Created by –Ankit Kumar

REVENUE LEAKAGE & PROFIT LOSS DETECTION SYSTEM

Revenue Leakage & Profit Loss Detection System

Built an enterprise-level Revenue Leakage Detection System using **SQL**, **Power BI**, and **Excel** to identify unbilled revenue, excess **discount losses**, **pricing mismatches**, and **return-driven** profit erosion, enabling data-driven corrective actions. To identify, measure, and analyze revenue loss caused by **discount misuse, billing gaps, delayed payments, and excessive returns**, using SQL-driven analytics and KPI monitoring.

Business Problem

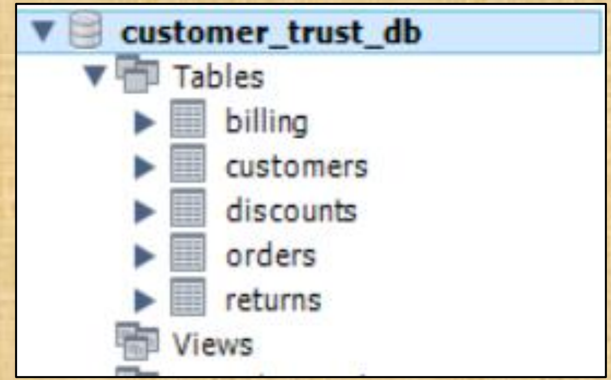
Many companies lose money due to:

- High discounts without control
- Incorrect billing amounts
- Excessive product returns
- Delayed or failed payments

These losses are often **hidden** and not tracked properly.

Data Model Used

- Customers - Customer profile & segmentation
- Orders - Order quantity & pricing
- Discounts - Discount % and approvals
- Billing - Invoice amount & payment status
- Returns - Refund amounts & reasons



```
▼ customer_trust_db
  ▼ Tables
    ▶ billing
    ▶ customers
    ▶ discounts
    ▶ orders
    ▶ returns
    Views
```

Tools & Technologies

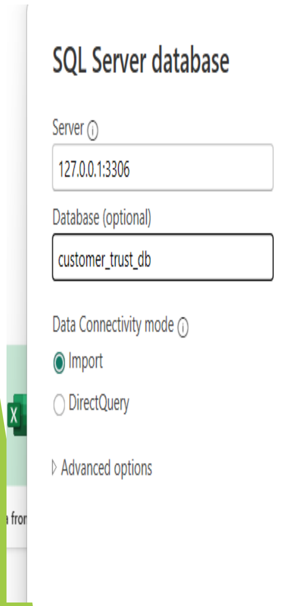
- **SQL** – KPI calculations & analysis
- **Excel** – Validation & intermediate analysis
- **Power BI** – Interactive dashboards & insights

Skills

- **Data Analysis**
- **Business Analytics**
- **KPI Design**
- **Financial Analysis**
- **Stakeholder Reporting**

Power BI Dashboard (SQL + Power BI)

- This Power BI dashboard provides a **comprehensive view of revenue performance, cost structure, and profit leakage points** across the business
- **Total Revenue (₹299K)** is lower than **Total Cost Price (₹326K)**, clearly highlighting an **overall loss scenario**
- **Refunds (₹16.5K)** and **discounts (₹2K)** are key contributors to revenue leakage.
- Monthly analysis shows a **sharp revenue decline in July**, signaling potential operational or demand issues.
- **Sales Managers approve the majority of discounts (61%)**, indicating a need for stronger discount control policies.
- Major return reasons include **Damaged Products, Late Delivery, and Quality Issues**, directly impacting profitability.
- Product-level analysis identifies **low-performing SKUs**, enabling targeted corrective actions.

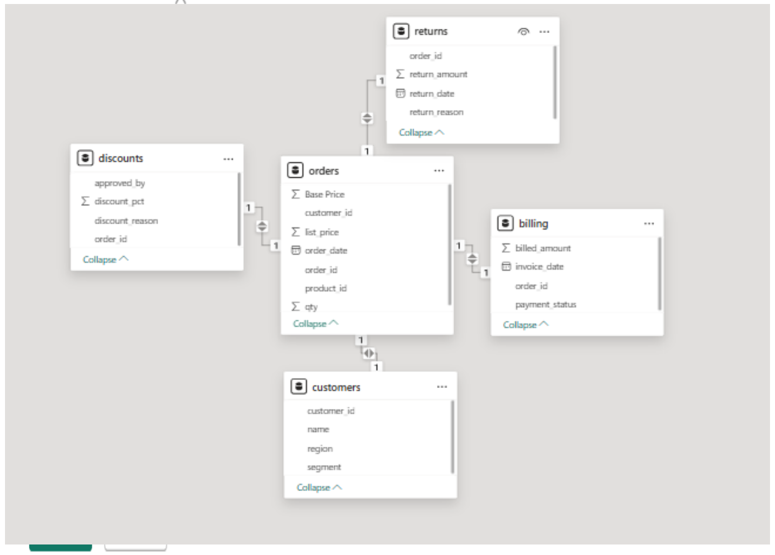

SQL Server database

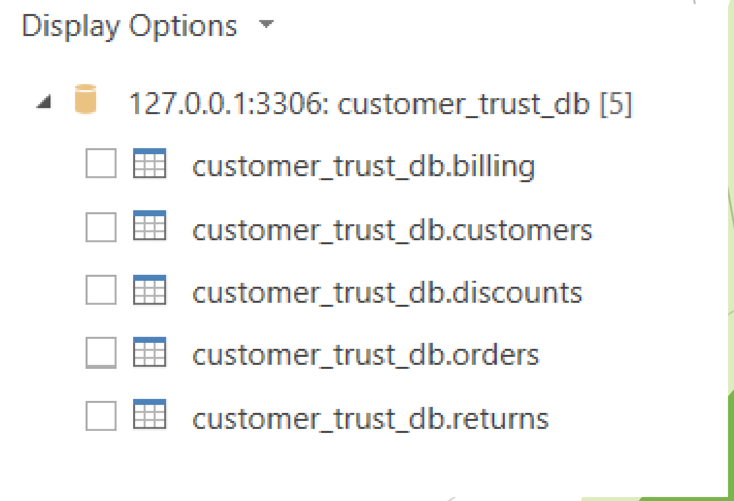
Server: 127.0.0.1:3306

Database (optional): customer_trust_db

Data Connectivity mode: [✓ Import] DirectQuery

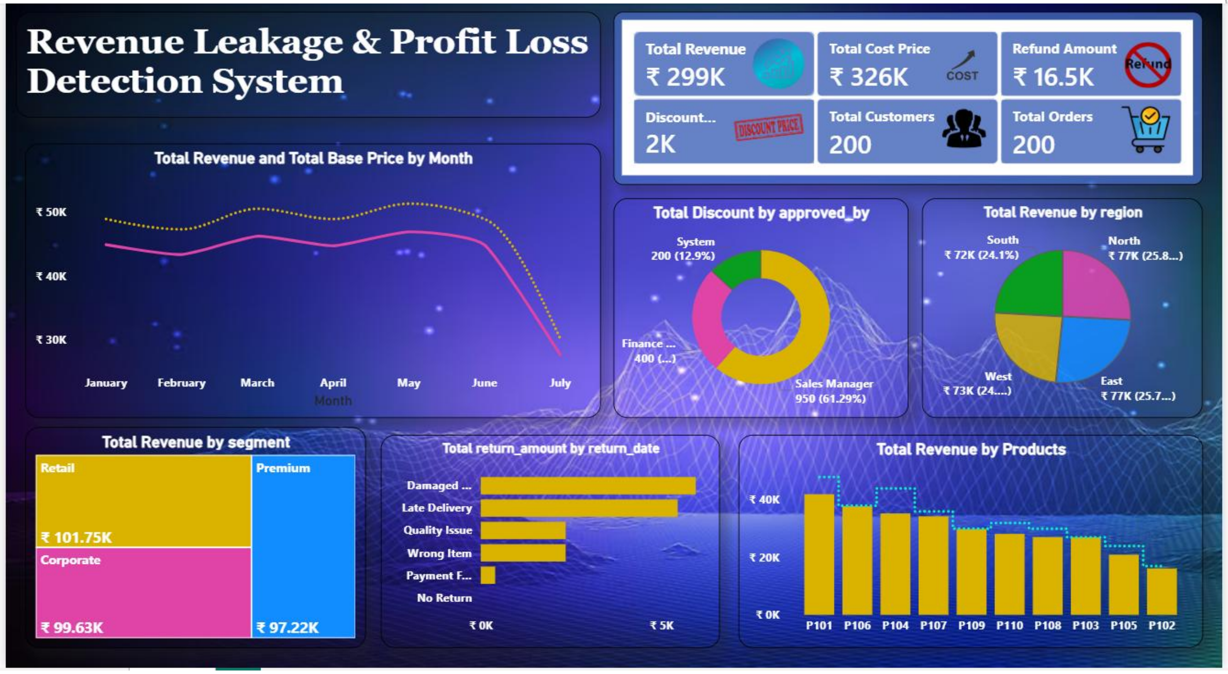
Advanced options

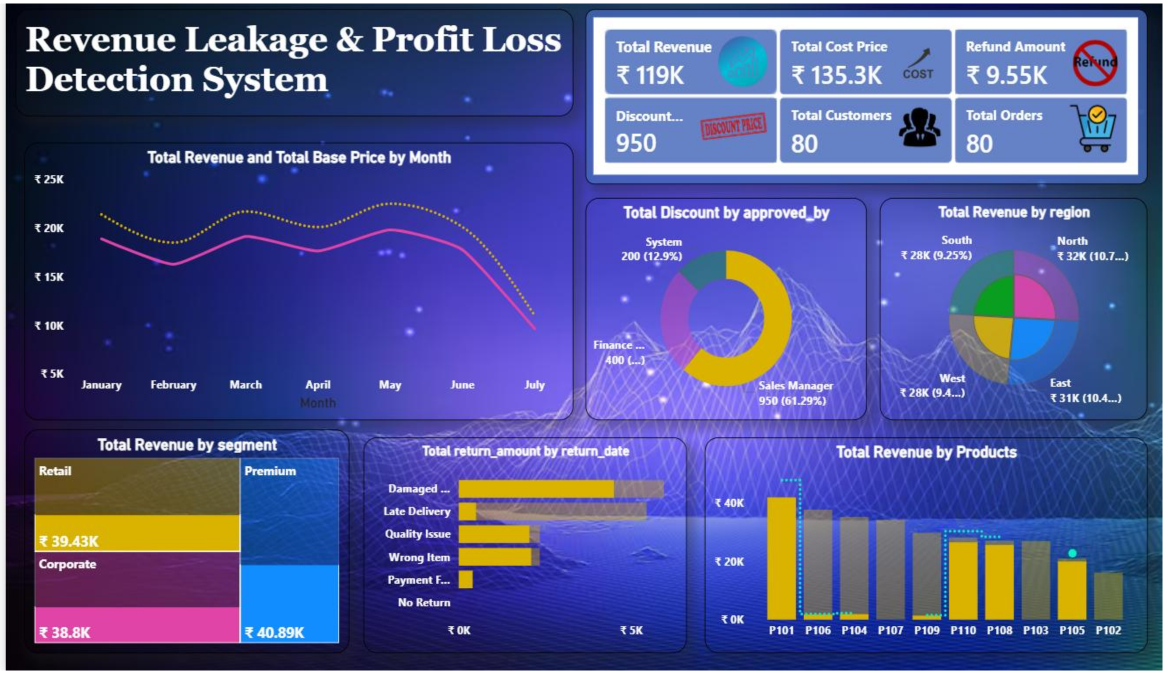



Display Options

127.0.0.1:3306: customer_trust_db [5]
- [ ] customer_trust_db.billing
- [ ] customer_trust_db.customers
- [ ] customer_trust_db.discounts
- [ ] customer_trust_db.orders
- [ ] customer_trust_db.returns

