SQL Analytics Projects

MavenMovies & ClassicModels Databases

Author: Vishnu Vardhan Reddy Kukudala

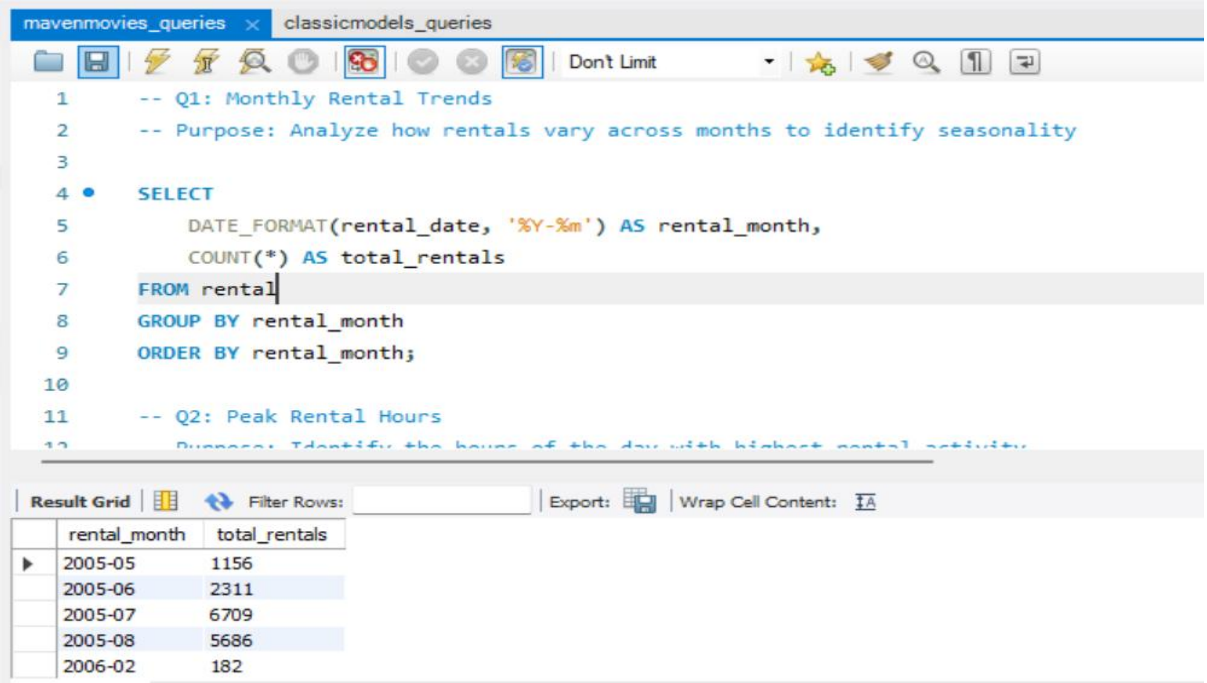
Role: Aspiring Data Scientist / Data Analyst

Section 1: SQL Analysis on MavenMovies Database

Q1. Monthly Rental Trends

Objective & Explanation:

This analysis examines how movie rentals vary across different months to identify seasonal demand patterns. By aggregating rentals at the monthly level, we can observe peaks and dips in customer activity over time. These trends help management understand high-demand periods and plan inventory, promotions, and staffing accordingly.



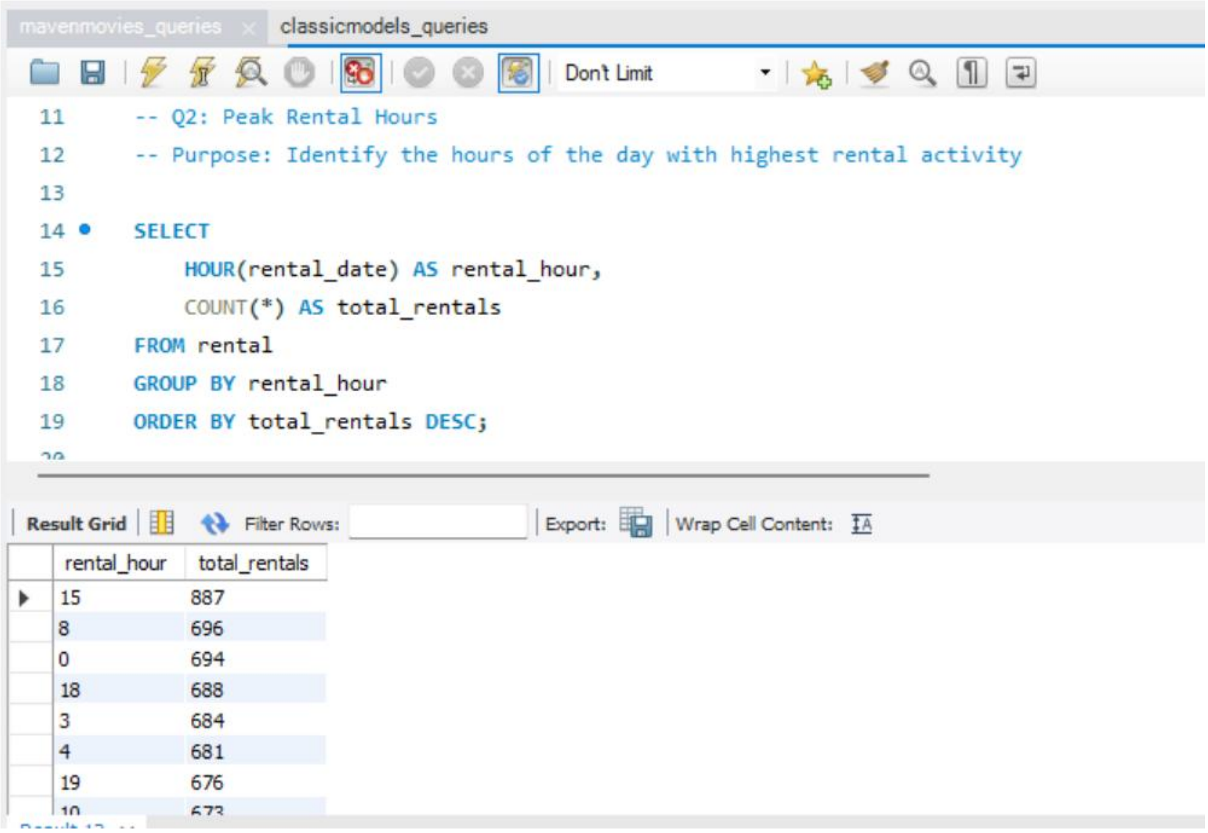
```
-- Q1: Monthly Rental Trends
-- Purpose: Analyze how rentals vary across months to identify seasonality

SELECT
    DATE_FORMAT(rental_date, '%Y-%m') AS rental_month,
    COUNT(*) AS total_rentals
FROM rental
GROUP BY rental_month
ORDER BY rental_month;

-- Q2: Peak Rental Hours
   Purpose: Identify the hours of the day with highest rental activity
```

rental_month	total_rentals
2005-05	1156
2005-06	2311
2005-07	6709
2005-08	5686
2006-02	182

Q2. Peak Rental Hours

Objective & Explanation:

This query identifies the hours of the day during which rental activity is highest. Understanding peak rental hours allows the business to allocate staff more efficiently and ensure smooth operations during busy periods. It also provides insights into customer behaviour and preferred rental times.



