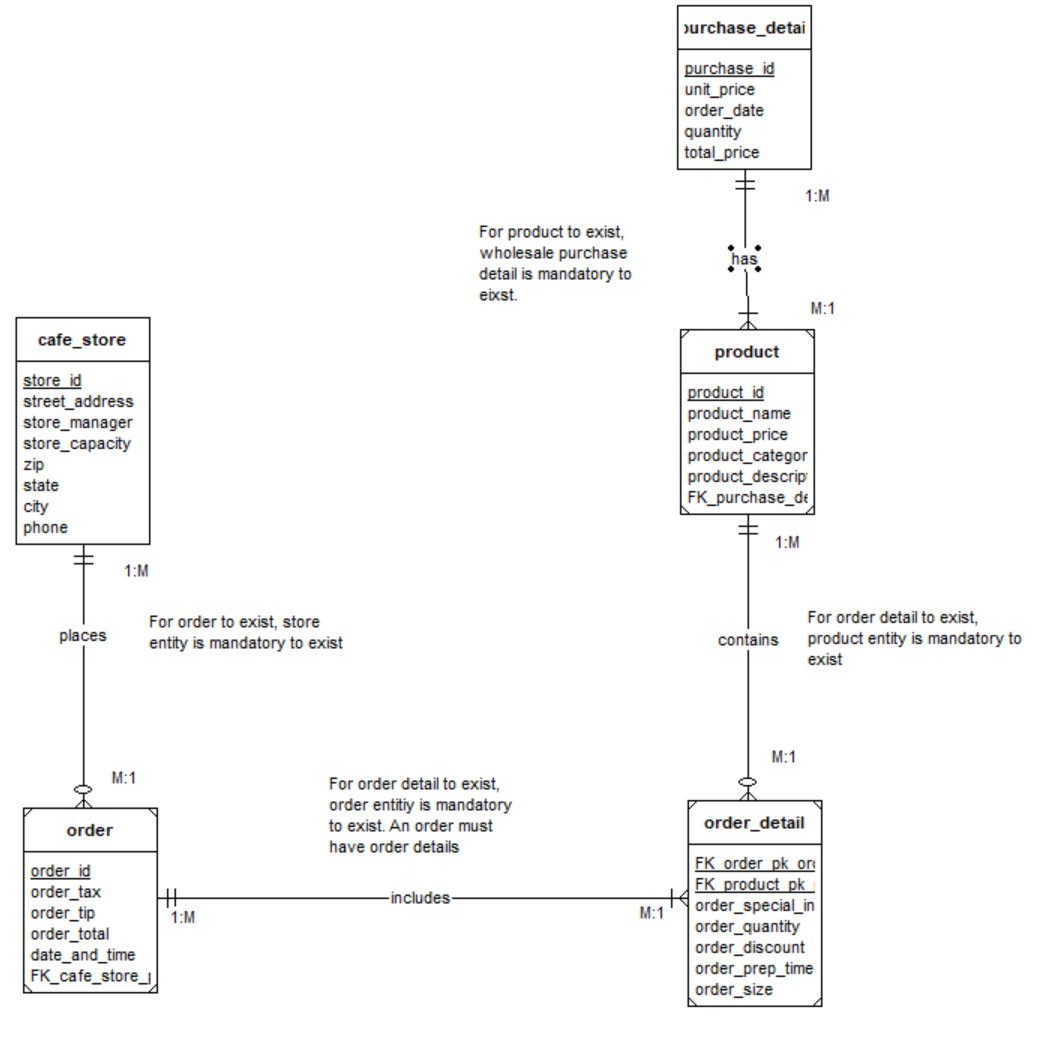
This initial phase involved gathering detailed requirements from various stakeholders, including café managers, inventory specialists, and marketing personnel, to ensure the database would meet the diverse needs of the business. Key requirements identified included:

- Robust order management capabilities to track orders from placement through fulfillment.

- Detailed inventory management to monitor stock levels, track shipments, and predict inventory needs based on sales trends.

- Customer management features to maintain customer profiles, preferences, and transaction histories to enhance customer service and marketing efforts.

Based on these requirements, an initial entity-relationship (ER) diagram was created to visualize the database structure, which included entities such as Products, Orders, Inventory, and Customers, along with their relationships.



1. **Entity and Attribute Description**

**Cafe\_Store**

**Description**: Represents a physical location where customers can purchase food and beverages.

**Attributes**:

* + **store\_id** (PK): A unique identifier for each cafe store.
  + **street\_address:** The street address of café store.
  + **store\_manager**: The name of the individual managing the store.
  + **store\_capacity**: The maximum number of people the store can accommodate at one time.
  + **zip**: The zip code for the café\_store.
  + **state:** The US state in which the store is located.
  + **city:** TheUScity in which the store is located.
  + **phone:** The phone number of the café store.

**Order**

**Description**: A record of a transaction initiated by a customer to purchase products from a cafe store.

**Attributes**:

* + **order\_id** (PK): A unique identifier for each order.
  + **order\_total**: The total cost of the order.
  + **order\_tax**: The total tax placed on the order.
  + **order\_tip**: The total amount tipped by the customer.
  + **date\_and\_time**: The exact date and time at which the order was placed.
  + **FK\_cafe\_store\_pk\_store\_id** (FK): A foreign key linking the order to the specific cafe store where it was made.

**Order\_Detail**

**Description**: Details of individual items within an order.

**Attributes**:

* + **FK\_order\_pk\_order\_id** (FK, PK): A foreign key to the Order entity, part of the composite primary key that uniquely identifies each order detail.
  + **FK\_product\_pk\_product\_id** (FK, PK): A foreign key to the Product entity, identifying what product the detail refers to.
  + **order\_special\_info**: Contains special information provided by customer e.g.   
    regarding their dietary restrictions.
  + **order\_prep\_time**: The average time it takes to process this item.
  + **order\_quantity**: The number of items purchased.
  + **order\_size**: The size of the item purchased.
  + **order\_discount**: The total discount on that item.   
      
    **Product**
  + **Description**: Goods available for purchase at the cafe.
  + **Attributes**:
  + **product\_id** (PK): A unique identifier for each product.
  + **product\_name**: The name of the product.
  + **product\_description**: A brief description of the product.
  + **product\_price**: The standard price of the product.
  + **product\_category**: The category in which the product lies.
  + **FK\_purchase\_detail\_pk\_purchase\_id**: A foreign key to Purchase detail entity, identifying what wholesale purchase it refers to.

