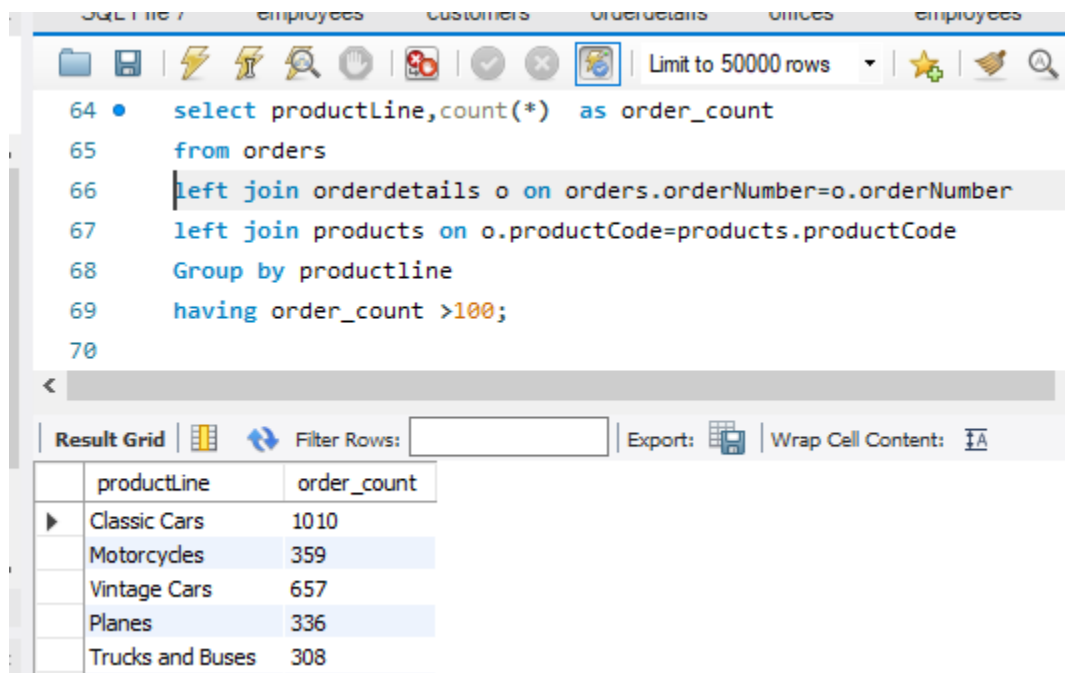


1. Order count for each productline where orders are more than 100 (Hint: Use Having)

```
select productLine,count(*) as order_count
from orders
left join orderdetails o on orders.orderNumber=o.orderNumber
left join products on o.productCode=products.productCode
Group by productline
having order_count >100;
```

