

## Task 3: SQL for Data Analysis

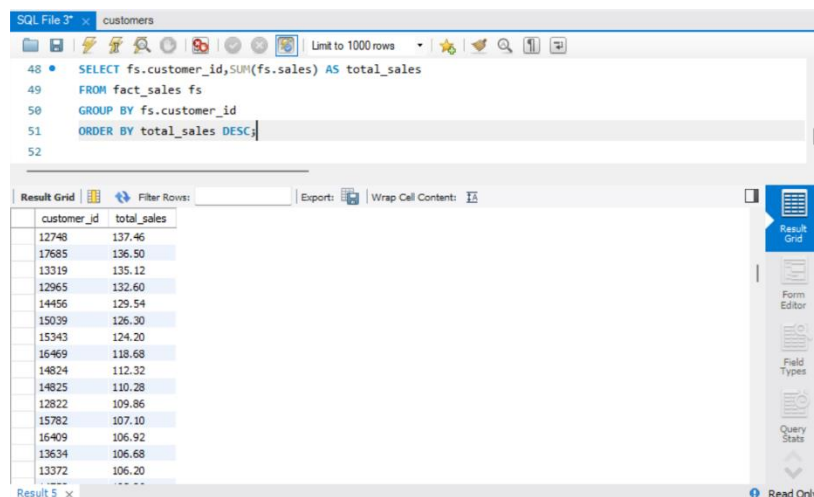
-- 1. Total sales per customer, ordered by total sales:

```
SELECT fs.customer_id, SUM(fs.sales) AS total_sales
```

```
FROM fact_sales fs
```

```
GROUP BY fs.customer_id
```

```
ORDER BY total_sales DESC;
```



The screenshot shows a SQL IDE window titled 'SQL File 3' with a tab 'customers'. The SQL editor contains the following query:

```
48 SELECT fs.customer_id, SUM(fs.sales) AS total_sales
49 FROM fact_sales fs
50 GROUP BY fs.customer_id
51 ORDER BY total_sales DESC;
52
```

Below the editor, the 'Result Grid' displays the results of the query. The table has two columns: 'customer\_id' and 'total\_sales'. The results are ordered by total sales in descending order.

customer_id	total_sales
12740	137.46
17685	136.50
13319	135.12
12965	132.60
14456	129.54
15039	126.30
15343	124.20
16469	118.68
14824	112.32
14825	110.28
12822	109.86
15782	107.10
16409	106.92
13634	106.68
13372	106.20

-- 2. Products sold more than 100 times:

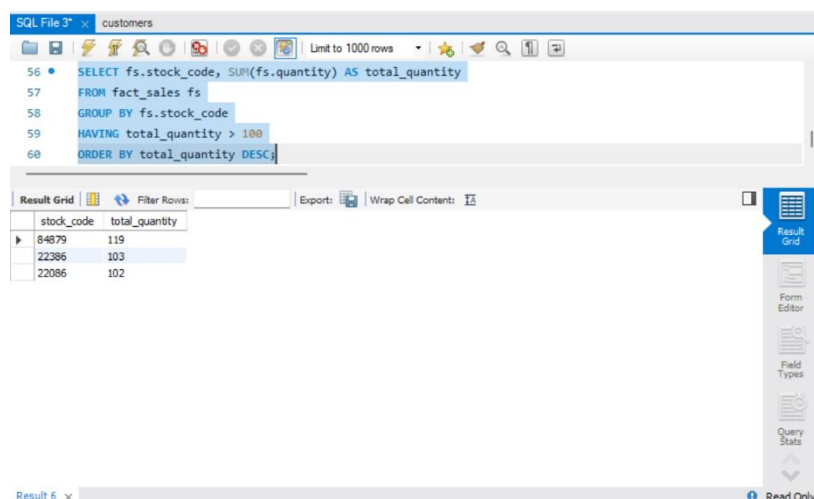
```
SELECT fs.stock_code, SUM(fs.quantity) AS total_quantity
```

```
FROM fact_sales fs
```

```
GROUP BY fs.stock_code
```

```
HAVING total_quantity > 100
```

```
ORDER BY total_quantity DESC;
```



The screenshot shows the same SQL IDE window with the second query entered:

```
56 SELECT fs.stock_code, SUM(fs.quantity) AS total_quantity
57 FROM fact_sales fs
58 GROUP BY fs.stock_code
59 HAVING total_quantity > 100
60 ORDER BY total_quantity DESC;
```

