31 DAYS OF MYSQL

DATA December

31 Days. 31 MySQL Projects. One Skill Level Up

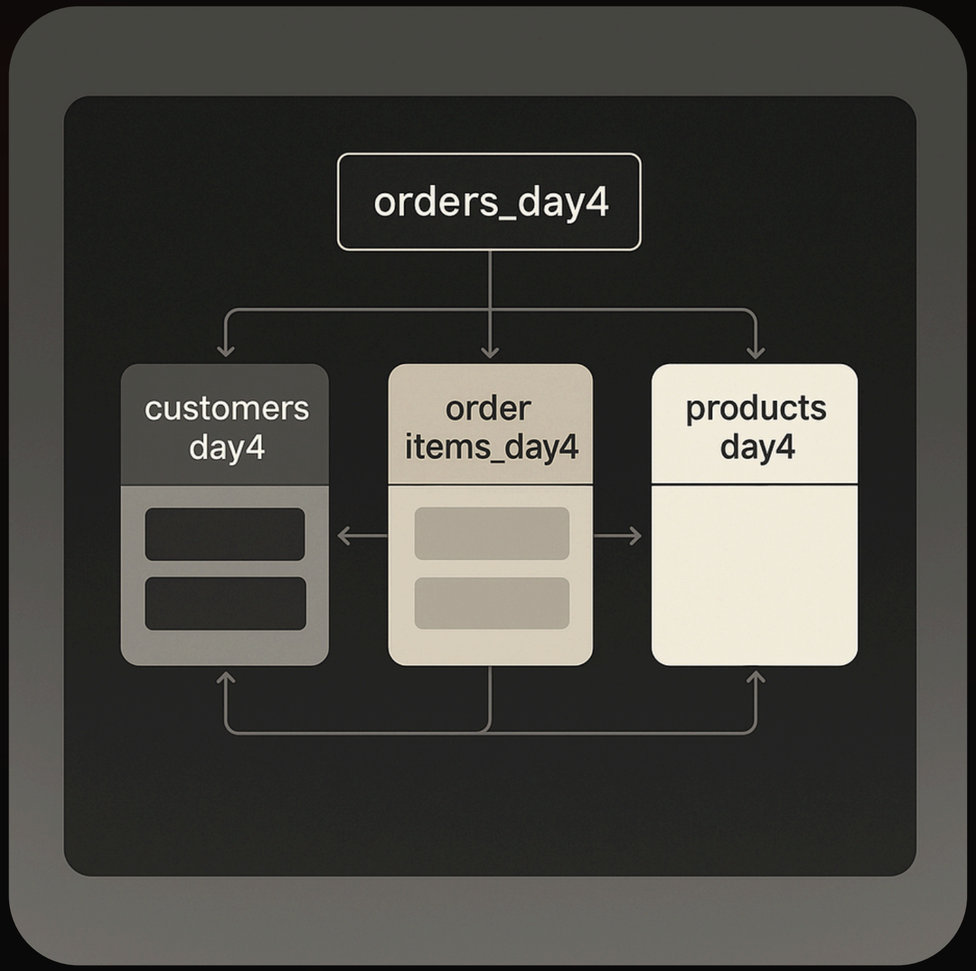
Day 4 — E-Commerce Project

PROJECT OVERVIEW

This project focused on analyzing a real-world E-commerce relational database consisting of customers, orders, order items, and product catalog tables.

The objective was to transform raw transactional data into meaningful business insights using SQL helping understand customer behavior, product performance, and sales trends across the platform.

