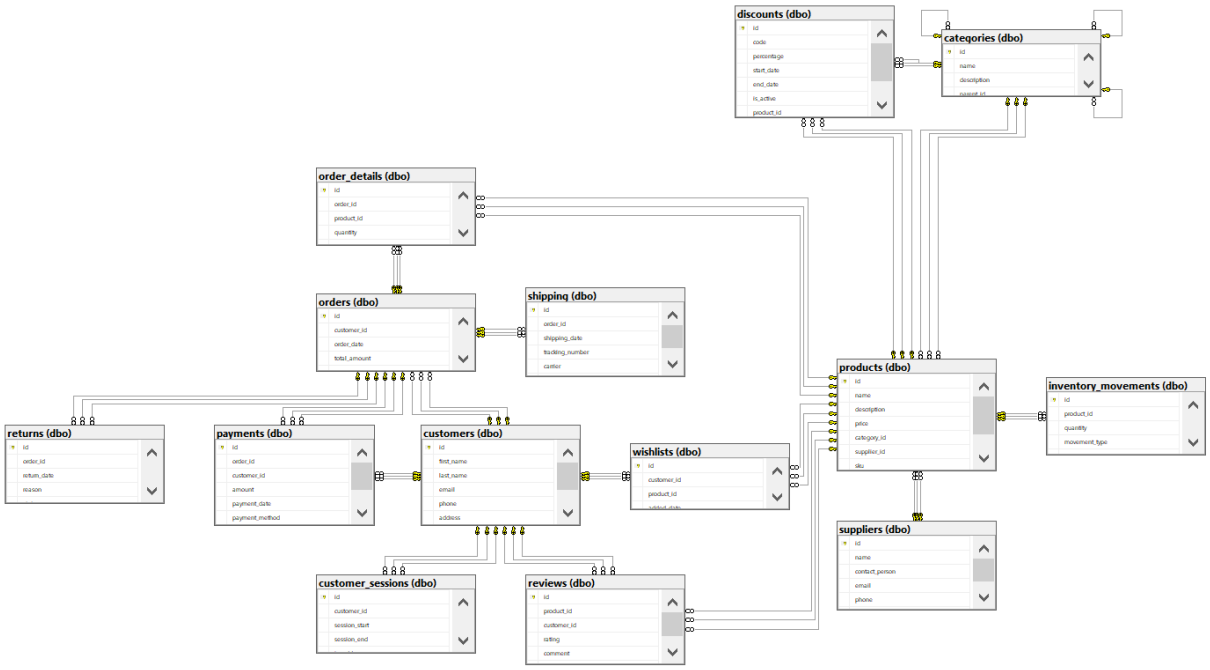
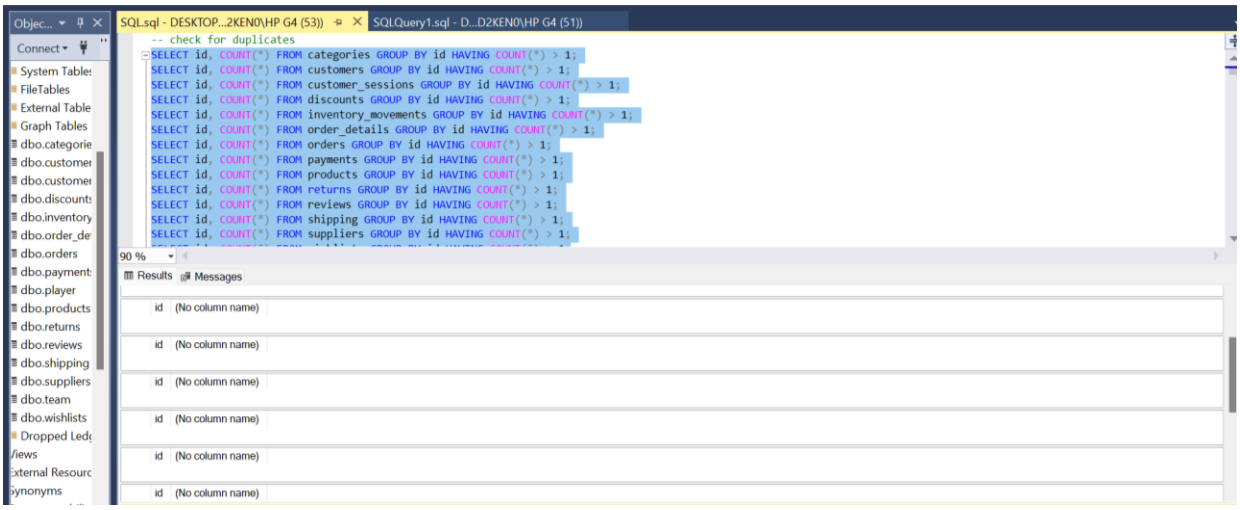
SQL



1. Cleaning the data

- Checking for duplicates

- 

Checking for nulls

```sql
-- check for nulls
SELECT * FROM customers
WHERE first_name IS NULL
   OR last_name IS NULL
   OR email IS NULL
   OR phone IS NULL
   OR registration_date IS NULL;

SELECT * FROM orders
WHERE customer_id IS NULL
   OR order_date IS NULL
   OR total_amount IS NULL
   OR status IS NULL;

SELECT * FROM categories
WHERE id IS NULL
   OR name IS NULL
   OR description IS NULL
   OR parent_id IS NULL;
```

