**SQL Capstone Project**

**Project Title: Online Chocolate Shop Management System**

Project Overview:

Design and implement a database system for an online chocolate shop. The system should handle various aspects of managing the shop, including inventory management, customers, orders and user authentication

Skills Demonstrated:

**Beginner:**

* Creating tables and defining relationships
* Basic CRUD(Create, Read, Update, Delete) operations

USE:

USE chocolate;

DROP:

–Dropped tables due to Foreign Key related errors/order of operations

DROP TABLE chocolates;

DROP TABLE orders;

DROP TABLE customers, contact\_info; –Foreign key work around

CREATE:

create table chocolates(

chocolate\_id int not null auto\_increment,

chocolate\_name varchar(255),

price\_per\_ounce\_dollars float(5,2),

primary key (chocolate\_id)

);

create table customers(

customer\_id int not null auto\_increment,

first\_name varchar(255),

last\_name varchar(255),

contact\_info\_id int,

primary key (customer\_id)

);

create table orders(

order\_id int not null auto\_increment,

order\_timestamp date,

customer\_id int,

chocolate\_id int,

order\_amount\_ounces float(5,2),

order\_price float(5,2),

primary key (order\_id)

);

create table contact\_info(

contact\_info\_id int not null auto\_increment,

customer\_id int,

address\_number int,

street varchar(255),

city varchar(255),

zip int,

phone\_number int,

email varchar(255),

primary key (contact\_info\_id)

);

-INSERT:

INSERT INTO chocolates (chocolate\_name, price\_per\_ounce\_dollars) VALUES

('Dark Chocolate', 2.50),

('Milk Chocolate', 2.00),

('White Chocolate', 2.75),

('Semisweet Chocolate', 2.75),

('Bittersweet Chocolate', 2.85),

('Unsweetened Chocolate', 2.50),

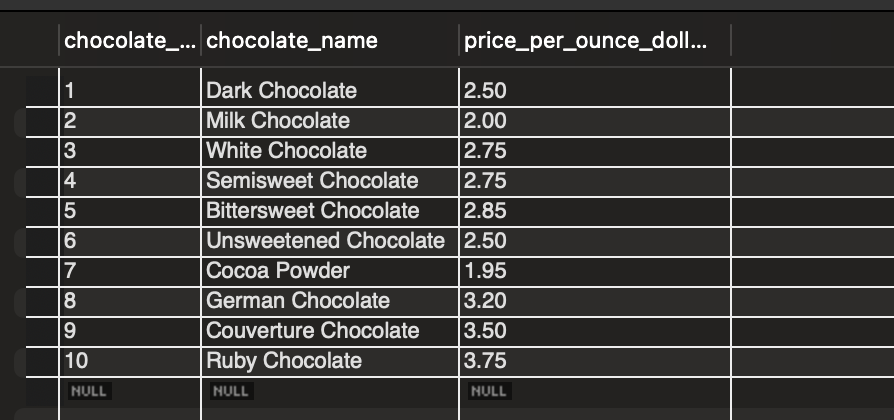
('Cocoa Powder', 1.95),

('German Chocolate', 3.20),

('Couverture Chocolate', 3.50),

('Ruby Chocolate', 3.75);

SELECT \* FROM chocolates;



INSERT INTO customers (first\_name, last\_name, contact\_info\_id) VALUES

('John', 'Doe', 1),

('Jane', 'Smith', 2),

('Michael', 'Johnson', 3),

('Alice', 'Johnson', 4),

('Bob', 'Smith', 5),

('Emily', 'Davis', 6),

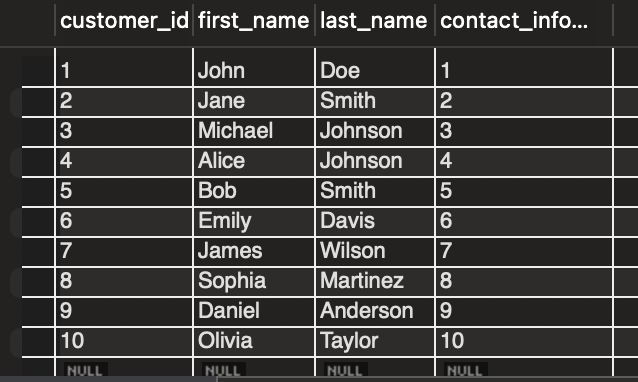
('James', 'Wilson', 7),

('Sophia', 'Martinez', 8),

('Daniel', 'Anderson', 9),

('Olivia', 'Taylor', 10);

SELECT \* from customers;