DATABASE SCHEMA



customers_day4

- Order_id
- customer_name
- city
- State
- signup_year

order_items_day4

- item_id
- order_id
- product_id
- quantity
- price

orders_day4

- order_id
- customer_id
- order_date
- payment_method
- order_amount

Products_day4

- product_id
- product_id
- category
- brand

Query 1-2: High-Spending Customers & Premium-Priced Products

Query 1: Customers whose total spending > average spending

```
WITH customer_spend AS (
 SELECT
 o.customer_id,
 SUM(oi.price * oi.quantity) AS total_spend
 FROM orders_day4 o
 JOIN order_items_day4 oi ON o.order_id = oi.order_id
 GROUP BY o.customer_id
),
avg_spend AS (
 SELECT AVG(total_spend) AS avg_total_spend FROM
customer_spend
)
SELECT cs.customer_id, c.customer_name,
cs.total_spend
FROM customer_spend cs
JOIN avg_spend a ON 1=1
JOIN customers_day4 c ON c.customer_id =
cs.customer_id
WHERE cs.total_spend > a.avg_total_spend
ORDER BY cs.total_spend DESC;
```

Query 2: Products priced above average product price

```
SELECT
 p.product_id,
 p.product_name,
 AVG(oi.price) AS avg_price
FROM products_day4 p
JOIN order_items_day4 oi ON p.product_id =
oi.product_id
GROUP BY p.product_id, p.product_name
HAVING AVG(oi.price) > (SELECT AVG(price) FROM
order_items_day4);
```

Query 3-4-5: Order Size Insights & Early Customer Patterns

Query 3: Orders with more items than average items per order

```
SELECT
 oi.order_id, SUM(oi.quantity) AS
total_items FROM order_items_day4 oi
GROUP BY oi.order_id HAVING
SUM(oi.quantity) > (SELECT AVG(items)
FROM (SELECT SUM(quantity) AS items
FROM order_items_day4 GROUP BY order_id )
t);
```

Query 4: Customers who joined before the earliest order date

```
SELECT *
FROM customers_day4
WHERE signup_year < (
 SELECT MIN(YEAR(order_date)) FROM
orders_day4
);
```

Query 5: Customers who made more orders than average

```
SELECT customer_id, customer_name, orders_count
FROM (SELECT o.customer_id, c.customer_name, COUNT(*) AS orders_count
FROM orders_day4 o JOIN customers_day4 c ON c.customer_id = o.customer_id
GROUP BY o.customer_id, c.customer_name) t
WHERE orders_count > (SELECT AVG(cnt) FROM (
SELECT COUNT(*) AS cnt
FROM orders_day4
GROUP BY customer_id) x)
ORDER BY orders_count DESC;
```

Query 6-7: Inactive Customers & Electronics-Based Orders

Q6: Customers who have never placed an order

```
SELECT *
FROM customers_day4 c
WHERE NOT EXISTS (
 SELECT 1 FROM orders_day4 o WHERE o.customer_id =
c.customer_id
);
```

Q7: Orders that contain at least one Electronics product

```
SELECT DISTINCT o.order_id
FROM orders_day4 o
JOIN order_items_day4 oi ON o.order_id = oi.order_id
JOIN products_day4 p ON oi.product_id = p.product_id
WHERE p.category = 'Electronics';
```

