

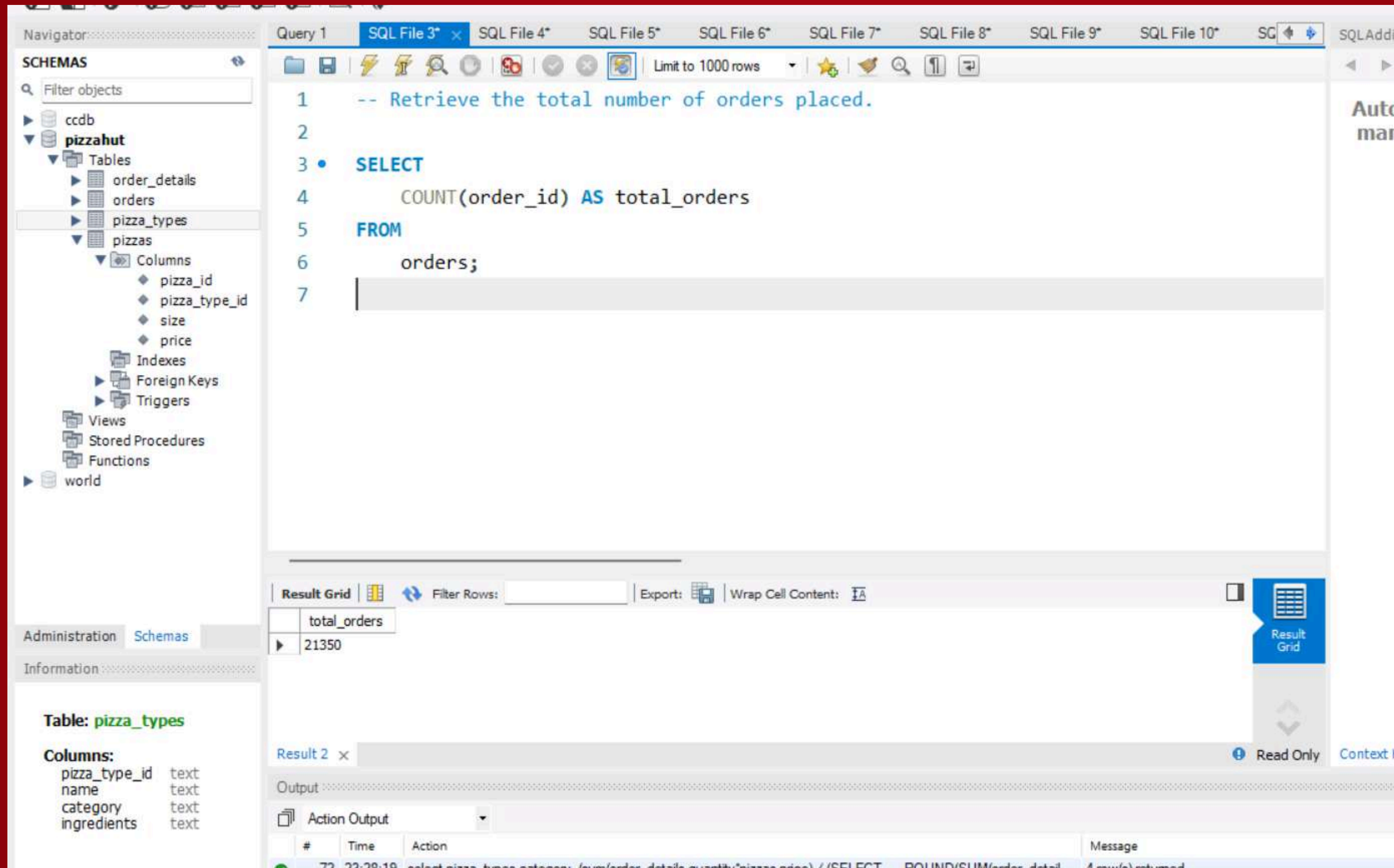
PIZZAHUT SALES ANALYSIS (USING MY SQL)



INTRODUCTION :

This project presents a comprehensive sales analysis of a pizza business using SQL querying techniques on transactional and product data. The analysis involves joining multiple related tables such as pizza types, individual pizzas, and order details to calculate key performance metrics like revenue by pizza category and size. Advanced SQL functions including aggregation and window functions are utilized to rank pizza categories based on sales performance.

