

```

91 -- 2. Use SQL Clauses (WHERE, HAVING, LIMIT)
92 -- A. Find all orders placed in the last 6 months
93 • SELECT * FROM Orders
94 WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
95

```

Result Grid	Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
order_id	customer_id	order_date	total_amount	status
NULL	NULL	NULL	NULL	NULL

```

96 -- B. Get the top 5 highest-priced products
97 • SELECT * FROM Products ORDER BY price DESC LIMIT 5;
98
99 -- C. Find customers who have placed more than 3 orders

```

Result Grid	Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:	Fetch rows:
product_id	name	category_id	price	stock_quantity	added_date
102	Laptop	1	55000.00	20	2023-02-15
101	Smartphone	1	15000.00	49	2023-01-10
105	Blender	3	3000.00	30	2024-01-20
104	Jeans	2	2500.00	60	2023-03-05
106	Gaming Mouse	1	1200.00	45	2025-11-21

```

113 -- B. Find all products that are NOT out of stock
114 • SELECT * FROM Products
115 WHERE NOT stock_quantity = 0;
116

```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content

	product_id	name	category_id	price	stock_quantity	added_date
▶	101	Smartphone	1	15000.00	49	2023-01-10
	102	Laptop	1	55000.00	20	2023-02-15
	103	T-Shirt	2	800.00	100	2023-03-01
	104	Jeans	2	2500.00	60	2023-03-05
	105	Blender	3	3000.00	30	2024-01-20

```

117 -- C. Retrieve customers who registered after 2022 OR have made purchases above 10,000
118 • SELECT DISTINCT c.name, c.registration_date
119 FROM Customers c
120 JOIN Orders o ON c.customer_id = o.customer_id
121 WHERE YEAR(c.registration_date) > 2022 OR o.total_amount > 10000;
122

```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	name	registration_date
▶	Rahul Sharma	2021-06-15
	Priya Verma	2023-01-10

```

124 -- A. List all products sorted by price in descending order
125 • SELECT * FROM Products ORDER BY price DESC;
126
127 -- B. Display the number of orders placed by each customer
128 • SELECT customer_id, COUNT(order_id) as total_orders
129 FROM Orders

```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap C

	product_id	name	category_id	price	stock_quantity	added_date
▶	102	Laptop	1	55000.00	20	2023-02-15
	101	Smartphone	1	15000.00	49	2023-01-10
	105	Blender	3	3000.00	30	2024-01-20
	104	Jeans	2	2500.00	60	2023-03-05
	106	Gaming Mouse	1	1200.00	45	2025-11-21
	103	T-Shirt	2	800.00	100	2023-03-01
	NULL	NULL	NULL	NULL	NULL	NULL

```

120
127 -- B. Display the number of orders placed by each customer
128 • SELECT customer_id, COUNT(order_id) as total_orders
129 FROM Orders
130 GROUP BY customer_id;
131

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
customer_id	total_orders		
201	2		
202	2		
204	1		

```

132 -- C. Show total revenue generated per category
133 • SELECT c.category_name, SUM(oi.subtotal) as total_revenue
134 FROM Categories c
135 JOIN Products p ON c.category_id = p.category_id
136 JOIN Order_Items oi ON p.product_id = oi.product_id
137 GROUP BY c.category_name;

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
category_name	total_revenue		
Electronics	85000.00		
Home & Kitchen	3000.00		
Clothing	2500.00		

```

139 -- 5. Use Aggregate Functions
140 -- A. Find the total revenue generated by the store
141 • SELECT SUM(total_amount) as store_revenue FROM Orders;
142

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
store_revenue			
76300.00			

```

143 -- B. Identify the most purchased product
144 • SELECT p.name, SUM(oi.quantity) as total_sold
145 FROM Order_Items oi
146 JOIN Products p ON oi.product_id = p.product_id
147 GROUP BY p.name
148 ORDER BY total_sold DESC
149 LIMIT 1;
150