INSERT INTO orders (order\_timestamp, customer\_id, chocolate\_id, order\_amount\_ounces, order\_price) VALUES

('2024-05-09', 1, 1, 10.0, 25.00),

('2024-05-09', 2, 2, 8.0, 16.00),

('2024-05-09', 3, 3, 6.0, 16.50),

('2024-05-10', 2, 1, 5.0, 12.50),

('2024-05-10', 3, 3, 4.0, 11.00),

('2024-05-11', 1, 5, 3.0, 9.60),

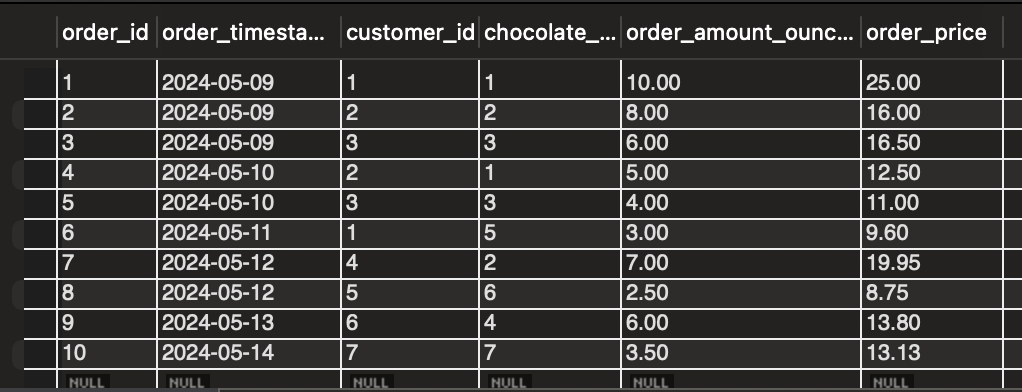
('2024-05-12', 4, 2, 7.0, 19.95),

('2024-05-12', 5, 6, 2.5, 8.75),

('2024-05-13', 6, 4, 6.0, 13.80),

('2024-05-14', 7, 7, 3.5, 13.13);

SELECT \* from customers;



INSERT INTO contact\_info (customer\_id, address\_number, street, city, zip, phone\_number, email) VALUES

(1, 123, 'Main St', 'Anytown', 12345, 1551234567, 'john@example.com'),

(2, 456, 'Broadway', 'Othertown', 54321, 1559876543, 'jane@example.com'),

(3, 789, 'Oak St', 'Somewhere', 98765, 1555555555, 'michael@example.com'),

(4, 321, 'Maple Ave', 'Somewhere', 54321, 1553334444, 'bob@example.com'),

(5, 654, 'Elm St', 'Othertown', 98765, 1554445555, 'emily@example.com'),

(6, 987, 'Pine Rd', 'Anytown', 12345, 1555556666, 'james@example.com'),

(7, 123, 'Oak St', 'Somewhere', 54321, 1556667777, 'sophia@example.com'),

(8, 456, 'Cedar Ln', 'Anytown', 12345, 1557778888, 'daniel@example.com'),

(9, 789, 'Birch Dr', 'Othertown', 98765, 1558889999, 'olivia@example.com'),

(10, 987, 'Spruce Ct', 'Somewhere', 54321, 1559990000, 'customer@example.com');

SELECT \* FROM contact\_info;



* Simple queries using SELECT, WHERE, and ORDER BY clauses

-- Retrieve all chocolates with a price per ounce greater than $3.00, ordered by chocolate name:

SELECT chocolate\_name, price\_per\_ounce\_dollars

FROM chocolates

WHERE price\_per\_ounce\_dollars > 3.00

ORDER BY chocolate\_name;



-- Retrieve all customers who have placed an order, along with their contact information, ordered by last name:

SELECT c.first\_name, c.last\_name, ci.street, ci.city, ci.phone\_number, ci.email

FROM customers c

JOIN contact\_info ci ON c.contact\_info\_id = ci.contact\_info\_id

WHERE c.customer\_id IN (SELECT DISTINCT customer\_id FROM orders)

ORDER BY c.last\_name;



-- Retrieve the top 5 customers with the highest total order amounts,

-- along with the total order amount for each customer, ordered by total order amount (descending):

SELECT c.first\_name, c.last\_name, SUM(o.order\_price) AS total\_order\_amount

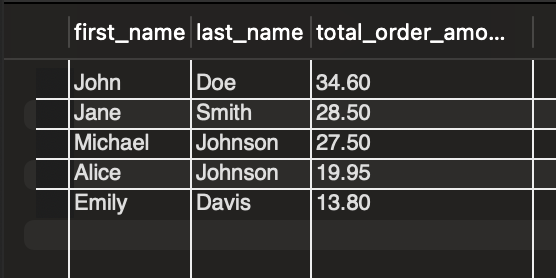
FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id