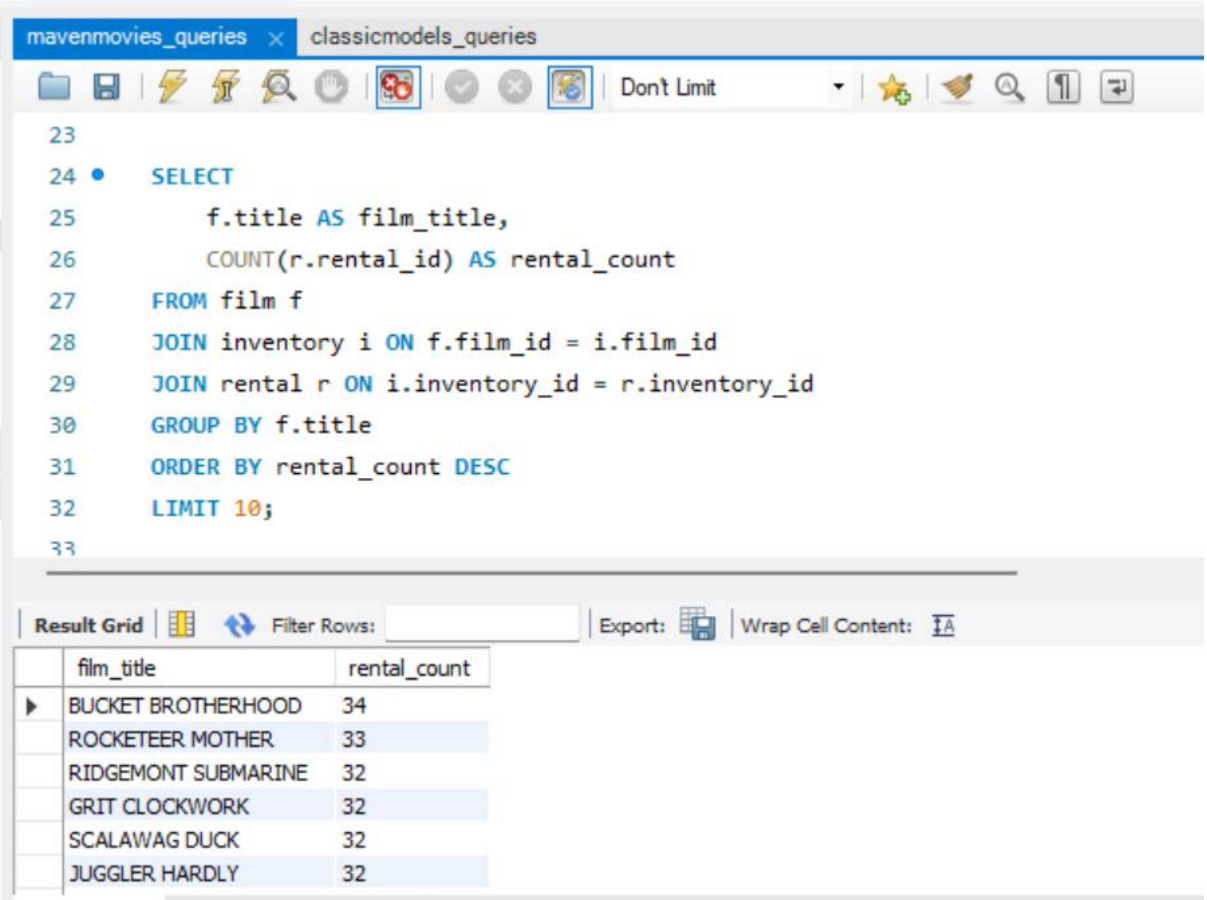
```sql
-- Q2: Peak Rental Hours
-- Purpose: Identify the hours of the day with highest rental activity

SELECT
    HOUR(rental_date) AS rental_hour,
    COUNT(*) AS total_rentals
FROM rental
GROUP BY rental_hour
ORDER BY total_rentals DESC;
```

rental_hour	total_rentals
15	887
8	696
0	694
18	688
3	684
4	681
19	676
10	673

Q3. Top 10 Most Rented Films

Objective & Explanation:

This analysis identifies the most frequently rented films based on total rental count. Highlighting popular titles helps in optimizing inventory decisions and ensuring high-demand movies are always available. It also provides insights into customer preferences and content popularity.



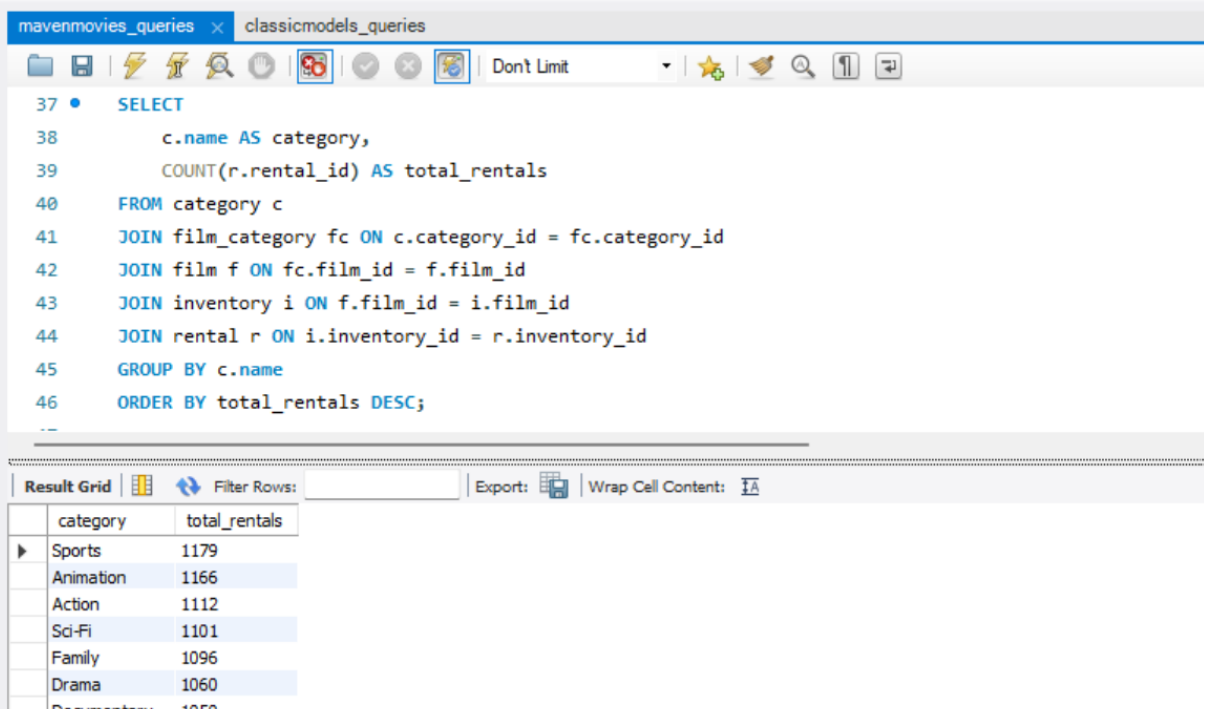
```sql

SELECT
    f.title AS film_title,
    COUNT(r.rental_id) AS rental_count
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY f.title
ORDER BY rental_count DESC
LIMIT 10;

```

film_title	rental_count
BUCKET BROTHERHOOD	34
ROCKETEER MOTHER	33
RIDGEMONT SUBMARINE	32
GRIT CLOCKWORK	32
SCALAWAG DUCK	32
JUGGLER HARDLY	32

Q4. Film Categories with Highest Rentals

Objective & Explanation:

This query evaluates rental demand across different film categories to determine which genres are most popular. Understanding category-level demand helps in targeted content acquisition and marketing strategies. It also supports data-driven decisions related to genre-focused promotions.