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:	Fetch rows:
name	total_sold			
Smartphone	2			

```

151 -- C. Calculate the average order value
152 • SELECT AVG(total_amount) as average_order_value FROM Orders;
153
154 -- 7. Implement Joins

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
average_order_value			
▶ 15260.000000			

```

154 -- 7. Implement Joins
155 -- A. Retrieve products with category names using INNER JOIN
156 • SELECT p.name, c.category_name
157 FROM Products p
158 INNER JOIN Categories c ON p.category_id = c.category_id;
159

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
name	category_name		
▶ Smartphone	Electronics		
Laptop	Electronics		
T-Shirt	Clothing		
Jeans	Clothing		
Blender	Home & Kitchen		
Gaming Mouse	Electronics		

```

160 -- B. Get all orders with customer details using LEFT JOIN
161 • SELECT o.order_id, c.name, c.email
162 FROM Orders o
163 LEFT JOIN Customers c ON o.customer_id = c.customer_id;
164

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
order_id	name	email	
▶ 301	Rahul Sharma	rahul@example.com	
302	Rahul Sharma	rahul@example.com	
303	Priya Verma	priya@example.com	
304	Priya Verma	priya@example.com	
305	Sneha Gupta	sneha@example.com	

```

171 -- D. Show customers who have never placed an order (Simulated FULL OUTER JOIN using LEFT JOIN)
172 • SELECT c.name
173 FROM Customers c
174 LEFT JOIN Orders o ON c.customer_id = o.customer_id
175 WHERE o.order_id IS NULL;
176

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
name			
▶ Amit Singh			
John Doe			

```

177 -- 8. Use Subqueries
178 -- A. Find orders placed by customers who registered after 2022
179 • SELECT * FROM Orders
180 WHERE customer_id IN (SELECT customer_id FROM Customers WHERE YEAR(registration_date) > 2022);
181

```

Result Grid  Filter Rows: Edit:    Export/Import:   Wrap Cell Content: 

	order_id	customer_id	order_date	total_amount	status
▶	303	202	2024-09-01	55000.00	Shipped
	304	202	2024-11-01	2500.00	Delivered
•	NULL	NULL	NULL	NULL	NULL

```

182 -- B. Identify the customer who has spent the most
183 • SELECT name FROM Customers
184 WHERE customer_id = (
185     SELECT customer_id FROM Orders
186     GROUP BY customer_id
187     ORDER BY SUM(total_amount) DESC
188     LIMIT 1
189 );

```

Result Grid  Filter Rows: Export:  Wrap Cell Content: 




	name
▶	Priya Verma

190

```

191 -- C. Get products that have never been ordered
192 • SELECT name FROM Products
193 WHERE product_id NOT IN (SELECT DISTINCT product_id FROM Order_Items);

```



Result Grid  Filter Rows: Export:  Wrap Cell Content: 

	name
▶	T-Shirt
	Gaming Mouse

```

195 -- 9. Implement Date & Time Functions
196 -- A. Extract month from order_date to count orders per month
197 • SELECT MONTH(order_date) as order_month, COUNT(*) as count
198 FROM Orders
199 GROUP BY MONTH(order_date);
200

```

Result Grid  Filter Rows: Export:  Wrap Cell Content: 

	order_month	count
▶	10	2
	9	1
	11	1
	1	1

```

201 -- B. Calculate delivery time (difference between shipping_date and delivery_date)
202 • SELECT order_id, DATEDIFF(delivery_date, shipping_date) as days_to_deliver
203 FROM Shipping
204 WHERE delivery_date IS NOT NULL;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	order_id	days_to_deliver
▶	301	3
	303	3
	304	3
	305	2

```

206 -- C. Format order_date as DD-MM-YYYY
207 • SELECT order_id, DATE_FORMAT(order_date, '%d-%m-%Y') as formatted_date
208 FROM Orders;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	order_id	formatted_date
▶	301	01-10-2024
	302	15-10-2024
	303	01-09-2024
	304	01-11-2024
	305	01-01-2023

```

210 -- 10. Use String Manipulation Functions
211 -- A. Convert all product names to uppercase
212 • SELECT UPPER(name) FROM Products;
213

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	UPPER(name)
▶	SMARTPHONE
	LAPTOP
	T-SHIRT
	JEANS
	BLENDER
	GAMING MOUSE