Q)1 - RETRIEVE THE TOTAL NUMBER OF ORDERS PLACED



The screenshot shows a SQL IDE interface with a Navigator pane on the left, a central query editor, and a bottom pane for results and messages.

Navigator: The left pane shows a database schema for 'pizzahut'. Under the 'Tables' folder, 'orders' is selected. Other tables listed include 'order_details', 'pizza_types', and 'pizzas'. The 'pizzas' table has columns: 'pizza_id', 'pizza_type_id', 'size', and 'price'. There are also sections for 'Indexes', 'Foreign Keys', 'Triggers', 'Views', 'Stored Procedures', and 'Functions'.

Query Editor: The central pane shows a SQL query in 'Query 1'.

```
1  -- Retrieve the total number of orders placed.
2
3  •  SELECT
4      COUNT(order_id) AS total_orders
5  FROM
6      orders;
7
```

Result Grid: Below the query editor, the 'Result Grid' tab is active. It shows a single column 'total_orders' with a value of 21350.

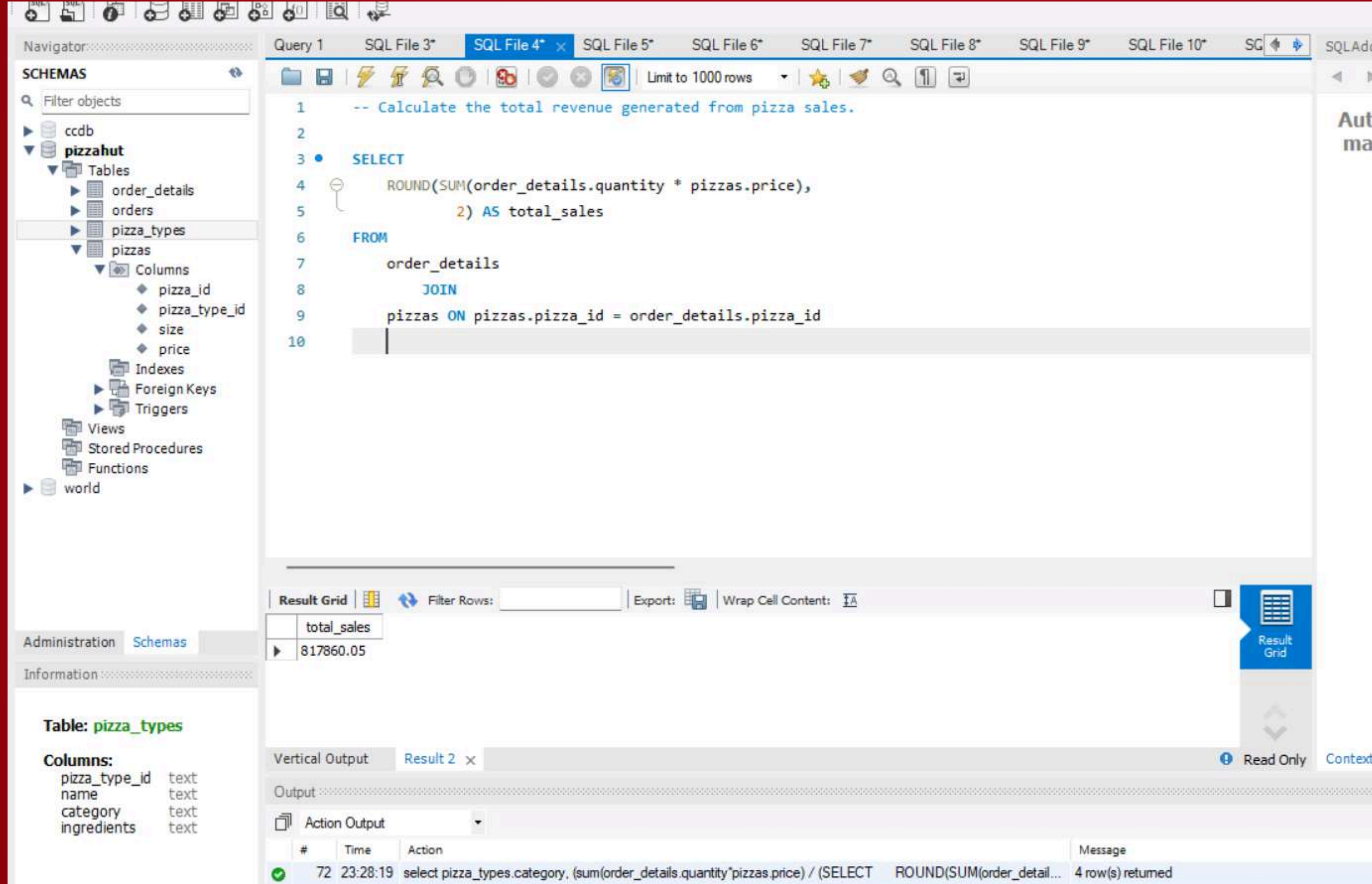
total_orders
21350

Table: pizza_types: The bottom left pane shows the structure of the 'pizza_types' table.

Column	Type
pizza_type_id	text
name	text
category	text
ingredients	text

Output: The bottom right pane shows the 'Output' tab with a message: '4 row(s) returned'.

Q2) CALCULATE THE TOTAL REVENUE GENERATED FROM PIZZA SALES.



The screenshot displays a SQL IDE interface with a query editor and a results pane. The query editor shows a SQL query to calculate the total revenue generated from pizza sales. The results pane shows the output of the query, which is a single row with the value 817860.05.

Query:

```
-- Calculate the total revenue generated from pizza sales.

SELECT
  ROUND(SUM(order_details.quantity * pizzas.price),
        2) AS total_sales
FROM
  order_details
  JOIN
  pizzas ON pizzas.pizza_id = order_details.pizza_id
```

Result Grid:

total_sales
817860.05

Table: pizza_types

Columns:

Column Name	Data Type
pizza_type_id	text
name	text
category	text
ingredients	text

Output:

