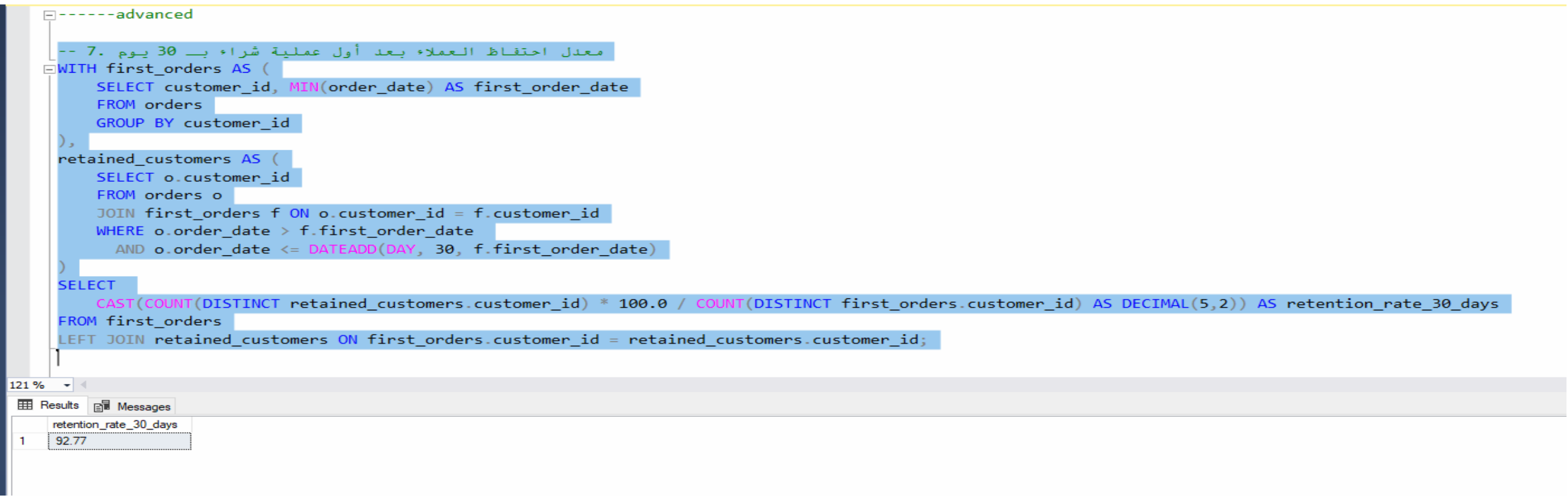
- **Compute the 30-daycustomer retention rate after their first purchase**

  - **Description:**
    This query calculates 30-day customer retention rate by identifying customers who made second purchase after 30 days from the first one
  - **Objective:**
    Calculate the percentage of customers who made a second purchase within 30 days of their first purchase
  - **Output columns:**

Column Name	Description
Retention_rate_30_days	percentage of customers who made a second purchase within 30 days of their first purchase

  - **Explanation of components:**
    -MIN function
    Retrieve first order date
    -DATEADD function
    Adds 30 days to the first order date to retrieve the second orders made within 30 days
    - COUNT function
    Counts total number of orders

  - **Assumptions:**
    - each row in **orders** represents a single customer order
    - A customer is considered “retained” if they place a second order within 30 days of their first order.
  - **SQL query**



```sql
------advanced

-- 7. معدل احتفاظ العملاء بعد أول عملية شراء بـ 30 يوم
WITH first_orders AS (
    SELECT customer_id, MIN(order_date) AS first_order_date
    FROM orders
    GROUP BY customer_id
),
retained_customers AS (
    SELECT o.customer_id
    FROM orders o
    JOIN first_orders f ON o.customer_id = f.customer_id
    WHERE o.order_date > f.first_order_date
      AND o.order_date <= DATEADD(DAY, 30, f.first_order_date)
)
SELECT
    CAST(COUNT(DISTINCT retained_customers.customer_id) * 100.0 / COUNT(DISTINCT first_orders.customer_id) AS DECIMAL(5,2)) AS retention_rate_30_days
FROM first_orders
LEFT JOIN retained_customers ON first_orders.customer_id = retained_customers.customer_id;
```

```
   retention_rate_30_days
1  92.77
```

- **Recommend products frequently bought together with items in customer wishlists**

- **Description:**

  This query identifies products that are frequently bought in the same orders as products from a customer's wishlist

- **Objective:**

  Recommend products that are commonly co-purchased with wishlist items by any customer, helping suggest relevant additional products to each customer.

