



Optimizing Pizza Hut Sales with SQL

Discover how SQL analysis unlocks valuable insights from Pizza Hut sales data. Drive smarter business decisions using efficient data extraction and trend visualization.

 by Harshika Nirwan

Essential SQL Queries: Data Extraction

1 Retrieve Data

Basic SELECT to get records from tables.

2 Filter Results

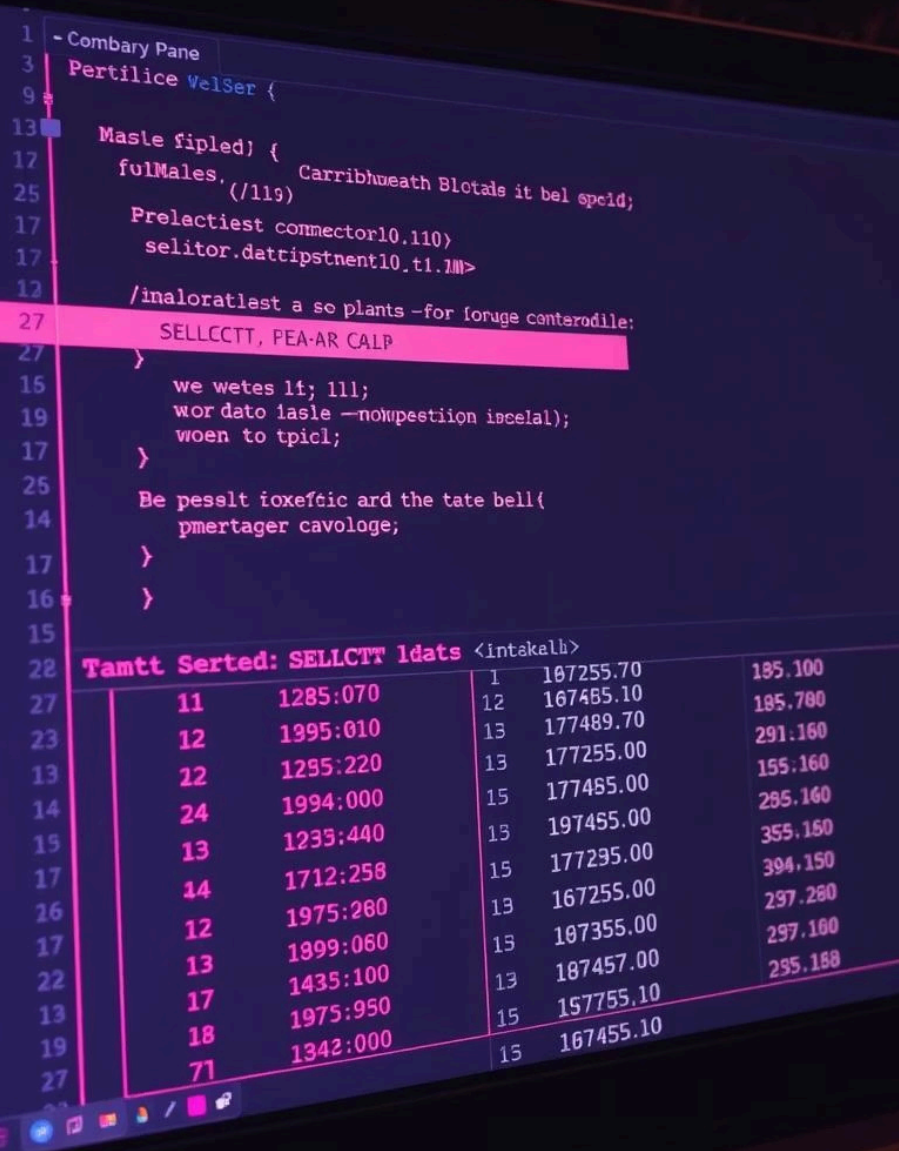
Use WHERE for targeted data (e.g., sales in Q1 2015).

3 Sort Data

ORDER BY for top-selling pizzas or dates.