**Purchase\_Detail**

**Description**: Represents the wholesale purchases made by the cafe for stocking products.

**Attributes**:

* + **purchase\_id** (PK): A unique identifier for each purchase detail record.
  + **order\_date:** The date when the wholesale purchase was made.
  + **quantity**: The amount of the products ordered in this purchase transaction.
  + **unit\_price**: The cost per unit of the product at the time of purchase.
  + **total\_price**: The overall cost of the purchase transaction.

1. **Relationship and Cardinality Description**

**Relationship**: “places” between entities cafe\_store and order.

**Cardinality**: 1:M between cafe\_store and order.

**Business rule:** A cafe\_store can make zero or many orders; an order must only be associated with one cafe\_store.

**Relationship**: “includes” between order and order\_detail.

**Cardinality**: 1:M between order and order\_detail.

**Business rule**: An order must contain one or many order\_details; an order\_detail must be from only one order.

**Relationship**: “contains” between product and order\_detail.

**Cardinality**: 1:M between product and order\_detail.

**Business rule**: A product can be in zero or many order\_details; an order\_detail must contain only one product.

**Relationship**: “has” between purchase\_detail and product.

**Cardinality**: 1:M between purchase\_detail and product.

**Business rule**: A purchase\_details may include zero or many products. A product must have only one purchase\_detail.

**3. Assumptions and Special Considerations**

**Assumptions**:

1. Uniform Pricing: It's assumed that within a single purchase order (as captured in the Purchase\_Detail entity), the unit price of a product is uniform across the quantity ordered. This simplifies the calculation of the total price but assumes no tiered pricing for bulk orders.
2. Inventory Updates: The system is presumed to automatically update inventory levels based on the Quantity attribute in the Purchase\_Detail entity. This assumes a direct integration between the purchasing system and inventory management functionalities.
3. Product Availability: For the Order\_Detail entity, it's assumed that all products listed as part of an order are available at the time of the order. This overlooks backorders or inventory shortages, simplifying order processing.

**Special Consideration:** Order\_Detail is treated as a weak entity due to its dependence on Order and Product for uniqueness and existence, necessitating an identifying relationship.

**Data Definition Language (DDL)**

The DDL part of this project involves the creation of SQL scripts to define the database schema. This includes the creation of tables, constraints, indices, and relationships. Detailed SQL statements were crafted to ensure all aspects of the café's operations were adequately modeled in the database. For instance, tables for storing product details, order information, inventory records, and customer data were created with relationships that enforce data integrity and facilitate complex queries.

set search\_path to 'cafe';

/\*

Project BAUN 6320 - UTD

\*/

/\* DROP statements to clean up objects from previous run \*/

-- Triggers

DROP TRIGGER IF EXISTS TRG\_cafe\_store\_store\_id ON cafe\_store;

DROP TRIGGER IF EXISTS TRG\_orders\_order\_id ON orders;

DROP TRIGGER IF EXISTS TRG\_purchase\_detail\_purchase\_id ON purchase\_detail;

DROP TRIGGER IF EXISTS TRG\_product\_product\_id ON product;

-- Trigger Functions

DROP FUNCTION IF EXISTS TRG\_cafe\_store\_store\_id\_function();

DROP FUNCTION IF EXISTS TRG\_orders\_order\_id\_function();

DROP FUNCTION IF EXISTS TRG\_purchase\_detail\_purchase\_id\_function();

DROP FUNCTION IF EXISTS TRG\_product\_product\_id\_function();

-- Sequences

DROP SEQUENCE IF EXISTS SEQ\_cafe\_store\_store\_id;

DROP SEQUENCE IF EXISTS SEQ\_orders\_order\_id;

DROP SEQUENCE IF EXISTS SEQ\_purchase\_detail\_purchase\_id;

DROP SEQUENCE IF EXISTS SEQ\_product\_product\_id;

-- Indices

DROP INDEX IF EXISTS IDX\_cafe\_store\_store\_id;

DROP INDEX IF EXISTS IDX\_orders\_order\_id;

DROP INDEX IF EXISTS IDX\_orders\_FK\_cafe\_store\_pk\_store\_id;

DROP INDEX IF EXISTS IDX\_purchase\_detail\_purchase\_id;

DROP INDEX IF EXISTS IDX\_product\_product\_id;

DROP INDEX IF EXISTS IDX\_product\_FK\_purchase\_detail\_pk\_purchase\_id;

DROP INDEX IF EXISTS IDX\_order\_detail\_FK\_order\_pk\_order\_id;

DROP INDEX IF EXISTS IDX\_order\_detail\_FK\_product\_pk\_product\_id;

-- Tables

DROP TABLE IF EXISTS order\_detail;

DROP TABLE IF EXISTS product;

DROP TABLE IF EXISTS orders;

DROP TABLE IF EXISTS purchase\_detail;

DROP TABLE IF EXISTS cafe\_store;

/\* Create Sequences \*/

CREATE SEQUENCE SEQ\_cafe\_store\_store\_id

START 100

INCREMENT 100;

CREATE SEQUENCE SEQ\_orders\_order\_id

START 1

INCREMENT 1;

CREATE SEQUENCE SEQ\_purchase\_detail\_purchase\_id

START 10

INCREMENT 5;

CREATE SEQUENCE SEQ\_product\_product\_id

START 10

INCREMENT 10;

/\* Create tables based on entities \*/

CREATE TABLE cafe\_store(

store\_id INTEGER NOT NULL,

store\_manager VARCHAR(30) NOT NULL,

store\_capacity INTEGER NOT NULL,

phone NUMERIC(10,0) NOT NULL,

street\_address VARCHAR(50) NOT NULL,

city VARCHAR(30) NOT NULL,

state VARCHAR(2) NOT NULL,

zip VARCHAR(5) NOT NULL,

CONSTRAINT PK\_cafe\_store PRIMARY KEY (store\_id)

);

CREATE TABLE orders(

order\_id INTEGER NOT NULL,

order\_tax NUMERIC(10, 2) NOT NULL,

order\_tip NUMERIC(10, 2) NOT NULL,

order\_total NUMERIC(10, 2) NOT NULL,

date\_and\_time DATE NOT NULL,

FK\_cafe\_store\_pk\_store\_id INTEGER NOT NULL,

CONSTRAINT PK\_order\_id PRIMARY KEY (order\_id),

CONSTRAINT FK\_cafe\_store FOREIGN KEY (FK\_cafe\_store\_pk\_store\_id) REFERENCES cafe\_store(store\_id)

);

CREATE TABLE purchase\_detail(

purchase\_id INTEGER NOT NULL,

unit\_price NUMERIC(10, 2) NOT NULL,

order\_date DATE NOT NULL,

quantity NUMERIC(10, 2) NOT NULL,

total\_price NUMERIC(15, 2) NOT NULL,

CONSTRAINT PK\_purchase\_detail PRIMARY KEY (purchase\_id)