75 %

Results | Messages

id	first_name	last_name	email	phone	address	registration_date

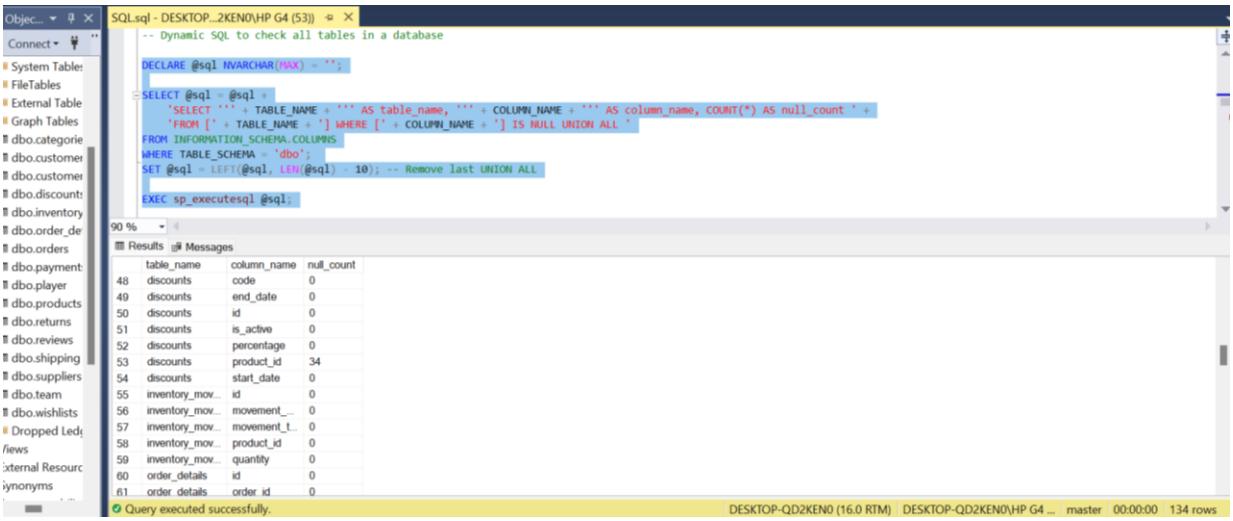
id	customer_id	order_date	total_amount	status

	id	name	description	parent_id
1	1	Part	Win color election her. Idea society understand ar...	NULL
2	2	Program	Remain great line dog alone either level. Arm mon...	NULL
3	3	Large	Table people job civil here. View seat according st...	NULL
4	4	Hold	Yard call test investment state hundred. Scene h...	NULL
5	5	Speech	Forward upon provide. Live world bring if continue...	NULL
6	1...	Table	Although him again natural yeah maintain. Call up...	NULL
7	1...	Mission	Doctor hair create agency white protect front. Ter...	NULL
8	1...	Share	Trouble yard state. Might and special deep owner ...	NULL

- Dynamic SQL to check all tables in a database

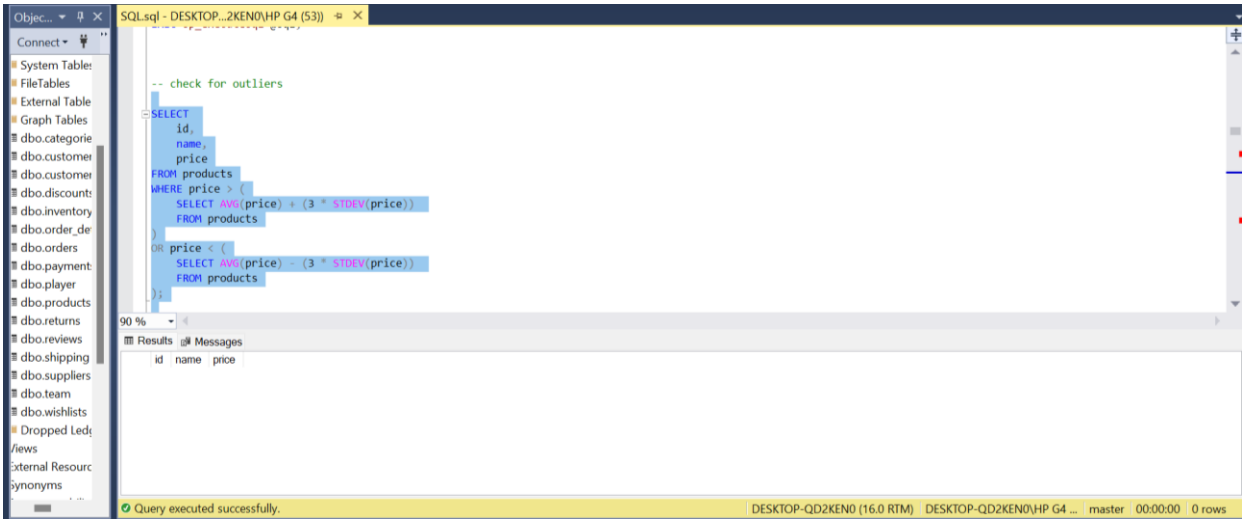
  Instead of repeating the same query for each table, we used a dynamic SQL query that scans all tables in the database and reports the number of NULLs per column.

  The script uses the INFORMATION_SCHEMA.COLUMNS view to automatically build and run the NULL-check query.