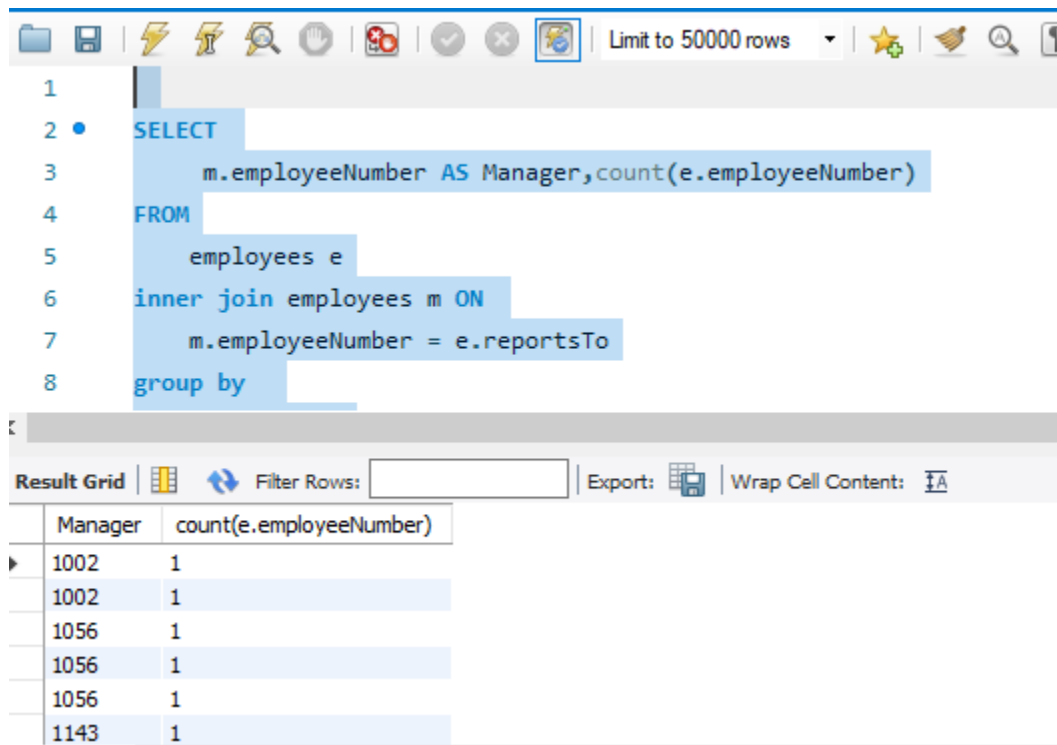


The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 50000 rows' dropdown. The SQL editor contains the query from the previous block. Below the editor, the 'Result Grid' tab is active, displaying the query results in a table. The table has two columns: 'productLine' and 'order_count'. The results are as follows:

productLine	order_count
Classic Cars	1010
Motorcycles	359
Vintage Cars	657
Planes	336
Trucks and Buses	308

2. Count of employees against each manager name (Hint: Use Self Join)

```
SELECT  
    m.employeeNumber AS Manager,count(e.employeeNumber)  
FROM  
    employees e  
inner join employees m ON  
    m.employeeNumber = e.reportsTo  
group by  
e.employeeNumber
```



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 50000 rows' dropdown. The SQL editor contains the following query:

```
1  
2 • SELECT  
3     m.employeeNumber AS Manager,count(e.employeeNumber)  
4 FROM  
5     employees e  
6 inner join employees m ON  
7     m.employeeNumber = e.reportsTo  
8 group by
```

Below the editor is the 'Result Grid' section, which includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are displayed in a table with two columns: 'Manager' and 'count(e.employeeNumber)'.

Manager	count(e.employeeNumber)
1002	1
1002	1
1056	1
1056	1
1056	1
1143	1

3. For each city, create individual columns of order count by each Order Status available in Database (Hint: Use CASE)

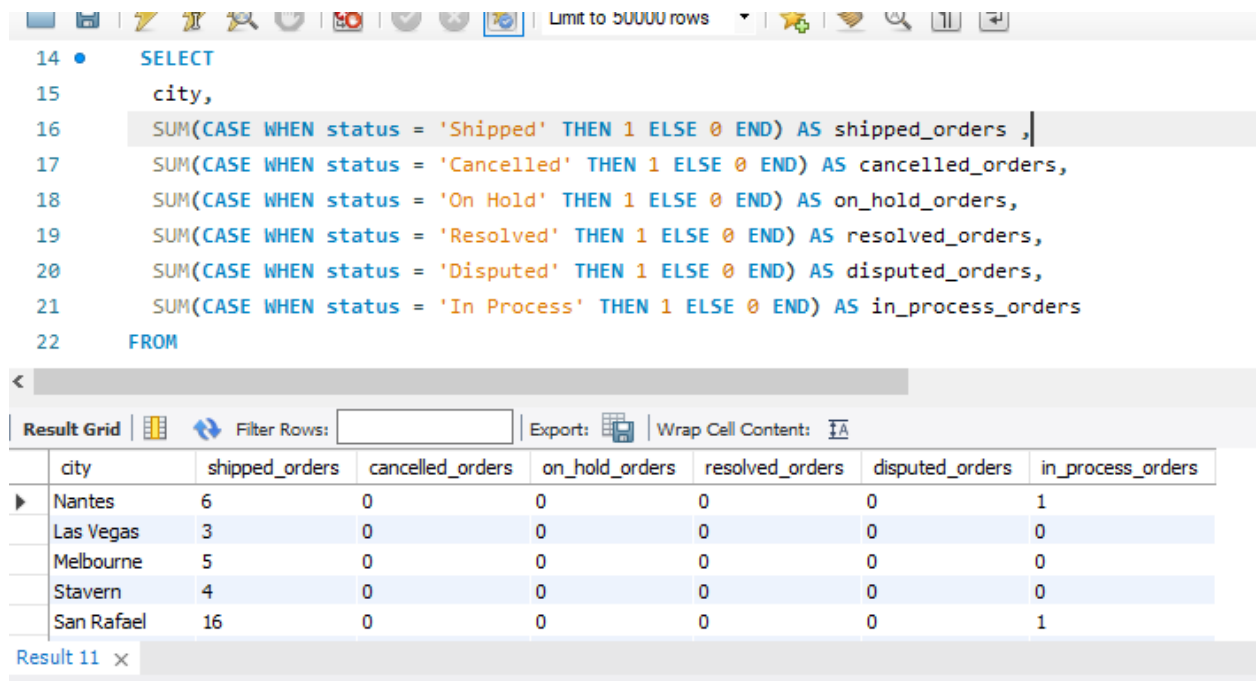
```
SELECT
    city,
    SUM(CASE WHEN status = 'Shipped' THEN 1 ELSE 0 END) AS shipped_orders ,
    SUM(CASE WHEN status = 'Cancelled' THEN 1 ELSE 0 END) AS cancelled_orders,
    SUM(CASE WHEN status = 'On Hold' THEN 1 ELSE 0 END) AS on_hold_orders,
    SUM(CASE WHEN status = 'Resolved' THEN 1 ELSE 0 END) AS resolved_orders,
    SUM(CASE WHEN status = 'Disputed' THEN 1 ELSE 0 END) AS disputed_orders,
    SUM(CASE WHEN status = 'In Process' THEN 1 ELSE 0 END) AS in_process_orders
```