);

CREATE TABLE product(

product\_id INTEGER NOT NULL,

product\_name VARCHAR(25) NOT NULL,

product\_price NUMERIC(10, 2) NOT NULL,

product\_category VARCHAR(30) NOT NULL,

product\_description VARCHAR(255) NOT NULL,

FK\_purchase\_detail\_pk\_purchase\_id INTEGER NOT NULL,

CONSTRAINT PK\_product\_id PRIMARY KEY (product\_id),

CONSTRAINT FK\_purchase\_detail FOREIGN KEY (FK\_purchase\_detail\_pk\_purchase\_id) REFERENCES purchase\_detail(purchase\_id)

);

CREATE TABLE order\_detail(

FK\_order\_pk\_order\_id INTEGER NOT NULL,

FK\_product\_pk\_product\_id INTEGER NOT NULL,

order\_special\_info VARCHAR(100),

order\_quantity INTEGER NOT NULL,

order\_discount DECIMAL(4, 2),

order\_prep\_time INTERVAL NOT NULL,

order\_size VARCHAR(10) NOT NULL,

CONSTRAINT PK\_order\_detail PRIMARY KEY (FK\_order\_pk\_order\_id, FK\_product\_pk\_product\_id),

CONSTRAINT FK\_orders FOREIGN KEY (FK\_order\_pk\_order\_id) REFERENCES orders(order\_id),

CONSTRAINT FK\_products FOREIGN KEY (FK\_product\_pk\_product\_id) REFERENCES product(product\_id)

);

/\* Create Trigger Functions \*/

CREATE OR REPLACE FUNCTION TRG\_cafe\_store\_store\_id\_function()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' OR TG\_OP = 'UPDATE' THEN

IF NEW.store\_id IS NULL THEN

NEW.store\_id := NEXTVAL('SEQ\_cafe\_store\_store\_id');

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION TRG\_orders\_order\_id\_function()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' OR TG\_OP = 'UPDATE' THEN

IF NEW.order\_id IS NULL THEN

NEW.order\_id := NEXTVAL('SEQ\_orders\_order\_id');

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION TRG\_purchase\_detail\_purchase\_id\_function()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' OR TG\_OP = 'UPDATE' THEN

IF NEW.purchase\_id IS NULL THEN

NEW.purchase\_id := NEXTVAL('SEQ\_purchase\_detail\_purchase\_id');

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION TRG\_product\_product\_id\_function()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' OR TG\_OP = 'UPDATE' THEN

IF NEW.product\_id IS NULL THEN

NEW.product\_id := NEXTVAL('SEQ\_product\_product\_id');

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

/\* Create Triggers \*/

CREATE TRIGGER TRG\_cafe\_store\_store\_id

BEFORE INSERT OR UPDATE OF store\_id ON cafe\_store

FOR EACH ROW

EXECUTE FUNCTION TRG\_cafe\_store\_store\_id\_function();

CREATE TRIGGER TRG\_orders\_order\_id

BEFORE INSERT OR UPDATE OF order\_id ON orders

FOR EACH ROW

EXECUTE FUNCTION TRG\_orders\_order\_id\_function();

CREATE TRIGGER TRG\_purchase\_detail\_purchase\_id

BEFORE INSERT OR UPDATE OF purchase\_id ON purchase\_detail

FOR EACH ROW

EXECUTE FUNCTION TRG\_purchase\_detail\_purchase\_id\_function();

CREATE TRIGGER TRG\_product\_product\_id

BEFORE INSERT OR UPDATE OF product\_id ON product

FOR EACH ROW

EXECUTE FUNCTION TRG\_product\_product\_id\_function();

/\* Create indices for natural keys, foreign keys, and frequently-queried columns \*/

-- cafe\_store

-- Primary Key

CREATE INDEX IDX\_cafe\_store\_store\_id ON cafe\_store(store\_id);

-- orders

-- Primary Key

CREATE INDEX IDX\_orders\_order\_id ON orders(order\_id);

-- Foreign Key

CREATE INDEX IDX\_orders\_FK\_cafe\_store\_pk\_store\_id ON orders(FK\_cafe\_store\_pk\_store\_id);

-- purchase\_detail

-- Primary Key

CREATE INDEX IDX\_purchase\_detail\_purchase\_id ON purchase\_detail(purchase\_id);

-- product

-- Primary Key

CREATE INDEX IDX\_product\_product\_id ON product(product\_id);

-- Foreign Key

CREATE INDEX IDX\_product\_FK\_purchase\_detail\_pk\_purchase\_id ON product(FK\_purchase\_detail\_pk\_purchase\_id);

-- order\_detail

-- Primary Key / Foreign Key

CREATE INDEX IDX\_order\_detail\_FK\_order\_pk\_order\_id ON order\_detail(FK\_order\_pk\_order\_id);

CREATE INDEX IDX\_order\_detail\_FK\_product\_pk\_product\_id ON order\_detail(FK\_product\_pk\_product\_id);

**Data Manipulation Language (DML)**

Following the creation of the database schema, DML scripts were developed to populate the database with initial data sets, including sample products, customer records, and preliminary orders. These scripts are crucial for testing the database's performance and ensuring that the data flows correctly through the various components of the database system.

set search\_path to 'cafe';

/\*

Project BAUN 6320 - UTD

\*/

-- cafe\_store table

INSERT INTO

cafe\_store (store\_manager, store\_capacity, phone, street\_address, city, state, zip)

VALUES

('Nisha Shah', 80, 4693853946, '1235 Main Street', 'Richardson', 'TX', 75080),

('William Wilson', 70, 2125439674, '261 Boradway', 'New York', 'NY', 10001),

('Ava Miller', 100, 3055552468, '2468 Oak Lane', 'Miami', 'FL', 33125),

('Nathan Shaw', 50, 5035455678, '5678 Pine Avenue', 'Portland', 'OR', 97201),

('Sofia Brown', 90, 3038649100, '910 Maple Drive', 'Denver', 'CO', 80202),

('Abigail Jones', 110, 4159451357, '1357 Cedar Road', 'San Francisco', 'CA', 94110),

('Dev Patel', 120, 2147427859, '7859 Willow Avenue', 'Dallas', 'TX', 75201),

('Ehan Smith', 40, 4699376925, '123 Frisco Avenue', 'Frisco', 'TX', 75034),

('Mateo Flores', 80, 3129376543, '6543 Walnut Boulevard', 'Chicago', 'IL', 60601),

('Santiago Lopez', 90, 3050583980, '456 Ocean Drive', 'Miami', 'FL', 33139);