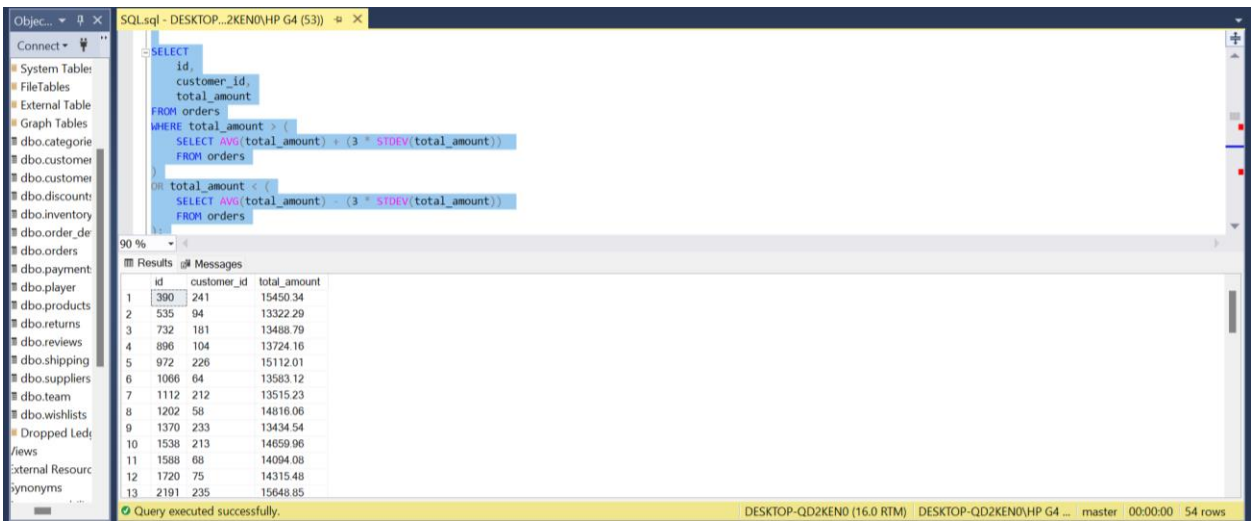


- Checking for outliers

  Products table



Orders table



- Check for Impossible or Suspicious Values
    - Negative or zero prices

- Orders with zero or negative total

```
SELECT * FROM orders
WHERE total_amount <= 0;
```

Results | Messages

id	customer_id	order_date	total_amount	status

- Inventory movements with zero quantity

```
SELECT * FROM inventory_movements
WHERE quantity = 0;
```

Results | Messages

id	product_id	quantity	movement_type	movement_date

- customers with no orders

```
select * from customers c where not exists (select 1 from orders o where c.id = o.customer_id)
```

Results | Messages

	id	first_name	last_name	email	phone	address	registration_date
1	48	Allison	Robinson	leematthew@example.net	+10000000048	03718 Andrew Motorway Apt. 522 West Patriciapo...	2025-01-28 14:20:47.000

- Check Foreign Key Violations

  - Orders without valid customers

```
SELECT * FROM orders
WHERE customer_id NOT IN (SELECT id FROM customers);
```

Results | Messages

id	customer_id	order_date	total_amount	status

  - Order details with invalid products

```sql
SELECT * FROM order_details
WHERE product_id NOT IN (SELECT id FROM products);
```

90 %

Results | Messages

id	order_id	product_id	quantity	unit_price

- Check for Unused Products

```sql
SELECT * FROM products p
WHERE NOT EXISTS (SELECT 1 FROM order_details od WHERE od.product_id = p.id)
  AND NOT EXISTS (SELECT 1 FROM wishlists w WHERE w.product_id = p.id)
  AND NOT EXISTS (SELECT 1 FROM reviews r WHERE r.product_id = p.id);
```

90 %

Results | Messages

id	name	description	price	category_id	supplier_id	sku	stock_quantity

2. Project requirements

- Identify customers with the highest number of orders.

```sql
SELECT
    RANK() OVER (ORDER BY COUNT(o.id) DESC) AS rank_by_orders,
    c.id AS customer_id,
    c.first_name + ' ' + c.last_name AS customer_name,
    c.email,
    COUNT(o.id) AS num_orders,
    SUM(o.total_amount) AS total_spend,
    AVG(o.total_amount) AS avg_order_value,
    CAST(MIN(o.order_date) AS DATE) AS first_order_date,
    CAST(MAX(o.order_date) AS DATE) AS last_order_date
FROM customers c
JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, c.first_name, c.last_name, c.email
ORDER BY num_orders DESC;
```

90 %

Results | Messages

	rank_by_orders	customer_id	customer_name	email	num_orders	total_spend	avg_order_value	first_order_date	last_order_date
1	1	212	Ryan Chang	rachel30@example.org	73	302643.41	4145.800136	2025-01-03	2025-05-03
2	2	90	Suzanne Bennett	kyle00@example.net	68	297327.19	4372.458676	2025-01-01	2025-05-06
3	2	134	Robyn Reed	carsonaustin@example.net	68	307115.89	4516.410147	2025-01-01	2025-05-06
4	4	116	Zachary Aguirre	johnsonjennifer@example.com	66	329685.39	4995.233181	2025-01-01	2025-05-03
5	5	189	Mark Spence	smithheather@example.org	65	265418.79	4083.366000	2025-01-06	2025-05-06
6	6	190	Leslie Alvarado	jaredkelly@example.net	64	274766.12	4293.220625	2025-01-03	2025-05-07
7	6	188	Tiffany Boone	rebecca45@example.net	64	290029.65	4531.713281	2025-01-02	2025-05-07
8	6	91	Dana Farrell	shelly85@example.com	64	308903.76	4826.621250	2025-01-01	2025-05-06
9	6	55	John Perez	christianrobert@example.net	64	275604.08	4306.313750	2025-01-03	2025-05-07

Query executed successfully. DESKTOP-QD2KEN0 (16.0 RTM) DESKTOP-QD2KEN(

1. Launch a Loyalty Program

- Reward top customers with points, discounts, or exclusive benefits.

- Encourage repeat purchases and increase customer retention.

2. Create VIP Customer Tiers

- Introduce “Gold” or “Platinum” levels for frequent buyers.
- Offer perks like early access, premium support, or free shipping.

3. Collect Direct Feedback

- Engage top customers through surveys or feedback forms.
- Use insights to enhance user experience and service quality.

  - Recommend products frequently bought together with items in customer wishlists

```sql
CREATE OR ALTER PROCEDURE sp_GetFrequentlyBoughtWithWishlistProducts
AS
BEGIN
    -- CTE for wishlisted products per customer
    WITH WishlistProducts AS (
        SELECT customer_id, product_id
        FROM wishlists
    ),

    -- Orders made by those customers
    CustomerOrders AS (
        SELECT o.customer_id, od.product_id
        FROM orders o
        INNER JOIN order_details od ON o.id = od.order_id
    ),

    -- Join to find frequently bought-together items
    ProductsBoughtWithWishlist AS (
        SELECT
            wp.product_id AS wishlist_product_id,
            co.product_id AS bought_product_id,
            COUNT(*) AS times_bought_together
        FROM WishlistProducts wp
        INNER JOIN CustomerOrders co
            ON wp.customer_id = co.customer_id
            AND wp.product_id <> co.product_id
        GROUP BY wp.product_id, co.product_id
    )
```

1. Implement Smart Product Bundles

- Create dynamic bundles featuring wishlist items and their frequently bought companions.
- Offer bundle discounts to encourage larger cart sizes.