ORDER BY total\_order\_amount DESC

LIMIT 5;



-- Retrieve the orders placed on May 10, 2024, along with the customer's first name,

-- last name, and order amount, ordered by order amount (descending):

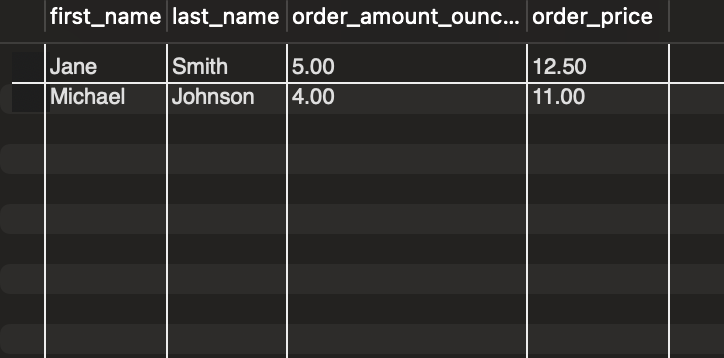
SELECT c.first\_name, c.last\_name, o.order\_amount\_ounces, o.order\_price

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_timestamp = '2024-05-10'

ORDER BY o.order\_amount\_ounces DESC;



-- Retrieve the contact information for customers who have placed orders for cocoa powder, ordered by city:

SELECT ci.\*

FROM contact\_info ci

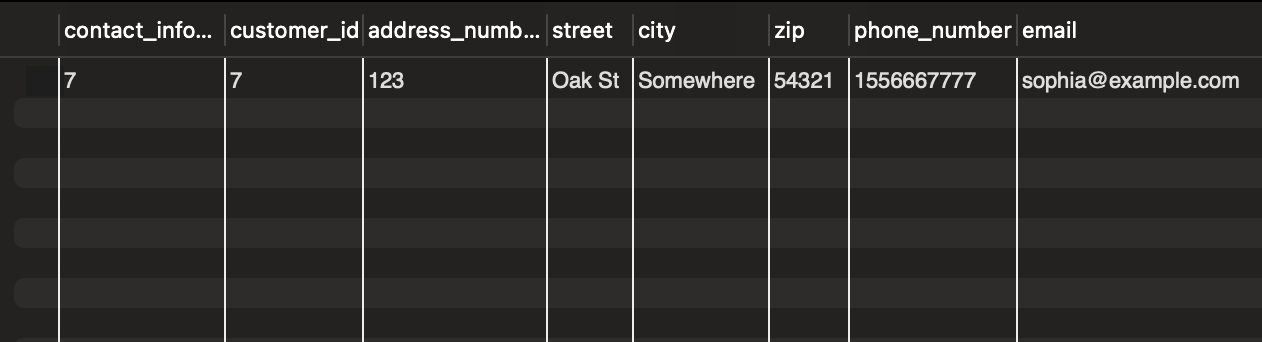
JOIN customers c ON ci.customer\_id = c.customer\_id

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN chocolates ch ON o.chocolate\_id = ch.chocolate\_id

WHERE ch.chocolate\_name = 'Cocoa Powder'

ORDER BY ci.city;



**Intermediate:**

* Implementing constraints and indexes for data integrity and performance optimization

Foreign Keys:

alter table customers add foreign key (contact\_info\_id) references contact\_info(contact\_info\_id);

alter table contact\_info add foreign key (customer\_id) references customers(customer\_id);

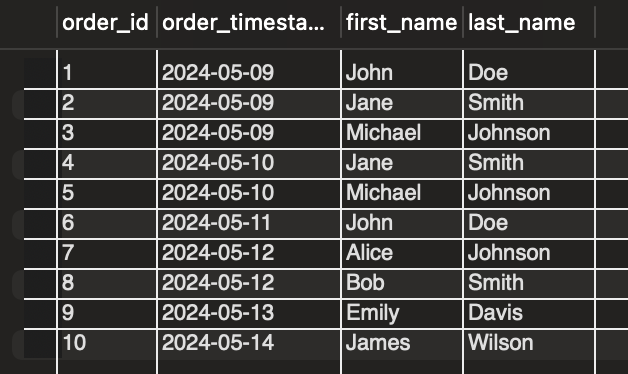
* Utilizing JOIN operations for querying data from multiple tables