Query 8-9: Unpurchased Products & Order-Level Summaries

1

Q8: Products that were never purchased

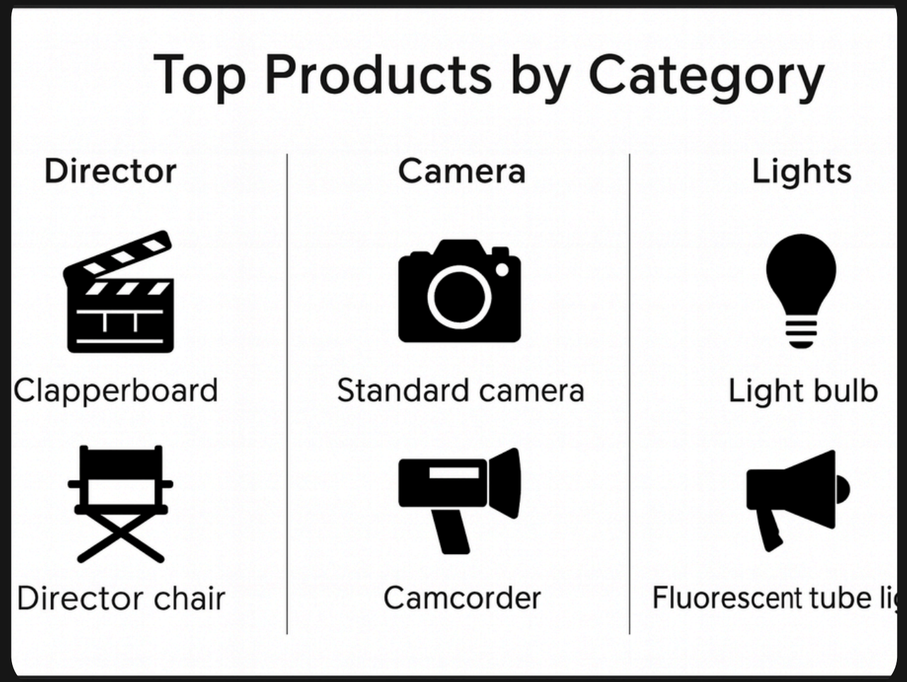
```
select m.movie_id, m.title, count(distinct
a.actor_id)
from imdb_movies m
join imdb_cast c
on m.movie_id = c.movie_id
join imdb_actors a
on c.actor_id = a.actor_id
group by m.movie_id, m.title;
```

2

Q9: Show each order with: Customer name, total item, total amount

```
SELECT
 o.order_id, c.customer_name, SUM(oi.quantity) AS total_items, SUM(oi.price * oi.quantity)
AS total_amount
FROM orders_day4 o JOIN customers_day4 c ON o.customer_id = c.customer_id
JOIN order_items_day4 oi ON o.order_id = oi.order_id GROUP BY o.order_id, c.customer_name
ORDER BY total_amount DESC;
```

Query 10-11: Category Leaders & Diverse Shoppers



Query 10: For each category, list top 3 most frequently purchased products

```
WITH prod_sales AS (
 SELECT p.category, p.product_id, p.product_name,
SUM(oi.quantity) AS qty_sold
 FROM products_day4 p JOIN order_items_day4 oi ON
p.product_id = oi.product_id GROUP BY p.category,
p.product_id, p.product_name)
SELECT category, product_id, product_name, qty_sold
FROM (SELECT ps.*, ROW_NUMBER() OVER (PARTITION BY
ps.category ORDER BY ps.qty_sold DESC) AS rn
 FROM prod_sales ps) t
WHERE rn <= 3
ORDER BY category, qty_sold DESC;
```