2. Cart Suggestions Based on Wishlist

- At checkout, recommend products commonly bought with items from the customer’s wishlist.
- Increase average order value with timely, relevant prompts.

3. Targeted Ads Based on Wishlist Associations

- Retarget users with ads showing complementary products.

- Use high-conversion bundles in social and display ads.

  - Calculate the time taken to deliver orders in days.

```sql
SELECT
    o.id AS order_id,
    c.first_name + ' ' + c.last_name AS customer_name,
    o.order_date,
    s.shipping_date,
    DATEDIFF(day, o.order_date, s.shipping_date) AS delivery_days,
    CASE
        WHEN DATEDIFF(day, o.order_date, s.shipping_date) <= 3 Then 'Fast'
        WHEN DATEDIFF(day, o.order_date, s.shipping_date) <= 7 THEN 'Standard'
        ELSE 'Slow'
    END AS delivery_speed
from orders o
INNER JOIN customers c on o.customer_id = c.id
INNER JOIN shipping s on o.id = s.order_id
where s.status = 'delivered'
Order by delivery_days DESC;
```

Results | Messages

	order_id	customer_name	order_date	shipping_date	delivery_days	delivery_speed
1	8	Jason Lewis	2025-03-12 04:09:19.000	2025-03-13 04:09:19.000	1	Fast
2	9	Kimberly Wiley	2025-01-03 14:50:19.000	2025-01-04 14:50:19.000	1	Fast
3	17	Carly Duran	2025-01-24 00:10:42.000	2025-01-25 00:10:42.000	1	Fast
4	33	Sheila Cohen	2025-03-20 16:32:46.000	2025-03-21 16:32:46.000	1	Fast
5	34	Martin Thomas	2025-05-03 01:18:51.000	2025-05-04 01:18:51.000	1	Fast
6	36	James Nguyen	2025-03-15 02:35:21.000	2025-03-16 02:35:21.000	1	Fast
7	38	Kevin Edwards	2025-01-02 21:19:27.000	2025-01-03 21:19:27.000	1	Fast
8	39	Kimberly Wiley	2025-03-26 07:52:44.000	2025-03-27 07:52:44.000	1	Fast
9	44	Lindsey Lutz	2025-04-28 20:26:05.000	2025-04-29 20:26:05.000	1	Fast
10	45	Michael Perez	2025-02-04 11:26:24.000	2025-02-05 11:26:24.000	1	Fast

1. Optimize Shipping Methods

- Offer customers faster shipping options at checkout (e.g., same-day, express).
- Negotiate better delivery SLAs with logistics providers.

2. Enhance Order Tracking & Transparency

- Display estimated delivery time clearly on product and order pages.
- Keep customers informed with real-time tracking updates.

3. Track Impact of Delivery Time on Customer Satisfaction

- Correlate delivery delays with refund requests or low ratings.