- **Output columns:**

Column name	**Description**
Wishlist_product	Products in the wishlists
Frequently_bought_together	Product frequently bought with wishlist product
Times_bought_together	Number of times these products were bought together

- **Explanation of components:**

  -Join

  To find orders that included wishlists items and other products in the same orders

  -Group by

  To group recommendations per customer and count pairs of products bought together

  -Order by

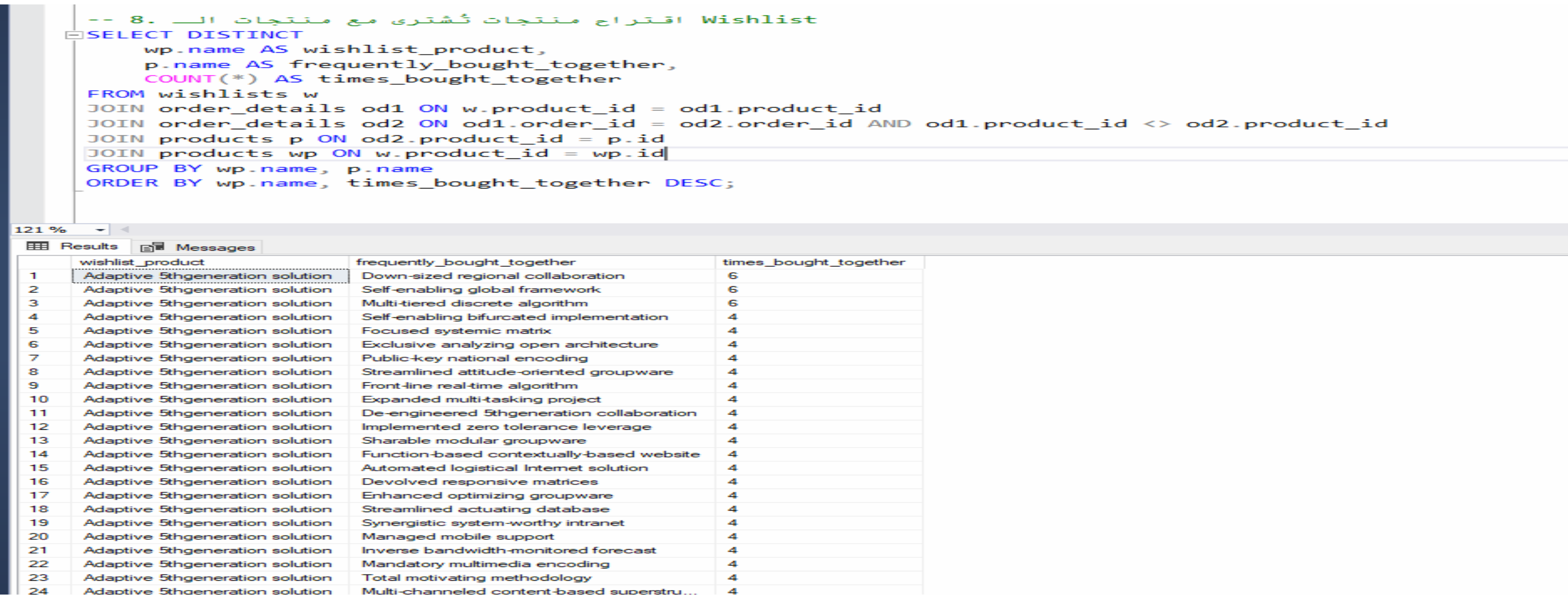
  Frequency to get most pairs that were bought together

- **Assumptions:**

  - A product can appear in multiple orders.