-- orders table

INSERT INTO

orders (order\_tax, order\_tip, order\_total, date\_and\_time, FK\_cafe\_store\_pk\_store\_id)

VALUES

(2.01, 3, 24.49,'2023-04-20',800),

(1.38, 4, 16.78,'2023-06-23',100),

(0.8, 1, 9.75,'2023-07-11',900),

(5.36, 7, 65,'2023-07-14',300),

(0.91, 0, 11,'2023-08-03',200),

(2.90, 6, 35.18,'2023-11-27',700),

(4.34, 8, 52.87,'2024-01-04',600),

(1.11, 2, 13.5,'2024-02-13',400),

(2.26, 3, 27.45,'2024-03-19',100),

(1.45, 4, 17.56,'2024-03-22',500);

-- purchase\_detail table

INSERT INTO

purchase\_detail (unit\_price, order\_date, quantity, total\_price)

VALUES

(10, '2023-08-20', 8, 80),

(1.49, '2023-09-13', 30, 44.7),

(6, '2023-10-18', 20, 120),

(17.99, '2023-12-20', 7, 125.93),

(3.49, '2024-01-10', 25, 87.25),

(7.99, '2024-02-16', 13, 103.87),

(0.99, '2024-03-17', 40, 39.6),

(9.99, '2024-03-19', 14, 139.86),

(9.49, '2024-03-23', 8, 75.92),

(15, '2024-03-28', 12, 180);

-- product table

INSERT INTO

product (product\_name, product\_price, product\_category, product\_description, FK\_purchase\_detail\_pk\_purchase\_id)

VALUES

('Espresso Beans', 19.99, 'Coffee', 'High-quality espresso beans for brewing rich and flavorful shots.', 55),

('Vanilla Syrup', 12.99, 'Syrups', 'A sweet and aromatic syrup for adding flavor to coffee and tea.', 10),

('Croissant', 2.49, 'Pastries', 'A buttery and flaky pastry perfect for breakfast or a snack.', 40),

('Green Tea', 4.99, 'Tea', 'A refreshing and healthy green tea known for its antioxidants.', 35),

('Chocolate Muffin', 3.99, 'Pastries', 'A moist and chocolaty muffin that pairs well with coffee.', 25),

('Caramel Macchiato', 5.49, 'Coffee Drinks', 'A popular espresso-based drink with caramel and steamed milk.', 50),

('Iced Tea', 3.99, 'Tea', 'A cool and refreshing iced tea available in various flavors.', 20),

('Ham and Cheese Sandwich', 13.99, 'Sandwiches', 'A classic sandwich with ham, cheese, lettuce, and tomato.', 45),

('Almond Milk', 5.49, 'Milk Alternatives', 'A dairy-free alternative made from almonds, perfect for lattes.', 30),

('Blueberry Scone', 2.99, 'Pastries', 'A delicious scone filled with blueberries and topped with glaze.', 15);

-- order\_detail table

INSERT INTO

order\_detail (FK\_order\_pk\_order\_id, FK\_product\_pk\_product\_id, order\_special\_info, order\_quantity, order\_discount, order\_prep\_time, order\_size)

VALUES

(1, 10, 'Double shot', 2, 0.1, INTERVAL '15 minutes', 'Large'),

(2, 60, 'Extra foam', 1, 0.0, INTERVAL '9 minutes', 'Medium'),

(3, 30, 'None', 2, 0.15, INTERVAL '13 minutes', 'Medium'),

(4, 20, 'None', 1, 0.05, INTERVAL '6 minutes', 'Large'),

(5, 70, 'Extra ice', 2, 0.0, INTERVAL '8 minutes', 'Small'),

(6, 90, 'None', 2, 0.0, INTERVAL '11 minutes', 'Large'),

(7, 40, 'No sugar', 1, 0.10, INTERVAL '10 minutes', 'Medium'),

(8, 80, 'Extra cheese', 2, 0.0, INTERVAL '15 minutes', 'Medium'),

(9, 100, 'None', 1, 0.10, INTERVAL '10 minutes', 'Small'),

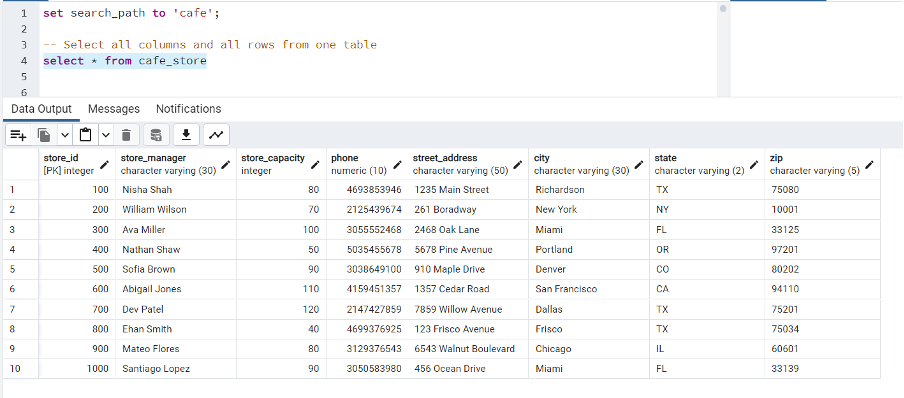
(10, 50, 'Extra chocolate', 2, 0.0, INTERVAL '9 minutes', 'Medium');

**SQL Queries**

The SQL queries developed for this project range from simple data retrieval to complex analytical queries that provide insights into business operations. These queries are vital for daily operations, such as generating sales reports, tracking inventory levels, and understanding customer buying patterns.