Prioritize improvements in high-impact areas to enhance customer loyalty

Preparing views for different departments which can access the specific data they need

```sql
CREATE VIEW vw_CustomerFeatures AS
SELECT
    c.id AS customer_id,
    c.first_name,
    c.last_name,
    c.email,
    c.registration_date,
    DATEDIFF(DAY, c.registration_date, GETDATE()) AS customer_tenure_days,

    -- Order behavior
    COUNT(DISTINCT o.id) AS total_orders,
    SUM(o.total_amount) AS total_spend,
    CASE WHEN COUNT(DISTINCT o.id) > 0 THEN SUM(o.total_amount)/COUNT(DISTINCT o.id) ELSE 0 END AS avg_order_value,
    MAX(o.order_date) AS last_order_date,
    DATEDIFF(DAY, MAX(o.order_date), GETDATE()) AS days_since_last_order,

    -- Return behavior
    COUNT(DISTINCT r.id) AS total_returns,
    CASE WHEN COUNT(DISTINCT o.id) > 0 THEN CAST(COUNT(DISTINCT r.id) AS FLOAT)/COUNT(DISTINCT o.id) ELSE 0 END AS return_rate,

    -- Session behavior
    COUNT(DISTINCT cs.id) AS total_sessions,
    AVG(DATEDIFF(MINUTE, cs.session_start, cs.session_end)) AS avg_session_duration,

    -- Wishlist behavior
    COUNT(DISTINCT w.id) AS wishlist_items,

    -- Payment behavior
    COUNT(DISTINCT CASE WHEN p.status = 'completed' THEN p.id END) AS successful_payments,
    COUNT(DISTINCT CASE WHEN p.status = 'failed' THEN p.id END) AS failed_payments,

    -- Calculate customer value metrics
    SUM(o.total_amount) / NULLIF(DATEDIFF(DAY, MIN(o.order_date), GETDATE()), 0) * 30 AS monthly_revenue,
    DATEDIFF(DAY, MIN(o.order_date), MAX(o.order_date)) / NULLIF(COUNT(DISTINCT o.id) - 1, 0) AS avg_days_between_orders
FROM
    customers c
LEFT JOIN orders o ON c.id = o.customer_id
LEFT JOIN returns r ON o.id = r.order_id
LEFT JOIN customer_sessions cs ON c.id = cs.customer_id
LEFT JOIN wishlists w ON c.id = w.customer_id
LEFT JOIN payments p ON c.id = p.customer_id
GROUP BY
    c.id, c.first_name, c.last_name, c.email, c.registration_date;
```

```sql
-- This creates a wide table with categories as columns

CREATE VIEW vw_CustomerCategoryPivot AS
WITH CategoryPurchases AS (
    SELECT
        o.customer_id,
        p.category_id,
        c.name AS category_name,
        COUNT(DISTINCT o.id) AS purchase_count,
        SUM(od.quantity) AS items_purchased
    FROM
        orders o
    JOIN order_details od ON o.id = od.order_id
    JOIN products p ON od.product_id = p.id
    JOIN categories c ON p.category_id = c.id
    GROUP BY
        o.customer_id, p.category_id, c.name
)
SELECT
    cp.customer_id,
    MAX(CASE WHEN cp.category_name = 'Adult' THEN cp.purchase_count ELSE 0 END) AS Adult_purchases,
    MAX(CASE WHEN cp.category_name = 'Almost' THEN cp.purchase_count ELSE 0 END) AS Almost_purchases,
    MAX(CASE WHEN cp.category_name = 'Base' THEN cp.purchase_count ELSE 0 END) AS Base_purchases,
    MAX(CASE WHEN cp.category_name = 'Born' THEN cp.purchase_count ELSE 0 END) AS Born_purchases,
    MAX(CASE WHEN cp.category_name = 'Carry' THEN cp.purchase_count ELSE 0 END) AS Carry_purchases,
    MAX(CASE WHEN cp.category_name = 'Choose' THEN cp.purchase_count ELSE 0 END) AS Choose_purchases,
    MAX(CASE WHEN cp.category_name = 'Energy' THEN cp.purchase_count ELSE 0 END) AS Energy_purchases,
    MAX(CASE WHEN cp.category_name = 'Fire' THEN cp.purchase_count ELSE 0 END) AS Fire_purchases,
    MAX(CASE WHEN cp.category_name = 'Free' THEN cp.purchase_count ELSE 0 END) AS Free_purchases,
    MAX(CASE WHEN cp.category_name = 'Game' THEN cp.purchase_count ELSE 0 END) AS Game_purchases,
    MAX(CASE WHEN cp.category_name = 'Glass' THEN cp.purchase_count ELSE 0 END) AS Glass_purchases,
    MAX(CASE WHEN cp.category_name = 'Half' THEN cp.purchase_count ELSE 0 END) AS Half_purchases,
    MAX(CASE WHEN cp.category_name = 'Himself' THEN cp.purchase_count ELSE 0 END) AS Himself_purchases,
    MAX(CASE WHEN cp.category_name = 'Hold' THEN cp.purchase_count ELSE 0 END) AS Hold_purchases,
    MAX(CASE WHEN cp.category_name = 'Large' THEN cp.purchase_count ELSE 0 END) AS Large_purchases,
    MAX(CASE WHEN cp.category_name = 'Letter' THEN cp.purchase_count ELSE 0 END) AS Letter_purchases,
    MAX(CASE WHEN cp.category_name = 'Marriage' THEN cp.purchase_count ELSE 0 END) AS Marriage_purchases,
    MAX(CASE WHEN cp.category_name = 'Mission' THEN cp.purchase_count ELSE 0 END) AS Mission_purchases,
    MAX(CASE WHEN cp.category_name = 'Onto' THEN cp.purchase_count ELSE 0 END) AS Onto_purchases,
    MAX(CASE WHEN cp.category_name = 'Or' THEN cp.purchase_count ELSE 0 END) AS Or_purchases,
    MAX(CASE WHEN cp.category_name = 'Over' THEN cp.purchase_count ELSE 0 END) AS Over_purchases,
    MAX(CASE WHEN cp.category_name = 'Part' THEN cp.purchase_count ELSE 0 END) AS Part_purchases,
    MAX(CASE WHEN cp.category_name = 'Program' THEN cp.purchase_count ELSE 0 END) AS Program_purchases,
    MAX(CASE WHEN cp.category_name = 'Recent' THEN cp.purchase_count ELSE 0 END) AS Recent_purchases,
    MAX(CASE WHEN cp.category_name = 'Remember' THEN cp.purchase_count ELSE 0 END) AS Remember_purchases,
    MAX(CASE WHEN cp.category_name = 'Share' THEN cp.purchase_count ELSE 0 END) AS Share_purchases,
    MAX(CASE WHEN cp.category_name = 'Speech' THEN cp.purchase_count ELSE 0 END) AS Speech_purchases,
    MAX(CASE WHEN cp.category_name = 'Table' THEN cp.purchase_count ELSE 0 END) AS Table_purchases,
    MAX(CASE WHEN cp.category_name = 'Who' THEN cp.purchase_count ELSE 0 END) AS Who_purchases,
    MAX(CASE WHEN cp.category_name = 'Window' THEN cp.purchase_count ELSE 0 END) AS Window_purchases,
   COUNT(DISTINCT cp.category_id) AS unique_categories_purchased
FROM
    CategoryPurchases cp
GROUP BY
    cp.customer_id;
```