-- Retrieve the details of all orders, including the customer's first and last name:

SELECT orders.order\_id, orders.order\_timestamp, customers.first\_name, customers.last\_name

FROM orders

JOIN customers ON orders.customer\_id = customers.customer\_id;



-- Retrieve the total order amount and customer's email for each order placed on a specific date:

SELECT orders.order\_id, SUM(orders.order\_price) AS total\_order\_amount, contact\_info.email

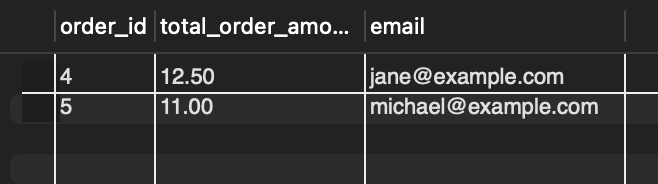
FROM orders

JOIN customers ON orders.customer\_id = customers.customer\_id

JOIN contact\_info ON customers.contact\_info\_id = contact\_info.contact\_info\_id

WHERE orders.order\_timestamp = '2024-05-10'

GROUP BY orders.order\_id;



-- Retrieve the details of customers who ordered a specific chocolate, along with their address information:

SELECT customers.first\_name, customers.last\_name, contact\_info.address\_number, contact\_info.street, contact\_info.city, contact\_info.zip

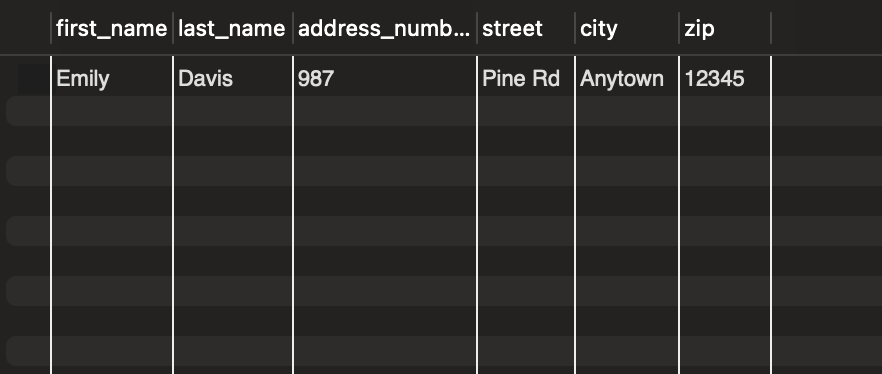
FROM customers

JOIN orders ON customers.customer\_id = orders.customer\_id

JOIN chocolates ON orders.chocolate\_id = chocolates.chocolate\_id

JOIN contact\_info ON customers.contact\_info\_id = contact\_info.contact\_info\_id

WHERE chocolates.chocolate\_name = 'Semisweet Chocolate';



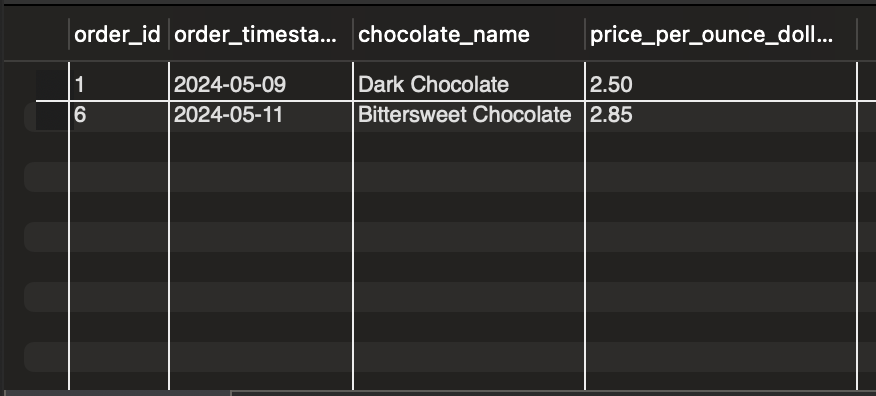
-- Retrieve the order details for a specific customer, including the chocolate name and price per ounce:

SELECT orders.order\_id, orders.order\_timestamp, chocolates.chocolate\_name, chocolates.price\_per\_ounce\_dollars

FROM orders

JOIN chocolates ON orders.chocolate\_id = chocolates.chocolate\_id

WHERE orders.customer\_id = 1;