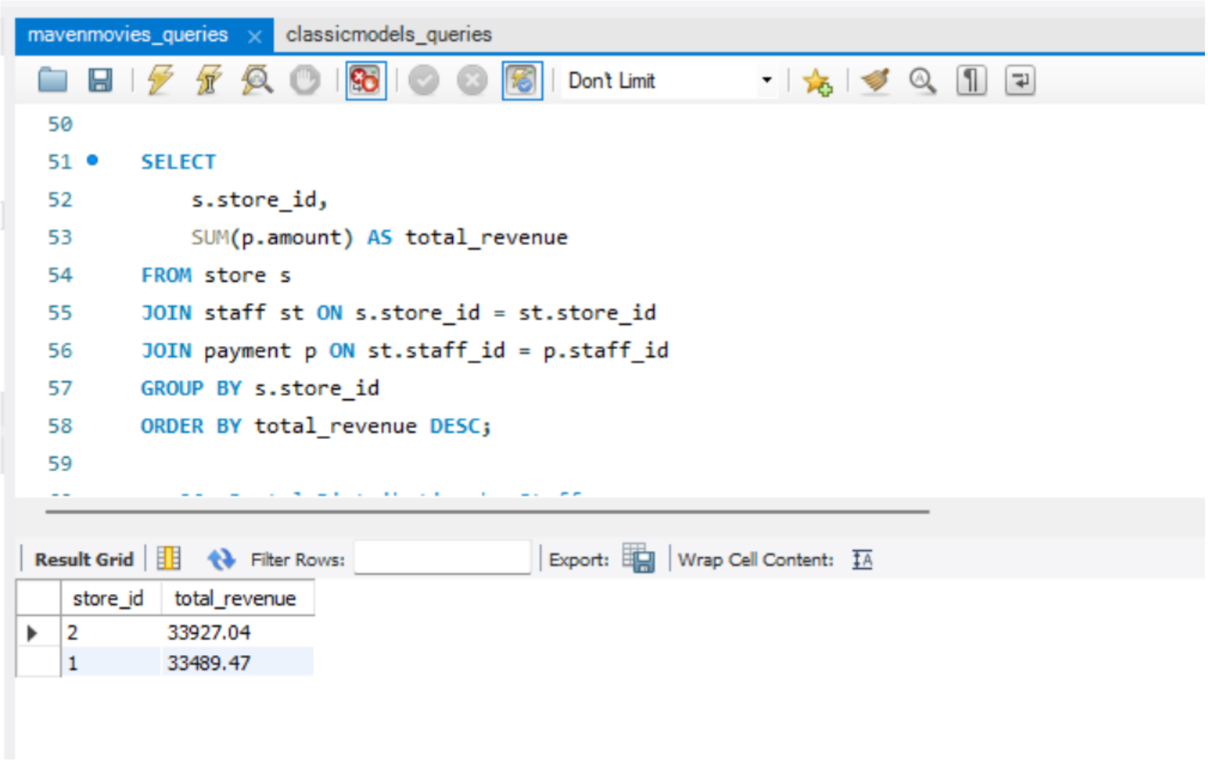
```sql
SELECT
    c.name AS category,
    COUNT(r.rental_id) AS total_rentals
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
JOIN film f ON fc.film_id = f.film_id
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY c.name
ORDER BY total_rentals DESC;
```

category	total_rentals
Sports	1179
Animation	1166
Action	1112
Sci-Fi	1101
Family	1096
Drama	1060
Documentary	1050

Q5. Store with Highest Rental Revenue

Objective & Explanation:

This analysis calculates total rental revenue generated by each store to identify the top-performing location. Comparing store-level revenue helps management evaluate operational efficiency and profitability. These insights can guide strategic decisions such as resource allocation and performance benchmarking.



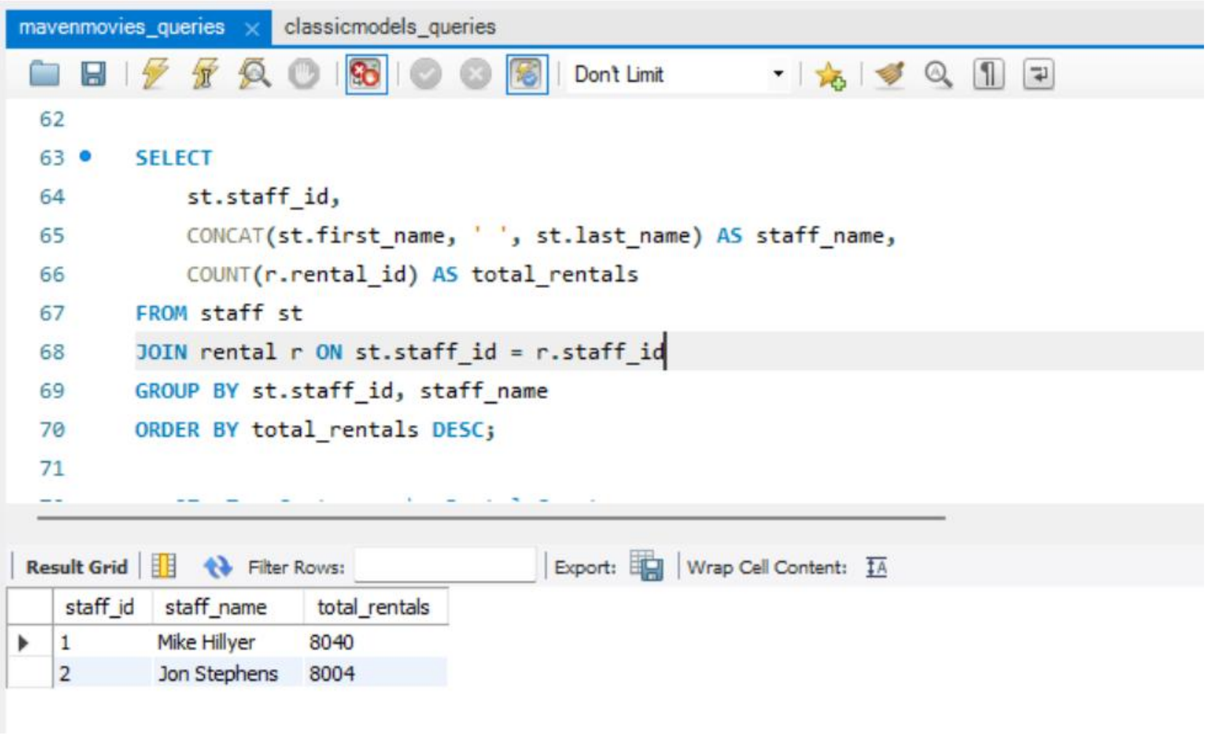
```sql
SELECT
    s.store_id,
    SUM(p.amount) AS total_revenue
FROM store s
JOIN staff st ON s.store_id = st.store_id
JOIN payment p ON st.staff_id = p.staff_id
GROUP BY s.store_id
ORDER BY total_revenue DESC;
```

store_id	total_revenue
2	33927.04
1	33489.47

Q6. Rental Distribution by Staff Members

Objective & Explanation:

This query measures staff performance by analyzing the number of rentals processed by each staff member. It helps identify high-performing employees and assess workload distribution. Such insights can support staffing decisions, training needs, and performance evaluations.



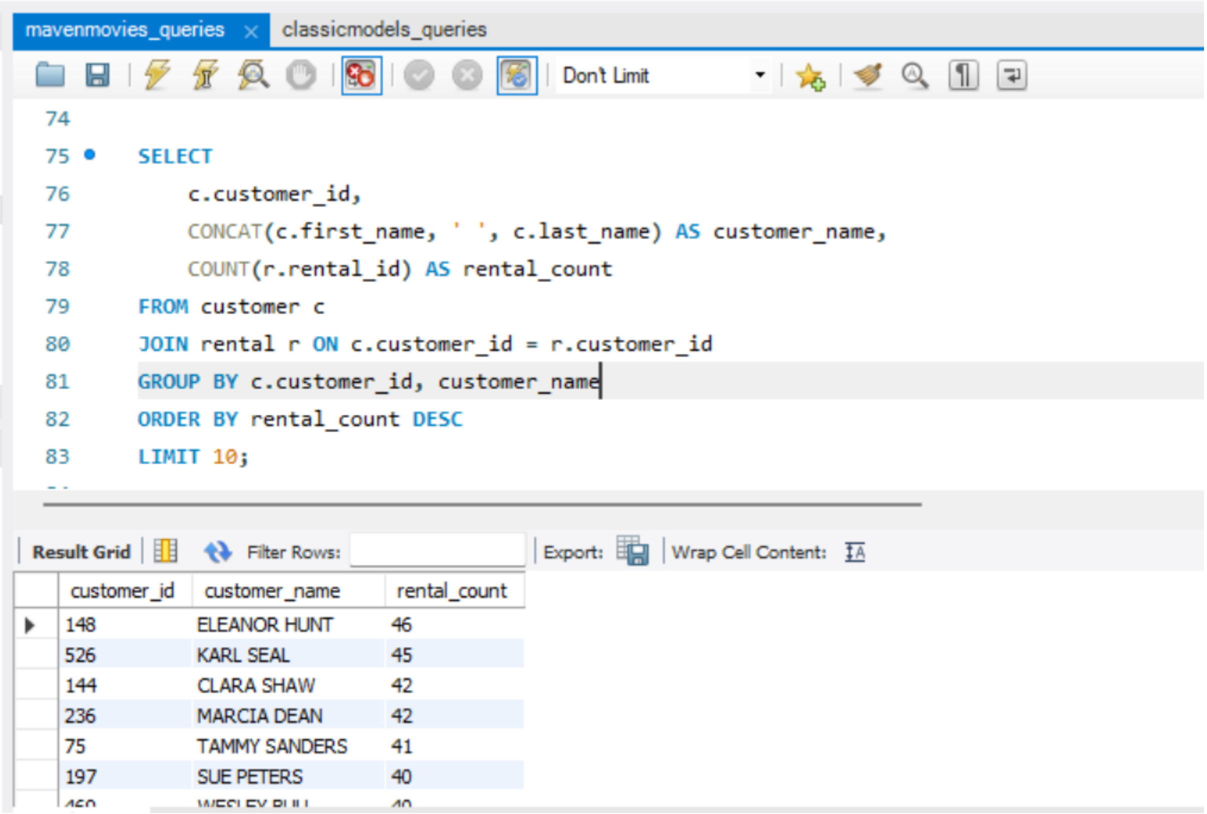
```sql
SELECT
    st.staff_id,
    CONCAT(st.first_name, ' ', st.last_name) AS staff_name,
    COUNT(r.rental_id) AS total_rentals
FROM staff st
JOIN rental r ON st.staff_id = r.staff_id
GROUP BY st.staff_id, staff_name
ORDER BY total_rentals DESC;
```