The 'Result Grid' shows the results of this query. The table has two columns: 'stock\_code' and 'total\_quantity'. Only three rows are visible, all with total quantities greater than 100.

stock_code	total_quantity
84879	119
22386	103
22086	102

-- 3. Sales with Product Details

```
SELECT fs.invoice_no, fs.stock_code, dp.description, fs.quantity, fs.sales
```

```
FROM fact_sales fs
```

```
INNER JOIN products dp ON fs.stock_code = dp.stock_code;
```

SQL File 3\* customers

```
-- 3. Sales with Product Details
63 * SELECT fs.invoice_no, fs.stock_code, dp.description, fs.quantity, fs.sales
64 FROM fact_sales fs
65 INNER JOIN products dp ON fs.stock_code = dp.stock_code;
66
```

invoice_no	stock_code	description	quantity	sales
565971	23203	Pet Hair Remover	1	12.48
565971	22197	Sheba Perfect Portions Paté Wet Cat Food	2	10.20
565967	22386	Milk-Bone MaroSnaDogs Dog Treats with Real Bone	4	49.92
565990	850998	Earth Rated Dog Poop Bags	4	49.92
565990	22386	Milk-Bone MaroSnaDogs Dog Treats with Real Bone	2	24.96
565990	23203	Pet Hair Remover	2	24.96
565993	22114	ChomChom Pet Hair Remover - Reusable Cat an...	2	51.00
565995	85123A	Dog and Puppy Pads	1	17.70
565995	20725	Pet Grooming Brush	1	9.90
565995	23298	Purina ONE SmartBlend Natural Adult Chicken 10lb	1	29.70
565995	22960	Rechargeable Pet Nail Grinder	1	25.50
566008	850998	Earth Rated Dog Poop Bags	1	12.48
566008	22386	Milk-Bone MaroSnaDogs Dog Treats with Real Bone	1	12.48
566013	22386	Milk-Bone MaroSnaDogs Dog Treats with Real Bone	2	24.96
566013	23203	Pet Hair Remover	4	49.92
566017	22386	Milk-Bone MaroSnaDogs Dog Treats with Real Bone	2	24.96
566018	850998	Earth Rated Dog Poop Bags	1	12.48

Result 7 x

-- 4. All customers and their regions (even if region data is missing):

```
SELECT dc.customer_id, dc.order_state, srm.region
```

```
FROM customers dc
```

```
LEFT JOIN state_region_mapping srm ON dc.order_state = srm.order_state;
```

SQL File 3\* customers

```
-- 4.All customers and their regions (even if region data is missing):
67 * SELECT dc.customer_id, dc.order_state, srm.region
68 FROM customers dc
69 LEFT JOIN state_region_mapping srm ON dc.order_state = srm.order_state;
70
71
```

customer_id	order_state	region
18287	WY	West
18283	WY	West
18282	WY	West
18281	WY	West
18280	WY	West
18278	WV	East
18277	WV	East
18276	WV	East
18274	WV	East
18273	WV	East
18272	WV	East
18270	WV	East
18269	WV	East
18268	WV	East
18265	WV	East
18263	WV	East
18262	WV	East

Result 8 x

-- 5. All regions and customers from those regions (even if no customer exists):

```
SELECT srm.region, dc.customer_id, dc.order_state
```

```
FROM state_region_mapping srm
```

```
RIGHT JOIN customers dc ON srm.order_state = dc.order_state;
```

SQL File 3\* customers

Limit to 1000 rows

```
-- 5.All regions and customers from those regions (even if no customer exists):
73 SELECT srm.region, dc.customer_id, dc.order_state
74 FROM state_region_mapping srm
75 RIGHT JOIN customers dc ON srm.order_state = dc.order_state;
76
```

Result Grid

region	customer_id	order_state
West	18287	WY
West	18283	WY
West	18282	WY
West	18281	WY
West	18280	WY
East	18278	WV
East	18277	WV
East	18276	WV
East	18274	WV
East	18273	WV
East	18272	WV
East	18270	WV
East	18269	WV
East	18268	WV
East	18265	WV
East	18263	WV
East	18262	WV