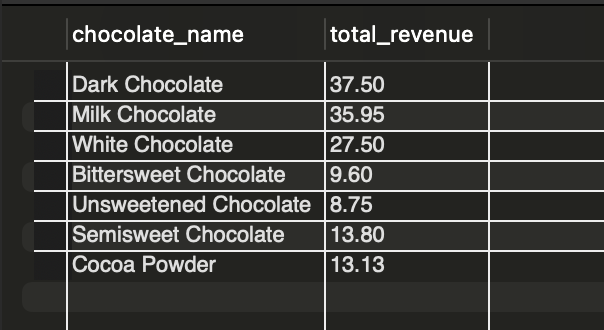
-- Retrieve the total revenue generated by each chocolate type:

SELECT chocolates.chocolate\_name, SUM(orders.order\_price) AS total\_revenue

FROM orders

JOIN chocolates ON orders.chocolate\_id = chocolates.chocolate\_id

GROUP BY chocolates.chocolate\_name;



* Aggregating data using GROUP BY and aggregate functions

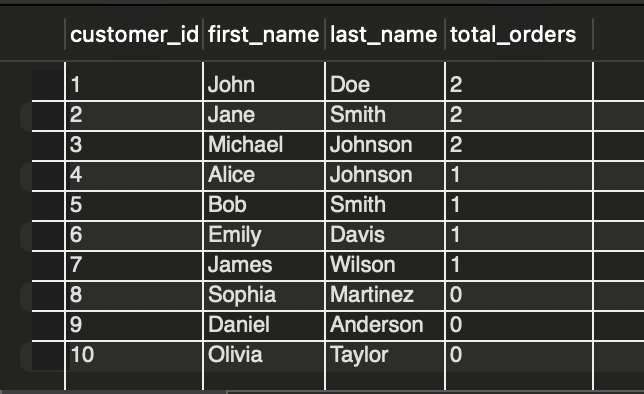
-- Calculate the total number of orders placed by each customer:

SELECT customers.customer\_id, customers.first\_name, customers.last\_name, COUNT(orders.order\_id) AS total\_orders

FROM customers

LEFT JOIN orders ON customers.customer\_id = orders.customer\_id

GROUP BY customers.customer\_id;



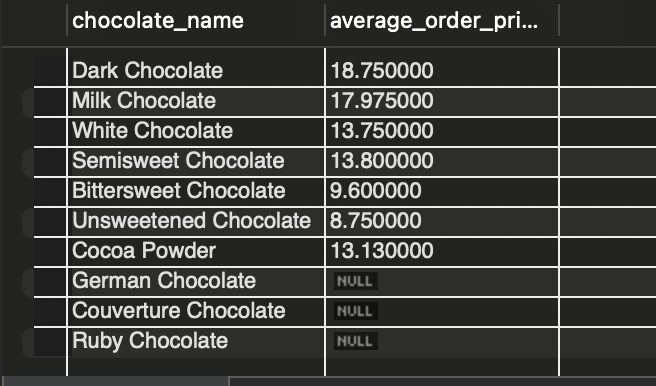
-- Find the average order price for each chocolate:

SELECT chocolates.chocolate\_name, AVG(orders.order\_price) AS average\_order\_price

FROM chocolates

LEFT JOIN orders ON chocolates.chocolate\_id = orders.chocolate\_id

GROUP BY chocolates.chocolate\_name;



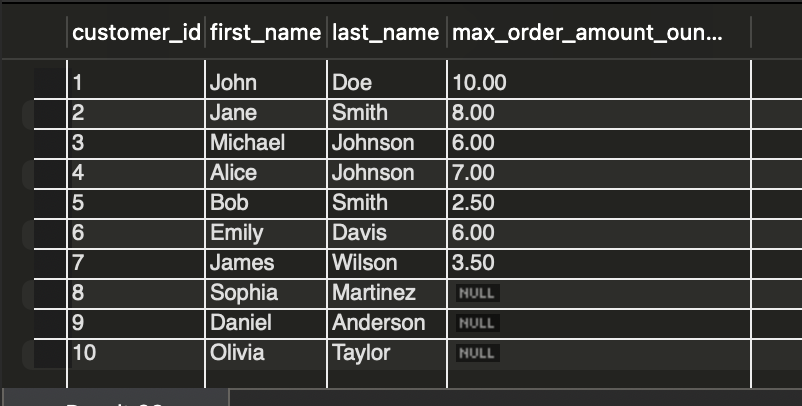
-- Determine the maximum order amount (in ounces) placed by each customer:

SELECT customers.customer\_id, customers.first\_name, customers.last\_name, MAX(orders.order\_amount\_ounces) AS max\_order\_amount\_ounces