Full Report View

```sql
CREATE VIEW vw_CustomerChurnDataset AS
WITH ChurnDefinition AS (
    SELECT
        c.id AS customer_id,
        CASE WHEN MAX(o.order_date) IS NULL OR DATEDIFF(DAY, MAX(o.order_date), GETDATE()) > 90 THEN 1 ELSE 0 END AS is_churned
    FROM
        customers c
    LEFT JOIN orders o ON c.id = o.customer_id
    GROUP BY
        c.id
)
SELECT
    cf.customer_id,
    cf.customer_tenure_days,
    cf.total_orders,
    cf.total_spend,
    cf.avg_order_value,
    cf.days_since_last_order,
    cf.return_rate,
    cf.total_sessions,
    cf.avg_session_duration,
    cf.wishlist_items,
    cf.successful_payments,
    cf.failed_payments,
    cf.monthly_revenue,
    cf.avg_days_between_orders,
    ccp.unique_categories_purchased,
    ccp.Adult_purchases,
    ccp.Almost_purchases,
    ccp.Base_purchases,
    ccp.Born_purchases,
    ccp.Carry_purchases,
    ccp.Choose_purchases,
    ccp.Energy_purchases,
    ccp.Fire_purchases,
    ccp.Free_purchases,
    ccp.Game_purchases,
    ccp.Glass_purchases,
    ccp.Half_purchases,
    ccp.Himself_purchases,
    ccp.Hold_purchases,
    ccp.Large_purchases,
    ccp.Letter_purchases,
    ccp.Marriage_purchases,
    ccp.Mission_purchases,
    ccp.Onto_purchases,
    ccp.Or_purchases,
    ccp.Over_purchases,
    ccp.Part_purchases,
    ccp.Program_purchases,
    ccp.Recent_purchases,
    ccp.Remember_purchases,
    ccp.Share_purchases,
    ccp.Speech_purchases,
    ccp.Table_purchases,
    ccp.Who_purchases,
    ccp.Window_purchases,
    -- Target variable
    cd.is_churned
FROM
    vw_CustomerFeatures cf
JOIN ChurnDefinition cd ON cf.customer_id = cd.customer_id
LEFT JOIN vw_CustomerCategoryPivot ccp ON cf.customer_id = ccp.customer_id;

-- VIEW THE FULL REPORT
SELECT top 5 * FROM vw_CustomerChurnDataset
ORDER BY customer_id;
```

- Calculate the total sales revenue from all orders.

```sql
-- Cancelled orders: total amount and count
SELECT
    COUNT(DISTINCT o.id) AS cancelled_order_count,
    SUM(od.quantity * od.unit_price) AS cancelled_order_revenue
FROM
    orders o
JOIN
    order_details od ON o.id = od.order_id
WHERE
    o.status = 'cancelled';


SELECT
    ROUND(SUM(od.quantity * od.unit_price), 2) AS total_sales_revenue
FROM
    order_details od
```

Results | Messages

cancelled_order_count	cancelled_order_revenue
2012	8867722.70

total_sales_revenue
44943966.02

```sql
SELECT
    ROUND(SUM(od.quantity * od.unit_price), 2) AS total_sales_revenue,
    COUNT(DISTINCT o.id) AS total_orders,
    COUNT(DISTINCT o.customer_id) AS unique_customers,
    ROUND(SUM(od.quantity * od.unit_price) / NULLIF(COUNT(DISTINCT o.id), 0), 2) AS avg_order_value
FROM
    orders o
JOIN
    order_details od ON o.id = od.order_id
WHERE
    o.status != 'cancelled';
```

90 %

Results | Messages

	total_sales_revenue	total_orders	unique_customers	avg_order_value
1	36076243.32	8188	248	4405.990000

Insights:

high revenue per customer, high cancellation rate

Recommendation

expand without losing quality

improving the user experience and providing fast, effective support during the process can

significantly help reduce the cancellation rate.

- List the top 5 best-selling products by quantity sold.
  (Ranked by units sold, with revenue)

```sql
SELECT TOP 5
    p.id AS product_id,
    p.name AS product_name,
    SUM(od.quantity) AS total_units_sold,
    ROUND(SUM(od.quantity * od.unit_price), 2) AS total_revenue
FROM
    order_details od
JOIN
    products p ON od.product_id = p.id
JOIN
    orders o ON od.order_id = o.id
WHERE
    o.status != 'cancelled'
GROUP BY
    p.id, p.name
ORDER BY
    total_units_sold DESC;
```

0 %

Results | Messages

	product_id	product_name	total_units_sold	total_revenue
1	216	Up-sized interactive time-frame	214	132288.38
2	591	Vision-oriented scalable archive	208	165917.44
3	499	Versatile holistic help-desk	205	124541.60
4	222	Compatible optimal knowledgebase	204	58301.16
5	531	Vision-oriented mission-critical application	202	57163.98

Insights

These products are in consistently high demand, indicating strong customer preference

Recommendation

Due to high turnover, these items should be prioritized in stock planning, reordering, and logistics management to avoid stockouts.

- Generate an alert for products with stock quantities below 20 units.

```sql
SELECT
    RANK() OVER (ORDER BY p.stock_quantity ASC) AS stock_rank,
    p.id AS product_id,
    p.name AS product_name,
    p.stock_quantity,
    c.name AS category_name,
    s.name AS supplier_name,
    p.price,
    CASE
        WHEN p.stock_quantity < 20 THEN 'Low'
        ELSE 'OK'
    END AS stock_status
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
LEFT JOIN suppliers s ON p.supplier_id = s.id
ORDER BY p.stock_quantity ASC;
```

90 %

Results | Messages

	stock_rank	product_id	product_name	stock_quantity	category_name	supplier_name	price	stock_status
55	52	48	Compatible modular policy	19	Half	Rodriguez, Smith and Gonzal...	144.16	Low
56	52	29	Synchronized multi-tasking hierarchy	19	Hold	Williams Ltd	658.88	Low
57	52	35	Inverse holistic open architecture	19	Recent	Crawford, Adams and Savage	23.95	Low
58	52	16	Innovative context-sensitive matrices	19	Adult	Williams Group	248.85	Low
59	59	6	Multi-channeled foreground adapter	20	Speech	Williams Ltd	779.25	OK
60	59	66	Upgradable optimal interface	20	Half	Figueroa PLC	973.60	OK
61	59	82	Adaptive empowering alliance	20	Window	Morrison and Sons	443.71	OK
62	59	76	Cloned incremental hierarchy	20	Large	Schmitt, Irwin and Dixon	92.83	OK
63	59	99	Horizontal methodical parallelism	20	Large	Davies, Jackson and Randolph	395.38	OK

Query executed successfully. DESKTOP-QD2KEN0 (16.0 RTM) DESKTOP-QD

Insights:
Some products have stock levels below 20 units, indicating a risk of running out of stock.

```sql
    RANK() OVER (ORDER BY p.stock_quantity ASC) AS stock_rank,
    p.id AS product_id,
    p.name AS product_name,
    p.stock_quantity,
    c.name AS category_name,
    s.name AS supplier_name,
    p.price,
    CASE
        WHEN p.stock_quantity < 20 THEN 'Low'
        ELSE 'OK'
    END AS stock_status
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
LEFT JOIN suppliers s ON p.supplier_id = s.id
ORDER BY p.stock_quantity ASC;
```