  - Each row in order_details represents a unique product in an order

- **SQL query:**



```
-- 8. اقتراح منتجات تُشترى مع منتجات الـ Wishlist
SELECT DISTINCT
    wp.name AS wishlist_product,
    p.name AS frequently_bought_together,
    COUNT(*) AS times_bought_together
FROM wishlists w
JOIN order_details od1 ON w.product_id = od1.product_id
JOIN order_details od2 ON od1.order_id = od2.order_id AND od1.product_id <> od2.product_id
JOIN products p ON od2.product_id = p.id
JOIN products wp ON w.product_id = wp.id
GROUP BY wp.name, p.name
ORDER BY wp.name, times_bought_together DESC;
```

	wishlist_product	frequently_bought_together	times_bought_together
1	Adaptive 5thgeneration solution	Down-sized regional collaboration	6
2	Adaptive 5thgeneration solution	Self-enabling global framework	6
3	Adaptive 5thgeneration solution	Multi-tiered discrete algorithm	6
4	Adaptive 5thgeneration solution	Self-enabling bifurcated implementation	4
5	Adaptive 5thgeneration solution	Focused systemic matrix	4
6	Adaptive 5thgeneration solution	Exclusive analyzing open architecture	4
7	Adaptive 5thgeneration solution	Public-key national encoding	4
8	Adaptive 5thgeneration solution	Streamlined attitude-oriented groupware	4
9	Adaptive 5thgeneration solution	Front-line real-time algorithm	4
10	Adaptive 5thgeneration solution	Expanded multi-tasking project	4
11	Adaptive 5thgeneration solution	De-engineered 5thgeneration collaboration	4
12	Adaptive 5thgeneration solution	Implemented zero tolerance leverage	4
13	Adaptive 5thgeneration solution	Sharable modular groupware	4
14	Adaptive 5thgeneration solution	Function-based contextually-based website	4
15	Adaptive 5thgeneration solution	Automated logistical Internet solution	4
16	Adaptive 5thgeneration solution	Devolved responsive matrices	4
17	Adaptive 5thgeneration solution	Enhanced optimizing groupware	4
18	Adaptive 5thgeneration solution	Streamlined actuating database	4
19	Adaptive 5thgeneration solution	Synergistic system-worthy intranet	4
20	Adaptive 5thgeneration solution	Managed mobile support	4
21	Adaptive 5thgeneration solution	Inverse bandwidth-monitored forecast	4
22	Adaptive 5thgeneration solution	Mandatory multimedia encoding	4
23	Adaptive 5thgeneration solution	Total motivating methodology	4
24	Adaptive 5thgeneration solution	Multi-channeled content-based superstru...	4