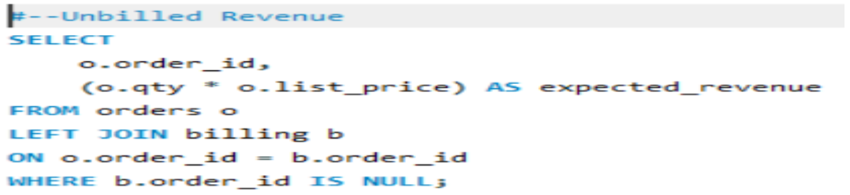
Power BI Dashboard





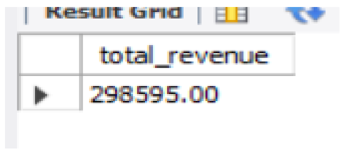
KEY KPIs

- **Unbilled Revenue**



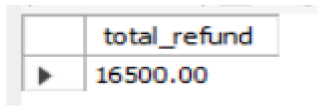
```
#--Unbilled Revenue
SELECT
    o.order_id,
    (o.qty * o.list_price) AS expected_revenue
FROM orders o
LEFT JOIN billing b
ON o.order_id = b.order_id
WHERE b.order_id IS NULL;
```

- **Total Revenue**



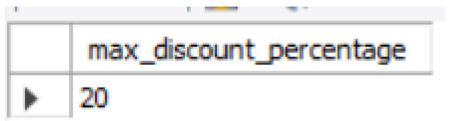
total_revenue
298595.00

- **Total Refund By Reasons**



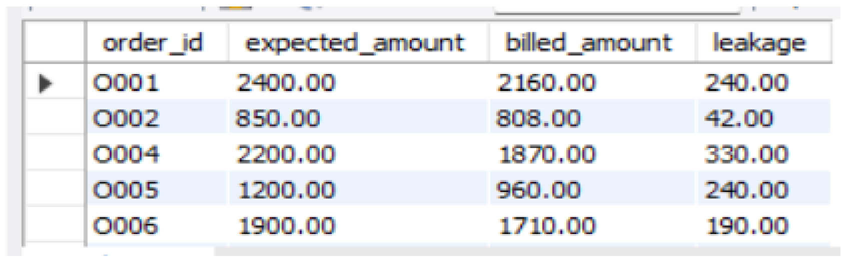
total_refund
16500.00

- **Max Discounts Percentage**



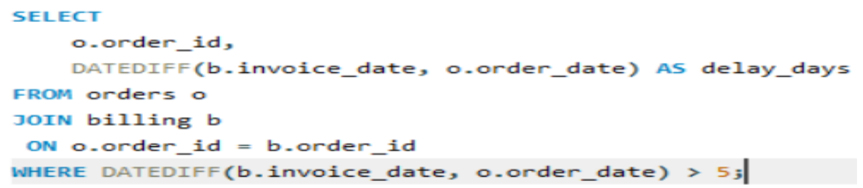
max_discount_percentage
20

- **Actual price VS Billing Mismatch**



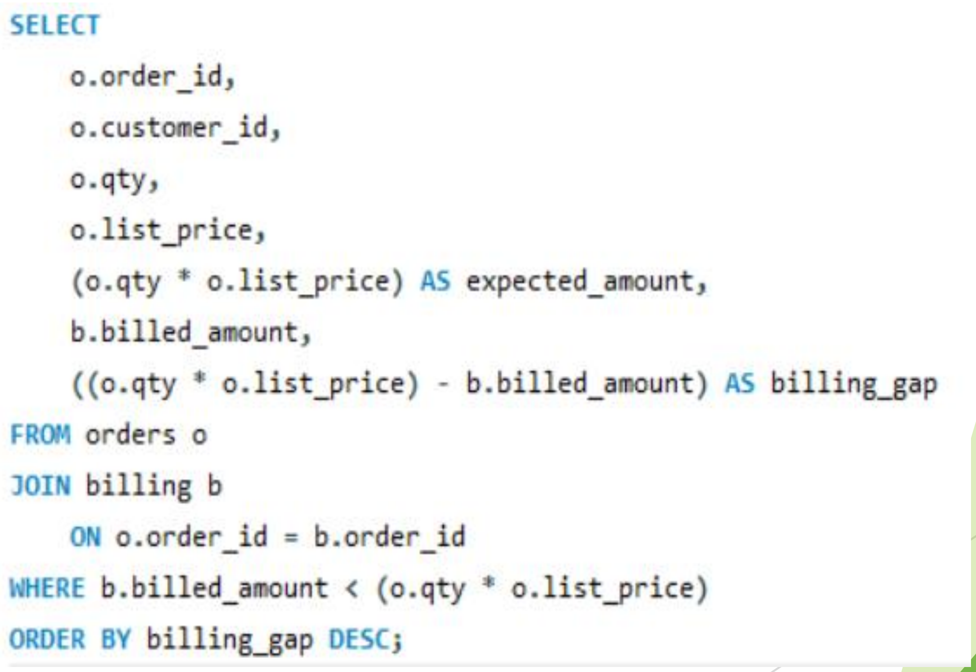
order_id	expected_amount	billed_amount	leakage
O001	2400.00	2160.00	240.00
O002	850.00	808.00	42.00
O004	2200.00	1870.00	330.00
O005	1200.00	960.00	240.00
O006	1900.00	1710.00	190.00

- **Late Invoicing Impact**



```
SELECT
    o.order_id,
    DATEDIFF(b.invoice_date, o.order_date) AS delay_days
FROM orders o
JOIN billing b
 ON o.order_id = b.order_id
WHERE DATEDIFF(b.invoice_date, o.order_date) > 5;
```

- **Price vs Billing Gap**



```
SELECT
    o.order_id,
    o.customer_id,
    o.qty,
    o.list_price,
    (o.qty * o.list_price) AS expected_amount,
    b.billed_amount,
    ((o.qty * o.list_price) - b.billed_amount) AS billing_gap
FROM orders o
JOIN billing b
    ON o.order_id = b.order_id
WHERE b.billed_amount < (o.qty * o.list_price)
ORDER BY billing_gap DESC;
```

Customer-Level Analysis

1. Which customers generated the highest total billed revenue?
2. Which customers have the most orders?
3. Which customers have never returned any products?
4. Identify customers with recurring return issues (>=3 returns).
5. Find customers with average discount received above 15%.
6. Find customers whose payments are often delayed or overdue.
7. Identify customers with the highest total returns (return amount).
8. Which customers have the highest Revenue Leakage due to discounts?
9. Calculate the average billed amount per customer.
10. Which customers have the lowest trust score (Revenue - Returns - Discounts)?

Order-Level Analysis

1. List all orders that received discounts greater than 20%.
2. Identify orders with returns exceeding 50% of billed amount.
3. Which orders have payment status as 'Overdue' or 'Failed'?
4. Find orders where billed amount is less than list price × qty (Price vs Billing Gap).
5. List orders that received both a discount and had a return.

Product-Level Analysis

1. Which products generated the most revenue?
2. Top 5 products with most returns.
3. Find the products with zero returns.
4. Which products have the highest return rate?
5. Which products were never sold?

Region / Segment Analysis

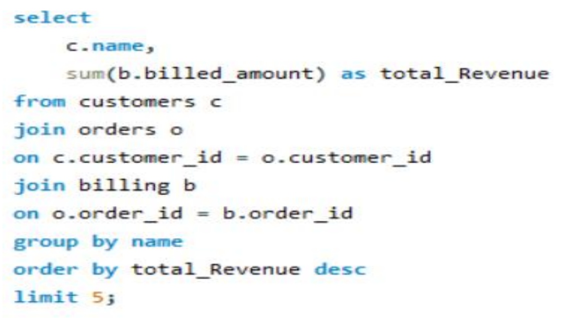
1. Which region has the highest total sales?
2. Which regions have the most pending payments?
3. Find the total revenue lost due to returns per region.
4. Which segment of customers has the highest average trust score?
5. Which segments have highest average order value?

Discounts / Approver Analysis

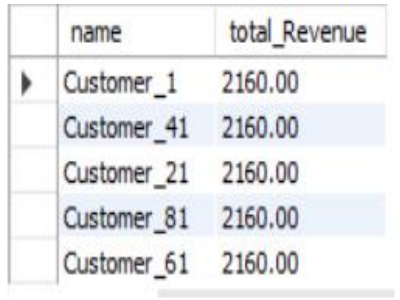
1. Find the total discount given by each approver.
2. Which approvers gave the maximum number of high-value discounts (>20%)?
3. Analyze the correlation between discount pct and return amount.
4. Find orders where discount pct > 20% and return amount > 0.
5. Identify customers who received the most total discounts.

Customer Level Analysis---

1. **Which customers generated the highest total billed revenue?**

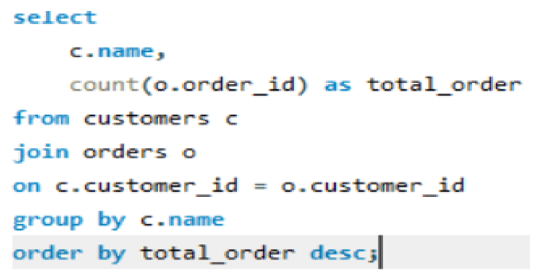


```
select
    c.name,
    sum(b.billed_amount) as total_Revenue
from customers c
join orders o
on c.customer_id = o.customer_id
join billing b
on o.order_id = b.order_id
group by name
order by total_Revenue desc
limit 5;
```

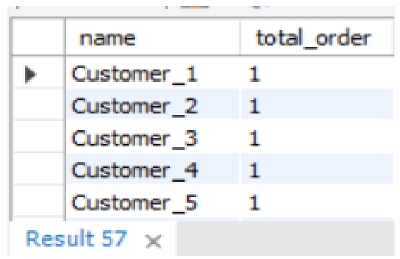


name	total_Revenue
Customer_1	2160.00
Customer_41	2160.00
Customer_21	2160.00
Customer_81	2160.00
Customer_61	2160.00

2. **Which customers have the most orders?**



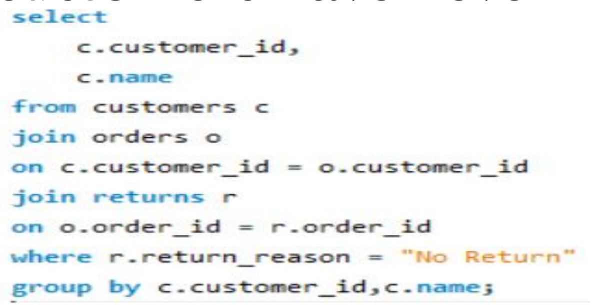
```
select
    c.name,
    count(o.order_id) as total_order
from customers c
join orders o
on c.customer_id = o.customer_id
group by c.name
order by total_order desc;
```



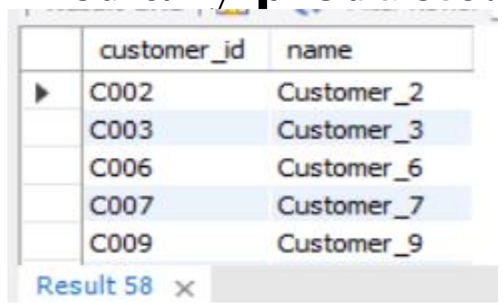
name	total_order
Customer_1	1
Customer_2	1
Customer_3	1
Customer_4	1
Customer_5	1