%

Results | Messages

stock_rank	product_id	product_name	stock_quantity	category_name	supplier_name	price	stock_status
1	84	Upgradable fresh-thinking model	0	Adult	Davies, Jackson and Randolph	69.27	Low
2	12	Advanced modular capacity	3	Part	Freeman-Valenzuela	716.47	Low
3	15	Focused radical protocol	6	Adult	Wright-Davis	228.16	Low
3	3	Function-based well-modulated intranet	6	Half	Figueroa PLC	712.42	Low
3	20	Reverse-engineered fresh-thinking success	6	Half	Cox-Williams	306.61	Low
3	25	Exclusive actuating open system	6	Adult	Wright-Davis	777.38	Low
3	33	Distributed empowering leverage	6	Recent	Crawford, Adams and Savage	210.22	Low
3	100	Reactive tertiary moratorium	6	Window	Osborn-Stewart	437.07	Low
9	34	Configurable optimizing extranet	7	Hold	Williams Group	561.51	Low
9	30	Synergistic intangible product	7	Program	Roberts-Dickerson	253.38	Low
9	19	Right-sized didactic function	7	Adult	Cox-Williams	294.79	Low
9	71	Distributed fault-tolerant process improve	7	Half	Sanders Ltd	864.49	Low

Recommendations:
Restock low inventory items before they run out.

- Determine the percentage of orders that used a discount.

SQL.sql - DESKTOP...2KEN0\HP G4 (52))

```sql
WITH DiscountedOrders AS (
    SELECT DISTINCT od.order_id
    FROM order_details od
    JOIN discounts d ON od.product_id = d.product_id
    JOIN orders o ON od.order_id = o.id
    WHERE
        d.is_active = 1
)

-- Final percentage calculation
SELECT
    COUNT(DISTINCT do.order_id) AS discounted_orders,
    (SELECT COUNT(DISTINCT id) FROM orders) AS total_orders,
    ROUND(
        COUNT(DISTINCT do.order_id) * 100.0 /
        NULLIF((SELECT COUNT(DISTINCT id) FROM orders), 0), 2
    ) AS discount_usage_percentage
FROM
    DiscountedOrders do;
```

90 %

Results | Messages

	discounted_orders	total_orders	discount_usage_percentage
1	637	10200	6.250000000000

Insights:
Despite having active discounts, only a small fraction of customers used them during checkout.

Recommendations:
Recheck the rules for applying discounts to make sure they match what customers actually want.

Consider offering more attractive or targeted discounts to increase customer engagement and boost usage rates.

- **Calculate the average rating for each product.**

```sql
--Calculate the average rating for each product.
select
p.id as Product_id,
p.name as Product_name,
avg(r.rating) as average_rating
from products p
inner join reviews r
on p.id=r.product_id
group by p.name ,p.id
```

90 %

Results | Messages

	Product_id	Product_name	average_rating
1	1	Enterprise-wide stable synergy	2
2	2	Synergized uniform contingency	1
3	3	Function-based well-modulated intranet	3
4	4	Monitored bifurcated database	2
5	5	Versatile modular info-mediaries	3
6	6	Multi-channeled foreground adapter	3
7	7	Fundamental grid-enabled definition	3
8	8	Business-focused zero tolerance func...	1
9	9	Networked bandwidth-monitored secu...	3
10	10	Secured reciprocal support	3
11	11	Visionary regional budgetary manage...	4
12	12	Advanced modular capacity	2
13	13	Horizontal didactic standardization	2
14	14	Customizable object-oriented support	3
15	15	Focused radical protocol	2

Query executed successfully.

Insights:

Several products consistently receive low ratings (3 or below).

Recommendations:

- **Focus improvement efforts on products with ratings ≤ 3.**
- **Run surveys for low-rated products to gather detailed feedback.**
- **Send follow-up emails asking happy customers to leave reviews.**
- **Offer small incentives (discounts, loyalty points) for leaving a review.**

- Advanced Queries
  - Compute the 30-day customer retention rate after their first purchase.

SQL.sql - DESKTOP...2KEN0\HP G4 (52))

```sql
WITH
-- First purchase date per valid customer
FirstPurchase AS (
    SELECT
        customer_id,
        MIN(order_date) AS first_purchase_date
    FROM
        orders
    WHERE
        status != 'cancelled'
        AND customer_id NOT LIKE 'test%'  -- Exclude test accounts
    GROUP BY
        customer_id
),

-- Customers who made a follow-up purchase within 30 days
RetainedCustomers AS (
    SELECT DISTINCT
        o.customer_id
    FROM
        orders o
    JOIN
        FirstPurchase fp ON o.customer_id = fp.customer_id
    WHERE
        o.order_date > fp.first_purchase_date  -- Exclude same-day purchases
        AND o.order_date <= DATEADD(day, 30, fp.first_purchase_date)
        AND o.status != 'cancelled'
)
```

SQL.sql - DESKTOP...2KEN0\HP G4 (52))

```sql
    SELECT DISTINCT
        o.customer_id
    FROM
        orders o
    JOIN
        FirstPurchase fp ON o.customer_id = fp.customer_id
    WHERE
        o.order_date > fp.first_purchase_date  -- Exclude same-day purchases
        AND o.order_date <= DATEADD(day, 30, fp.first_purchase_date)
        AND o.status != 'cancelled'
)

-- Final retention rate calculation
SELECT
    COUNT(DISTINCT fp.customer_id) AS total_customers,
    COUNT(DISTINCT rc.customer_id) AS retained_customers,
    ROUND(COUNT(DISTINCT rc.customer_id) * 100.0 /
          NULLIF(COUNT(DISTINCT fp.customer_id), 0), 1) AS retention_rate_pct
FROM
    FirstPurchase fp
LEFT JOIN
    RetainedCustomers rc ON fp.customer_id = rc.customer_id;
```

90 %

Results | Messages

	total_customers	retained_customers	retention_rate_pct
1	248	227	91.500000000000

Insights

A 91% 30-day retention rate indicates excellent early customer satisfaction and product-market fit.