staff_id	staff_name	total_rentals
1	Mike Hillyer	8040
2	Jon Stephens	8004

Q7. Top Customers by Rental Count

Objective & Explanation:

This analysis identifies the most loyal customers based on their rental frequency. Recognizing high-engagement customers enables the business to design loyalty programs and personalized offers. It also helps in understanding customer retention and long-term value.



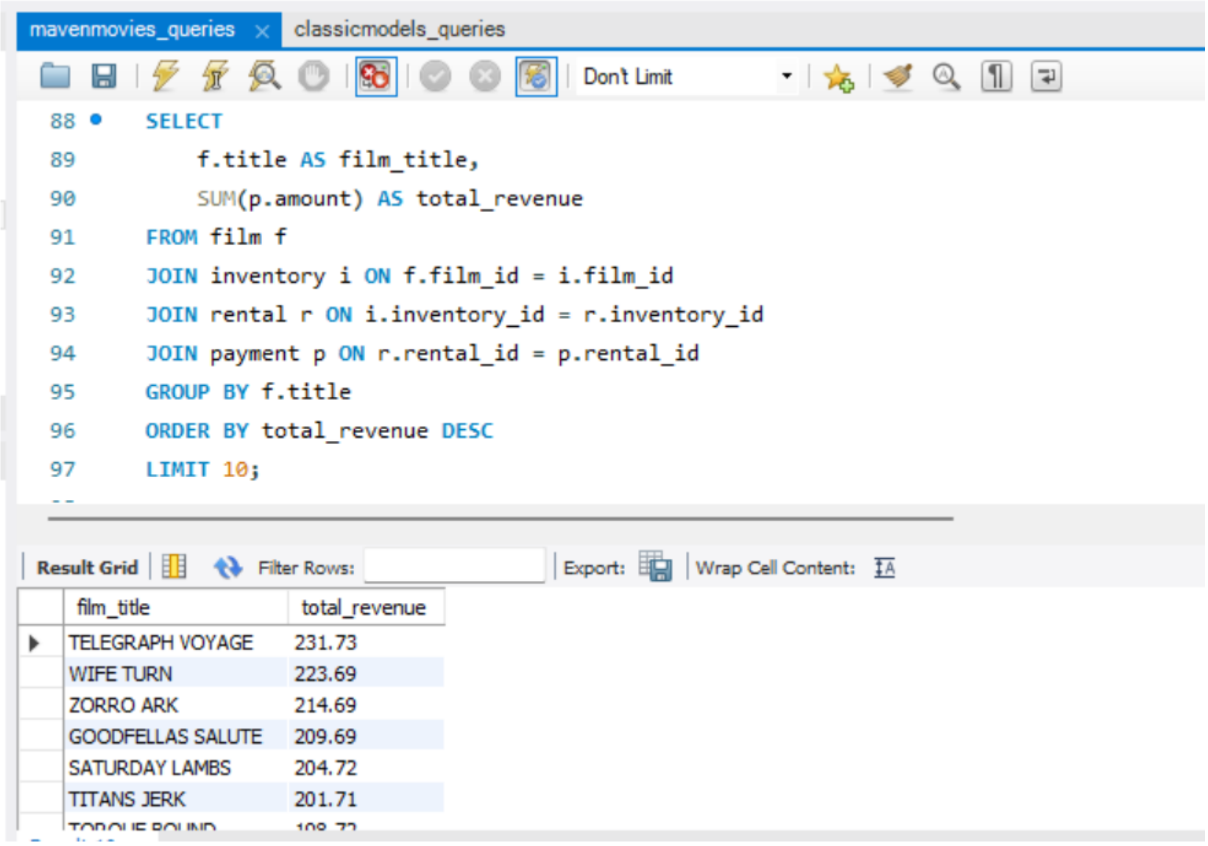
```
SELECT
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    COUNT(r.rental_id) AS rental_count
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
GROUP BY c.customer_id, customer_name
ORDER BY rental_count DESC
LIMIT 10;
```

customer_id	customer_name	rental_count
148	ELEANOR HUNT	46
526	KARL SEAL	45
144	CLARA SHAW	42
236	MARCIA DEAN	42
75	TAMMY SANDERS	41
197	SUE PETERS	40
459	WESLEY BULL	40

Q8. Films Generating the Highest Revenue

Objective & Explanation:

This query identifies films that contribute the most to overall revenue, not just rental volume. It highlights cases where fewer rentals still generate higher income due to pricing factors. These insights help in pricing strategy evaluation and revenue optimization.



```sql
SELECT
    f.title AS film_title,
    SUM(p.amount) AS total_revenue
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
JOIN payment p ON r.rental_id = p.rental_id
GROUP BY f.title
ORDER BY total_revenue DESC
LIMIT 10;
```

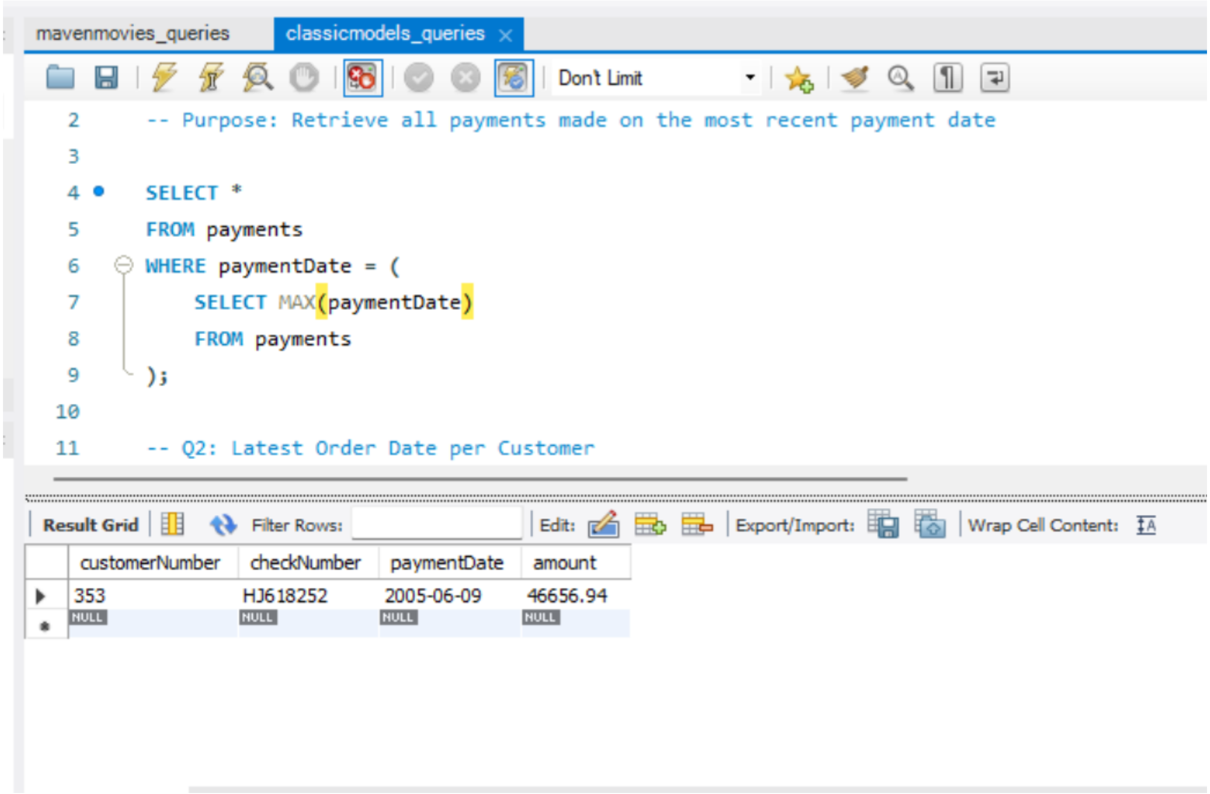
film_title	total_revenue
TELEGRAPH VOYAGE	231.73
WIFE TURN	223.69
ZORRO ARK	214.69
GOODFELLAS SALUTE	209.69
SATURDAY LAMBS	204.72
TITANS JERK	201.71
TORQUE BOUND	198.72