FROM

orders

right JOIN customers ON orders.customerNumber = customers.customerNumber

GROUP BY city;



The screenshot shows a SQL IDE interface. At the top, there's a toolbar with various icons and a dropdown menu set to 'Limit to 50000 rows'. Below the toolbar, a SQL query is entered in a text area. The query is as follows:

```
14 • SELECT
15     city,
16     SUM(CASE WHEN status = 'Shipped' THEN 1 ELSE 0 END) AS shipped_orders ,
17     SUM(CASE WHEN status = 'Cancelled' THEN 1 ELSE 0 END) AS cancelled_orders,
18     SUM(CASE WHEN status = 'On Hold' THEN 1 ELSE 0 END) AS on_hold_orders,
19     SUM(CASE WHEN status = 'Resolved' THEN 1 ELSE 0 END) AS resolved_orders,
20     SUM(CASE WHEN status = 'Disputed' THEN 1 ELSE 0 END) AS disputed_orders,
21     SUM(CASE WHEN status = 'In Process' THEN 1 ELSE 0 END) AS in_process_orders
22 FROM
```

Below the query editor, there's a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The result grid displays the following data:

city	shipped_orders	cancelled_orders	on_hold_orders	resolved_orders	disputed_orders	in_process_orders
Nantes	6	0	0	0	0	1
Las Vegas	3	0	0	0	0	0
Melbourne	5	0	0	0	0	0
Stavern	4	0	0	0	0	0
San Rafael	16	0	0	0	0	1

At the bottom of the result grid, it says 'Result 11' with a close button (X).

4. For each office total order sold (Using Multiple Joins)

```
SELECT o.city,o.country, SUM(od.quantityOrdered)

FROM offices o

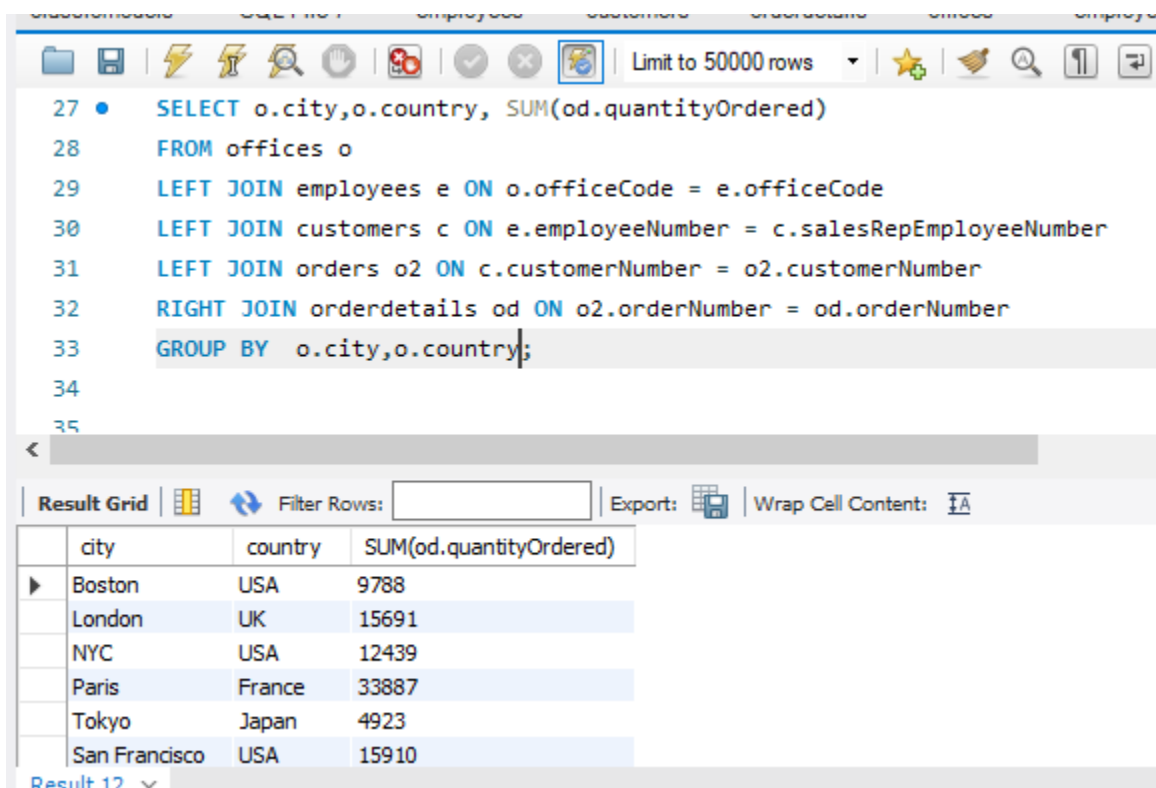
LEFT JOIN employees e ON o.officeCode = e.officeCode

LEFT JOIN customers c ON e.employeeNumber = c.salesRepEmployeeNumber

LEFT JOIN orders o2 ON c.customerNumber = o2.customerNumber

RIGHT JOIN orderdetails od ON o2.orderNumber = od.orderNumber

GROUP BY o.city,o.country;
```



