

## SQL QUERIES & OUTPUTS

-- EXECUTIVE SUMMARY

-- 1) TOTAL REVENUE

```
SELECT SUM(revenue)as Total_Revenue FROM sales;
```

```
1      -- EXECUTIVE SUMMARY  
2  
3      -- 1) TOTAL REVENUE  
4 •   SELECT SUM(revenue)as Total_Revenue FROM sales;  
5  
6
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	Total_Revenue			
▶	408093111.6294027			

-- 2) TOTAL PROFIT

```
SELECT SUM(Profit) AS Total_Profit FROM Sales;
```

```
9      -- 2) TOTAL PROFIT  
10 •   SELECT SUM(Profit) AS Total_Profit FROM Sales;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	Total_Profit			
▶	217399424.66940337			

-- 3) PROFIT MARGIN

```
SELECT ROUND((SUM(Profit) / SUM(revenue)) * 100,2) AS Profit_Margin FROM Sales;
```

```

13      -- 3) PROFIT MARGIN
14 •   SELECT ROUND((SUM(Profit) / SUM(revenue)) * 100,2) AS Profit_Margin FROM Sales;
15
16

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
▶	Profit_Margin			

53.27

-- 4) REVENUE TREND(MONTH WISE)

```

SELECT DATE_FORMAT(date, '%Y-%m') AS Month,
       SUM(revenue) AS Revenue
  FROM Sales
 GROUP BY Month
 ORDER BY Month;

```

```

20      -- 4) REVENUE TREND(MONTH WISE)
21 •   SELECT DATE_FORMAT(date, '%Y-%m') AS Month,
22           SUM(revenue) AS Revenue
23     FROM Sales
24   GROUP BY Month
25 ORDER BY Month;
26
27

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
▶	Month	Revenue		
▶	2023-01	35689085.96856286		
	2023-02	30722968.034204114		
	2023-03	35810727.46382958		
	2023-04	34356645.73118096		
	2023-05	34813133.107993856		
	2023-06	33070422.405125678		
	2023-07	33529562.729336854		
	2023-08	36171108.01645611		
	2023-09	33913815.84994974		
	2023-10	34010722.07977732		
	2023-11	32060951.031078		
	2023-12	33943969.21190674		

-- 5) CATEGORY CONTRIBUTION

```
SELECT Category, SUM(revenue) AS Revenue  
FROM Sales  
GROUP BY Category  
ORDER BY revenue DESC;
```

```
28      -- 5) CATEGORY CONTRIBUTION  
29 •  SELECT Category, SUM(revenue) AS Revenue  
30    FROM Sales  
31    GROUP BY Category  
32    ORDER BY revenue DESC;  
33
```

The screenshot shows a database query results grid. At the top, there are navigation buttons for 'Result Grid' (highlighted), 'Filter Rows', 'Export' (with icons for CSV and Excel), and 'Wrap Cell Content'. The results table has two columns: 'Category' and 'Revenue'. The data rows are: Electronics (84686637.21269211), Sports (75889447.11940283), Fashion (75235778.02945608), Beauty (69038507.78788902), Home Appliances (65098902.22639057), and Grocery (38143839.25357065). The 'Sports' row is currently selected.

	Category	Revenue
▶	Electronics	84686637.21269211
	Sports	75889447.11940283
	Fashion	75235778.02945608
	Beauty	69038507.78788902
	Home Appliances	65098902.22639057
	Grocery	38143839.25357065

-- 6) TOP 10 PRODUCTS

```
SELECT product_name, SUM(revenue) AS Revenue  
FROM Sales  
GROUP BY product_name  
ORDER BY revenue DESC  
LIMIT 10;
```

```
35      -- 6) TOP 10 PRODUCTS
36 •   SELECT product_name, SUM(revenue) AS Revenue
37     FROM Sales
38     GROUP BY product_name
39     ORDER BY revenue DESC
40     LIMIT 10;
41
```

Result Grid    Filter Rows: <input type="text"/> Export:  Wrap Cell Content:		
	product_name	Revenue
▶	Product_77	4561063.45050084
	Product_74	4542231.597170674
	Product_161	4467041.564946462
	Product_173	4452452.336504849
	Product_113	4285186.605278401
	Product_177	4245156.561944845
	Product_79	4198885.541076673
	Product_183	4142364.196524453
	Product_70	4058391.6244094064
	Product_187	4055879.092970333

#### -- 7) TOTAL QUANTITY

```
SELECT SUM(quantity) AS Total_Quantity
FROM sales;
```

```
44      -- 7) TOTAL QUANTITY
45 •   SELECT SUM(quantity) AS Total_Quantity
46     FROM sales;
47
48
49
50
```

Result Grid    Filter Rows: <input type="text"/> Export:  Wrap Cell Content:		
	Total_Quantity	
▶	149940	