High Potential for Long-Term Value

Recommendation

introduce loyalty programs, milestone rewards, or personalized content to keep customers .

```sql
    -- Return with product names
    SELECT
        pbw.wishlist_product_id,
        wp.name AS wishlist_product_name,
        pbw.bought_product_id,
        bp.name AS bought_product_name,
        pbw.times_bought_together
    FROM ProductsBoughtWithWishlist pbw
   INNER JOIN products wp ON wp.id = pbw.wishlist_product_id
   INNER JOIN products bp ON bp.id = pbw.bought_product_id
    ORDER BY
        pbw.times_bought_together DESC;
END;

EXEC sp_GetFrequentlyBoughtWithWishlistProducts;
```

90 %

Results | Messages

	wishlist_product_id	wishlist_product_name	bought_product_id	bought_product_name	times_bought_together
1	201	Streamlined tangible ability	568	Multi-channeled bottom-line orchestration	9
2	535	Customizable homogene...	458	Up-sized multi-tasking productivity	8
3	179	Triple-buffered homogen...	525	Enhanced discrete implementation	8
4	446	Versatile neutral help-desk	283	Pre-emptive tertiary implementation	8
5	201	Streamlined tangible ability	483	Synergized uniform protocol	8
6	201	Streamlined tangible ability	197	Down-sized regional collaboration	8
7	546	Switchable zero-defect p...	402	Organized motivating projection	8
8	546	Switchable zero-defect p...	456	Pre-emptive systematic circuit	7
9	201	Streamlined tangible ability	143	Devolved value-added help-desk	7
10	127	Balanced dedicated help...	397	Pre-emptive tangible encoding	7

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

```sql
WITH DailySales AS (
    SELECT
        product_id,
        CAST(movement_date AS DATE) AS sale_date,
        SUM(-quantity) AS daily_units_sold  -- Convert negative to positive sales
    FROM
        inventory_movements
    WHERE
        movement_type = 'sale'
    GROUP BY
        product_id, CAST(movement_date AS DATE)
)

SELECT
    product_id,
    sale_date,
    daily_units_sold,
    AVG(daily_units_sold * 1.0) OVER (
        PARTITION BY product_id
        ORDER BY sale_date
        ROWS BETWEEN 29 PRECEDING AND CURRENT ROW
    ) AS moving_avg_30day
FROM
    DailySales
ORDER BY
    product_id, sale_date;
```

```sql
WITH DailySales AS (
    SELECT
        product_id,
        CAST(movement_date AS DATE) AS sale_date,
        SUM(-quantity) AS daily_units_sold  -- Convert negative to positive sales
    FROM
        inventory_movements
    WHERE
        movement_type = 'sale'
    GROUP BY
        product_id, CAST(movement_date AS DATE)
)

SELECT
    product_id,
```

90 %

Results | Messages

	product_id	sale_date	daily_units_sold	moving_avg_30day
3	1	2025-03-16	1	1.666666
4	1	2025-04-11	3	2.000000
5	1	2025-04-15	3	2.200000
6	1	2025-04-24	3	2.333333
7	2	2025-01-03	5	5.000000
8	2	2025-01-30	2	3.500000
9	2	2025-03-05	5	4.000000
10	2	2025-03-18	3	3.750000
11	2	2025-03-28	1	3.200000
12	2	2025-04-11	1	2.833333
13	3	2025-03-13	3	3.000000

Insights:
The 30-day moving average reveals trends in product sales over time, helping identify high-demand and slow-moving products.

Recommendations:
Increase inventory for high-turnover products to meet demand, and consider reducing or phasing out slow-moving products to optimize stock levels.

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

```sql
SELECT
    c.id AS customer_id,
    c.first_name,
    c.last_name,
    cat.id AS category_id,
    cat.name AS category_name
FROM
    customers c
    JOIN orders o ON c.id = o.customer_id
    JOIN order_details od ON o.id = od.order_id
    JOIN products p ON od.product_id = p.id
    JOIN categories cat ON p.category_id = cat.id
WHERE
    cat.name = 'Window' -- Replace with the desired category name
GROUP BY
    c.id, c.first_name, c.last_name, cat.id, cat.name
```

90 %

Results | Messages

	customer_id	first_name	last_name	category_id	category_name
1	1	Loretta	Weaver	8	Window
2	2	Jennifer	Kelley	8	Window
3	3	Michael	Mcintosh	8	Window
4	4	James	Nguyen	8	Window
5	5	Sheila	Cohen	8	Window
6	6	Christina	Hines	8	Window
7	7	Susan	Barron	8	Window
8	9	Dana	Ross	8	Window

Insights:

These customers are **power users** who are not only interested but have completed purchases for the entire product range in this category.

Recommenditions:

- Target these customers with complementary product offers.
- Provide a discount or gift for bringing in new customers to purchase from the same category.

Find pairs of products commonly bought together in the same order.

```sql
SELECT
    od1.product_id AS product_1,
    od2.product_id 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
GROUP BY
    od1.product_id, od2.product_id
ORDER BY
    times_bought_together DESC;
```

0 %

Results | Messages

	product_1	product_2	times_bought_together
1	282	341	5
2	488	595	4
3	179	440	4
4	244	487	4
5	293	354	4
6	137	249	4
7	348	441	4
8	426	432	4
9	285	299	4
10	274	409	4
11	339	407	4

Insights:

Products are frequently bought together>1,2.

Recommenditions:

-These are good candidates for bundling or combo offers.

- Pairs that are bought together frequently should be stored close together in warehouses to speed up packing.