```

214 -- B. Trim whitespace from customer names
215 • SELECT TRIM(name) FROM Customers;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	TRIM(name)
▶	Rahul Sharma
	Priya Verma
	Amit Singh
	Sneha Gupta
	John Doe

```

217 -- C. Replace missing email values with "Not Provided"
218 • SELECT name, COALESCE(email, 'Not Provided') as email_status
219 FROM Customers;
220

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	name	email_status		
▶	Rahul Sharma	rahul@example.com		
	Priya Verma	priya@example.com		
	Amit Singh	Not Provided		
	Sneha Gupta	sneha@example.com		
	John Doe	john@test.com		

```

221 -- 11. Implement Window Functions
222 -- A. Rank customers based on total spending
223 • SELECT customer_id, SUM(total_amount) as total_spent,
224 RANK() OVER (ORDER BY SUM(total_amount) DESC) as spending_rank
225 FROM Orders
226 GROUP BY customer_id;

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	customer_id	total_spent	spending_rank	
▶	202	57500.00	1	
	201	18000.00	2	
	204	800.00	3	

```

227
228 -- B. Show the cumulative total revenue per month
229 • SELECT order_date, total_amount,
230 SUM(total_amount) OVER (ORDER BY order_date) as running_revenue
231 FROM Orders;

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	order_date	total_amount	running_revenue	
▶	2023-01-01	800.00	800.00	
	2024-09-01	55000.00	55800.00	
	2024-10-01	15000.00	70800.00	
	2024-10-15	3000.00	73800.00	
	2024-11-01	2500.00	76300.00	

```

232
233 -- C. Display the running total of orders placed
234 • SELECT order_id, order_date,
235 COUNT(order_id) OVER (ORDER BY order_date) as running_order_count
236 FROM Orders;


```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	order_id	order_date	running_order_count	
▶	305	2023-01-01	1	
	303	2024-09-01	2	
	301	2024-10-01	3	
	302	2024-10-15	4	
	304	2024-11-01	5	

```

238 -- 12. Apply SQL CASE Expressions
239 -- A. Assign Loyalty_Status to customers
240 • SELECT c.name, SUM(o.total_amount) as total_spent,
241 CASE
242     WHEN SUM(o.total_amount) > 50000 THEN 'Gold'
243     WHEN SUM(o.total_amount) BETWEEN 20000 AND 50000 THEN 'Silver'
244     ELSE 'Bronze'
245 END as Loyalty_Status
246 FROM Customers c
247 JOIN Orders o ON c.customer_id = o.customer_id
248 GROUP BY c.customer_id;

```

Result Grid   Filter Rows: Export:  Wrap Cell Content: 

	name	total_spent	Loyalty_Status
▶	Rahul Sharma	18000.00	Bronze
	Priya Verma	57500.00	Gold
	Sneha Gupta	800.00	Bronze

```

250 -- B. Categorize products (Best Seller, Popular, Regular)
251 • SELECT p.name, COALESCE(SUM(oi.quantity), 0) as total_sold,
252 CASE
253     WHEN SUM(oi.quantity) > 500 THEN 'Best Seller'
254     WHEN SUM(oi.quantity) BETWEEN 200 AND 500 THEN 'Popular'
255     ELSE 'Regular'
256 END as Sales_Category
257 FROM Products p
258 LEFT JOIN Order_Items oi ON p.product_id = oi.product_id
259 GROUP BY p.product_id;
260
261

```

Result Grid   Filter Rows: Export:  Wrap Cell Content: 

	name	total_sold	Sales_Category
▶	Smartphone	2	Regular
	Laptop	1	Regular
	T-Shirt	0	Regular
	Jeans	1	Regular
	Blender	1	Regular
	Gaming Mouse	0	Regular