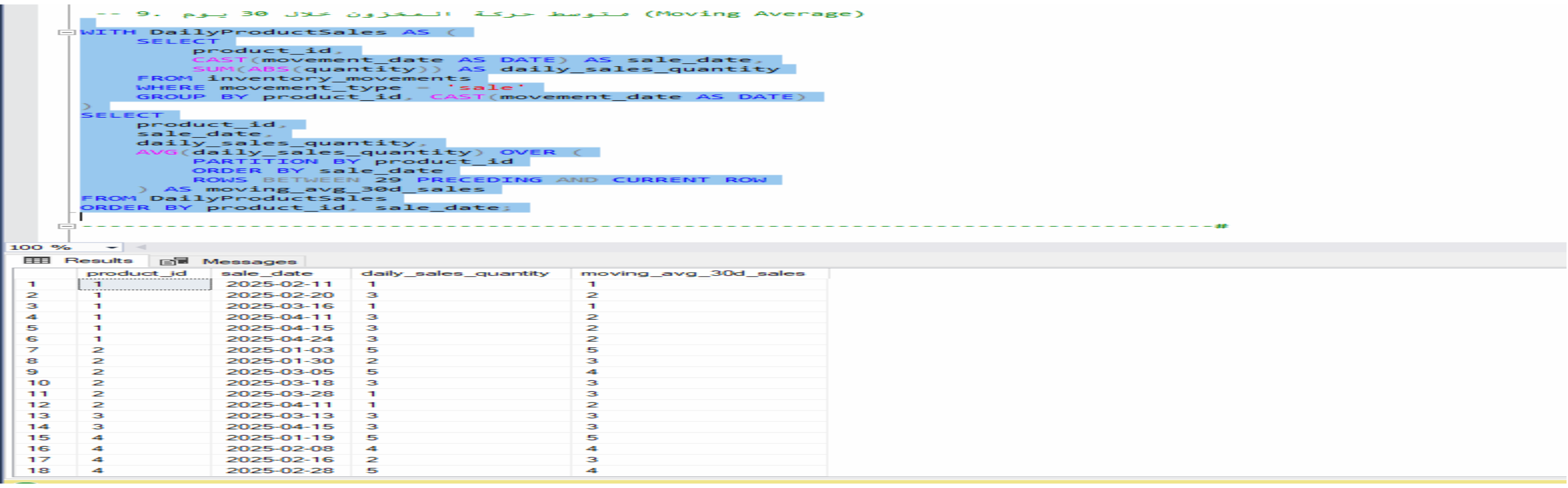
- **Track inventory turnover trends using a 30-day moving average**

- **Description:**
  This query computes the daily sales quantity for each product and calculates a 30-day moving average of those sales, helping track inventory turnover trends over time.
- **Objective:**
  analyze how product sales fluctuate over time by applying a rolling 30-day average
- **Output columns:**

Column Name	Description
Product_id	The product being analyzed
Sale_date	The date of the sale
Daily_sales_quantity	- Total quantity sold on that date.
Moving_avg_30d_sales	percentage of customers who made a second purchase within 30 days of their first purchase

- **Explanation of components:**
  - WITH CTE: DailyProductSales
  Aggregates daily total sales per product.
  - order by function
  Ordering result by product and date

- **Assumptions:**
  - Sales movements stored as negative quantities; ABS(quantity) makes them positive.
  - movement_type = 'sale' filters only sales movements.
  -Data includes sufficient history to compute 30-day rolling averages
- **SQL query:**



```
-- 9. متوسط حركة المخزون خلال 30 يوم (Moving Average)
WITH DailyProductSales AS (
    SELECT
        product_id,
        CAST(movement_date AS DATE) AS sale_date,
        SUM(ABS(quantity)) AS daily_sales_quantity
    FROM inventory_movements
    WHERE movement_type = 'sale'
    GROUP BY product_id, CAST(movement_date AS DATE)
)
SELECT
    product_id,
    sale_date,
    daily_sales_quantity,
    AVG(daily_sales_quantity) OVER (
        PARTITION BY product_id
        ORDER BY sale_date
        ROWS BETWEEN 29 PRECEDING AND CURRENT ROW
    ) AS moving_avg_30d_sales
FROM DailyProductSales
ORDER BY product_id, sale_date;
```

```
    product_id  sale_date   daily_sales_quantity  moving_avg_30d_sales
1   1           2025-02-11  1                     1
2   1           2025-02-20  3                     2
3   1           2025-03-16  1                     1
4   1           2025-04-11  3                     2
5   1           2025-04-15  3                     2
6   1           2025-04-24  3                     2
7   2           2025-01-03  5                     5
8   2           2025-01-30  2                     3
9   2           2025-03-05  5                     4
10  2           2025-03-18  3                     3
11  2           2025-03-28  1                     3
12  2           2025-04-11  1                     2
13  3           2025-03-13  3                     3
14  3           2025-04-15  3                     3
15  4           2025-01-19  5                     5
16  4           2025-02-08  4                     4
17  4           2025-02-16  2                     3
18  4           2025-02-28  5                     4
```

- **Identify customers who have purchased every product in a specific category**

- **Description:**

  This query identifies customers who have purchased all products in a specific category (e.g. category_id = 9)