**Query 1:** Select all columns and all rows from one table (5 points)

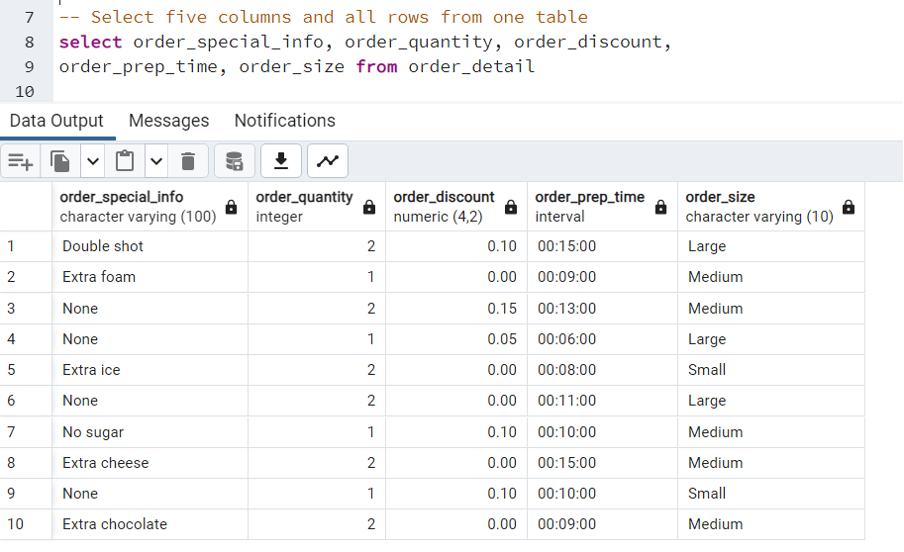
select \* from cafe\_store



**Query 2:** Select five columns and all rows from one table (5 points)

select order\_special\_info, order\_quantity, order\_discount,

order\_prep\_time, order\_size from order\_detail



**Query 3:** Select all columns from all rows from one view (5 points)

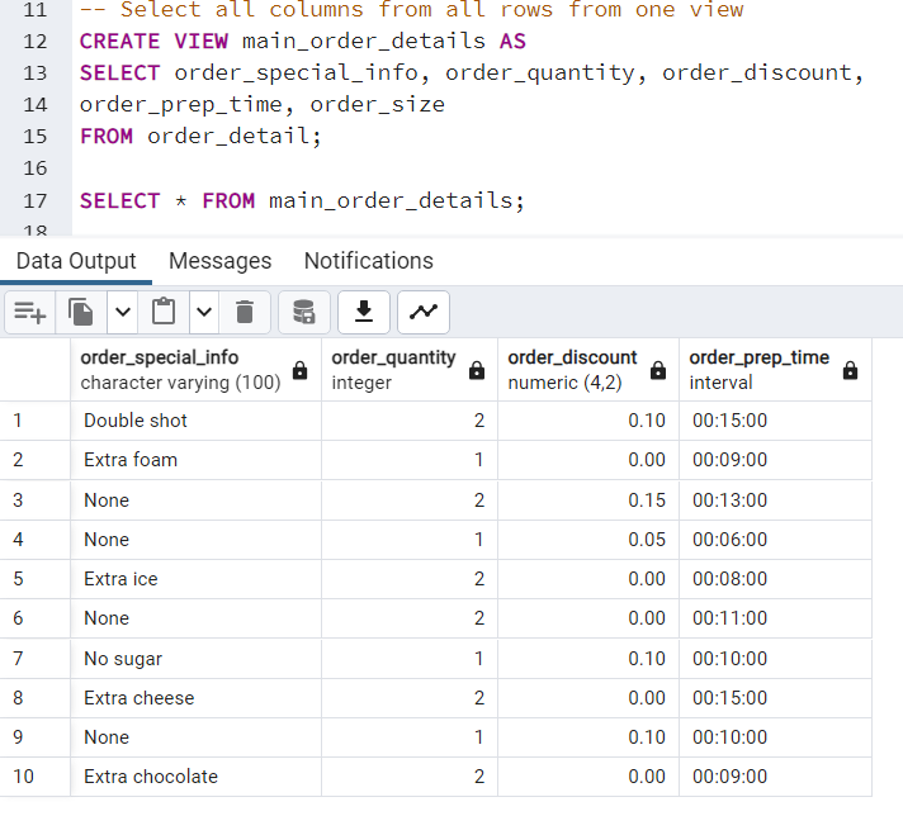
CREATE VIEW main\_order\_details AS

SELECT order\_special\_info, order\_quantity, order\_discount,

order\_prep\_time, order\_size

FROM order\_detail;

SELECT \* FROM main\_order\_details;

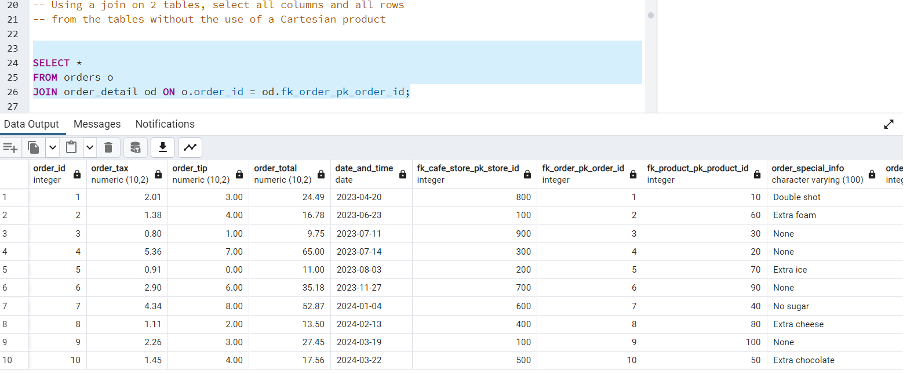


**Query 4:** Using a join on 2 tables, select all columns and all rows from the tables without the use of a Cartesian product (5 points)

SELECT \*

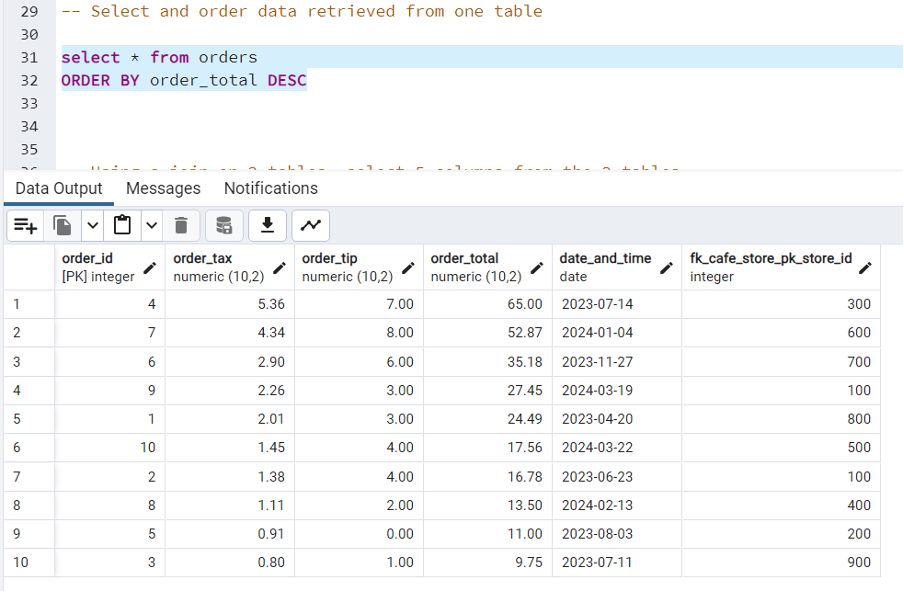
FROM orders o

JOIN order\_detail od ON o.order\_id = od.fk\_order\_pk\_order\_id;