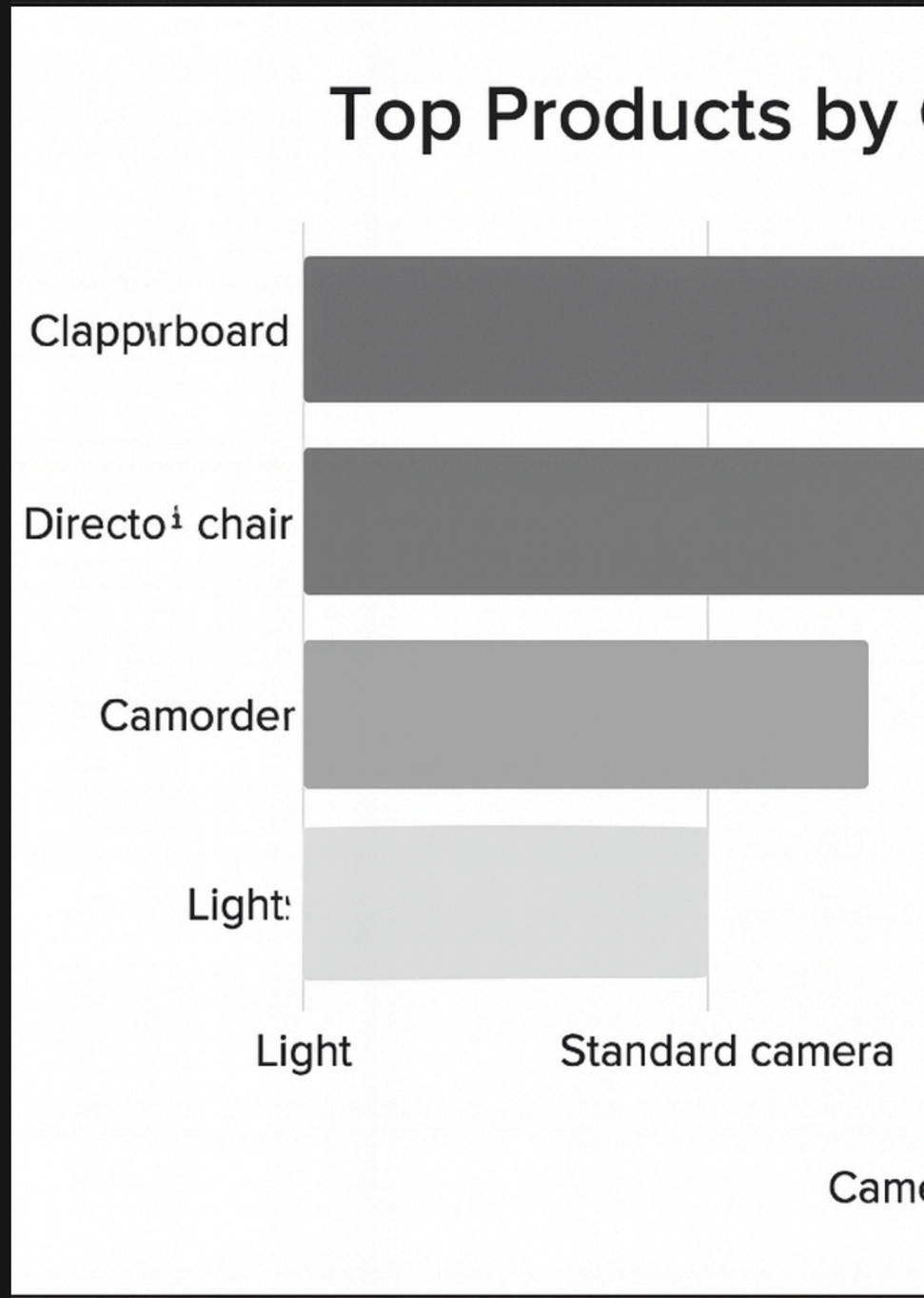
Query 11: Customers who purchased items from 5+ different categories

```
SELECT
 c.customer_id,
 c.customer_name,
 COUNT(DISTINCT p.category) AS categories_bought
FROM customers_day4 c
JOIN orders_day4 o ON c.customer_id = o.customer_id
JOIN order_items_day4 oi ON o.order_id = oi.order_id
JOIN products_day4 p ON oi.product_id = p.product_id
GROUP BY c.customer_id, c.customer_name
HAVING COUNT(DISTINCT p.category) >= 5
ORDER BY categories_bought DESC;
```

Query 12: For each customer, calculate:

First Purchase date, Last Purchase date, Gap in days

```
SELECT
 c.customer_id,
 c.customer_name,
 MIN(o.order_date) AS first_purchase,
 MAX(o.order_date) AS last_purchase,
 DATEDIFF(MAX(o.order_date), MIN(o.order_date)) AS gap_days
FROM customers_day4 c
JOIN orders_day4 o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.customer_name
ORDER BY gap_days DESC;
```



Key Skills Used

- Advanced SQL joins, aggregations, and category-based analysis across customers, orders, items, and products.
- Subqueries, NOT EXISTS logic, and temporal filtering to uncover customer behavior and product performance insights.

CONCLUSION

This project provided deep insights into customer behavior, product performance, and order patterns using real-world SQL techniques. Through advanced joins, aggregations, and filtering logic, we uncovered meaningful trends that support data-driven decision-making in an e-commerce environment.

31 DAYS OF MYSQL, DATA DECEMBER

THANK *You!*

Connect: https://www.linkedin.com/in/altaf-khan-093730245
GitHub: https://github.com/Altaf0099

✳ Day 4 — E-Commerce Project ✳