4 Example Query

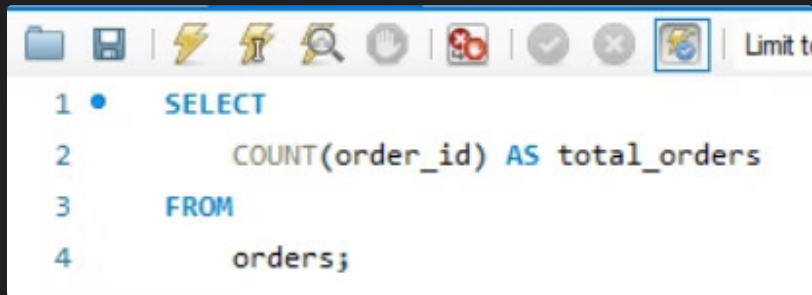
SELECT * FROM orders WHERE order_date LIKE '2015-01%';



The screenshot shows a code editor with a dark theme. The top pane contains SQL code for a stored procedure. The bottom pane displays the result of a SELECT query, titled 'Tamtt Serted: SELECTT ldat<intakal>'. The result is a table with 4 columns and 15 rows of data.

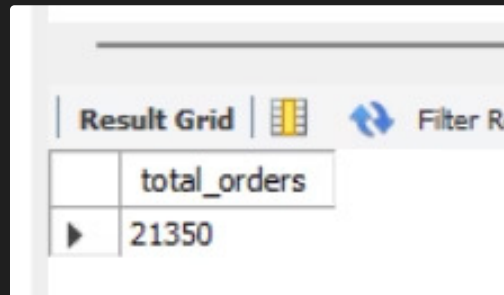
```
1 - Combarry Pane
3 Pertilice velSer {
9 #
13 Masle fipled) {
17   fulMales, (//119) Carrribhweath Blotade it bel spclid;
25   Prelectiest connector10.110)
17   selitor.dattipstnent10.t1.1M>
12   /inaloratlast a se plants -for foruge canterodile:
27   SELECTT, PEA-AR CALP
27 }
15   we wetes lf; 111;
19   wor dato lasle --nomupestiion isecelal);
17   woen to tpicl;
25 }
14 Be pesslt ioxeftic ard the tate bell{
17   pmertager cavologe;
16 }
15
22 Tamtt Serted: SELECTT ldat<intakal>
27 11 1285:070 12 167255.70 185.100
23 12 1995:010 13 167485.10 185.780
13 22 1255:220 13 177489.70 291.160
14 24 1994:000 13 177255.00 155.160
15 13 1233:440 15 177485.00 285.160
17 14 1712:258 13 197455.00 355.150
26 12 1975:280 15 177295.00 394.150
17 13 1999:060 13 167255.00 297.280
22 17 1435:100 13 107355.00 297.100
13 18 1975:950 13 187457.00 255.188
19 71 1342:000 15 157755.10
27 15 167455.10
```

Retrieve the total number of orders placed.



A screenshot of a SQL query editor window. The toolbar at the top includes icons for file operations, execution, and navigation. The query text is as follows:

```
1 • SELECT
2     COUNT(order_id) AS total_orders
3 FROM
4     orders;
```



A screenshot of a database result grid. The toolbar at the top includes icons for grid view, refresh, and filter. The result is displayed in a table with one column and one row.

	total_orders
▶	21350

Calculate the total revenue generated from pizza sales.

SQL File 5* SQL File 6* SQL File 7* x SQL File 8* SQL File 9* SQL File 11

Limit to 50000 rows

```
1 • SELECT
2   ROUND(SUM(orders_details.quantity * pizzas.price),
3         2) AS total_sales
4 FROM
5   orders_details
6   JOIN
7   pizzas ON pizzas.pizza_id = orders_details.pizza_id
8
```

Result Grid | Filter Rows

	total_sales
▶	817860.05



Identify the highest-priced pizza.

```
1 • SELECT
2     pizza_types.name, pizzas.price
3 FROM
4     pizza_types
5     JOIN
6     pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
7 ORDER BY pizzas.price DESC
8 LIMIT 1;
9
```