Section 2: Advanced SQL Analytics (ClassicModels)

Q1. Payments on the Latest Payment Date

Objective & Explanation:

This analysis retrieves all payments that occurred on the most recent payment date in the database. It helps identify the latest financial activity and ensures accurate reporting of recent transactions. Such queries are useful for end-of-day or period-close financial analysis.



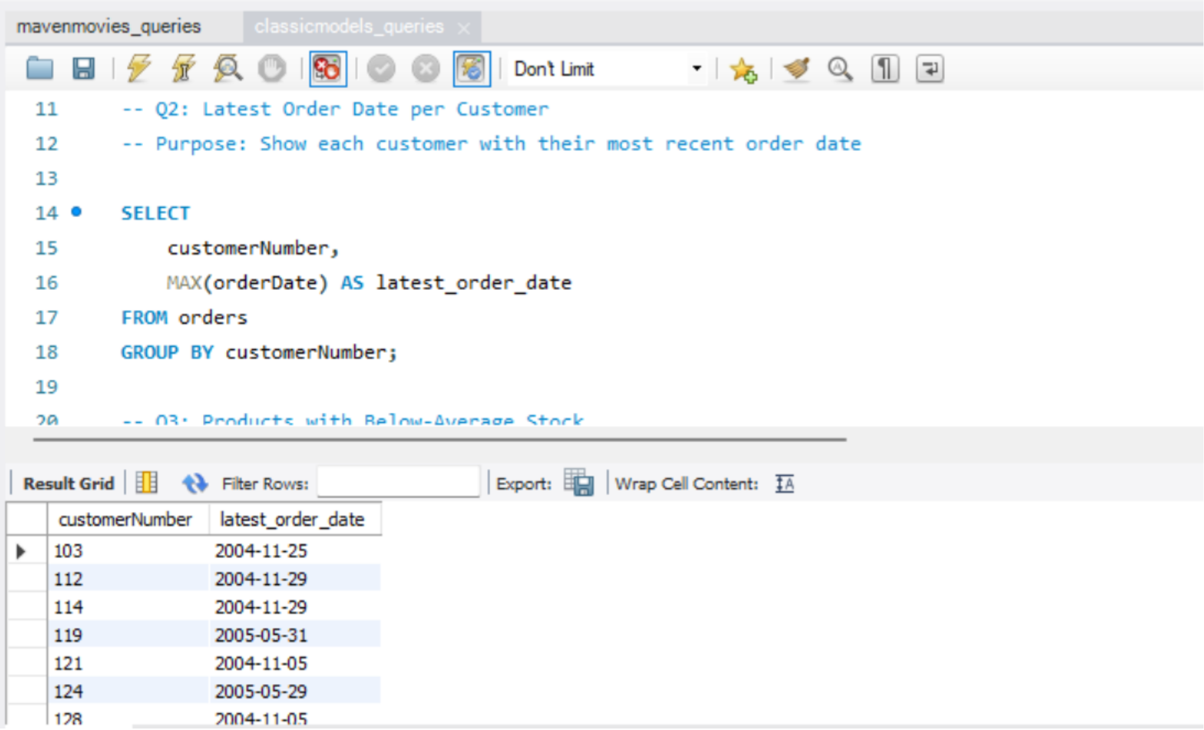
```sql
-- Purpose: Retrieve all payments made on the most recent payment date

SELECT *
FROM payments
WHERE paymentDate = (
    SELECT MAX(paymentDate)
    FROM payments
);

-- Q2: Latest Order Date per Customer
```

customerNumber	checkNumber	paymentDate	amount
353	HJ618252	2005-06-09	46656.94
NULL	NULL	NULL	NULL

Q2. Latest Order Date per Customer

Objective & Explanation:

This query identifies the most recent order placed by each customer. It provides insights into customer activity and engagement levels. Understanding recent customer interactions helps in targeting active customers and identifying inactive ones.



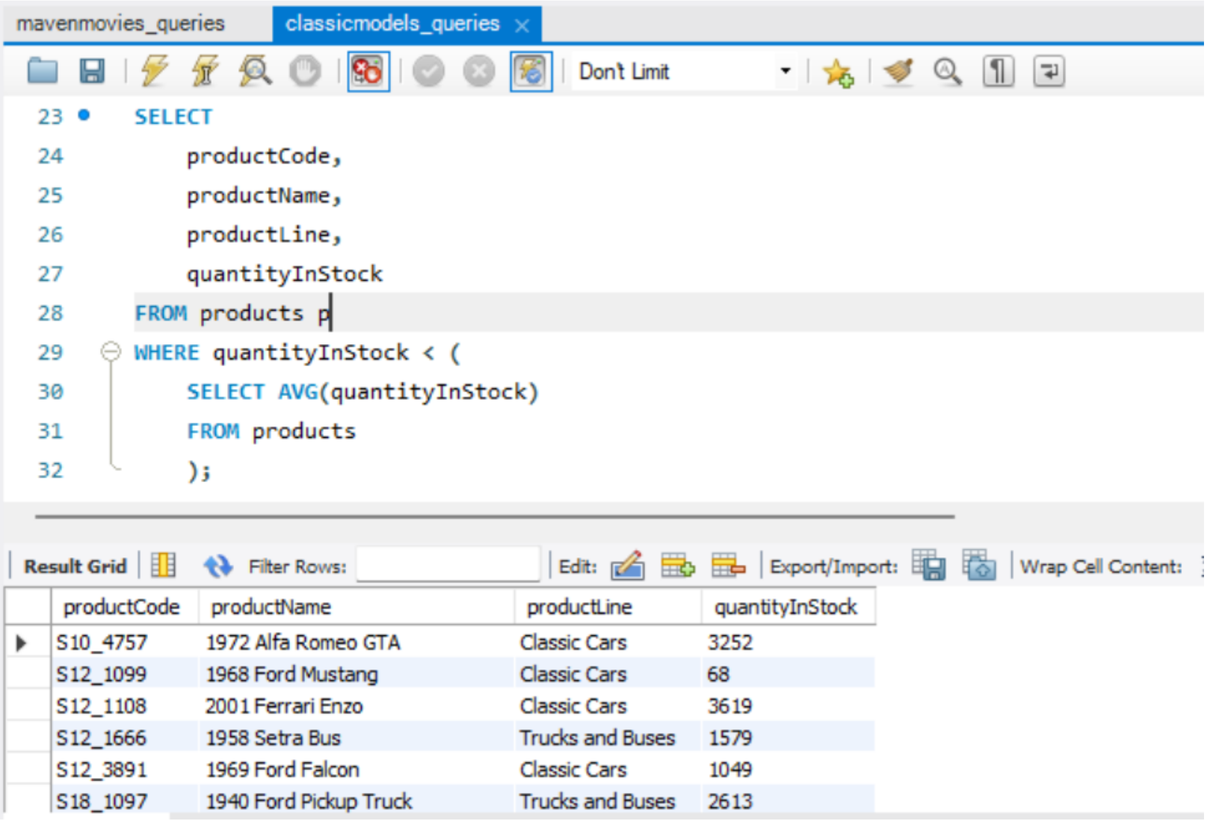
```sql
-- Q2: Latest Order Date per Customer
-- Purpose: Show each customer with their most recent order date

SELECT
    customerNumber,
    MAX(orderDate) AS latest_order_date
FROM orders
GROUP BY customerNumber;

-- Q3: Products with Below-Average Stock
```

customerNumber	latest_order_date
103	2004-11-25
112	2004-11-29
114	2004-11-29
119	2005-05-31
121	2004-11-05
124	2005-05-29
128	2004-11-05

Q3. Products with Below-Average Stock in Their Product Line

Objective & Explanation:

This analysis compares product stock levels against the average stock within the same product line. It helps identify items that may be at risk of stockouts relative to similar products. These insights support inventory planning and replenishment decisions.