- **Objective:**

  find highly engaged customers who have fully bought out a category

- **Output columns:**

Column Name	Description
id	Customer id
First_name	Customer first name
Last_name	Customer last name

- **Explanation of components:**

  - CTE: category_products , customer_products

  Get products id for products in the chosen category, combinations of customer and products purchased

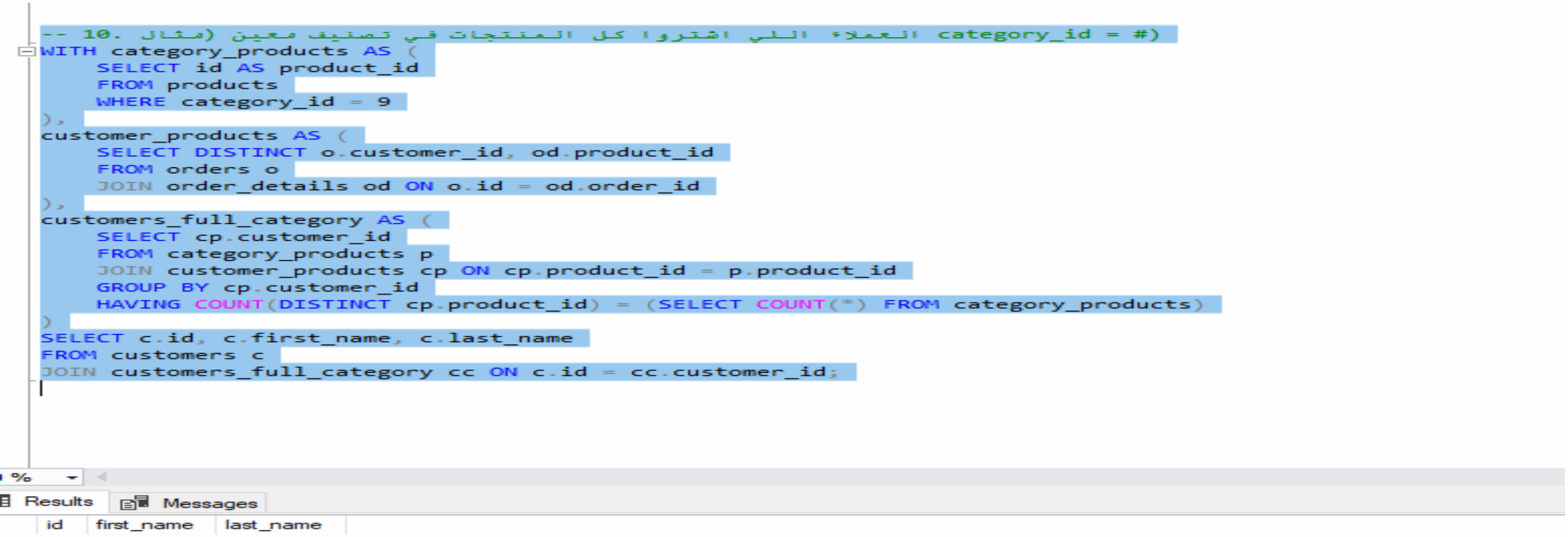
  -filtering

  Grouping by customers, matching customer purchases with category products

- **Assumptions:**

  - Category_ID 9 is the category being analyzed.
  - Each row in order_details represents one product in an order.
  -A customer is considered to have purchased a product if it appears in any of their orders.

- **SQL query:**



```
-- 10. العملاء اللي اشتروا كل المنتجات في تصنيف معين (مثال category_id = #)
WITH category_products AS (
    SELECT id AS product_id
    FROM products
    WHERE category_id = 9
),
customer_products AS (
    SELECT DISTINCT o.customer_id, od.product_id
    FROM orders o
    JOIN order_details od ON o.id = od.order_id
),
customers_full_category AS (
    SELECT cp.customer_id
    FROM category_products p
    JOIN customer_products cp ON cp.product_id = p.product_id
    GROUP BY cp.customer_id
    HAVING COUNT(DISTINCT cp.product_id) = (SELECT COUNT(*) FROM category_products)
)
SELECT c.id, c.first_name, c.last_name
FROM customers c
JOIN customers_full_category cc ON c.id = cc.customer_id;
```

id	first_name	last_name

- **Find pairs of products commonly bought together in the same order**
    - **Description:**