Result 57 ×

3. **Which customers have never returned any products?**



```
select
    c.customer_id,
    c.name
from customers c
join orders o
on c.customer_id = o.customer_id
join returns r
on o.order_id = r.order_id
where r.return_reason = "No Return"
group by c.customer_id,c.name;
```

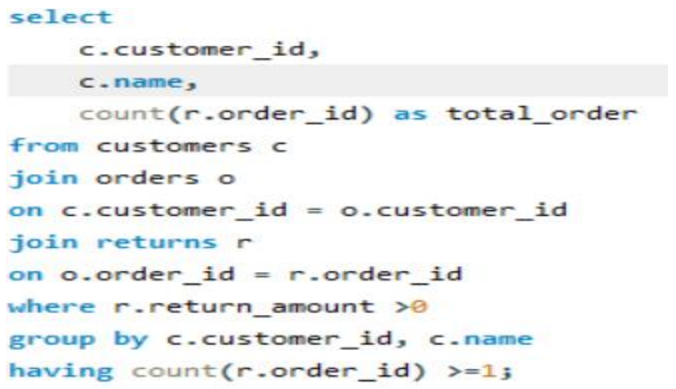


customer_id	name
C002	Customer_2
C003	Customer_3
C006	Customer_6
C007	Customer_7
C009	Customer_9

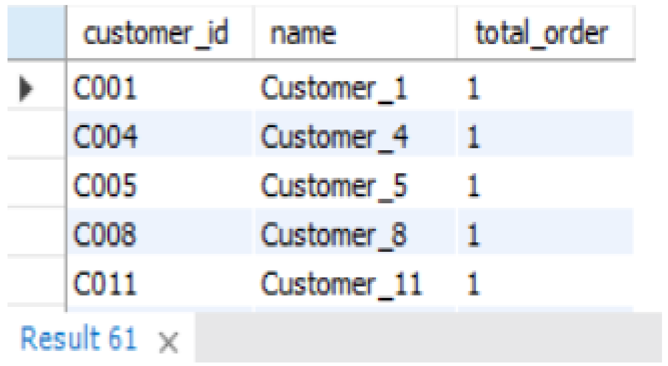
Result 58 ×

Customer Level Analysis

4. Identify customers with recurring return issues (>=3 returns)?



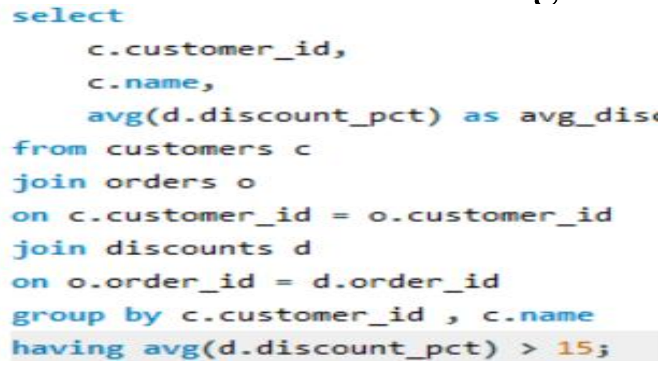
```
select
    c.customer_id,
    c.name,
    count(r.order_id) as total_order
from customers c
join orders o
on c.customer_id = o.customer_id
join returns r
on o.order_id = r.order_id
where r.return_amount >0
group by c.customer_id, c.name
having count(r.order_id) >=1;
```



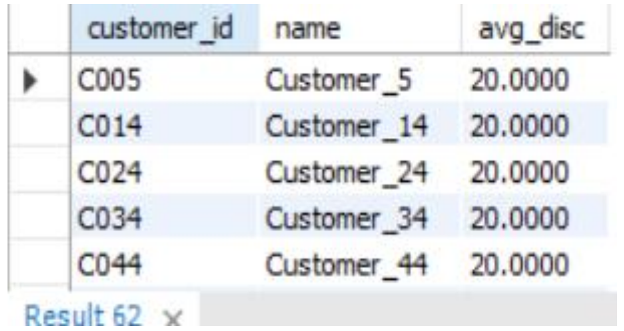
customer_id	name	total_order
C001	Customer_1	1
C004	Customer_4	1
C005	Customer_5	1
C008	Customer_8	1
C011	Customer_11	1

Result 61

5. Find customers with average discount received above 15%?



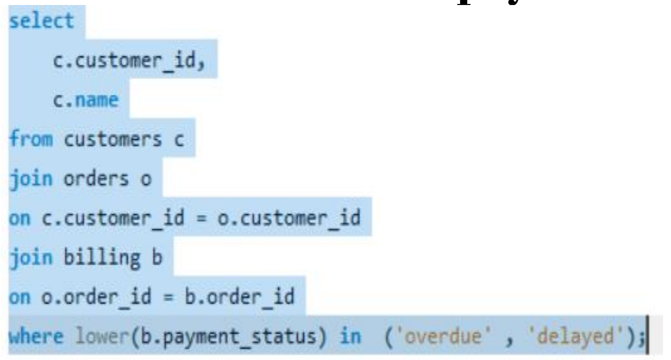
```
select
    c.customer_id,
    c.name,
    avg(d.discount_pct) as avg_dis
from customers c
join orders o
on c.customer_id = o.customer_id
join discounts d
on o.order_id = d.order_id
group by c.customer_id , c.name
having avg(d.discount_pct) > 15;
```



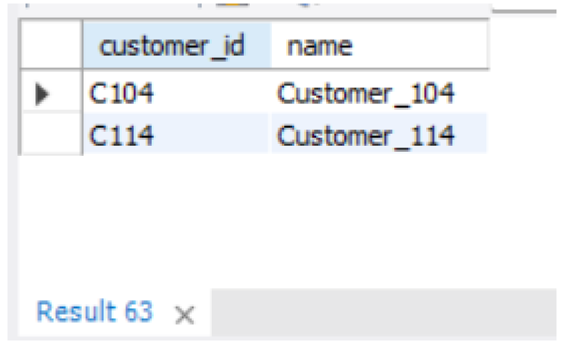
customer_id	name	avg_disc
C005	Customer_5	20.0000
C014	Customer_14	20.0000
C024	Customer_24	20.0000
C034	Customer_34	20.0000
C044	Customer_44	20.0000

Result 62

6. Find customers whose payments are often delayed or overdue?



```
select
    c.customer_id,
    c.name
from customers c
join orders o
on c.customer_id = o.customer_id
join billing b
on o.order_id = b.order_id
where lower(b.payment_status) in ('overdue' , 'delayed');
```

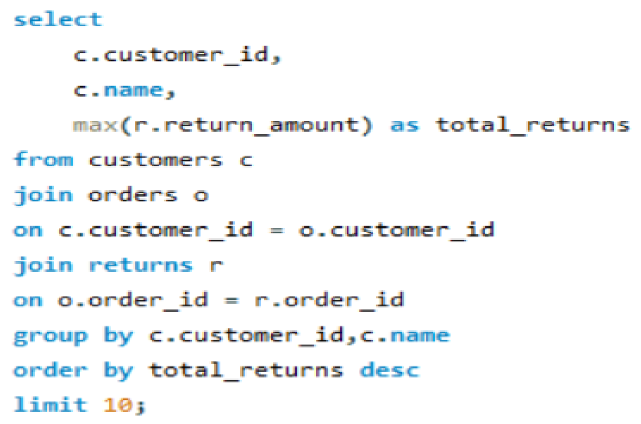


customer_id	name
C104	Customer_104
C114	Customer_114

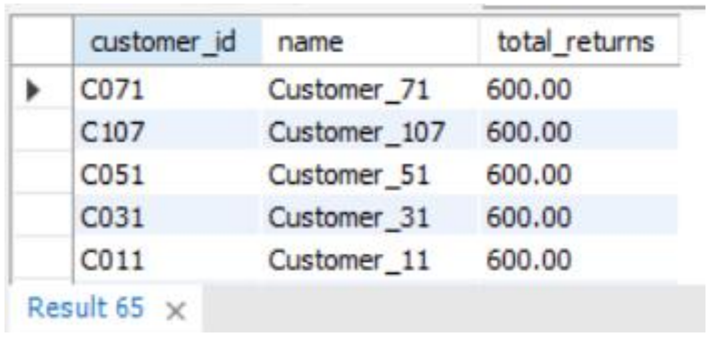
Result 63

Customer Level Analysis

7. Identify customers with the highest total returns (return amount).



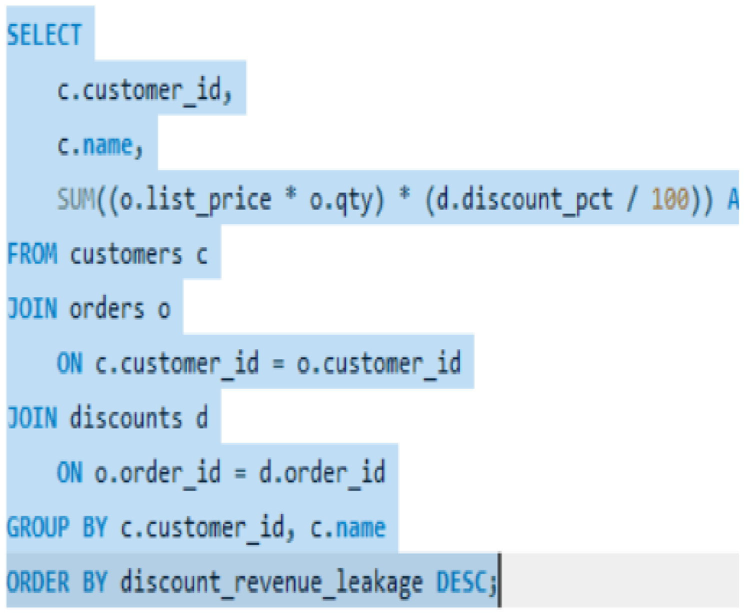
```
select
    c.customer_id,
    c.name,
    max(r.return_amount) as total_returns
from customers c
join orders o
on c.customer_id = o.customer_id
join returns r
on o.order_id = r.order_id
group by c.customer_id,c.name
order by total_returns desc
limit 10;
```



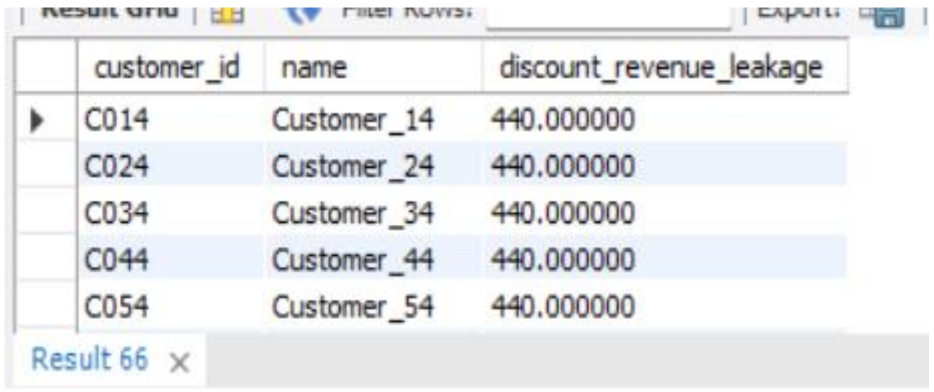
customer_id	name	total_returns
C071	Customer_71	600.00
C107	Customer_107	600.00
C051	Customer_51	600.00
C031	Customer_31	600.00
C011	Customer_11	600.00

Result 65 ×

8. Which customers have the highest Revenue Leakage due to discounts?



```
SELECT
    c.customer_id,
    c.name,
    SUM((o.list_price * o.qty) * (d.discount_pct / 100)) A
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id
JOIN discounts d
    ON o.order_id = d.order_id
GROUP BY c.customer_id, c.name
ORDER BY discount_revenue_leakage DESC;
```