Result 9 x Read Only

-- 6. Top 5 customers by sales using a subquery:

```
SELECT * FROM (
    SELECT customer_id, SUM(sales) AS total_sales
    FROM fact_sales
    GROUP BY customer_id
    ORDER BY total_sales DESC LIMIT 5)
AS top_customers;
```

SQL File 3\* customers

Limit to 1000 rows

```
-- 6.Top 5 customers by sales using a subquery:
77
78 SELECT * FROM (
79     SELECT customer_id, SUM(sales) AS total_sales
80     FROM fact_sales
81     GROUP BY customer_id
82     ORDER BY total_sales DESC LIMIT 5)
83 AS top_customers;
84
85
```

Result Grid

customer_id	total_sales
13098	1271.34
13324	591.30
14646	490.92
14110	445.50
18092	377.10

Result 10 x Read Only

-- 7. Find products that have higher-than-average unit price:

```
SELECT DISTINCT fs.stock_code, fs.unit_price
FROM fact_sales fs
WHERE fs.unit_price > (SELECT AVG(unit_price) FROM fact_sales);
```

SQL File 3\* customers

Limit to 1000 rows

```

85 -- 7. Find products that have higher-than-average unit price:
86 • SELECT DISTINCT fs.stock_code, fs.unit_price
87 FROM fact_sales fs
88 WHERE fs.unit_price > (SELECT AVG(unit_price) FROM fact_sales);
89

```

Result Grid

stock_code	unit_price
22114	25.50
23298	29.70
22960	25.50
82484	47.70
22720	29.70
22423	76.50
22423	65.70
79321	34.50
47566	29.70
82484	40.50
22720	25.50
79321	29.70
22086	34.74
22960	49.74
47566	24.90
22502	35.70

fact\_sales 11 x

Read Only

-- 8. Total revenue and average order value:

```

SELECT SUM(sales) AS total_revenue, AVG(sales) AS average_order_value
FROM fact_sales;

```

SQL File 3\* customers

Limit to 1000 rows

```

90 -- 8. Total revenue and average order value:
91 • SELECT SUM(sales) AS total_revenue, AVG(sales) AS average_order_value
92 FROM fact_sales;
93
94

```

Result Grid

total_revenue	average_order_value
19531.74	45.422651

Result 12 x

Read Only

-- 9. Create a view for total sales per customer with region info:

```

CREATE VIEW customer_sales_region AS
SELECT fs.customer_id, SUM(fs.sales) AS total_sales, srm.region
FROM fact_sales fs
JOIN customers dc ON fs.customer_id = dc.customer_id
LEFT JOIN state_region_mapping srm ON dc.order_state = srm.order_state
GROUP BY fs.customer_id, srm.region;

SELECT * FROM customer_sales_region WHERE region = 'west';

```

SQL File 3" customers

```

94 -- 9. Create a view for total sales per customer with region info:
95 • CREATE VIEW customer_sales_region AS
96   SELECT fs.customer_id, SUM(fs.sales) AS total_sales, srm.region
97   FROM fact_sales fs
98   JOIN customers dc ON fs.customer_id = dc.customer_id
99   LEFT JOIN state_region_mapping srm ON dc.order_state = srm.order_state
100  GROUP BY fs.customer_id, srm.region;
101
102 • SELECT * FROM customer_sales_region WHERE region = 'west';

```

Result Grid

customer_id	total_sales	region
18118	82.80	West
18092	1131.30	West
18069	298.62	West
18065	124.20	West
18044	306.00	West
18001	53.10	West
17997	76.50	West
17965	30.60	West
17757	29.70	West
17728	20.28	West
16871	106.20	West

customer\_sales\_region 15 x

Read Only

-- 10. Average quantity sold per product:

SELECT stock\_code, AVG(quantity) AS avg\_quantity

FROM fact\_sales

GROUP BY stock\_code;

SQL File 3" customers

```

104 -- 10. Average quantity sold per product:
105 • SELECT stock_code, AVG(quantity) AS avg_quantity
106   FROM fact_sales
107   GROUP BY stock_code;
108

```

Result Grid

stock_code	avg_quantity
23203	2.5806
22197	3.8095
22386	4.9048
850998	2.3667
22114	1.3913
85123A	2.2895
20725	2.0667
23298	1.0400
22960	1.3333
82484	1.0769
84879	4.2500
22720	1.1786
22423	1.4146
79321	3.4000
47566	1.4815
22086	3.0000

Result 16 x

Read Only