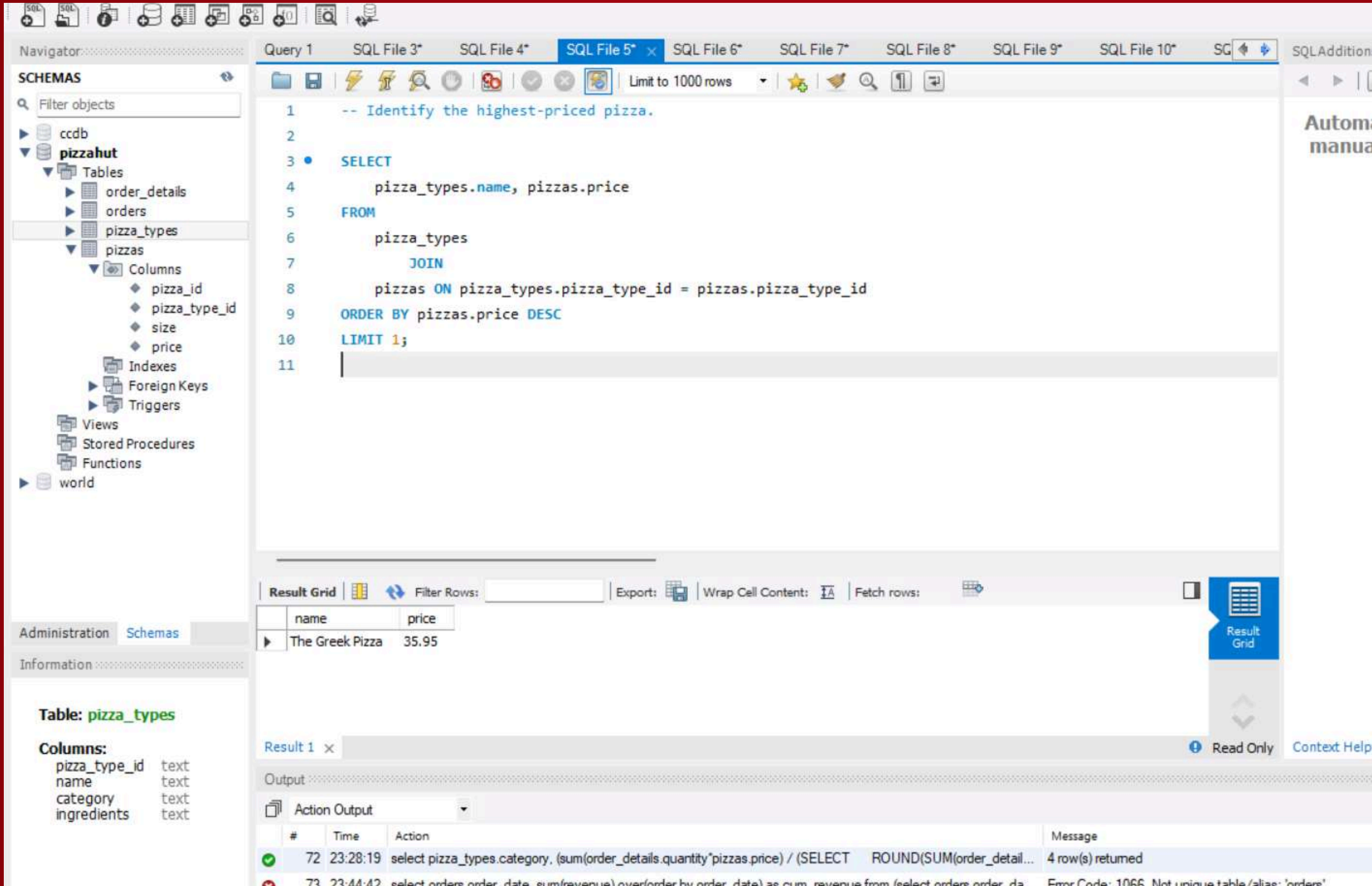
Vertical Output Result 2 x

Output

Action Output

Message: 4 row(s) returned

Q3) IDENTIFY THE HIGHEST-PRICED PIZZA.



The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
-- Identify the highest-priced pizza.