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 50000 rows' dropdown. The SQL editor contains the following query:

```
27 • SELECT o.city,o.country, SUM(od.quantityOrdered)
28 FROM offices o
29 LEFT JOIN employees e ON o.officeCode = e.officeCode
30 LEFT JOIN customers c ON e.employeeNumber = c.salesRepEmployeeNumber
31 LEFT JOIN orders o2 ON c.customerNumber = o2.customerNumber
32 RIGHT JOIN orderdetails od ON o2.orderNumber = od.orderNumber
33 GROUP BY o.city,o.country;
34
35
```

Below the editor is a 'Result Grid' section with a 'Filter Rows' input and an 'Export' button. The results are displayed in a table with 4 columns: city, country, and SUM(od.quantityOrdered). The table shows 6 rows of data.

city	country	SUM(od.quantityOrdered)
Boston	USA	9788
London	UK	15691
NYC	USA	12439
Paris	France	33887
Tokyo	Japan	4923
San Francisco	USA	15910

The bottom of the interface shows 'Result 12' with a dropdown arrow.

5. For each Employee total order sold (Exclude those Employees which are in USA) (Use Sub-Query in Where)

```
SELECT e.employeeNumber,e.firstName,e.lastName, SUM(od.quantityOrdered)
FROM employees e
LEFT JOIN customers c ON e.employeeNumber = c.salesRepEmployeeNumber
LEFT JOIN orders o ON c.customerNumber = o.customerNumber
RIGHT JOIN orderdetails od ON o.orderNumber = od.orderNumber
WHERE e.employeeNumber NOT IN
```

```
(SELECT employeeNumber FROM employees WHERE officeCode IN
```

```
(SELECT officeCode FROM offices WHERE country = 'USA')
```

```
)
```

```
GROUP BY e.employeeNumber;
```

The screenshot shows a SQL IDE interface with a query editor and a result grid. The query editor contains the following SQL code:

```
34
35 • SELECT e.employeeNumber,e.firstName,e.lastName, SUM(od.quantityOrdered)
36 FROM employees e
37 LEFT JOIN customers c ON e.employeeNumber = c.salesRepEmployeeNumber
38 LEFT JOIN orders o ON c.customerNumber = o.customerNumber
39 RIGHT JOIN orderdetails od ON o.orderNumber = od.orderNumber
40 WHERE e.employeeNumber NOT IN
41 (SELECT employeeNumber FROM employees WHERE officeCode IN
42 (SELECT officeCode FROM offices WHERE country = 'USA')
43 )
44 GROUP BY e.employeeNumber;
```

The result grid displays the following data:

employeeNumber	firstName	lastName	SUM(od.quantityOrdered)
1504	Barry	Jones	7486
1370	Gerard	Hernandez	14231
1401	Pamela	Castillo	9290
1621	Mami	Nishi	4923
1501	Larry	Bott	8205
1337	Loui	Bondur	6186
1702	Martin	Corrad	4180

The interface also shows a toolbar with various icons, a 'Limit to 50000 rows' dropdown, and a 'Result Grid' tab. The 'Output' tab is also visible at the bottom.