      This query finds unique pairs of products that are commonly purchased together within the same customer order
    - **Objective:**

      To identify product combinations that frequently appear in the same order to discover product relationships and build product recommendation engines.
    - **Output columns:**

Column Name	Description
Product_1	First product
Product_2	Product bought with first product
Times_bought_together	Number of times these products were bought together

- **Explanation of components:**

  -Self-Join on order_details

  -group by

  Grouping by products pairs

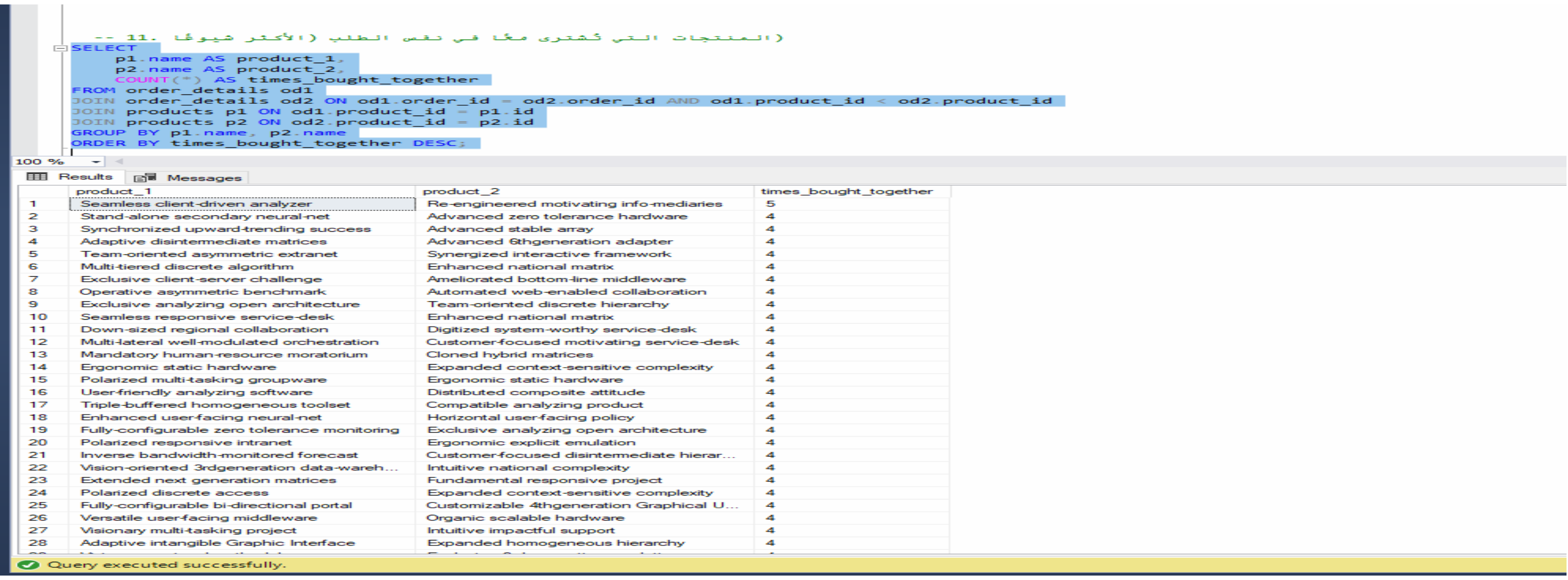
  -order by

  Sorts product pairs by how often they are purchased together

  -count

  the number of times each pair was bought together
- **Assumptions:**

  - Each row in order_details represents a product included in a customer's order
  - The query excludes duplicate and self-pairs
- **SQL query:**



```
-- 11. المنتجات التي تُشترى معًا في نفس الطلب (الأكثر شيوعًا)
SELECT
    p1.name AS product_1,
    p2.name AS product_2,
    COUNT(*) AS times_bought_together
FROM order_details od1
JOIN order_details od2 ON od1.order_id = od2.order_id AND od1.product_id < od2.product_id
JOIN products p1 ON od1.product_id = p1.id
JOIN products p2 ON od2.product_id = p2.id
GROUP BY p1.name, p2.name
ORDER BY times_bought_together DESC;
```