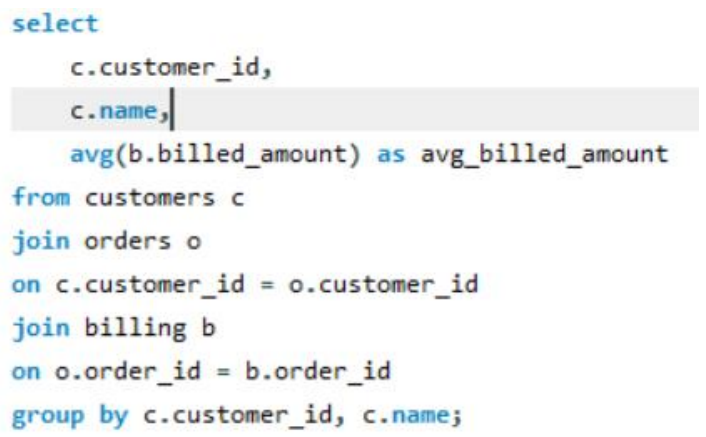


customer_id	name	discount_revenue_leakage
C014	Customer_14	440.000000
C024	Customer_24	440.000000
C034	Customer_34	440.000000
C044	Customer_44	440.000000
C054	Customer_54	440.000000

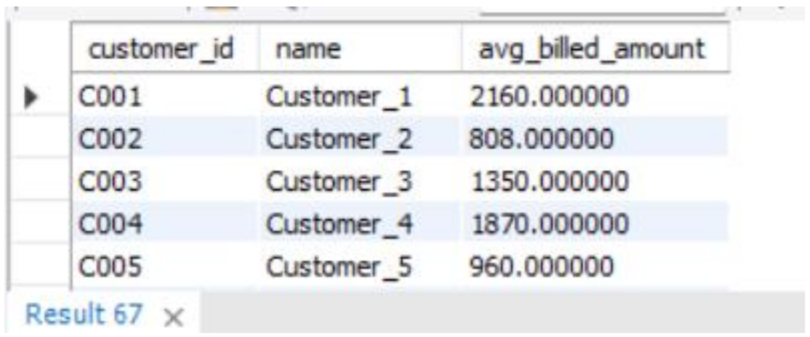
Result 66 ×

Customer Level Analysis

9.Calculate the average billed amount per customer.



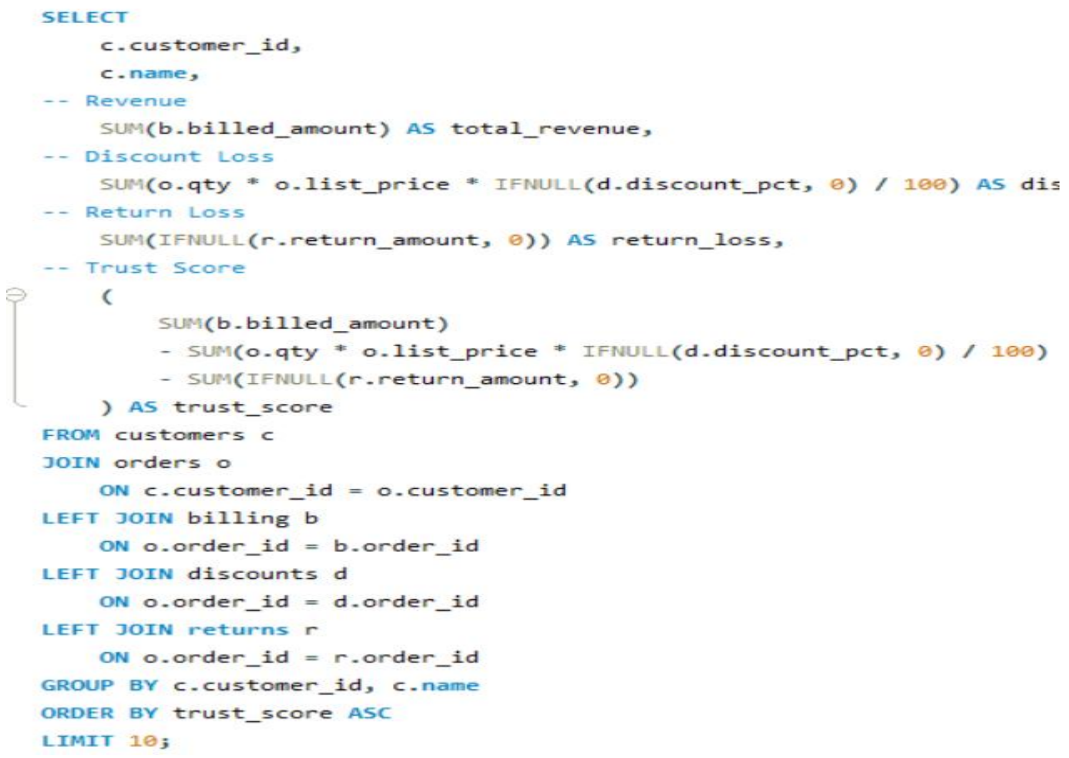
```
select
    c.customer_id,
    c.name,
    avg(b.billed_amount) as avg_billed_amount
from customers c
join orders o
on c.customer_id = o.customer_id
join billing b
on o.order_id = b.order_id
group by c.customer_id, c.name;
```



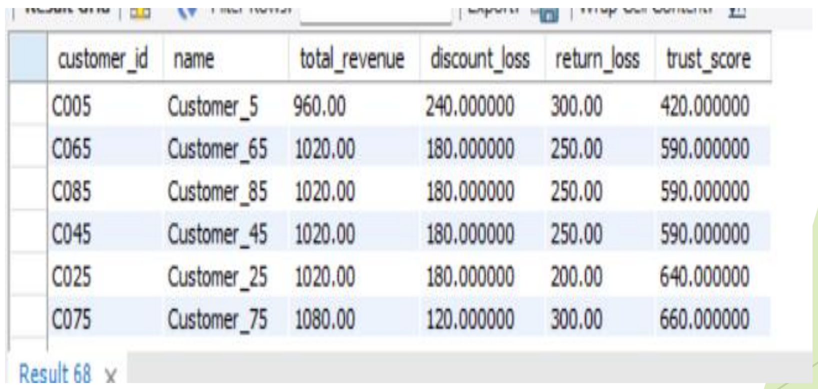
customer_id	name	avg_billed_amount
C001	Customer_1	2160.000000
C002	Customer_2	808.000000
C003	Customer_3	1350.000000
C004	Customer_4	1870.000000
C005	Customer_5	960.000000

Result 67

10.Which customers have the lowest trust score (Revenue - Returns - Discounts)?



```
SELECT
    c.customer_id,
    c.name,
-- Revenue
    SUM(b.billed_amount) AS total_revenue,
-- Discount Loss
    SUM(o.qty * o.list_price * IFNULL(d.discount_pct, 0) / 100) AS dis
-- Return Loss
    SUM(IFNULL(r.return_amount, 0)) AS return_loss,
-- Trust Score
    (
        SUM(b.billed_amount)
        - SUM(o.qty * o.list_price * IFNULL(d.discount_pct, 0) / 100)
        - SUM(IFNULL(r.return_amount, 0))
    ) AS trust_score
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id
LEFT JOIN billing b
    ON o.order_id = b.order_id
LEFT JOIN discounts d
    ON o.order_id = d.order_id
LEFT JOIN returns r
    ON o.order_id = r.order_id
GROUP BY c.customer_id, c.name
ORDER BY trust_score ASC
LIMIT 10;
```

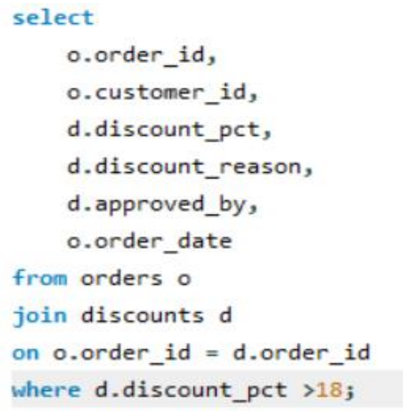


customer_id	name	total_revenue	discount_loss	return_loss	trust_score
C005	Customer_5	960.00	240.000000	300.00	420.000000
C065	Customer_65	1020.00	180.000000	250.00	590.000000
C085	Customer_85	1020.00	180.000000	250.00	590.000000
C045	Customer_45	1020.00	180.000000	250.00	590.000000
C025	Customer_25	1020.00	180.000000	200.00	640.000000
C075	Customer_75	1080.00	120.000000	300.00	660.000000

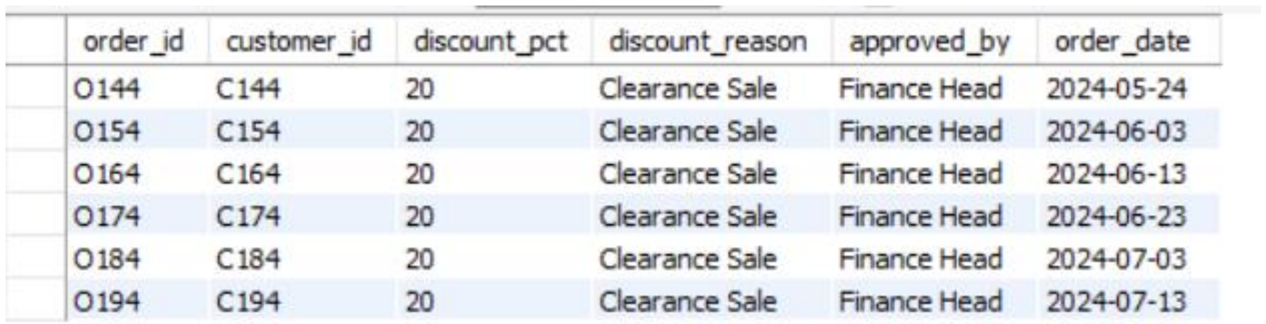
Result 68

Order-Level Analysis

1.List all orders that received discounts greater than 20%.