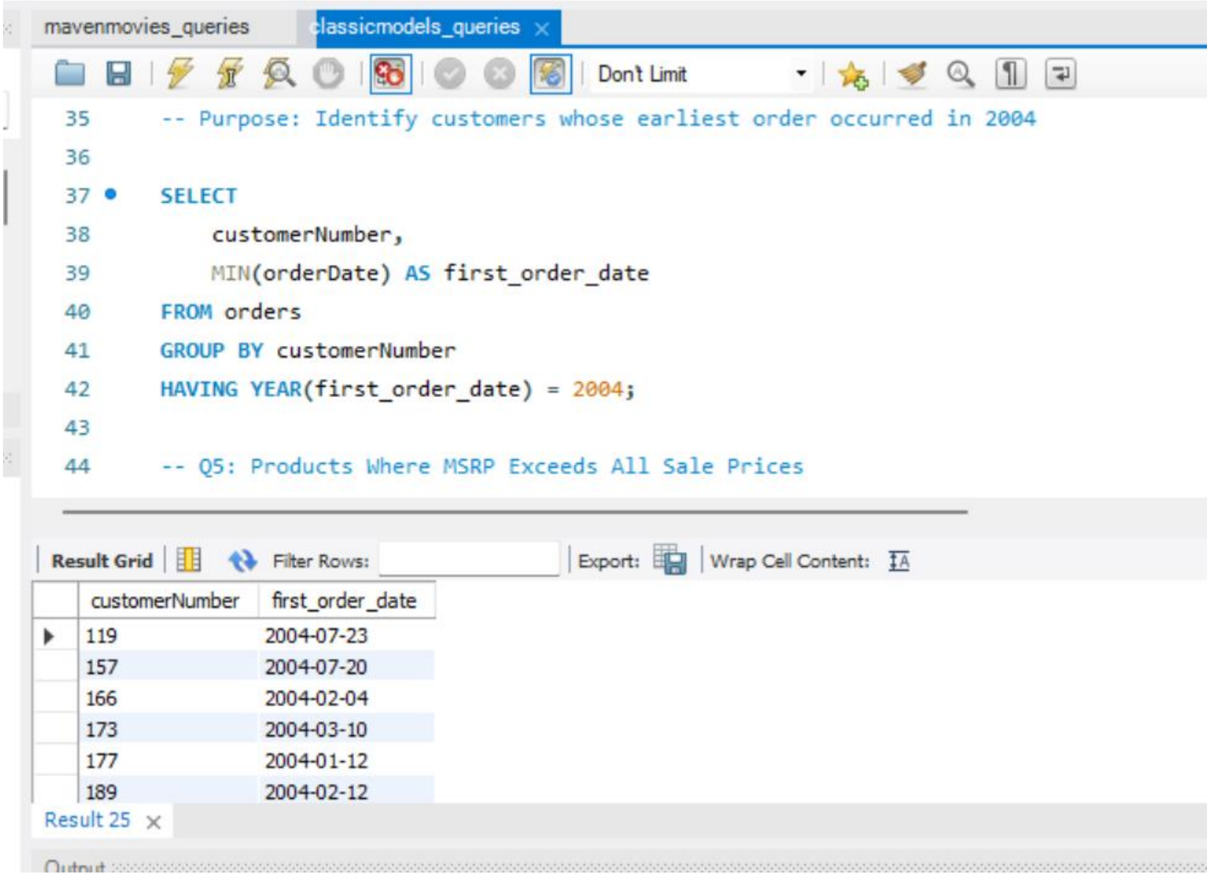
```sql
SELECT
    productCode,
    productName,
    productLine,
    quantityInStock
FROM products p
WHERE quantityInStock < (
    SELECT AVG(quantityInStock)
    FROM products
    );
```

productCode	productName	productLine	quantityInStock
S10_4757	1972 Alfa Romeo GTA	Classic Cars	3252
S12_1099	1968 Ford Mustang	Classic Cars	68
S12_1108	2001 Ferrari Enzo	Classic Cars	3619
S12_1666	1958 Setra Bus	Trucks and Buses	1579
S12_3891	1969 Ford Falcon	Classic Cars	1049
S18_1097	1940 Ford Pickup Truck	Trucks and Buses	2613

Q4. Customers Whose First Order Happened in 2004

Objective & Explanation:

This query identifies customers whose earliest recorded order occurred in the year 2004. It helps analyze customer acquisition during a specific time period. Such insights are useful for cohort analysis and long-term customer behavior studies.



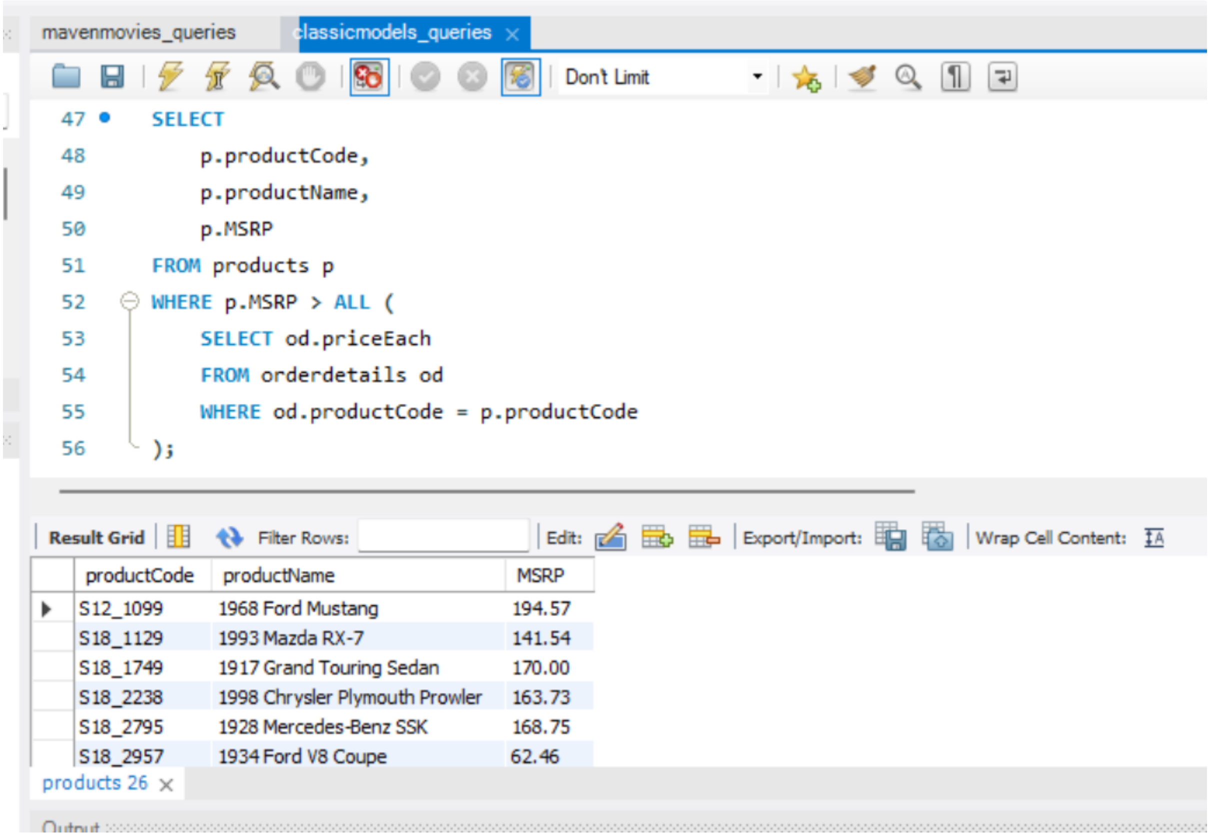
```
-- Purpose: Identify customers whose earliest order occurred in 2004

SELECT
    customerNumber,
    MIN(orderDate) AS first_order_date
FROM orders
GROUP BY customerNumber
HAVING YEAR(first_order_date) = 2004;

-- Q5: Products Where MSRP Exceeds All Sale Prices
```

customerNumber	first_order_date
119	2004-07-23
157	2004-07-20
166	2004-02-04
173	2004-03-10
177	2004-01-12
189	2004-02-12