	product_1	product_2	times_bought_together
1	Seamless client-driven analyzer	Re-engineered motivating info-mediaries	5
2	Stand-alone secondary neural-net	Advanced zero tolerance hardware	4
3	Synchronized upward-trending success	Advanced stable array	4
4	Adaptive disintermediate matrices	Advanced 6thgeneration adapter	4
5	Team-oriented asymmetric extranet	Synergized interactive framework	4
6	Multi-tiered discrete algorithm	Enhanced national matrix	4
7	Exclusive client-server challenge	Ameliorated bottom-line middleware	4
8	Operative asymmetric benchmark	Automated web-enabled collaboration	4
9	Exclusive analyzing open architecture	Team-oriented discrete hierarchy	4
10	Seamless responsive service-desk	Enhanced national matrix	4
11	Down-sized regional collaboration	Digitized system-worthy service-desk	4
12	Multi-lateral well-modulated orchestration	Customer-focused motivating service-desk	4
13	Mandatory human-resource moratorium	Cloned hybrid matrices	4
14	Ergonomic static hardware	Expanded context-sensitive complexity	4
15	Polarized multi-tasking groupware	Ergonomic static hardware	4
16	User-friendly analyzing software	Distributed composite attitude	4
17	Triple-buffered homogeneous toolset	Compatible analyzing product	4
18	Enhanced user-facing neural-net	Horizontal user-facing policy	4
19	Fully-configurable zero tolerance monitoring	Exclusive analyzing open architecture	4
20	Polarized responsive intranet	Ergonomic explicit emulation	4
21	Inverse bandwidth-monitored forecast	Customer-focused disintermediate hierar...	4
22	Vision-oriented 3rdgeneration data-wareh...	Intuitive national complexity	4
23	Extended next generation matrices	Fundamental responsive project	4
24	Polarized discrete access	Expanded context-sensitive complexity	4
25	Fully-configurable bi-directional portal	Customizable 4thgeneration Graphical U...	4
26	Versatile user-facing middleware	Organic scalable hardware	4
27	Visionary multi-tasking project	Intuitive impactful support	4
28	Adaptive intangible Graphic Interface	Expanded homogeneous hierarchy	4

Query executed successfully.

- **Calculate the time taken to deliver orders in days**

- **Description:**

  This query calculates the number of days taken to deliver each order by subtracting the order date from the shipping date

- **Objective:**

  To measure the delivery performance by determining how many days it took from when an order was placed to when it was delivered

- **Output columns:**

Column Name	Description
Order_id	Id for each order
Delivery_days	Time taken for orders to get delivered

- **Explanation of components:**

  -Join

  Joins the orders and shipping tables to link each order with its shipment.