FROM customers

LEFT JOIN orders ON customers.customer\_id = orders.customer\_id

GROUP BY customers.customer\_id;



-- Calculate the total revenue generated for each city:

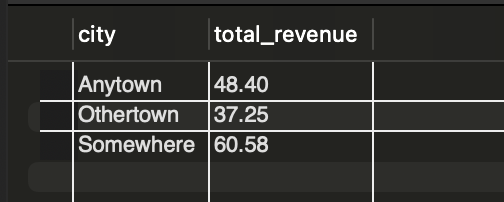
SELECT contact\_info.city, SUM(orders.order\_price) AS total\_revenue

FROM contact\_info

LEFT JOIN customers ON contact\_info.contact\_info\_id = customers.contact\_info\_id

LEFT JOIN orders ON customers.customer\_id = orders.customer\_id

GROUP BY contact\_info.city;



-- Find the minimum and maximum prices per ounce for each chocolate:

SELECT chocolates.chocolate\_name, MIN(chocolates.price\_per\_ounce\_dollars) AS min\_price\_per\_ounce, MAX(chocolates.price\_per\_ounce\_dollars) AS max\_price\_per\_ounce

FROM chocolates

GROUP BY chocolates.chocolate\_name;



* Implementing views and stored procedures for modularizing queries and operations

-- Customer Orders View:

CREATE VIEW customer\_orders AS

SELECT c.first\_name, c.last\_name, ci.email, o.order\_id, o.order\_timestamp, ch.chocolate\_name, o.order\_amount\_ounces, o.order\_price

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN chocolates ch ON o.chocolate\_id = ch.chocolate\_id

JOIN contact\_info ci ON c.contact\_info\_id = ci.contact\_info\_id;

SELECT \* FROM chocolate.customer\_orders;



-- Total Revenue by Chocolate View:

CREATE VIEW total\_revenue\_by\_chocolate AS

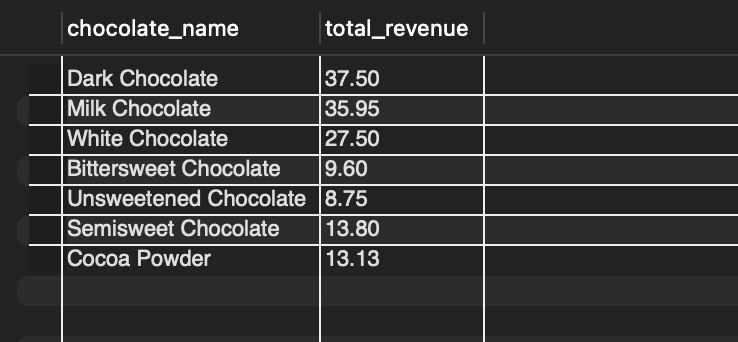
SELECT ch.chocolate\_name, SUM(o.order\_price) AS total\_revenue

FROM chocolates ch

JOIN orders o ON ch.chocolate\_id = o.chocolate\_id

GROUP BY ch.chocolate\_name;

SELECT \* FROM chocolate.total\_revenue\_by\_chocolate;



-- High-Value Customers View:

CREATE VIEW high\_value\_customers AS

SELECT c.customer\_id, c.first\_name, c.last\_name, ci.email, SUM(o.order\_price) AS total\_order\_amount

FROM customers c

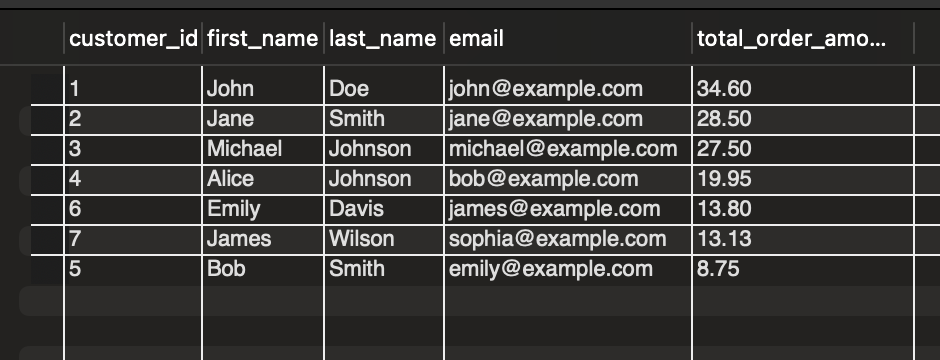
JOIN orders o ON c.customer\_id = o.customer\_id

JOIN contact\_info ci ON c.contact\_info\_id = ci.contact\_info\_id

GROUP BY c.customer\_id

ORDER BY total\_order\_amount DESC;