Q5. Products Whose MSRP Exceeds All Historical Sale Prices

Objective & Explanation:

This analysis identifies products where the listed MSRP is higher than any price at which the product has ever been sold. It highlights pricing discrepancies and potential overpricing issues. These insights can guide pricing strategy reviews and discount policies.



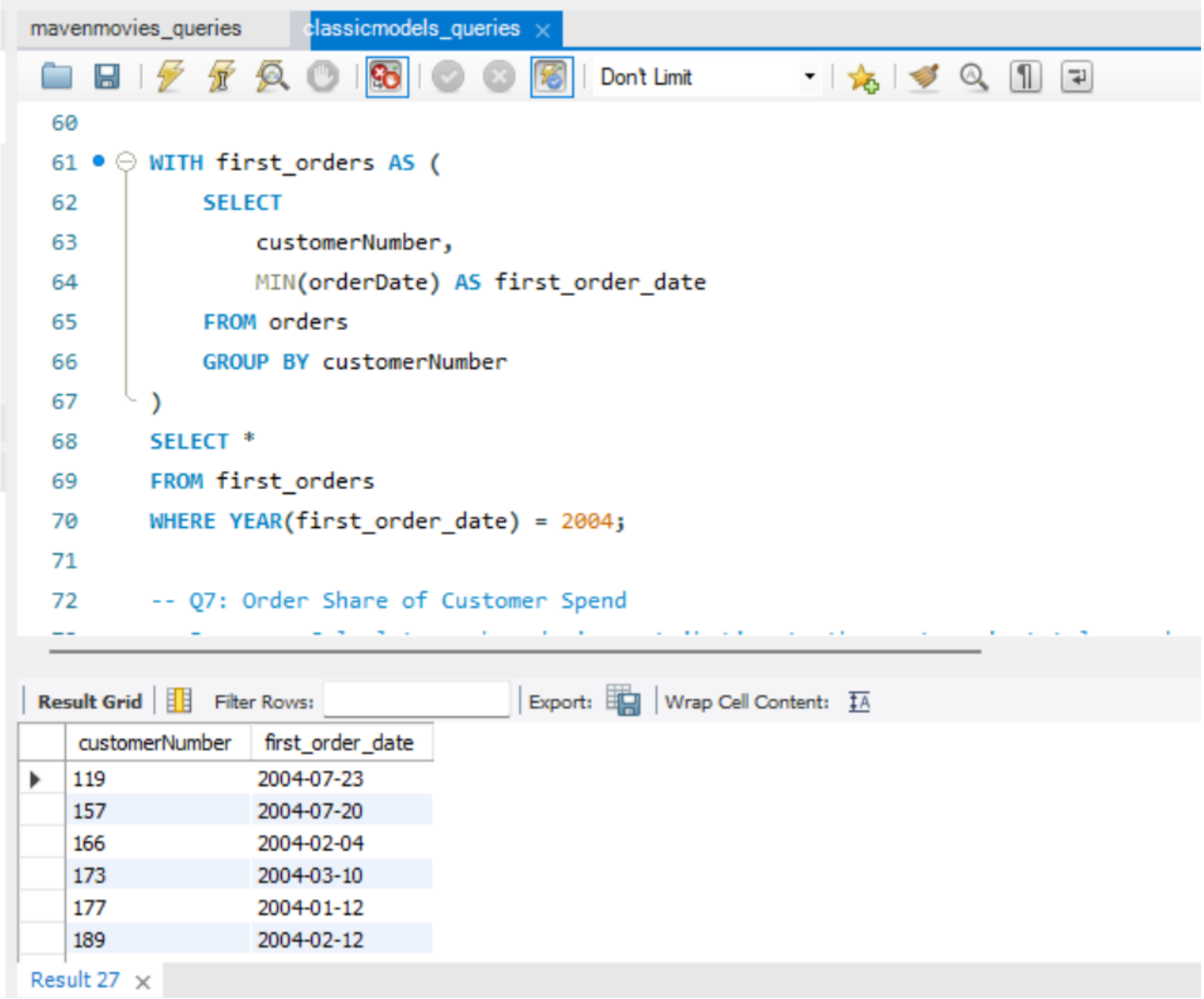
```
SELECT
    p.productCode,
    p.productName,
    p.MSRP
FROM products p
WHERE p.MSRP > ALL (
    SELECT od.priceEach
    FROM orderdetails od
    WHERE od.productCode = p.productCode
);
```

productCode	productName	MSRP
S12_1099	1968 Ford Mustang	194.57
S18_1129	1993 Mazda RX-7	141.54
S18_1749	1917 Grand Touring Sedan	170.00
S18_2238	1998 Chrysler Plymouth Prowler	163.73
S18_2795	1928 Mercedes-Benz SSK	168.75
S18_2957	1934 Ford V8 Coupe	62.46

Q6. First Order Date per Customer Using CTE

Objective & Explanation:

This query uses a Common Table Expression (CTE) to calculate each customer’s first order date in a structured and readable manner. It demonstrates modular query design and improves clarity when performing multi-step analysis. This approach is useful for scalable and maintainable SQL development.



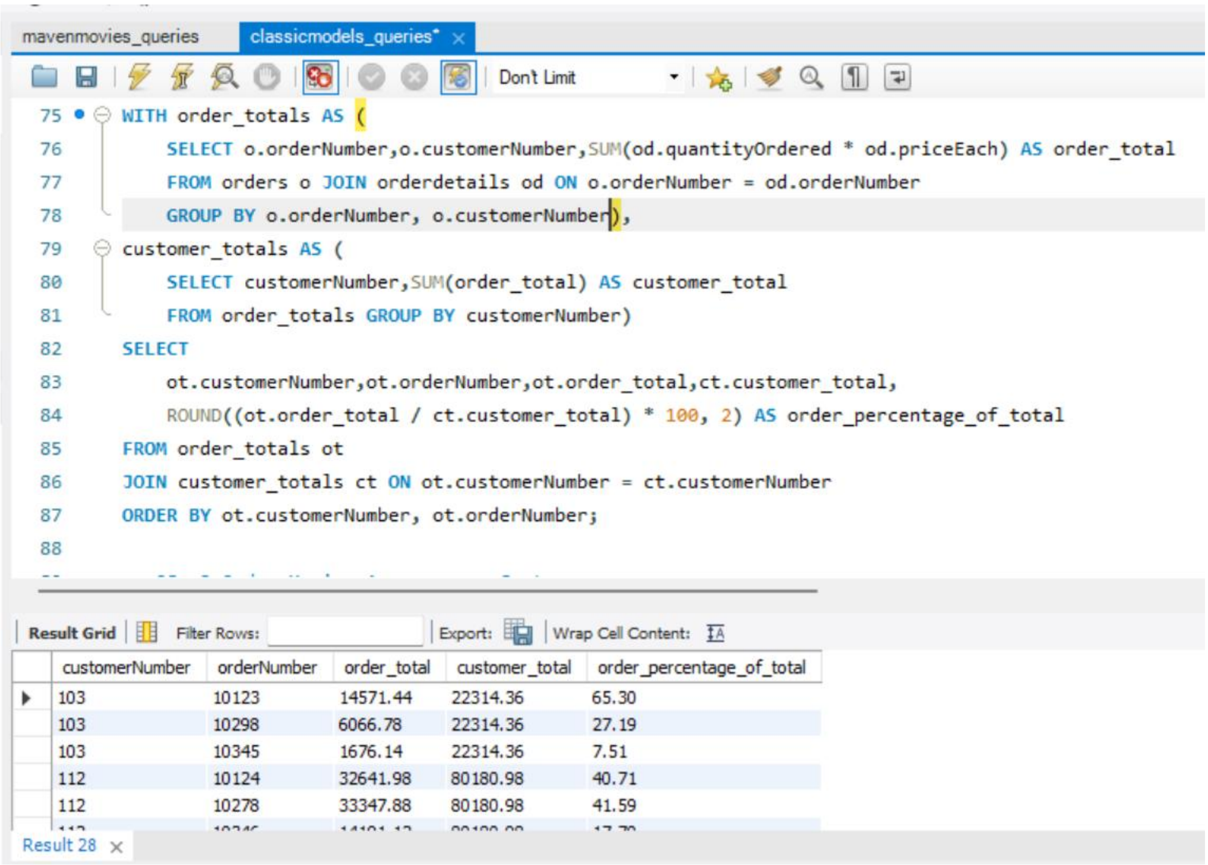
```sql
WITH first_orders AS (
    SELECT
        customerNumber,
        MIN(orderDate) AS first_order_date
    FROM orders
    GROUP BY customerNumber
)
SELECT *
FROM first_orders
WHERE YEAR(first_order_date) = 2004;

-- Q7: Order Share of Customer Spend
```