**Query 5:** Select and order data retrieved from one table (5 points)

select \* from orders  
ORDER BY order\_total DESC



**Query 6:** Using a join on 3 tables, select 5 columns from the 3 tables. Use syntax that would limit the output to 3 rows (5 points)

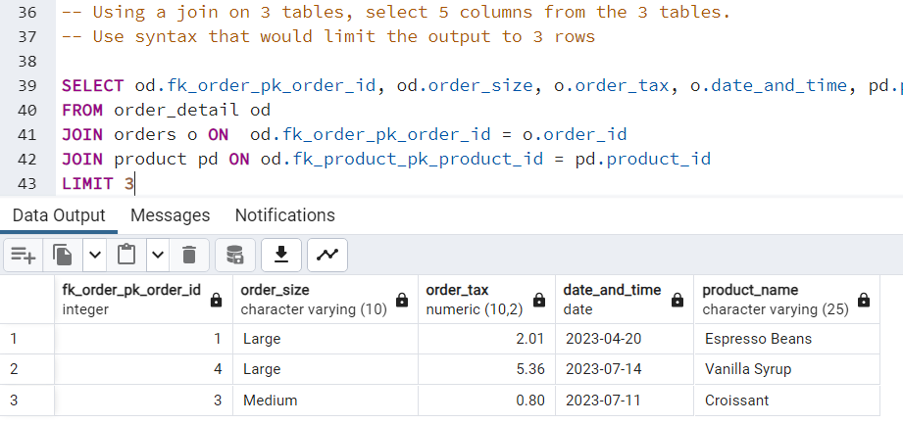
SELECT od.fk\_order\_pk\_order\_id, od.order\_size, o.order\_tax, o.date\_and\_time, pd.product\_name

FROM order\_detail od

JOIN orders o ON od.fk\_order\_pk\_order\_id = o.order\_id

JOIN product pd ON od.fk\_product\_pk\_product\_id = pd.product\_id

LIMIT 3



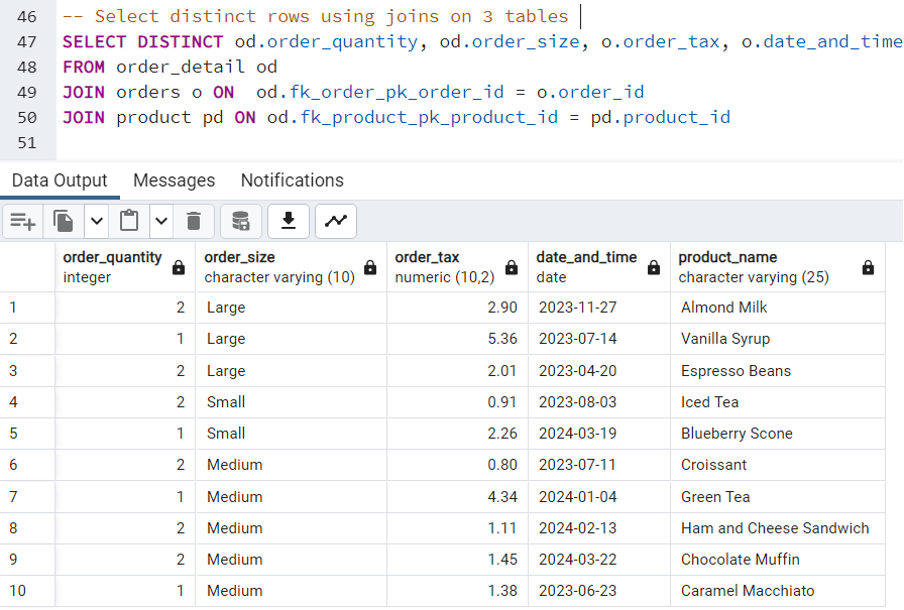
**Query 7:** Select distinct rows using joins on 3 tables (5 points)

SELECT DISTINCT od.order\_quantity, od.order\_size, o.order\_tax, o.date\_and\_time, pd.product\_name

FROM order\_detail od

JOIN orders o ON od.fk\_order\_pk\_order\_id = o.order\_id

JOIN product pd ON od.fk\_product\_pk\_product\_id = pd.product\_id



**Query 8:** Use GROUP BY and HAVING in a select statement using one or more tables (5 points)

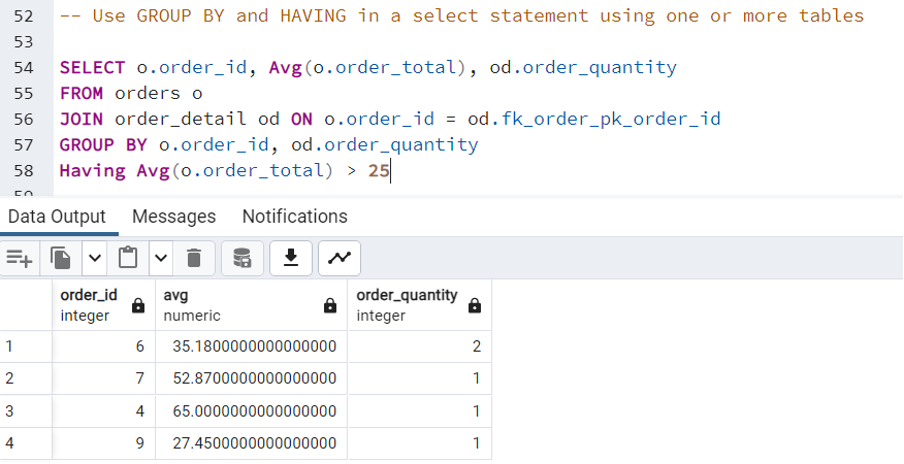
SELECT o.order\_id, Avg(o.order\_total), od.order\_quantity

