Coding Challenge MySQL

**1) Total Sales Revenue by Product**

SELECT

p.id AS product\_id,

p.name AS product\_name,

SUM(oi.quantity \* oi.price) AS total\_revenue

FROM products p

JOIN order\_items oi ON p.id = oi.product\_id

JOIN orders o ON o.id = oi.order\_id

WHERE o.status = 'completed'

GROUP BY p.id, p.name

ORDER BY total\_revenue DESC;

---------------------------------------------------------------------------------------------------------

**2) Top Customers by Spending**

SELECT c.id AS customer\_id, c.name AS customer\_name, SUM(oi.quantity \* oi.price) AS total\_spending FROM customers c JOIN orders o ON c.id = o.customer\_id JOIN order\_items oi ON o.id = oi.order\_id WHERE o.status = 'completed' GROUP BY c.id, c.name ORDER BY total\_spending DESC LIMIT 5;

---------------------------------------------------------------------------------------------------------

**3) Average Order Value per Customer**

SELECT

c.id AS customer\_id,

c.name AS customer\_name,

ROUND(SUM(oi.quantity \* oi.price) / COUNT(DISTINCT o.id), 2) AS avg\_order\_value

FROM customers c

JOIN orders o

ON c.id = o.customer\_id

JOIN order\_items oi

ON o.id = oi.order\_id

GROUP BY c.id, c.name

HAVING COUNT(DISTINCT o.id) > 0

ORDER BY avg\_order\_value DESC;

**4) Recent Orders**

SELECT o.id AS order\_id, c.name AS customer\_name, o.order\_date, o.status FROM orders o

JOIN customers c ON o.customer\_id = c.id

WHERE o.status = 'completed' AND o.order\_date >= NOW() - INTERVAL 30 DAY ORDER BY o.order\_date DESC;

**5)** **Running Total of Customer Spending**

WITH order\_totals AS (

SELECT

o.id AS order\_id,

o.customer\_id,

o.order\_date,

SUM(oi.quantity \* oi.price) AS order\_total

FROM orders o

JOIN order\_items oi

ON o.id = oi.order\_id

GROUP BY o.id, o.customer\_id, o.order\_date

)

SELECT

ot.customer\_id,

ot.order\_id,

ot.order\_date,

ot.order\_total,

SUM(ot.order\_total) OVER (

PARTITION BY ot.customer\_id

ORDER BY ot.order\_date, ot.order\_id

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

) AS running\_total

FROM order\_totals ot

ORDER BY ot.customer\_id, ot.order\_date, ot.order\_id;

**6) Product Review Summary**

SELECT

p.id AS product\_id,

p.name AS product\_name,

ROUND(AVG(r.rating), 2) AS avg\_review\_rating,

COUNT(r.id) AS total\_reviews

FROM products p

LEFT JOIN reviews r

ON p.id = r.product\_id

GROUP BY p.id, p.name

ORDER BY avg\_review\_rating DESC NULLS LAST, total\_reviews DESC;

**7) Customers Without Orders**

SELECT c.id, c.name FROM customers c LEFT JOIN orders o ON c.id = o.customer\_id WHERE o.id IS NULL;

--------------------------------------------------------------------------------------------------------------

**8) Update Last Purchased Date**

UPDATE products p

JOIN (

SELECT

oi.product\_id,

MAX(o.order\_date) AS last\_purchase\_date

FROM order\_items oi

JOIN orders o

ON oi.order\_id = o.id

GROUP BY oi.product\_id

) t ON p.id = t.product\_id

SET p.last\_purchased = t.last\_purchase\_date;

**9) Transaction Scenario**

-- Start the transaction

START TRANSACTION;

-- 1. Insert a new order

INSERT INTO orders (customer\_id, order\_date, status)

VALUES (1, NOW(), 'pending');

-- Capture the new order\_id

SET @order\_id = LAST\_INSERT\_ID();

-- Example: Assume customer orders 2 products

-- Product 1: quantity = 3

-- Product 2: quantity = 1

-- 2. Insert order items

INSERT INTO order\_items (order\_id, product\_id, quantity, price)

VALUES

(@order\_id, 1, 3, 100.00), -- product\_id = 1, qty = 3

(@order\_id, 2, 1, 250.00); -- product\_id = 2, qty = 1

-- 3. Deduct stock for each product

UPDATE products

SET stock = stock - 3,

last\_purchased = NOW()

WHERE id = 1 AND stock >= 3;

UPDATE products

SET stock = stock - 1,

last\_purchased = NOW()

WHERE id = 2 AND stock >= 1;

-- 4. Check if stock deduction was valid (no negative stock)

-- If affected rows do not match, rollback

IF ROW\_COUNT() = 0 THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient stock';

END IF;

-- If all good, commit transaction

COMMIT;

**Q 10: Query Optimization and Indexing**

E.g take query of question no 11. By putting the explain key word before the query. It will return select type,table names including joined tables.keys, rows processed.

Index on customer\_id will make the join faster.

CREATE INDEX idx\_orders\_customer\_id ON orders(customer\_id);

**11) Query Optimization Challenge**

SELECT c.id AS customer\_id, c.name AS customer\_name, SUM(oi.quantity \* oi.price) AS total\_spent FROM customers c JOIN orders o ON c.id = o.customer\_id JOIN order\_items oi ON o.id = oi.order\_id WHERE o.status = 'completed' GROUP BY c.id, c.name