customerNumber	first_order_date
119	2004-07-23
157	2004-07-20
166	2004-02-04
173	2004-03-10
177	2004-01-12
189	2004-02-12

Q7. Share of Customer Spend by Order (Percentage of Total)

Objective & Explanation:

This analysis calculates each order’s contribution to the customer’s total spending. It helps identify high-impact orders and spending patterns within individual customers. Such insights are valuable for understanding purchase behavior and revenue concentration.



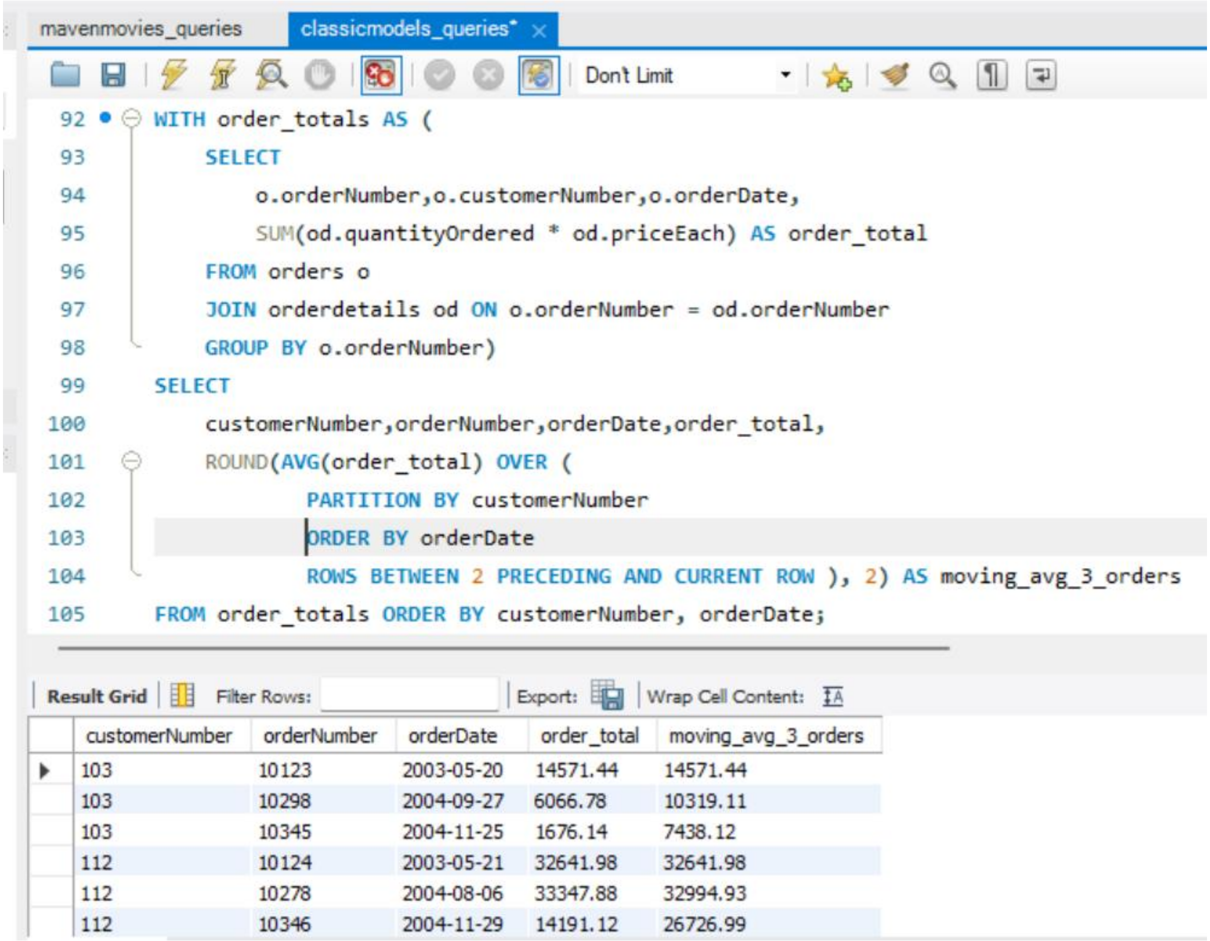
```sql
WITH order_totals AS (
    SELECT o.orderNumber,o.customerNumber,SUM(od.quantityOrdered * od.priceEach) AS order_total
    FROM orders o JOIN orderdetails od ON o.orderNumber = od.orderNumber
    GROUP BY o.orderNumber, o.customerNumber),
customer_totals AS (
    SELECT customerNumber,SUM(order_total) AS customer_total
    FROM order_totals GROUP BY customerNumber)
SELECT
    ot.customerNumber,ot.orderNumber,ot.order_total,ct.customer_total,
    ROUND((ot.order_total / ct.customer_total) * 100, 2) AS order_percentage_of_total
FROM order_totals ot
JOIN customer_totals ct ON ot.customerNumber = ct.customerNumber
ORDER BY ot.customerNumber, ot.orderNumber;
```

customerNumber	orderNumber	order_total	customer_total	order_percentage_of_total
103	10123	14571.44	22314.36	65.30
103	10298	6066.78	22314.36	27.19
103	10345	1676.14	22314.36	7.51
112	10124	32641.98	80180.98	40.71
112	10278	33347.88	80180.98	41.59

Q8. Three-Order Moving Average of Order Value per Customer

Objective & Explanation:

This query computes a rolling average of order values over the current and previous two orders for each customer. It smooths short-term fluctuations and reveals spending trends over time. Moving averages are commonly used in time-series and behavioral analysis.



```sql
WITH order_totals AS (
    SELECT
        o.orderNumber,o.customerNumber,o.orderDate,
        SUM(od.quantityOrdered * od.priceEach) AS order_total
    FROM orders o
    JOIN orderdetails od ON o.orderNumber = od.orderNumber
    GROUP BY o.orderNumber)
SELECT
    customerNumber,orderNumber,orderDate,order_total,
    ROUND(AVG(order_total) OVER (
            PARTITION BY customerNumber
            ORDER BY orderDate
            ROWS BETWEEN 2 PRECEDING AND CURRENT ROW ), 2) AS moving_avg_3_orders
FROM order_totals ORDER BY customerNumber, orderDate;
```

customerNumber	orderNumber	orderDate	order_total	moving_avg_3_orders
103	10123	2003-05-20	14571.44	14571.44
103	10298	2004-09-27	6066.78	10319.11
103	10345	2004-11-25	1676.14	7438.12
112	10124	2003-05-21	32641.98	32641.98
112	10278	2004-08-06	33347.88	32994.93
112	10346	2004-11-29	14191.12	26726.99