FROM orders o

JOIN order\_detail od ON o.order\_id = od.fk\_order\_pk\_order\_id

GROUP BY o.order\_id, od.order\_quantity

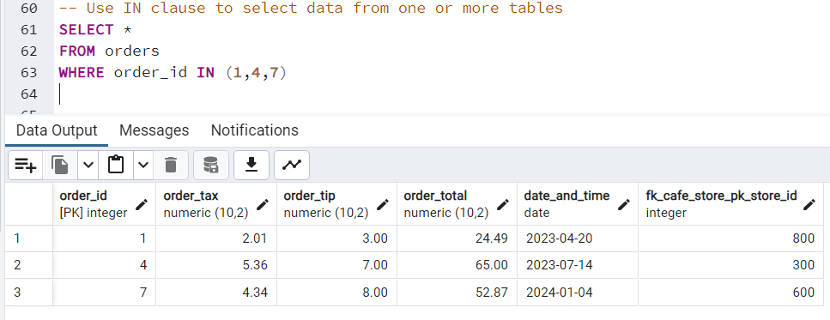
Having Avg(o.order\_total) > 25



**Query 9**: Use IN clause to select data from one or more tables (5 points)

SELECT \*

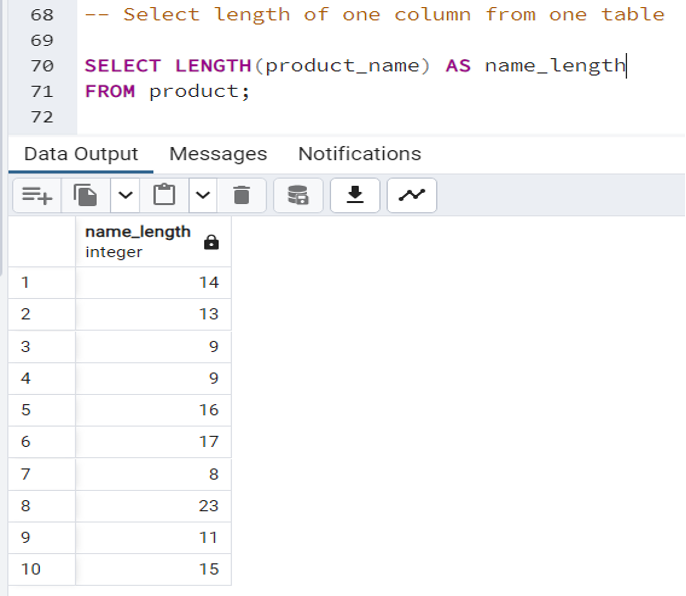
FROM orders  
WHERE order\_id IN (1,4,7)



**Query 10:** Select length of one column from one table (use LENGTH function) (5 points)

SELECT LENGTH(product\_name) AS name\_length

FROM product;



**Query 11:** Delete one record from one table. Use select statements to demonstrate the table contents before and after the DELETE statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed (5 points)

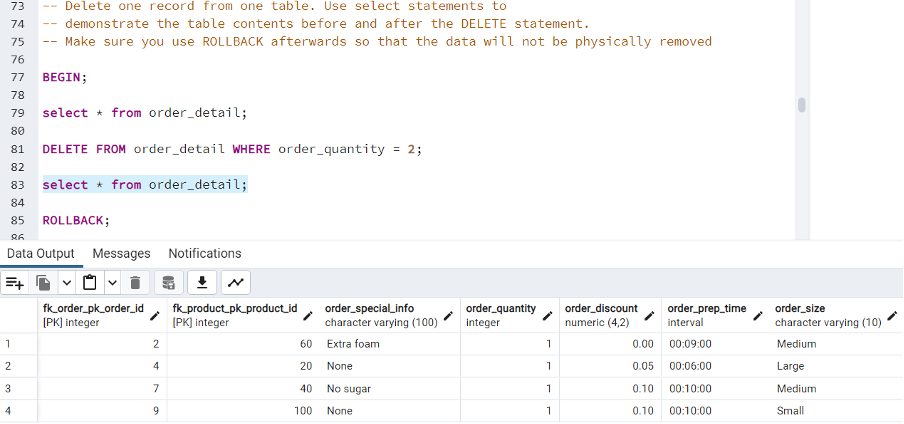
BEGIN;

select \* from order\_detail;

DELETE FROM order\_detail WHERE order\_quantity = 2;

select \* from order\_detail;

ROLLBACK;



**Query 12:** Update one record from one table. Use select statements to demonstrate the table contents before and after the UPDATE statement. Make sure you use ROLLBACK afterward so that the data will not be physically removed (5 points)

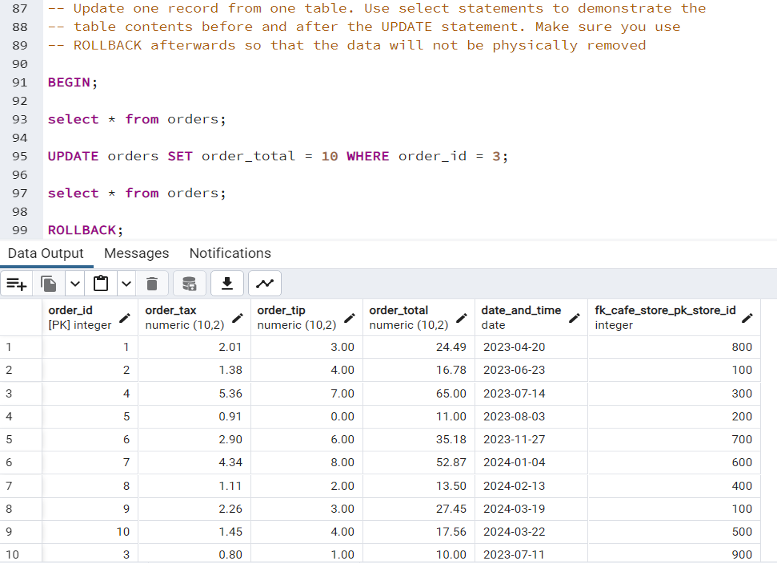
BEGIN;

select \* from orders;

UPDATE orders SET order\_total = 10 WHERE order\_id = 3;

select \* from orders;

ROLLBACK;