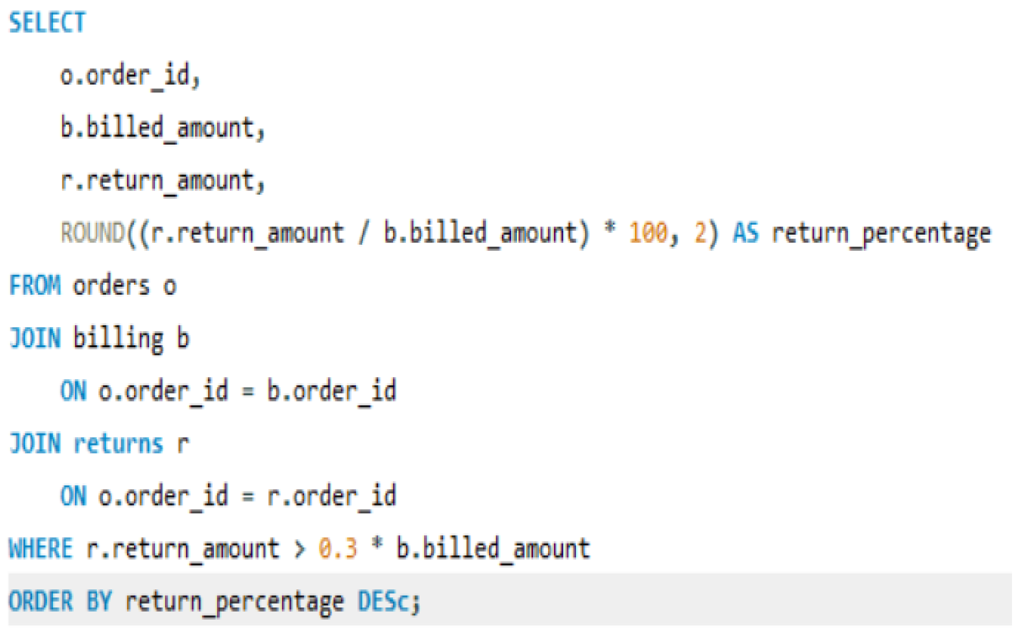


```
select
    o.order_id,
    o.customer_id,
    d.discount_pct,
    d.discount_reason,
    d.approved_by,
    o.order_date
from orders o
join discounts d
on o.order_id = d.order_id
where d.discount_pct >18;
```

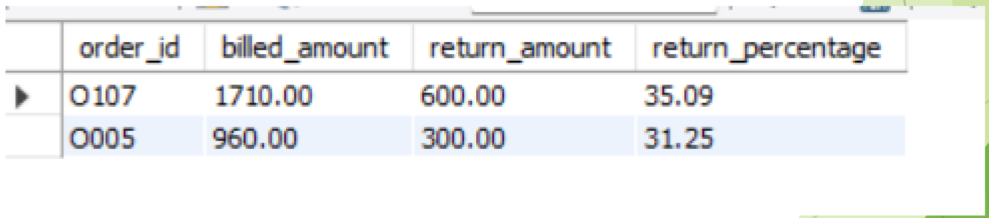


order_id	customer_id	discount_pct	discount_reason	approved_by	order_date
O144	C144	20	Clearance Sale	Finance Head	2024-05-24
O154	C154	20	Clearance Sale	Finance Head	2024-06-03
O164	C164	20	Clearance Sale	Finance Head	2024-06-13
O174	C174	20	Clearance Sale	Finance Head	2024-06-23
O184	C184	20	Clearance Sale	Finance Head	2024-07-03
O194	C194	20	Clearance Sale	Finance Head	2024-07-13

2.Identify orders with returns exceeding 50% of billed amount.



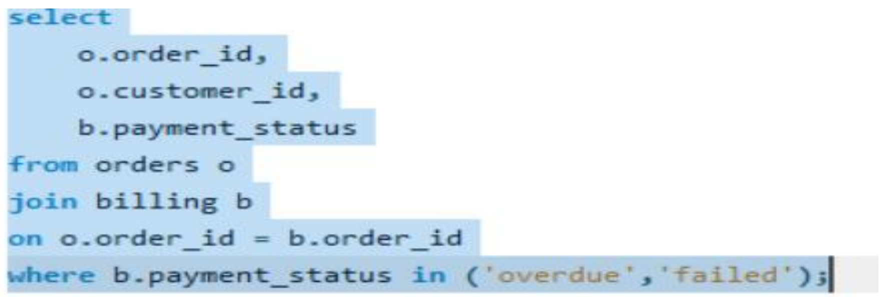
```
SELECT
    o.order_id,
    b.billed_amount,
    r.return_amount,
    ROUND((r.return_amount / b.billed_amount) * 100, 2) AS return_percentage
FROM orders o
JOIN billing b
    ON o.order_id = b.order_id
JOIN returns r
    ON o.order_id = r.order_id
WHERE r.return_amount > 0.3 * b.billed_amount
ORDER BY return_percentage DESC;
```



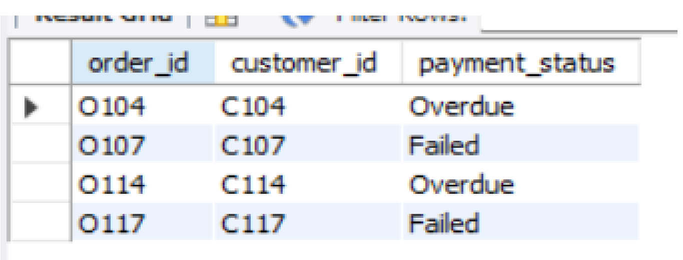
order_id	billed_amount	return_amount	return_percentage
O107	1710.00	600.00	35.09
O005	960.00	300.00	31.25

Order Level Analysis

3.Which orders have payment status as 'Overdue' or 'Failed'?

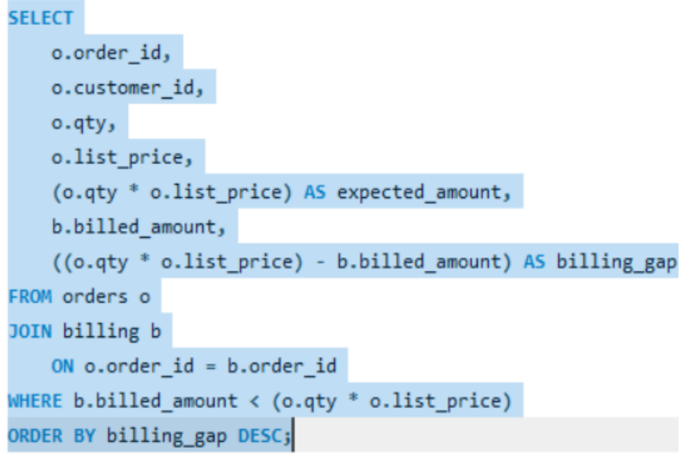


```
select
    o.order_id,
    o.customer_id,
    b.payment_status
from orders o
join billing b
on o.order_id = b.order_id
where b.payment_status in ('overdue','failed');
```

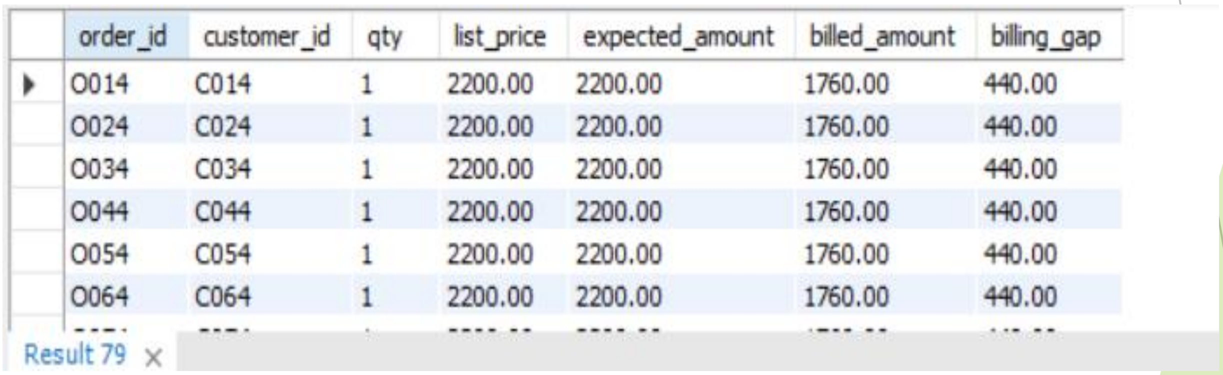


order_id	customer_id	payment_status
O104	C104	Overdue
O107	C107	Failed
O114	C114	Overdue
O117	C117	Failed

4.Find orders where billed amount is less than list price × qty (Price vs Billing Gap).



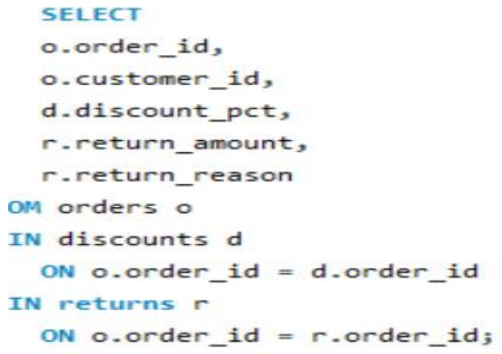
```
SELECT
    o.order_id,
    o.customer_id,
    o.qty,
    o.list_price,
    (o.qty * o.list_price) AS expected_amount,
    b.billed_amount,
    ((o.qty * o.list_price) - b.billed_amount) AS billing_gap
FROM orders o
JOIN billing b
    ON o.order_id = b.order_id
WHERE b.billed_amount < (o.qty * o.list_price)
ORDER BY billing_gap DESC;
```



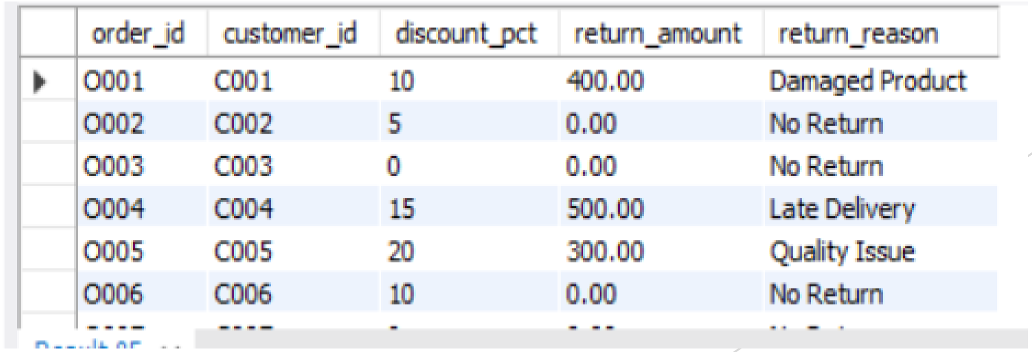
order_id	customer_id	qty	list_price	expected_amount	billed_amount	billing_gap
O014	C014	1	2200.00	2200.00	1760.00	440.00
O024	C024	1	2200.00	2200.00	1760.00	440.00
O034	C034	1	2200.00	2200.00	1760.00	440.00
O044	C044	1	2200.00	2200.00	1760.00	440.00
O054	C054	1	2200.00	2200.00	1760.00	440.00
O064	C064	1	2200.00	2200.00	1760.00	440.00

Result 79 ×

5. List orders that received both a discount and had a return.



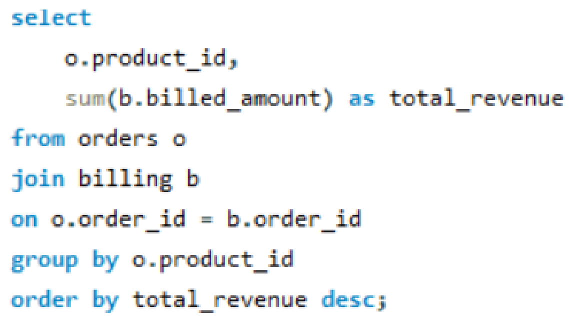
```
    SELECT
    o.order_id,
    o.customer_id,
    d.discount_pct,
    r.return_amount,
    r.return_reason
OM orders o
IN discounts d
    ON o.order_id = d.order_id
IN returns r
    ON o.order_id = r.order_id;
```



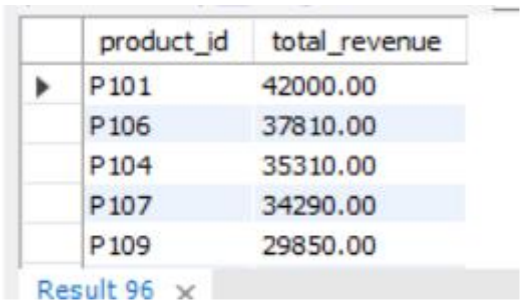
order_id	customer_id	discount_pct	return_amount	return_reason
O001	C001	10	400.00	Damaged Product
O002	C002	5	0.00	No Return
O003	C003	0	0.00	No Return
O004	C004	15	500.00	Late Delivery
O005	C005	20	300.00	Quality Issue
O006	C006	10	0.00	No Return