Result Grid			Filter Rows:
	name	price	
▶	The Greek Pizza	35.95	

Identify the most common pizza size ordered.



```
2
3 • SELECT
4     pizzas.size,
5     COUNT(orders_details.order_details_id) AS order_count
6 FROM
7     pizzas
8     JOIN
9     orders_details ON pizzas.pizza_id = orders_details.pizza_id
10 GROUP BY pizzas.size
11 ORDER BY order_count;
12
```

Result Grid |   Filter Rows:

	size	order_count
▶	XXL	28
	XL	544
	S	14137
	M	15385
	L	18526

List the top 5 most ordered pizza types along with their quantities.

```
• SELECT
    pizza_types.name, SUM(orders_details.quantity) AS quantity
FROM
    pizza_types
    JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
    JOIN
    orders_details ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.name
ORDER BY quantity DESC
LIMIT 5;
```

Result Grid |   Filter Rows:

	name	quantity
▶	The Classic Deluxe Pizza	2453
	The Barbecue Chicken Pizza	2432
	The Hawaiian Pizza	2422
	The Pepperoni Pizza	2418
	The Thai Chicken Pizza	2371



Join the necessary tables to find the total quantity of each pizza category ordered.

```
2
3 • SELECT
4     pizza_types.category,
5     SUM(orders_details.quantity) AS quantity
6 FROM
7     pizza_types
8     JOIN
9     pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
10    JOIN
11    orders_details ON orders_details.pizza_id = pizzas.pizza_id
12 GROUP BY pizza_types.category
13 ORDER BY quantity DESC;
```

Result Grid			Filter Rows
	category	quantity	
▶	Classic	14888	
	Supreme	11987	
	Veggie	11649	
	Chicken	11050	

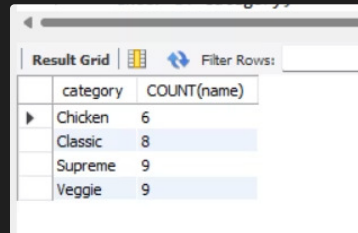
Determine the distribution of orders by hour of the day.

```
• SELECT
    HOUR(oder_time) AS hour, COUNT(order_id) AS orders
FROM
    orders
GROUP BY HOUR(oder_time);
```

Result Grid   Filter Ro		
	hour	orders
▶	11	1231
	12	2520
	13	2455
	14	1472
	15	1468
	16	1920
	17	2336
	18	2399
	19	2009
	20	1642
	21	1198
	22	663
	23	28
	10	8
	9	1

Join relevant tables to find the category-wise distribution of pizzas.

```
SELECT  
    category, COUNT(name)  
FROM  
    pizza_types  
GROUP BY category;
```

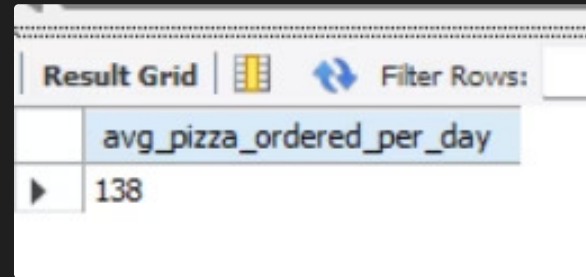


The screenshot shows a database interface with a 'Result Grid' tab. It displays a table with two columns: 'category' and 'COUNT(name)'. The table contains four rows of data: 'Chicken' with a count of 6, 'Classic' with a count of 8, 'Supreme' with a count of 9, and 'Veggie' with a count of 9. A 'Filter Rows' input field is visible at the top right of the grid.

category	COUNT(name)
Chicken	6
Classic	8
Supreme	9
Veggie	9

Group the orders by date and calculate the average number of pizzas ordered per day.

```
SELECT
    ROUND(AVG(quantity), 0) as avg_pizza_ordered_per_day
FROM
    (SELECT
        orders.order_date, SUM(orders_details.quantity) AS quantity
    FROM
        orders
    JOIN orders_details ON orders.order_id = orders_details.order_id
    GROUP BY orders.order_date) AS orders_quantity;
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid contains one column named 'avg_pizza_ordered_per_day' and one row with the value '138'. There are icons for 'Filter Rows' and a refresh button.

avg_pizza_ordered_per_day
138



Determine the top 3 most ordered pizza types based on revenue.