SELECT
    pizza_types.name, pizzas.price
FROM
    pizza_types
JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
ORDER BY pizzas.price DESC
LIMIT 1;
```

The results pane displays the following table:

name	price
The Greek Pizza	35.95

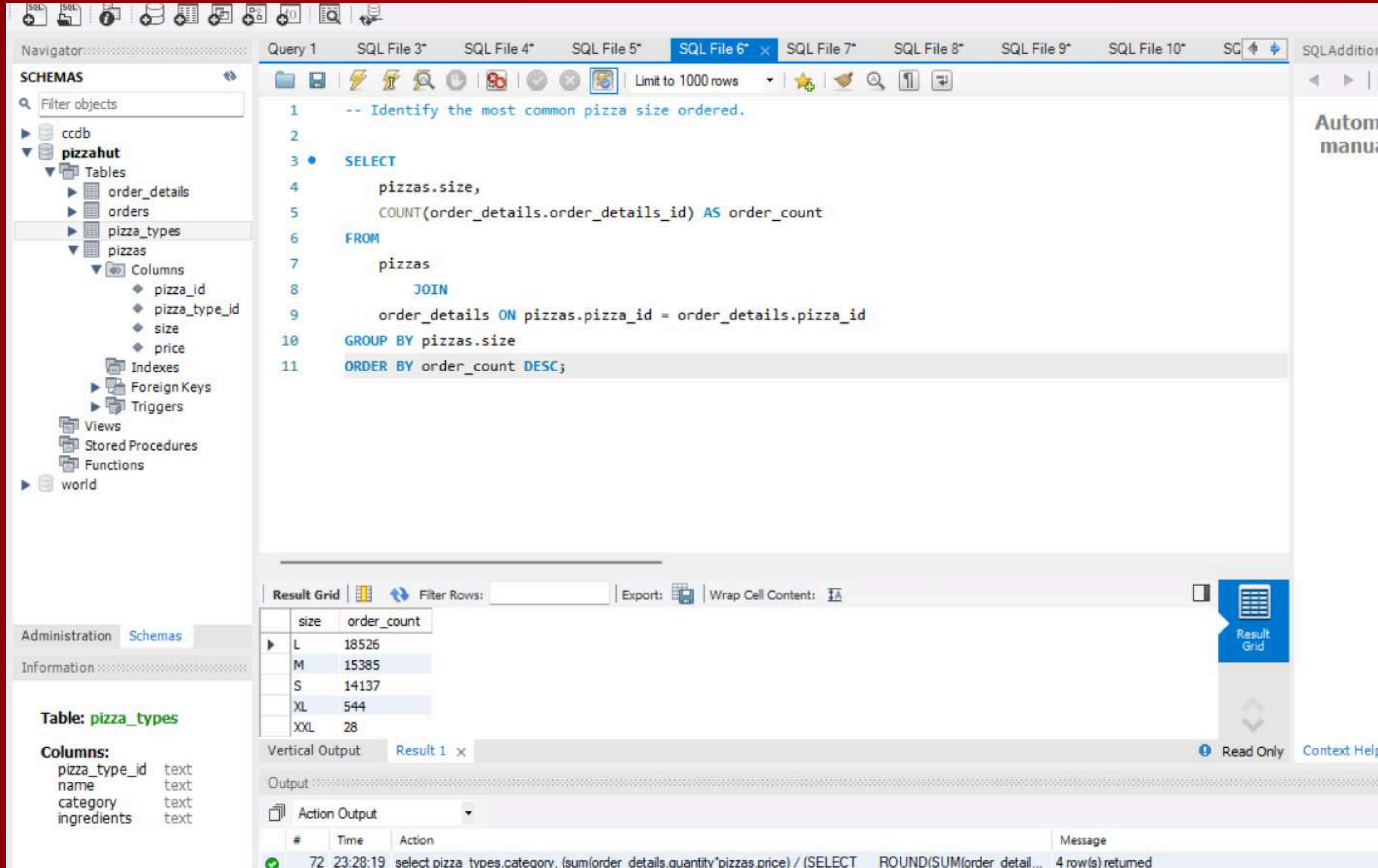
The interface also shows a schema browser on the left with the following structure:

- ccdb
 - pizzahut
 - Tables
 - order_details
 - orders
 - pizza_types
 - pizzas
 - Columns
 - pizza_id
 - pizza_type_id
 - size
 - price
 - Indexes
 - Foreign Keys
 - Triggers
- Views
- Stored Procedures
- Functions

- world

The bottom pane shows the output of the query, including a message: "4 row(s) returned".

Q4) IDENTIFY THE MOST COMMON PIZZA SIZE ORDERED.



The screenshot shows a SQL IDE interface with a query editor and a result grid. The query editor contains the following SQL code:

```
-- Identify the most common pizza size ordered.