-- FORECASTING & PRICING

-- 1) Actual vs Forecast Revenue

SELECT

p.product\_name,

COALESCE(SUM(s.revenue),0) AS Actual\_Revenue,

COALESCE(MAX(f.forecast\_next\_month),0) AS Forecast\_Revenue

FROM Sales s

JOIN Products p

ON s.product\_id = p.product\_id

LEFT JOIN Forecasts f

ON p.product\_id = f.product\_id

GROUP BY

p.product\_id, p.product\_name

ORDER BY

Actual\_Revenue DESC LIMIT 50;

Result Grid | Filter Rows: | Export: |

	product_name	Actual_Revenue	Forecast_Revenue
▶	Product_77	4561063.45050084	5219310.122876391
	Product_74	4542231.597170674	5190131.118357565
	Product_161	4467041.564946462	4349956.831777971
	Product_173	4452452.336504849	4810454.920736514
	Product_113	4285186.605278401	4456182.718884374
	Product_177	4245156.561944845	5019063.941672961
	Product_79	4198885.541076673	4344247.85091062
	Product_183	4142364.196524453	4391138.283528218
	Product_70	4058391.6244094064	4370582.975196742
	Product_187	4055879.092970333	3929433.4435128598
	Product_98	4036069.903435647	4793347.242473334
	Product_75	4029535.7222840767	4503069.664406863
	Product_86	4012184.2841345184	4373916.027957586
	Product_128	3964549.0512422863	4162006.8718045265
	Product_101	3952774.5223910506	4116805.900048231
	Product_54	3879311.8379420554	4612581.319577996

```

1      -- FORECASTING & PRICING
2      -- 1) Actual vs Forecast Revenue
3 •   SELECT
4          p.product_name,
5          COALESCE(SUM(s.revenue),0) AS Actual_Revenue,
6          COALESCE(MAX(f.forecast_next_month),0) AS Forecast_Revenue
7      FROM Sales s
8      JOIN Products p
9          ON s.product_id = p.product_id
10     LEFT JOIN Forecasts f
11        ON p.product_id = f.product_id
12     GROUP BY
13        p.product_id, p.product_name
14     ORDER BY
15        Actual_Revenue DESC | TMTT 50:

```

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

	product_name	Actual_Revenue	Forecast_Revenue
▶	Product_77	4561063.45050084	5219310.122876391
	Product_74	4542231.597170674	5190131.118357565
	Product_161	4467041.564946462	4349956.831777971
	Product_173	4452452.336504849	4810454.920736514
	Product_113	4285186.605278401	4456182.718884374

-- 2) Pricing Impact (Old vs New Price)

SELECT

```
product_id,  
old_price,  
new_price,  
(new_price - old_price) AS Price_Change,  
((new_price - old_price) / old_price) * 100 AS Price_Change_Percent  
FROM pricing_changes;
```

```
21      -- 2) Pricing Impact (Old vs New Price)  
22 •  SELECT  
23      product_id,  
24      old_price,  
25      new_price,  
26      (new_price - old_price) AS Price_Change,  
27      ((new_price - old_price) / old_price) * 100 AS Price_Change_Percent  
28  FROM pricing_changes;  
29
```

Result Grid				
product_id	old_price	new_price	Price_Change	Price_Change_Percent
28	3471.28	2691.43	-779.850000000004	-22.46577631306032
51	2190.09	1311.96	-878.130000000001	-40.095612509074975
3	1443.88	1492.06	48.17999999999836	3.336842396875075
157	3259.41	3008.66	-250.75	-7.693110102748657
116	701.3	1908.05	1206.75	172.07329245686583
158	252.3	339.12	86.82	34.41141498216408
183	1006.55	2441.08	1434.53	142.5194972927326
19	3349.58	4240.31	890.730000000005	26.592289182524393
148	957.62	2129.55	1171.930000000003	122.37944069672733
113	411.63	2778.28	2366.65	574.9459466025314

-- 3) Revenue Before & After Price Change

SELECT

```
pr.product_name,  
SUM(CASE WHEN DATE(s.date) < DATE(pc.change_date) THEN s.revenue END) AS  
Revenue_Before,
```

```

        SUM(CASE WHEN DATE(s.date) >= DATE(pc.change_date) THEN s.revenue END) AS
Revenue_After

FROM Sales s

JOIN Pricing_Changes pc

ON s.product_id = pc.product_id

JOIN Products pr

ON s.product_id = pr.product_id

GROUP BY pr.product_name;

```

```

31      -- 3) Revenue Before & After Price Change
32 •  SELECT
33      pr.product_name,
34      SUM(CASE WHEN DATE(s.date) < DATE(pc.change_date) THEN s.revenue END) AS Revenue_Before,
35      SUM(CASE WHEN DATE(s.date) >= DATE(pc.change_date) THEN s.revenue END) AS Revenue_After
36  FROM Sales s
37  JOIN Pricing_Changes pc
38    ON s.product_id = pc.product_id
39  JOIN Products pr
40    ON s.product_id = pr.product_id
41  GROUP BY pr.product_name;