Products Level Analysis

1. Which products generated the most revenue?



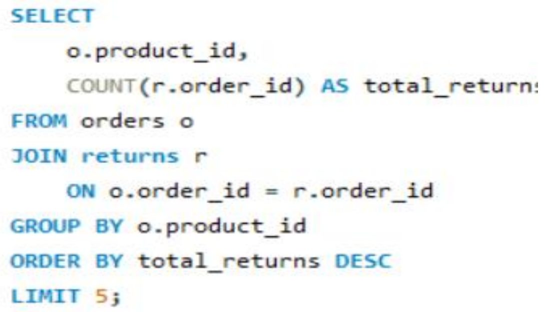
```sql
select
    o.product_id,
    sum(b.billed_amount) as total_revenue
from orders o
join billing b
on o.order_id = b.order_id
group by o.product_id
order by total_revenue desc;
```



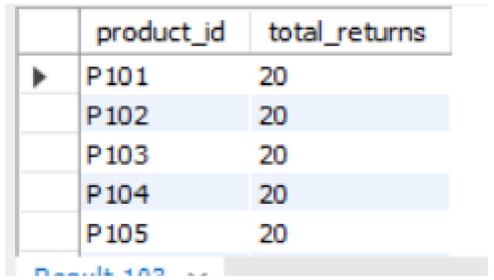
product_id	total_revenue
P101	42000.00
P106	37810.00
P104	35310.00
P107	34290.00
P109	29850.00

Result 96 ×

2. Top 5 products with most returns.

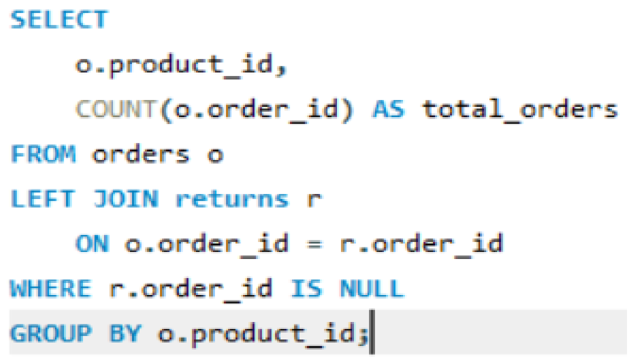


```sql
SELECT
    o.product_id,
    COUNT(r.order_id) AS total_return
FROM orders o
JOIN returns r
    ON o.order_id = r.order_id
GROUP BY o.product_id
ORDER BY total_returns DESC
LIMIT 5;
```

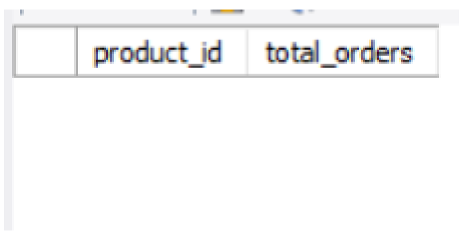


product_id	total_returns
P101	20
P102	20
P103	20
P104	20
P105	20

3. Find the products with zero returns.



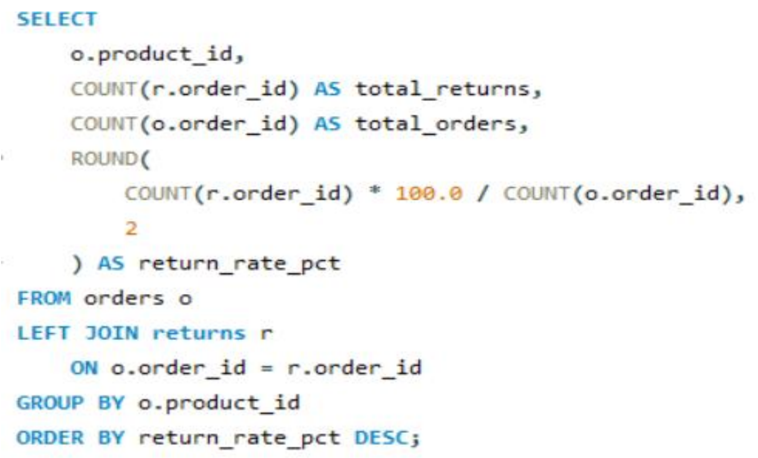
```sql
SELECT
    o.product_id,
    COUNT(o.order_id) AS total_orders
FROM orders o
LEFT JOIN returns r
    ON o.order_id = r.order_id
WHERE r.order_id IS NULL
GROUP BY o.product_id;
```



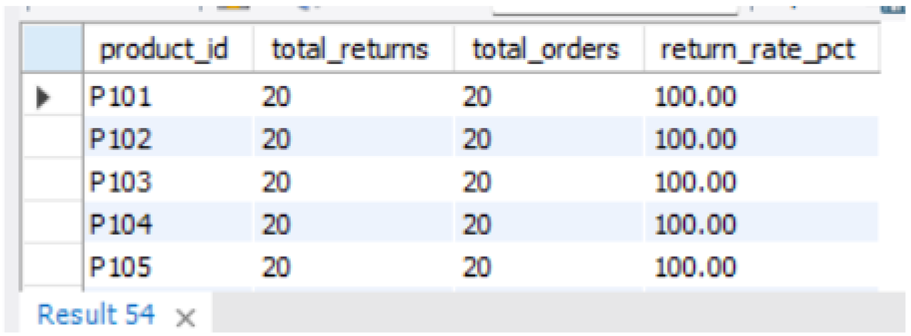
product_id	total_orders

Products Level Analysis

4. Which products have the highest return rate?



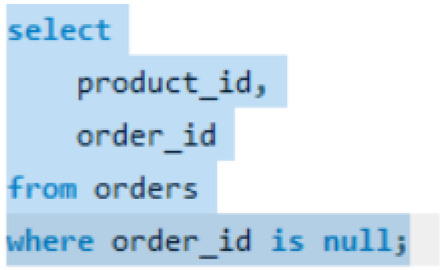
```
SELECT
    o.product_id,
    COUNT(r.order_id) AS total_returns,
    COUNT(o.order_id) AS total_orders,
    ROUND(
        COUNT(r.order_id) * 100.0 / COUNT(o.order_id),
        2
    ) AS return_rate_pct
FROM orders o
LEFT JOIN returns r
    ON o.order_id = r.order_id
GROUP BY o.product_id
ORDER BY return_rate_pct DESC;
```



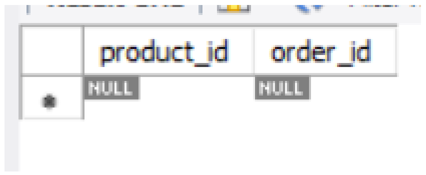
product_id	total_returns	total_orders	return_rate_pct
P101	20	20	100.00
P102	20	20	100.00
P103	20	20	100.00
P104	20	20	100.00
P105	20	20	100.00

Result 54

5. Which products were never sold?



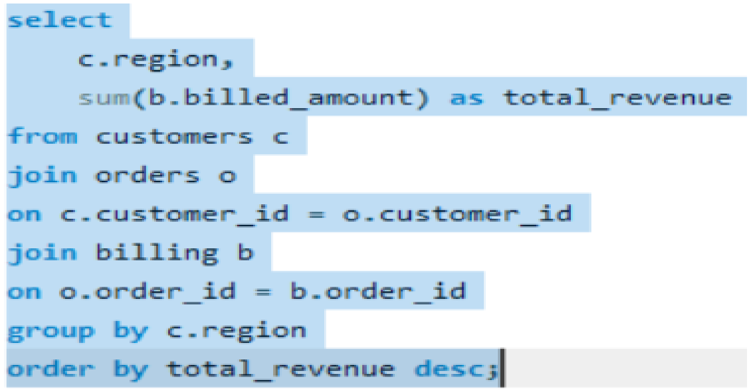
```
select
    product_id,
    order_id
from orders
where order_id is null;
```



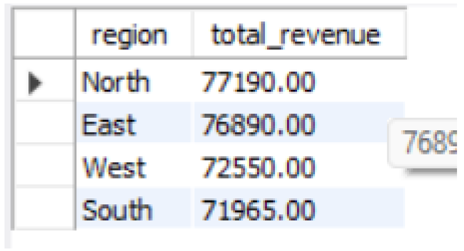
product_id	order_id
NULL	NULL

Region / Segment Analysis

1. **Which regions generate the most revenue?**



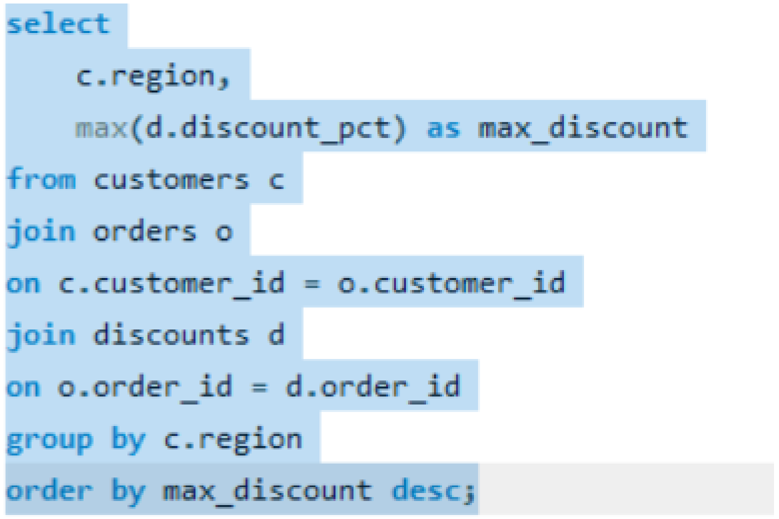
```
select
    c.region,
    sum(b.billed_amount) as total_revenue
from customers c
join orders o
on c.customer_id = o.customer_id
join billing b
on o.order_id = b.order_id
group by c.region
order by total_revenue desc;
```



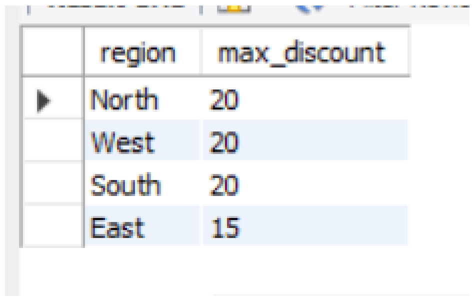
region	total_revenue
North	77190.00
East	76890.00
West	72550.00
South	71965.00