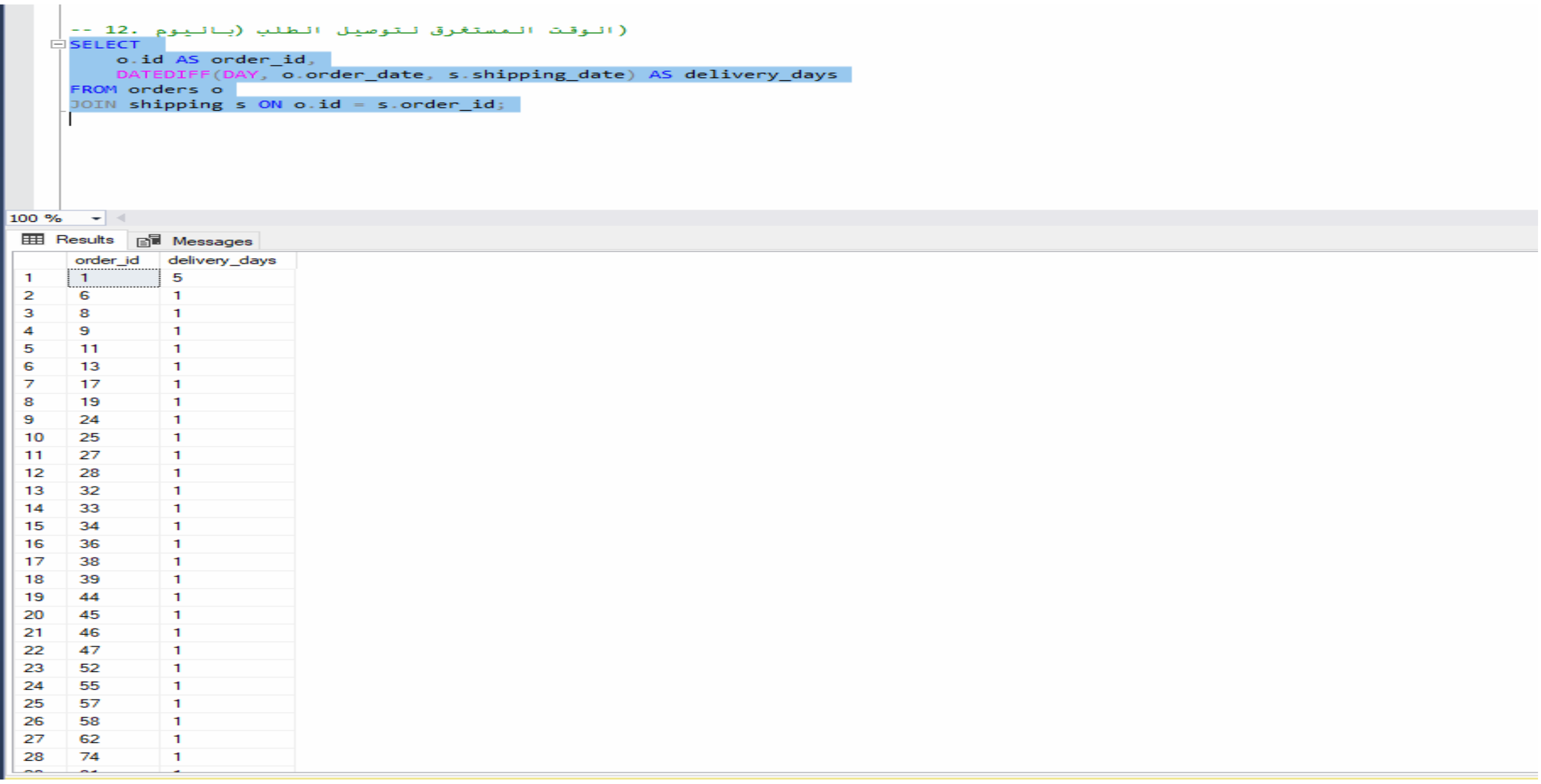
  -DATEDIFF

  Calculates the difference in days between the order date and the shipping date

- **Assumptions:**

  - Each order has a corresponding entry in the shipping table.
  - order_date and shipping_date are stored as DATE or DATETIME formats.

- **SQL query:**



```sql
-- 12. الوقت المستغرق لتوصيل الطلب (باليوم)
SELECT
    o.id AS order_id,
    DATEDIFF(DAY, o.order_date, s.shipping_date) AS delivery_days
FROM orders o
JOIN shipping s ON o.id = s.order_id;
```

```
    order_id  delivery_days
1   1         5
2   6         1
3   8         1
4   9         1
5   11        1
6   13        1
7   17        1
8   19        1
9   24        1
10  25        1
11  27        1
12  28        1
13  32        1
14  33        1
15  34        1
16  36        1
17  38        1
18  39        1
19  44        1
20  45        1
21  46        1
22  47        1
23  52        1
24  55        1
25  57        1
26  58        1
27  62        1
28  74        1
```