```
SELECT
    pizza_types.name,
    SUM(orders_details.quantity * pizzas.price) AS revenue
FROM
    pizza_types
    JOIN
    pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
    JOIN
    orders_details ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.name
ORDER BY revenue DESC
LIMIT 3;
```

Result Grid			Filter Rows:
	name	revenue	
▶	The Thai Chicken Pizza	43434.25	
	The Barbecue Chicken Pizza	42768	
	The California Chicken Pizza	41409.5	

Calculate the percentage contribution of each pizza type to total revenue.



```
SELECT
    pizza_types.category,
    round(SUM(orders_details.quantity * pizzas.price) / ( select  ROUND(SUM(orders_details.quantity * pizzas.price),
        2) AS total_sales
FROM
    orders_details
    JOIN
    pizzas ON pizzas.pizza_id = orders_details.pizza_id)* 100,2) as revenue
FROM
    pizza_types
    JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
    JOIN
    orders_details ON orders_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.category
ORDER BY revenue DESC;
```

Result Grid |   Filter R

	category	revenue
▶	Classic	26.91
	Supreme	25.46
	Chicken	23.96
	Veggie	23.68

Analyze the cumulative revenue generated over time.

```
• select order_date,  
  sum(revenue) over(order by order_date) as cum_revenue  
from  
(select  
  orders.order_date,  
  sum(orders_details.quantity*pizzas.price) as revenue  
from orders_details join pizzas  
on orders_details.pizza_id= pizzas.pizza_id  
join orders  
on orders.order_id = orders_details.order_details_id  
group by orders.order_date) as sales;
```

Result Grid			 Filter Rows:
	order_date	cum_revenue	
▶	2015-01-01	1171.45	
	2015-01-02	2316.10000000000004	1171.45
	2015-01-03	3433.8	
	2015-01-04	4341.8	
	2015-01-05	5247.25	
	2015-01-06	6299.9	
	2015-01-07	7284.7	
	2015-01-08	8542.35	

Determine the top 3 most ordered pizza types based on revenue for each pizza category.