7689

2. **Which regions have the highest total discounts given?**



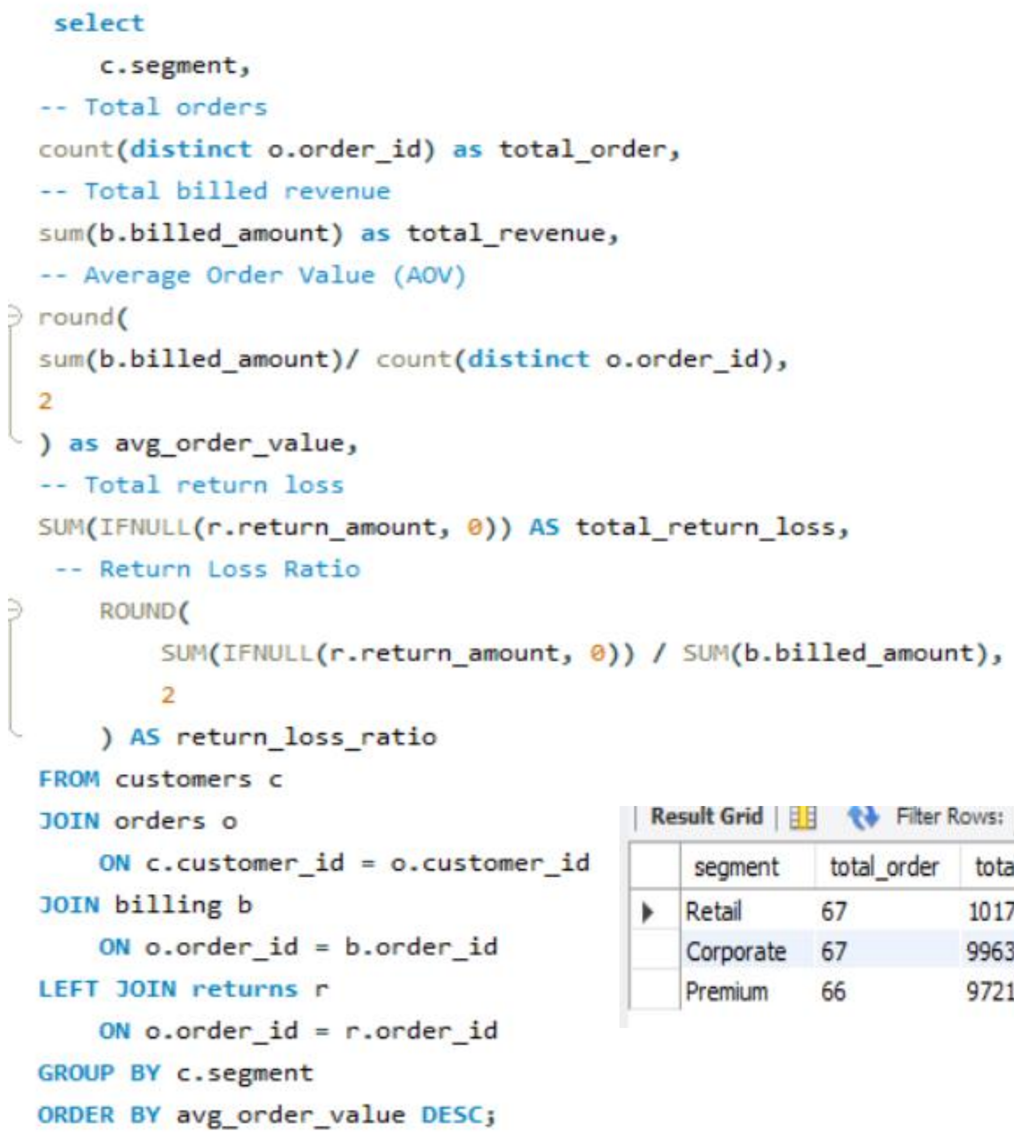
```
select
    c.region,
    max(d.discount_pct) as max_discount
from customers c
join orders o
on c.customer_id = o.customer_id
join discounts d
on o.order_id = d.order_id
group by c.region
order by max_discount desc;
```



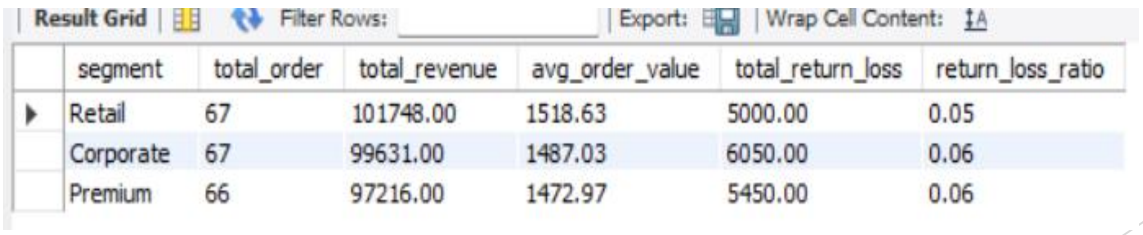
region	max_discount
North	20
West	20
South	20
East	15

Region / Segment Analysis

3. Segment-wise average order value and return loss ratio.



```
 select
    c.segment,
-- Total orders
count(distinct o.order_id) as total_order,
-- Total billed revenue
sum(b.billed_amount) as total_revenue,
-- Average Order Value (AOV)
round(
sum(b.billed_amount)/ count(distinct o.order_id),
2
) as avg_order_value,
-- Total return loss
SUM(IFNULL(r.return_amount, 0)) AS total_return_loss,
 -- Return Loss Ratio
    ROUND(
        SUM(IFNULL(r.return_amount, 0)) / SUM(b.billed_amount),
        2
    ) AS return_loss_ratio
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id
JOIN billing b
    ON o.order_id = b.order_id
LEFT JOIN returns r
    ON o.order_id = r.order_id
GROUP BY c.segment
ORDER BY avg_order_value DESC;
```

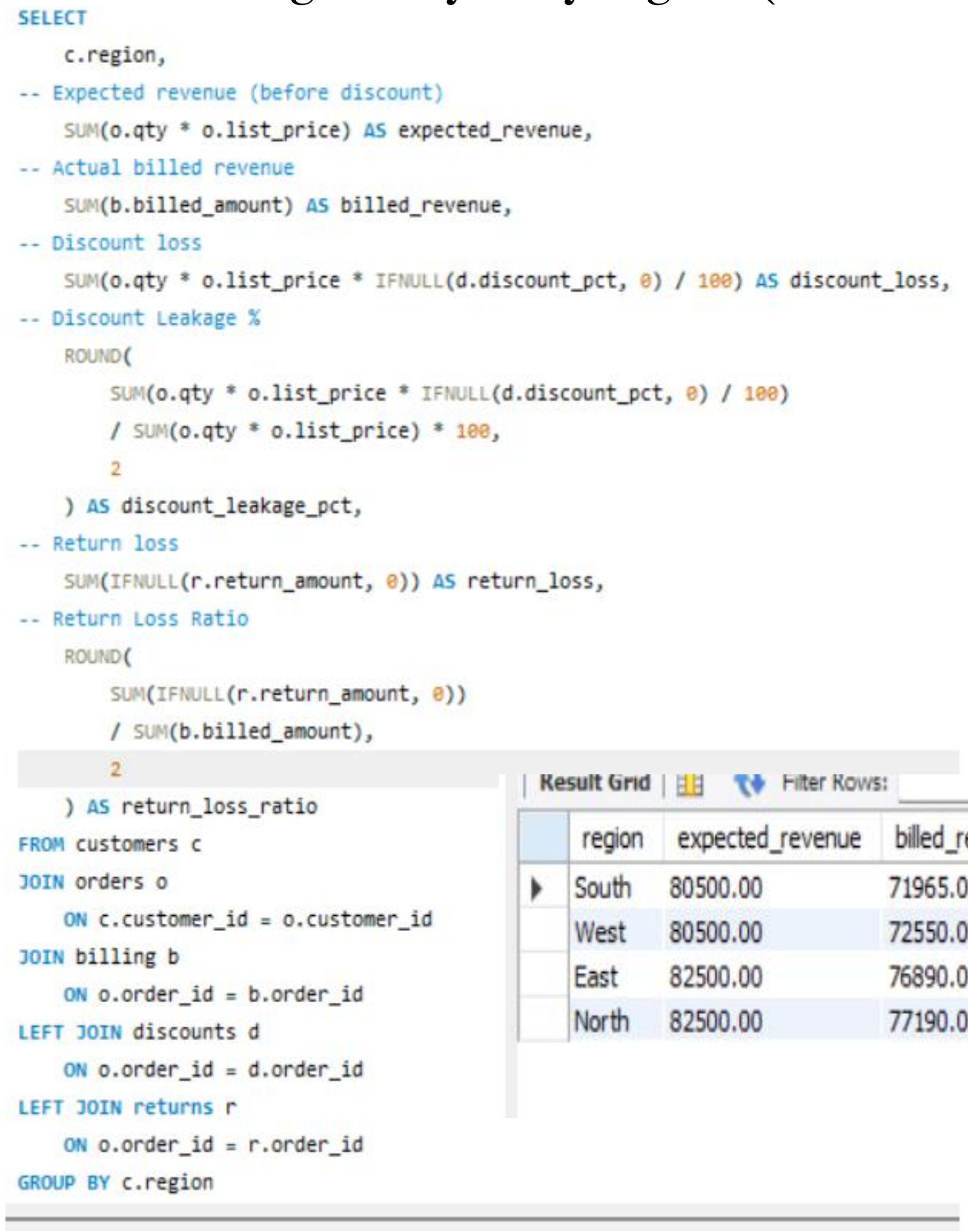


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

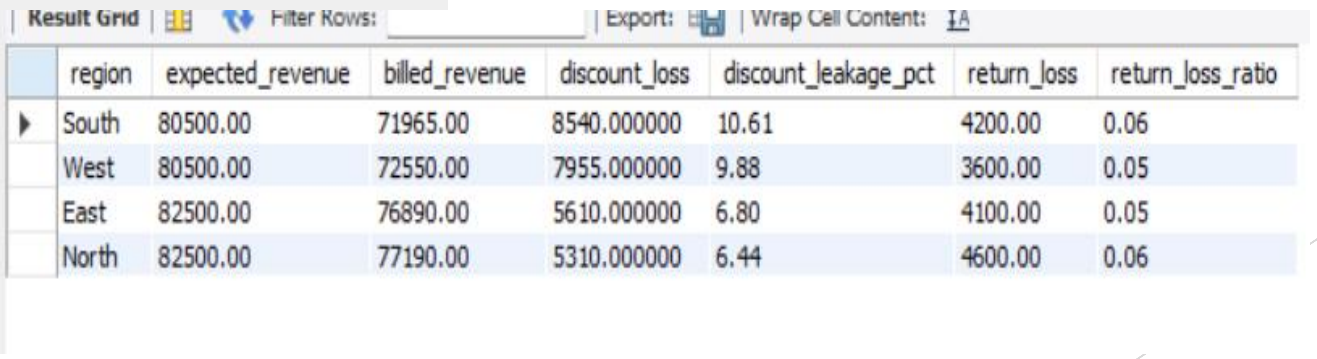
segment	total_order	total_revenue	avg_order_value	total_return_loss	return_loss_ratio
Retail	67	101748.00	1518.63	5000.00	0.05
Corporate	67	99631.00	1487.03	6050.00	0.06
Premium	66	97216.00	1472.97	5450.00	0.06

Region / Segment Analysis

4.Revenue leakage analysis by region (Discount Leakage %, Return Loss Ratio).



```sql
SELECT
    c.region,
-- Expected revenue (before discount)
    SUM(o.qty * o.list_price) AS expected_revenue,
-- Actual billed revenue
    SUM(b.billed_amount) AS billed_revenue,
-- Discount loss
    SUM(o.qty * o.list_price * IFNULL(d.discount_pct, 0) / 100) AS discount_loss,
-- Discount Leakage %
    ROUND(
        SUM(o.qty * o.list_price * IFNULL(d.discount_pct, 0) / 100)
        / SUM(o.qty * o.list_price) * 100,
        2
    ) AS discount_leakage_pct,
-- Return loss
    SUM(IFNULL(r.return_amount, 0)) AS return_loss,
-- Return Loss Ratio
    ROUND(
        SUM(IFNULL(r.return_amount, 0))
        / SUM(b.billed_amount),
        2
    ) AS return_loss_ratio
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id
JOIN billing b
    ON o.order_id = b.order_id
LEFT JOIN discounts d
    ON o.order_id = d.order_id
LEFT JOIN returns r
    ON o.order_id = r.order_id
GROUP BY c.region
```



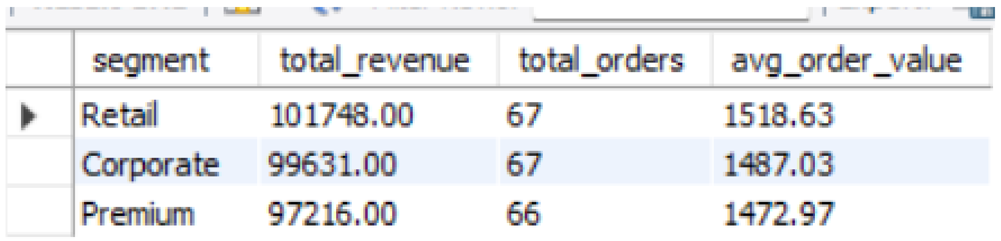
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

region	expected_revenue	billed_revenue	discount_loss	discount_leakage_pct	return_loss	return_loss_ratio
South	80500.00	71965.00	8540.000000	10.61	4200.00	0.06
West	80500.00	72550.00	7955.000000	9.88	3600.00	0.05
East	82500.00	76890.00	5610.000000	6.80	4100.00	0.05
North	82500.00	77190.00	5310.000000	6.44	4600.00	0.06

Region / Segment Analysis

5.Which segments have highest average order value?