**Performing 2 Additional Advanced Queries**

Advance Query 1. Revenue Analysis by Store and Product Category

SELECT

cs.store\_id,

cs.store\_manager,

p.product\_category AS category,

SUM(od.order\_quantity \* p.product\_price) AS total\_revenue

FROM

cafe\_store cs

JOIN

orders o ON cs.store\_id = o.FK\_cafe\_store\_pk\_store\_id

JOIN

order\_detail od ON o.order\_id = od.FK\_order\_pk\_order\_id

JOIN

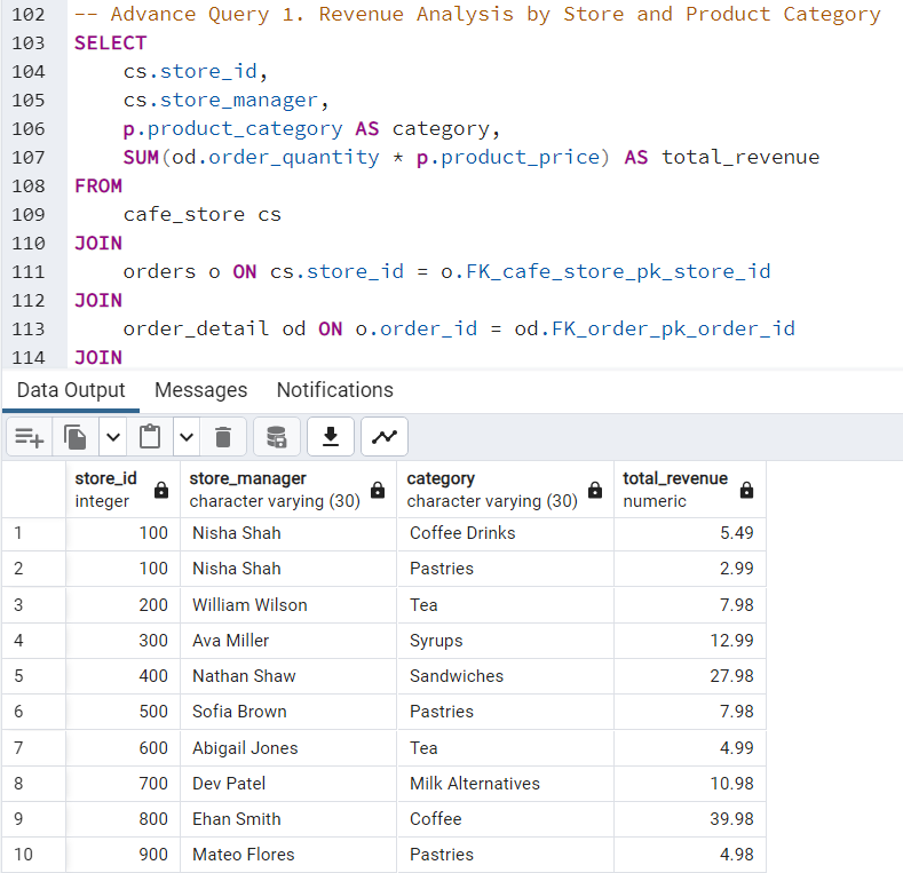
product p ON od.FK\_product\_pk\_product\_id = p.product\_id

GROUP BY

cs.store\_id, cs.store\_manager, p.product\_category

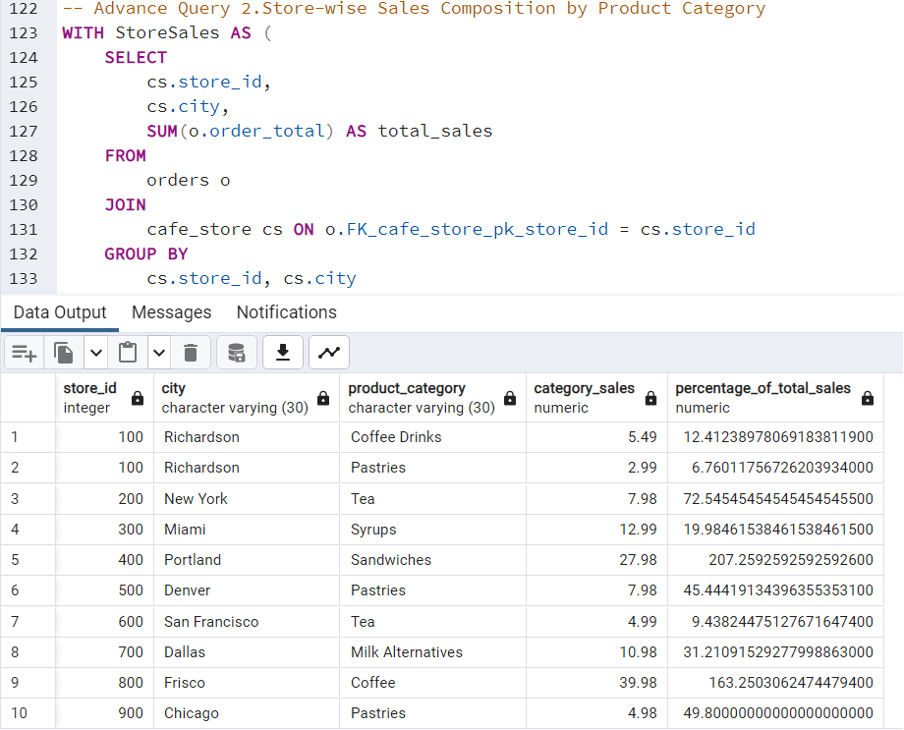
ORDER BY

cs.store\_id, total\_revenue DESC;



Advance Query 2.Store-wise Sales Composition by Product Category

WITH StoreSales AS (  
 SELECT  
 cs.store\_id,  
 cs.city,  
 SUM(o.order\_total) AS total\_sales  
 FROM  
 orders o  
 JOIN  
 cafe\_store cs ON o.FK\_cafe\_store\_pk\_store\_id = cs.store\_id  
 GROUP BY  
 cs.store\_id, cs.city  
),  
CategorySales AS (  
 SELECT  
 cs.store\_id,  
 p.product\_category,  
 SUM(od.order\_quantity \* p.product\_price) AS category\_sales  
 FROM  
 order\_detail od  
 JOIN  
 orders o ON od.FK\_order\_pk\_order\_id = o.order\_id  
 JOIN  
 product p ON od.FK\_product\_pk\_product\_id = p.product\_id  
 JOIN  
 cafe\_store cs ON o.FK\_cafe\_store\_pk\_store\_id = cs.store\_id  
 GROUP BY  
 cs.store\_id, p.product\_category  
)  
SELECT  
 ss.store\_id,  
 ss.city,  
 cs.product\_category,  
 cs.category\_sales,  
 (cs.category\_sales / ss.total\_sales \* 100) AS percentage\_of\_total\_sales  
FROM  
 StoreSales ss  
JOIN  
 CategorySales cs ON ss.store\_id = cs.store\_id  
ORDER BY  
 ss.store\_id, cs.category\_sales DESC;



**Conclusion**

The database system designed for the café chain is a comprehensive solution that integrates various aspects of the business operations into a unified data management framework. Through the careful design and implementation of the database schema, DML scripts, and SQL queries, the system is equipped to handle the current and future data management needs of the café chain. This project not only supports operational efficiency but also provides a foundation for strategic decision-making based on reliable data.

This long-form technical report provides a thorough overview of the database project from concept to implementation, ensuring it aligns with the detailed example provided earlier. This structure ensures that every phase of the project is well-documented, providing a comprehensive reference for the design, development, and operational strategy of the database system.