```
• select name, revenue from
  (select category, name , revenue,
   rank() over(partition by category order by revenue desc) as RN
   from
    (select pizza_types.category, pizza_types.name,
     sum(orders_details.quantity*pizzas.price) as revenue
     from pizza_types join pizzas
     on pizza_types.pizza_type_id = pizzas.pizza_type_id
     join orders_details
     on orders_details.pizza_id=pizzas.pizza_id
     group by pizza_types.category, pizza_types.name) as A) as B
 where RN <=3;
```

Result Grid			Filter Rows:	
	name	revenue		
▶	The Thai Chicken Pizza	43434.25		
	The Barbecue Chicken Pizza	42768		
	The California Chicken Pizza	41409.5		
	The Classic Deluxe Pizza	38180.5		
	The Hawaiian Pizza	32273.25		
	The Pepperoni Pizza	30161.75		
	The Spicy Italian Pizza	34831.25		
	The Italian Supreme Pizza	33476.75		

Advanced SQL Queries: Aggregation

1

GROUP BY

Aggregate sales by pizza type or category.

2

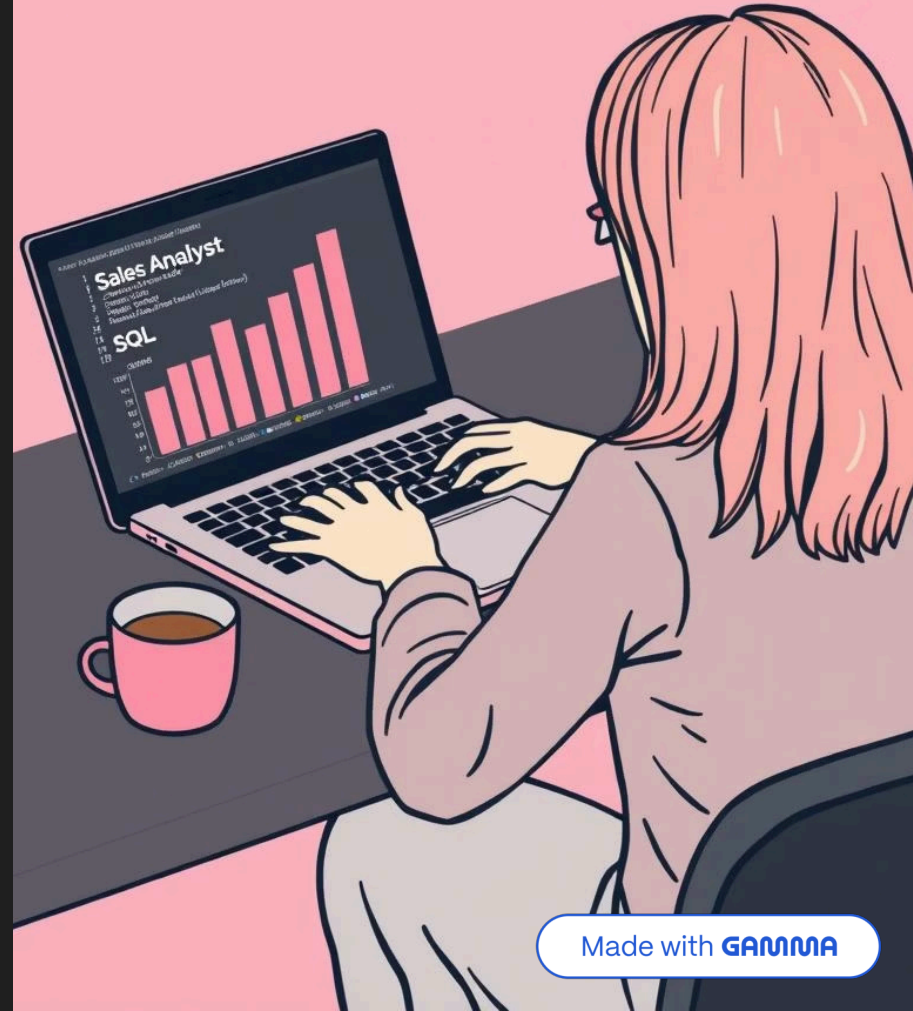
Aggregate Functions

- SUM: total sales
- AVG: average order value
- COUNT: number of orders

3

Example

```
SELECT pizza_type_id, SUM(quantity) AS total_quantity FROM  
order_details GROUP BY pizza_type_id ORDER BY total_quantity  
DESC;
```



SQL for Sales Trend Analysis

Trend Analysis

Use date functions to analyze sales over time.

Peak Identification

Find top sales days and months.

Moving Averages

Smooth sales fluctuations using averages.

Example Query

```
SELECT strftime('%Y-%m',  
order_date) AS month,  
SUM(price) AS  
monthly_revenue FROM  
orders JOIN order_details ON  
orders.order_id =  
order_details.order_id JOIN  
pizzas ON  
order_details.pizza_id =  
pizzas.pizza_id GROUP BY  
month ORDER BY month;
```

Business Implications and Recommendations

Inventory Optimization

Stock more ingredients for Classic and Chicken pizzas.

Targeted Promotions

Run offers during peak months and evening hours.

Boost Online Experience

Enhance digital ordering for growing demand.

Adjust Staffing

Increase team during busy periods for efficiency.

Marketing Focus

Promote heavily during slow sales months.



Conclusion: Data-Driven Decision Making

SQL analysis unlocks key insights for sales optimization. Keep monitoring data regularly to adapt fast. Data-driven choices drive revenue and business growth.