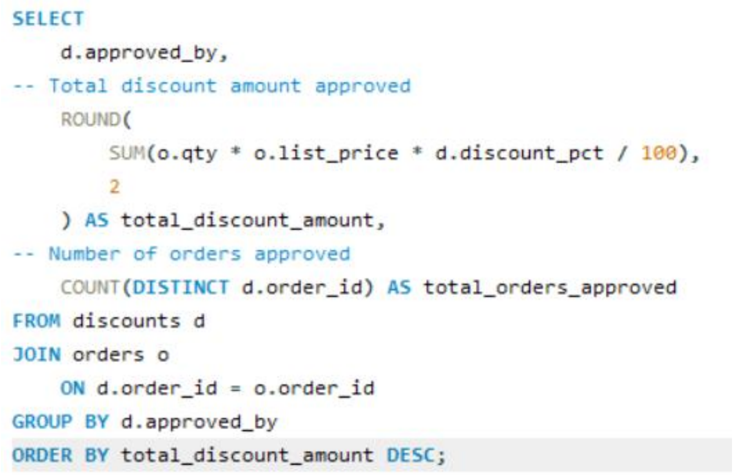
```
SELECT
    c.segment,
-- Total revenue
    SUM(b.billed_amount) AS total_revenue,
-- Total orders
    COUNT(DISTINCT o.order_id) AS total_orders,
-- Average Order Value
    ROUND(
        SUM(b.billed_amount) / COUNT(DISTINCT o.order_id),
        2
    ) AS avg_order_value
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id
JOIN billing b
    ON o.order_id = b.order_id
GROUP BY c.segment
ORDER BY avg_order_value DESC;
```



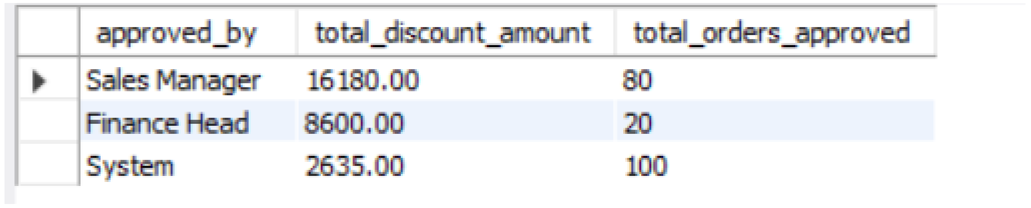
segment	total_revenue	total_orders	avg_order_value
Retail	101748.00	67	1518.63
Corporate	99631.00	67	1487.03
Premium	97216.00	66	1472.97

Discounts / Approver Analysis

1. **Find the total discount given by each approver.**

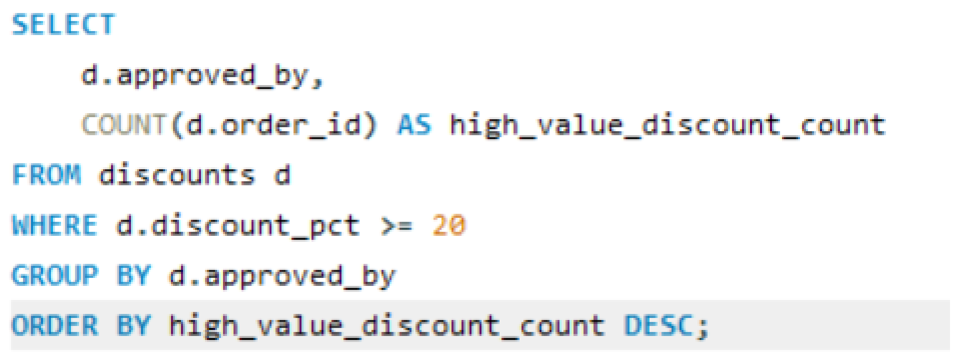


```sql
SELECT
    d.approved_by,
-- Total discount amount approved
    ROUND(
        SUM(o.qty * o.list_price * d.discount_pct / 100),
        2
    ) AS total_discount_amount,
-- Number of orders approved
    COUNT(DISTINCT d.order_id) AS total_orders_approved
FROM discounts d
JOIN orders o
    ON d.order_id = o.order_id
GROUP BY d.approved_by
ORDER BY total_discount_amount DESC;
```

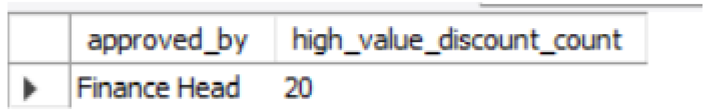


approved_by	total_discount_amount	total_orders_approved
Sales Manager	16180.00	80
Finance Head	8600.00	20
System	2635.00	100

2. **Which approvers gave the maximum number of high-value discounts (>=20%)?**



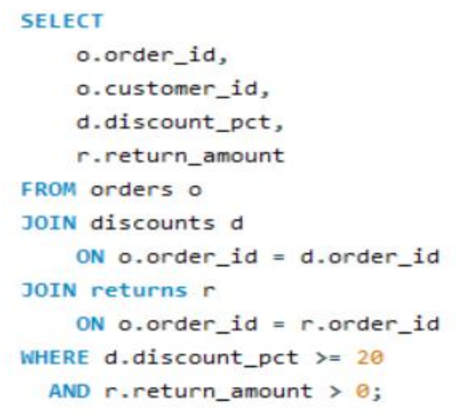
```sql
SELECT
    d.approved_by,
    COUNT(d.order_id) AS high_value_discount_count
FROM discounts d
WHERE d.discount_pct >= 20
GROUP BY d.approved_by
ORDER BY high_value_discount_count DESC;
```



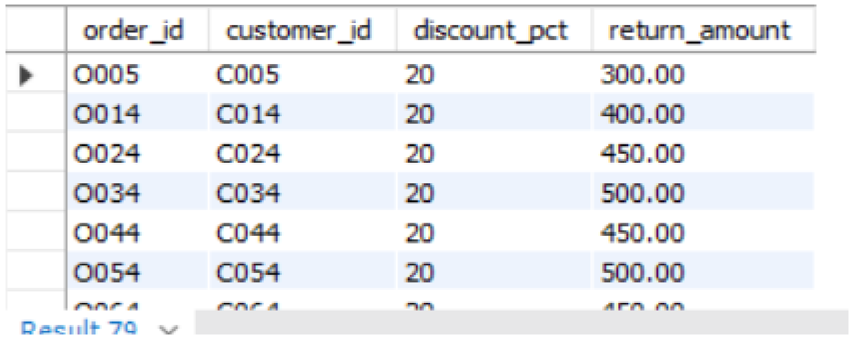
approved_by	high_value_discount_count
Finance Head	20

Discounts / Approver Analysis

3.Find orders where discount pct >= 20% and return amount > 0?

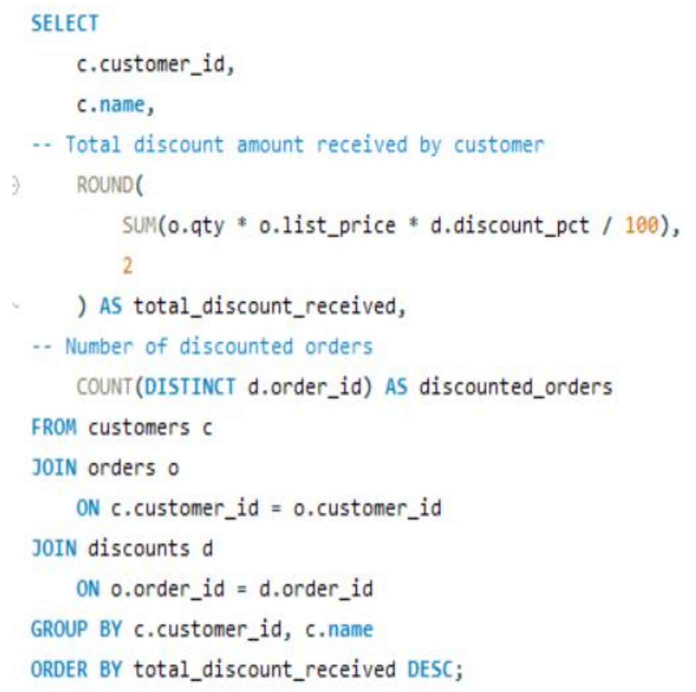


```
SELECT
    o.order_id,
    o.customer_id,
    d.discount_pct,
    r.return_amount
FROM orders o
JOIN discounts d
    ON o.order_id = d.order_id
JOIN returns r
    ON o.order_id = r.order_id
WHERE d.discount_pct >= 20
  AND r.return_amount > 0;
```

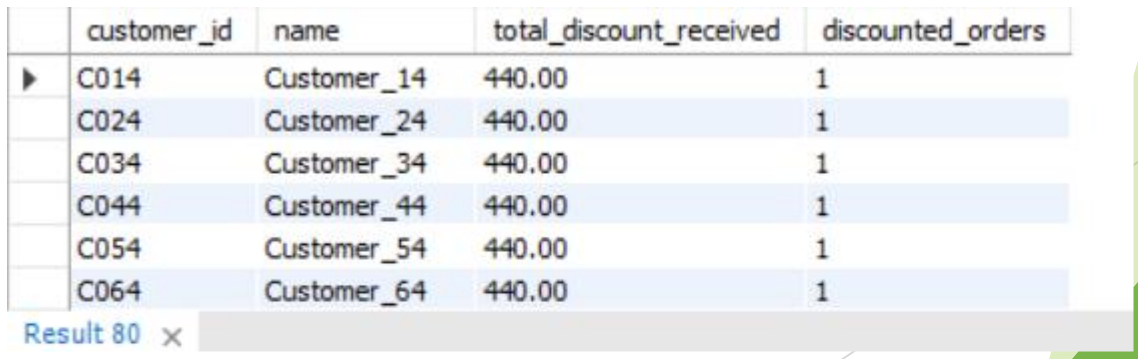


order_id	customer_id	discount_pct	return_amount
O005	C005	20	300.00
O014	C014	20	400.00
O024	C024	20	450.00
O034	C034	20	500.00
O044	C044	20	450.00
O054	C054	20	500.00
O064	C064	20	450.00

4. Identify customers who received the most total discounts?



```
SELECT
    c.customer_id,
    c.name,
-- Total discount amount received by customer
    ROUND(
        SUM(o.qty * o.list_price * d.discount_pct / 100),
        2
    ) AS total_discount_received,
-- Number of discounted orders
    COUNT(DISTINCT d.order_id) AS discounted_orders
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id
JOIN discounts d
    ON o.order_id = d.order_id
GROUP BY c.customer_id, c.name
ORDER BY total_discount_received DESC;
```



customer_id	name	total_discount_received	discounted_orders
C014	Customer_14	440.00	1
C024	Customer_24	440.00	1
C034	Customer_34	440.00	1
C044	Customer_44	440.00	1
C054	Customer_54	440.00	1
C064	Customer_64	440.00	1

Conclusion

This project demonstrates real-world financial analytics skills by identifying hidden revenue losses and providing actionable insights to improve profitability and governance.