SELECT \* FROM chocolate.high\_value\_customers;



-- Top-Selling Chocolates View:

CREATE VIEW top\_selling\_chocolates AS

SELECT ch.chocolate\_name, SUM(o.order\_amount\_ounces) AS total\_quantity\_sold, SUM(o.order\_price) AS total\_revenue

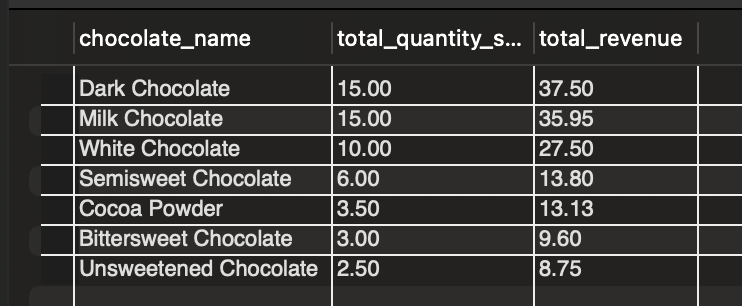
FROM chocolates ch

JOIN orders o ON ch.chocolate\_id = o.chocolate\_id

GROUP BY ch.chocolate\_name

ORDER BY total\_quantity\_sold DESC;

SELECT \* FROM chocolate.top\_selling\_chocolates;



-- Monthly Revenue Trend View:

CREATE VIEW monthly\_revenue\_trend AS

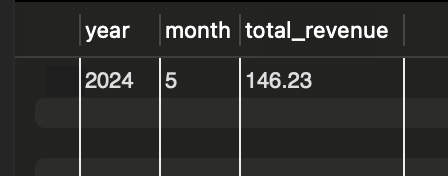
SELECT YEAR(o.order\_timestamp) AS year, MONTH(o.order\_timestamp) AS month, SUM(o.order\_price) AS total\_revenue

FROM orders o

GROUP BY YEAR(o.order\_timestamp), MONTH(o.order\_timestamp)

ORDER BY year, month;

SELECT \* FROM chocolate.monthly\_revenue\_trend;



-- Customer Location View:

CREATE VIEW customer\_location AS

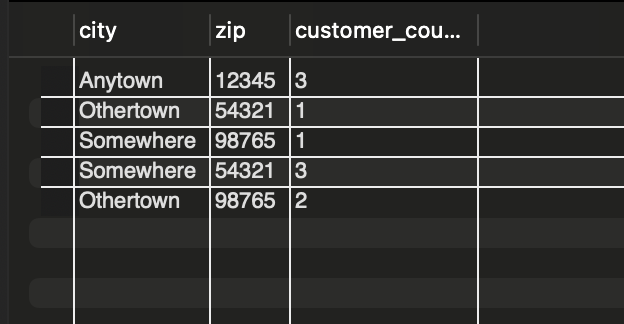
SELECT ci.city, ci.zip, COUNT(\*) AS customer\_count

FROM contact\_info ci

JOIN customers c ON ci.contact\_info\_id = c.contact\_info\_id

GROUP BY ci.city, ci.zip;

SELECT \* FROM chocolate.customer\_location;



**Advanced:**

* Handling transactions for maintaining data consistency and integrity

-- Add the inventory\_quantity column to the chocolates table

ALTER TABLE chocolates ADD COLUMN inventory\_quantity INT DEFAULT 0;

-- Begin a new transaction

START TRANSACTION;

-- Perform database operations within the transaction

INSERT INTO orders (customer\_id, chocolate\_id, order\_amount\_ounces, order\_price) VALUES (1, 3, 2.5, 15.00);

UPDATE chocolates SET inventory\_quantity = inventory\_quantity - 2.5 WHERE chocolate\_id = 3;

-- Commit the transaction to make its changes permanent

COMMIT;

-- Begin another transaction

START TRANSACTION;

-- Perform more database operations within the transaction

INSERT INTO orders (customer\_id, chocolate\_id, order\_amount\_ounces, order\_price) VALUES (2, 1, 3.0, 20.00);

UPDATE chocolates SET inventory\_quantity = inventory\_quantity - 3.0 WHERE chocolate\_id = 1;

-- Set a savepoint within the transaction

SAVEPOINT after\_first\_order;

-- Perform additional operations

INSERT INTO orders (customer\_id, chocolate\_id, order\_amount\_ounces, order\_price) VALUES (2, 4, 1.5, 12.00);

