

a. Use SELECT, WHERE, ORDER BY, GROUP BY

```
84 -- Select customers who signed up after Jan 1, 2023:
```

```
85 • SELECT * FROM customers WHERE created_at > '2023-01-01' ORDER BY created_at DESC;
```

```
86
```

result Grid					
Filter Rows:					
Edit:					
Export/Import:					
Wrap Cell Content:					
customer_id	first_name	last_name	email	phone	created_at
2	Bob	Smith	bob@example.com	2345678901	2023-02-15
1	Alice	Johnson	alice@example.com	1234567890	2023-01-10
NULL	NULL	NULL	NULL	NULL	NULL

```
87 -- Total sales per product:
```

```
88 • SELECT
```

```
89     p.product_name,
```

```
90     SUM(oi.quantity) AS total_quantity_sold,
```

```
91     SUM(oi.price * oi.quantity) AS total_sales
```

```
92 FROM order_items oi
```

```
93 JOIN products p ON oi.product_id = p.product_id
```

```
94 GROUP BY p.product_name
```

```
95 ORDER BY total_sales DESC;
```

```
96
```

Result Grid			
Filter Rows:			
Export:			
Wrap Cell Content:			
	product_name	total_quantity_sold	total_sales
▶	Laptop	1	1200.00
	Smartphone	1	500.00
	Keyboard	2	200.00

b. Use JOINS (INNER, LEFT, RIGHT)

```
97      -- INNER JOIN: Orders with customer info
98  •   SELECT
99      o.order_id,
100     c.first_name,
101     c.last_name,
102     o.order_date,
103     o.status
104  FROM orders o
105  INNER JOIN customers c ON o.customer_id = c.customer_id;
106
```

Result Grid					
		Filter Rows:		Export:	Wrap Cell Content:
	order_id	first_name	last_name	order_date	status
▶	1001	Alice	Johnson	2023-03-01	Shipped
	1002	Bob	Smith	2023-03-05	Delivered



```
106
107      -- LEFT JOIN: All customers with their orders (even those without orders)
108  •   SELECT
109      c.customer_id,
110      c.first_name,
111      o.order_id,
112      o.status
113  FROM customers c
114  LEFT JOIN orders o ON c.customer_id = o.customer_id;
115
```

Result Grid				
		Filter Rows:		Export:
	customer_id	first_name	order_id	status
▶	1	Alice	1001	Shipped
	2	Bob	1002	Delivered

```

115
116 -- RIGHT JOIN: All orders including those without matching customer
117 • SELECT
118     o.order_id,
119     o.customer_id,
120     c.first_name
121 FROM orders o
122 RIGHT JOIN customers c ON o.customer_id = c.customer_id;
123

```




Result Grid			
Filter Rows: <input type="text"/>			
Export:  Wrap Cell Content: 			
	order_id	customer_id	first_name
	1001	1	Alice
	1002	2	Bob

c. Write subqueries

```

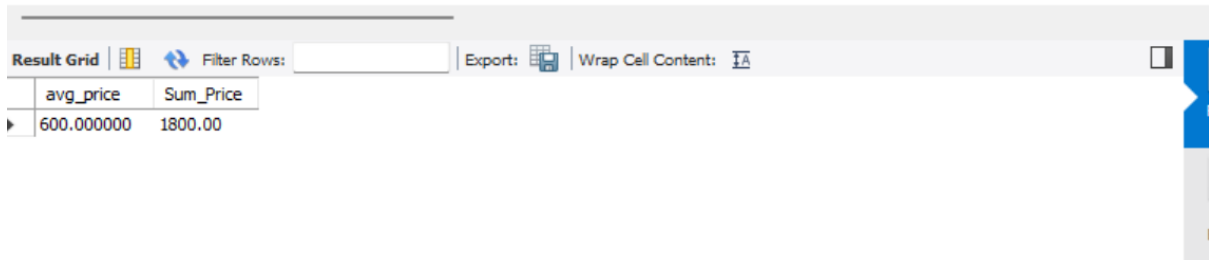
124 -- Find customers who made payments over $1000 total
125 • SELECT customer_id, first_name, last_name
126 FROM customers
127 WHERE customer_id IN (
128     SELECT o.customer_id
129     FROM orders o
130     JOIN payments p ON o.order_id = p.order_id
131     GROUP BY o.customer_id
132     HAVING SUM(p.amount) > 1000
133 );

```

Result Grid			
Filter Rows: <input type="text"/>			
Edit:    Export/Import			
	customer_id	first_name	last_name
	2	Bob	Smith
*	NULL	NULL	NULL

d. Use aggregate functions (SUM, AVG)

```
134
135 -- Average and Sum product price
136 • SELECT AVG(price) AS avg_price , sum(price) as Sum_Price FROM products;
137
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid contains two columns: 'avg_price' and 'Sum_Price'. The first row of data shows '600.000000' for 'avg_price' and '1800.00' for 'Sum_Price'. The interface includes a 'Filter Rows' section, an 'Export' button, and a 'Wrap Cell Content' option.

avg_price	Sum_Price
600.000000	1800.00

e. CREATE VIEWS FOR ANALYSIS

```
-- Create view Average and Sum product price
• Create View Average_and_Sum_product_price as
  SELECT AVG(price) AS avg_price , sum(price) as Sum_Price FROM products;
```

f. Optimize queries with indexes

```
-- Index for customer_id in orders
CREATE INDEX idx_orders_customer_id ON orders(customer_id);
```