```

Result Grid		
product_name	Revenue_Before	Revenue_After
Product_165	7686799.869920924	11917887.41617217
Product_159	209623.15980048373	424833.80919271737
Product_58	7703857.778279726	7480544.21427946
Product_191	4464554.512774957	2651196.624957235
Product_130	2265209.5918712616	1424928.574836495
Product_86	9976891.438887073	14096214.265920062
Product_125	5337537.4665228445	3176433.923023676
Product_4	21342125.884278998	14478188.326129656
Product_97	349662.57266196224	251797.66886625875

-- TOTAL ACTUAL REVENUE AND TOTAL PRODUCT LEVEL FORECAST

```

SELECT

SUM(s.revenue) AS Total_Actual_Revenue,

SUM(f.forecast_next_month) AS Total_Product_Level_Forecast

FROM Sales s

LEFT JOIN Forecasts f

ON s.product_id = f.product_id;

```

```
61      -- TOTAL ACTUAL REVENUE AND TOTAL PRODUCT LEVEL FORECAST
62 •  SELECT
63      SUM(s.revenue) AS Total_Actual_Revenue,
64      SUM(f.forecast_next_month) AS Total_Product_Level_Forecast
65  FROM Sales s
```

		Result Grid	Filter Rows:	Export:	Wrap Cell Content:
		Total_Actual_Revenue	Total_Product_Level_Forecast		
		408093111.6294027	110126714321.07805		

-- CUSTOMER ANALYTICS

-- 1) Customer Lifetime Revenue

SELECT

```
c.customer_name,
SUM(s.revenue) AS Customer_Lifetime_Revenue
FROM Sales s
JOIN Customers c
ON s.customer_id = c.customer_id
GROUP BY c.customer_name
ORDER BY Customer_Lifetime_Revenue DESC;
```

```

1      -- CUSTOMER ANALYTICS
2      -- 1) Customer Lifetime Revenue
3 •  SELECT
4          c.customer_name,
5          SUM(s.revenue) AS Customer_Lifetime_Revenue
6  FROM Sales s
7  JOIN Customers c
8      ON s.customer_id = c.customer_id
9  GROUP BY c.customer_name
10 ORDER BY Customer_Lifetime_Revenue DESC;
11

```

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

	customer_name	Customer_Lifetime_Revenue
▶	Customer_3443	222412.90936389234
	Customer_4210	215356.3470832561
	Customer_3046	212226.09453027186
	Customer_3033	211771.13432619558
	Customer_2131	208982.0194527773
	Customer_3038	207328.2895362899
	Customer_4703	206008.8583778586
	Customer_3867	202537.7461863457
	Customer_1443	197242.44076073807
	Customer_2057	196934.34135173453

## -- 2) SEGMENT PERFORMANCE

```

SELECT
    c.segment,
    SUM(s.revenue) AS Revenue
FROM Sales s
JOIN Customers c
    ON s.customer_id = c.customer_id
GROUP BY c.segment
ORDER BY Revenue DESC;

```

```
14      -- 2) SEGMENT PERFORMANCE
15 •   SELECT
16      c.segment,
17      SUM(s.revenue) AS Revenue
18  FROM Sales s
19  JOIN Customers c
20    ON s.customer_id = c.customer_id
21  GROUP BY c.segment
22  ORDER BY Revenue DESC;
```

The screenshot shows a database query results grid. At the top, there are buttons for 'Result Grid' (selected), 'Filter Rows', 'Export' (with a CSV icon), and 'Wrap Cell Content'. The grid itself has two columns: 'segment' and 'Revenue'. The data rows are: New (104242679.76397906), Loyal (102797536.23643586), High-Value (101233833.78884277), and Regular (99819061.84014456). The 'Loyal' row is currently selected.

segment	Revenue
New	104242679.76397906
Loyal	102797536.23643586
High-Value	101233833.78884277
Regular	99819061.84014456

### -- 3) REGION REVENUE MAP

```
SELECT
  c.location,
  SUM(s.revenue) AS Revenue
FROM Sales s
JOIN customers c
  ON s.customer_id = c.customer_id
GROUP BY c.location;
```

```
25      -- 3) REGION REVENUE MAP
26 •   SELECT
27      c.location,
28      SUM(s.revenue) AS Revenue
29  FROM Sales s
30  JOIN customers c
31      ON s.customer_id = c.customer_id
32  GROUP BY c.location;
33
```

The screenshot shows a database query results grid. At the top, there are navigation buttons for 'Result Grid' (with a grid icon), 'Filter Rows' (with a filter icon), 'Export' (with a file icon), and 'Wrap Cell Content' (with a text icon). The results grid has two columns: 'location' and 'Revenue'. The data rows are: Chennai (72585130.25691506), Hyderabad (65891688.73542326), Bangalore (65750784.00662776), Mumbai (70879061.79221648), Delhi (66999026.3853881), and Pune (65987420.45283058). The 'Hyderabad' row is currently selected.

	location	Revenue
▶	Chennai	72585130.25691506
	Hyderabad	65891688.73542326
	Bangalore	65750784.00662776
	Mumbai	70879061.79221648
	Delhi	66999026.3853881
	Pune	65987420.45283058