UPDATE chocolates SET inventory\_quantity = inventory\_quantity - 1.5 WHERE chocolate\_id = 4;

-- Rollback to the savepoint to undo changes made after it

ROLLBACK TO SAVEPOINT after\_first\_order;

-- Commit the transaction to make its changes permanent

COMMIT;

SELECT \* FROM chocolates;



* Implementing complex queries involving subqueries, common table expressions, and window functions

-- Find customers who have placed orders for chocolates with a price higher than the average price,

-- along with their rank based on total order amount

WITH customer\_order\_totals AS (

SELECT customer\_id, SUM(order\_price) AS total\_amount

FROM orders

GROUP BY customer\_id

)

SELECT customer\_id, total\_amount,

RANK() OVER (ORDER BY total\_amount DESC) AS total\_order\_rank

FROM customer\_order\_totals

WHERE EXISTS (

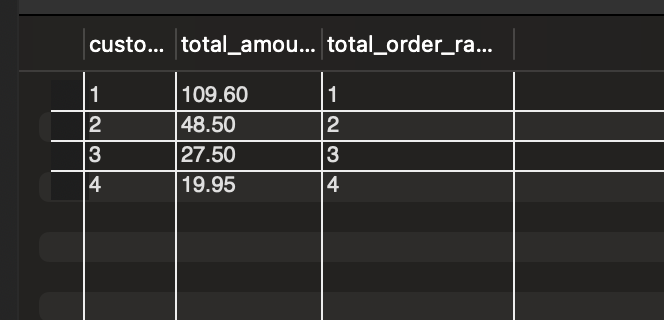
SELECT 1

FROM orders

WHERE customer\_id = customer\_order\_totals.customer\_id

AND order\_price > (SELECT AVG(order\_price) FROM orders)

);



* Performance tuning using query optimization techniques like query plans and indexes

-- Create a composite index on (order\_timestamp, customer\_id) columns in the orders table

CREATE INDEX idx\_order\_timestamp\_customer\_id ON orders(order\_timestamp, customer\_id);

-- Create a composite index on (customer\_id, chocolate\_id) columns in the orders table

CREATE INDEX idx\_customer\_id\_chocolate\_id ON orders(customer\_id, chocolate\_id);

* Implementing security measures such as roles and permissions to restrict access to sensitive data

– Error Code: 1410. You are not allowed to create a user with GRANT

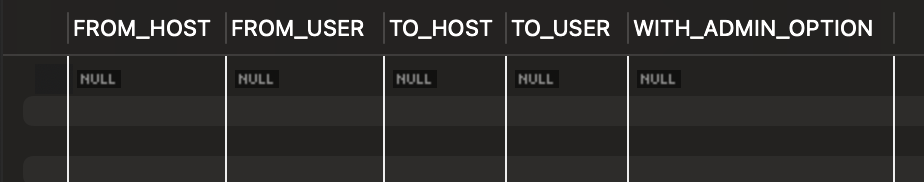
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';

-- Grant privileges to the user account

GRANT SELECT, INSERT, UPDATE ON chocolate.\* TO 'username'@'localhost';

GRANT role TO 'username'@'localhost';

SELECT \* FROM mysql.role\_edges;



SHOW GRANTS;

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'localhost' WITH GRANT OPTION;

– no dice

GRANT GRANT OPTION ON \*.\* TO 'root'@'localhost' WITH GRANT OPTION;

– no dice

Project Components:

1. **Database Design:**
   1. Design the database schema, including tables for chocolate, customers, and orders
   2. Define relationships between tables
   3. Implement constraints for data integrity
2. **Data Population:**
   1. Populate the database with sample data for chocolate, customers, and orders
   2. Ensure data consistency and referential integrity
3. **Basic Operations:**
   1. Implement CRUD operations for managing chocolate, customers, and orders
   2. Allow users to add, update and delete records from the database
4. **Intermediate Queries:**
   1. Create queries to retrieve information such as:
      1. Top-selling chocolate
      2. Customers with the highest order amounts
      3. Total revenue generated over a specific period
      4. Chocolate with low inventory levels
   2. Utilize JOIN operations to fetch data from multiple related tables
5. **Advanced Features:**
   1. Implement transactions to ensure atomicity and consistency during order processing
   2. Develop complex queries to answer specific business questions(e.g., identifying patterns in customer behavior)
   3. Optimize query performance by analyzing query execution plans and indexing strategies
   4. Implement role-based access control to restrict access to sensitive data and operations
6. **Documentation and Presentation:**
   1. Document the database schema, including tables, relationships, and constraints
   2. Provide explanations for the implemented features and query optimization techniques