6. 2nd highest selling product for each Productline. (Use Window Function & CTE Approach)

```
WITH ranked_products AS (  
    SELECT productLine, productName, total_sales,  
           ROW_NUMBER() OVER (PARTITION BY productline ORDER BY total_sales DESC) AS sales_rank  
    FROM (  
        SELECT productline, productName, SUM(quantityOrdered * priceEach) AS total_sales  
        FROM products  
        left join orderdetails on orderdetails.productCode=products.productCode  
        GROUP BY productline, productName  
    ) AS sales_by_product  
)  
SELECT productline, productName, total_sales  
FROM ranked_products  
WHERE sales_rank = 2;
```

```

62
63 WITH ranked_products AS (
64     SELECT productline, productName, total_sales,
65            ROW_NUMBER() OVER (PARTITION BY productline ORDER BY total_sales DESC) AS sales_rank
66 FROM (
67     SELECT productline, productName, SUM(quantityOrdered * priceEach) AS total_sales
68     FROM products
69     left join orderdetails on orderdetails.productCode=products.productCode
70     GROUP BY productline, productName
71 ) AS sales_by_product

```

Result Grid			
Filter Rows: <input type="text"/>			
Export: 			
Wrap Cell Content: 			
productline	productName	total_sales	
Classic Cars	2001 Ferrari Enzo	190755.86	
Motorcycles	2002 Suzuki XREO	135767.03	
Planes	ATA: B757-300	102786.38	
Ships	The Titanic	84992.25	
Trains	1950's Chicago Surface Lines Streetcar	53791.99	
Trucks and Buses	1962 Volkswagen Microbus	118774.33	
Vintage Cars	1928 Mercedes-Benz SSK	132275.98	

1. Creating a View (5 Marks):

Create a view "complaints_last_3_months_sum" of Number of Complaints received against the following attributes in last three months use "date_received" column, use the Original Table & Data you have from Airflow Assignment Dump:

1. state
2. product
3. issue
4. sub_product
5. sub_issue

```
CREATE VIEW complaints_last_3_months_sum AS  
  
SELECT  
  
    state,  
  
    product,  
  
    issue,  
  
    sub_product,  
  
    sub_issue,  
  
    COUNT(*) AS num_complaints  
  
FROM complaints  
  
WHERE date_received >= DATE_SUB(CURRENT_DATE, INTERVAL 3 MONTH)  
  
GROUP BY state, product, issue, sub_product, sub_issue;
```

2. Creating a Procedure (10 Marks):

Create a procedure that intakes a Date Parameter and uses it to migrate data from Original table to another table for all the Complaints received in Last 3 Months use “date_received” column to replicate data into another table. Name the table as “complaints_last_3_months”

For Example if the Parameter Date is 28-Feb-2023 then the Data pulled from one table to the another must be between 01-DEC-22 to 28-FEB-23

Table to Migrate from: use the Original Table & Data you have from Airflow Assignment Dump

Table to Migrate to: "complaints_last_3_months"

```
DELIMITER $$
```

```
CREATE PROCEDURE migrate_complaints_last_3_months (p_date DATE)
```

```
BEGIN
```

```
    INSERT INTO complaints_last_3_months
```

```
    SELECT *
```

```
    FROM complaints
```

```
    WHERE date_received >= DATE_SUB(LAST_DAY(DATE_SUB(p_date, INTERVAL 3 MONTH)),  
INTERVAL 3 MONTH)
```

```
    AND date_received <= p_date;
```

```
END$$
```

```
DELIMITER ;
```

3. Creating a Trigger (5 Marks):

Create a trigger that updates "complaints_last_3_months" table with new record whenever the data is inserted in original financial consumer complaints table based on same criteria of Complaints received in Last 3 Months use "date_received" column. (Hint: On Insert Trigger will be used)

```
DELIMITER $$
```

```
CREATE TRIGGER insert_complaints_trigger
```

```
AFTER INSERT ON complaindb.complaints
FOR EACH ROW
BEGIN
    INSERT INTO complaints_last_3_months (complaint_id, date_received, complaints)
    SELECT id, date_received, complaints
    FROM original_complaints_table
    WHERE date_received >= DATE_SUB(SYSDATE(), INTERVAL '3' MONTH);
END$$
DELIMITER ;
```