SELECT
    pizzas.size,
    COUNT(order_details.order_details_id) AS order_count
FROM
    pizzas
JOIN
    order_details ON pizzas.pizza_id = order_details.pizza_id
GROUP BY pizzas.size
ORDER BY order_count DESC;
```

The result grid displays the following data:

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

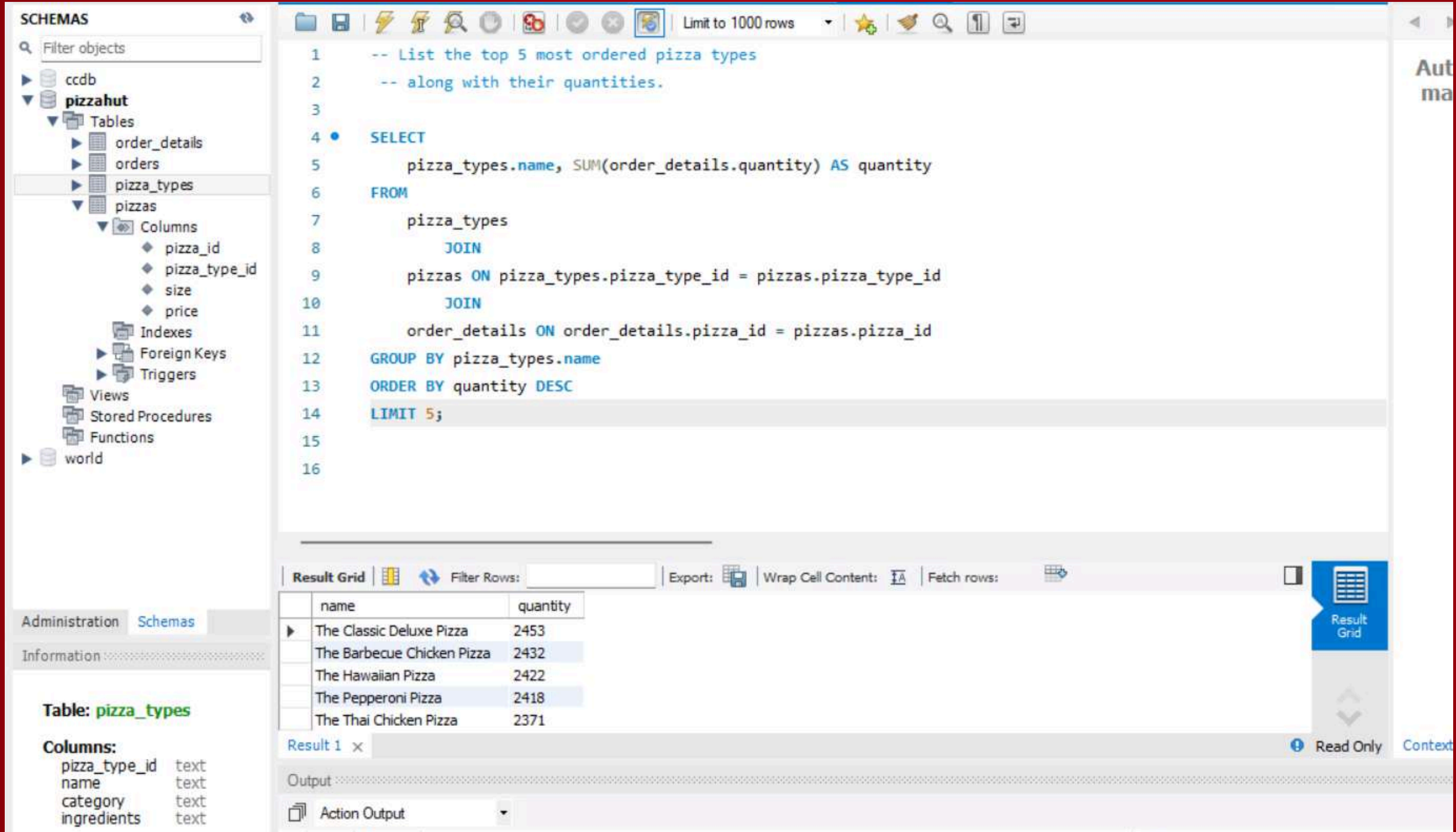
The IDE also shows a schema browser on the left with the following structure:

- ccdb
- pizzahut
 - Tables
 - order_details
 - orders
 - pizza_types
 - pizzas
 - Columns
 - pizza_id
 - pizza_type_id
 - size
 - price
 - Indexes
 - Foreign Keys
 - Triggers
 - Views
 - Stored Procedures
 - Functions
- world

The bottom of the IDE shows a message bar with the following text:

72 23:28:19 select pizza_types.category, (sum(order_details.quantity*pizzas.price) / (SELECT ROUND(SUM(order_detail... 4 row(s) returned

Q5) LIST THE TOP 5 MOST ORDERED PIZZA TYPES ALONG WITH THEIR QUANTITIES.



The screenshot displays a database management interface. On the left, a 'SCHEMAS' panel shows a tree view of the database structure, including a 'pizzahut' database with tables like 'order_details', 'orders', 'pizza_types', and 'pizzas'. The 'pizza_types' table is selected. The main area shows a SQL query:

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

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

 Below the query, the 'Result Grid' shows the top 5 most ordered pizza types:

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

 The bottom left shows the 'Table: pizza_types' with columns: pizza_type_id (text), name (text), category (text), and ingredients (text). The bottom right shows the 'Output' panel with 'Action Output'.

Administration Schemas

Information

Table: **pizza_types**

Columns:

- pizza_type_id text
- name text
- category text
- ingredients text

Result Grid

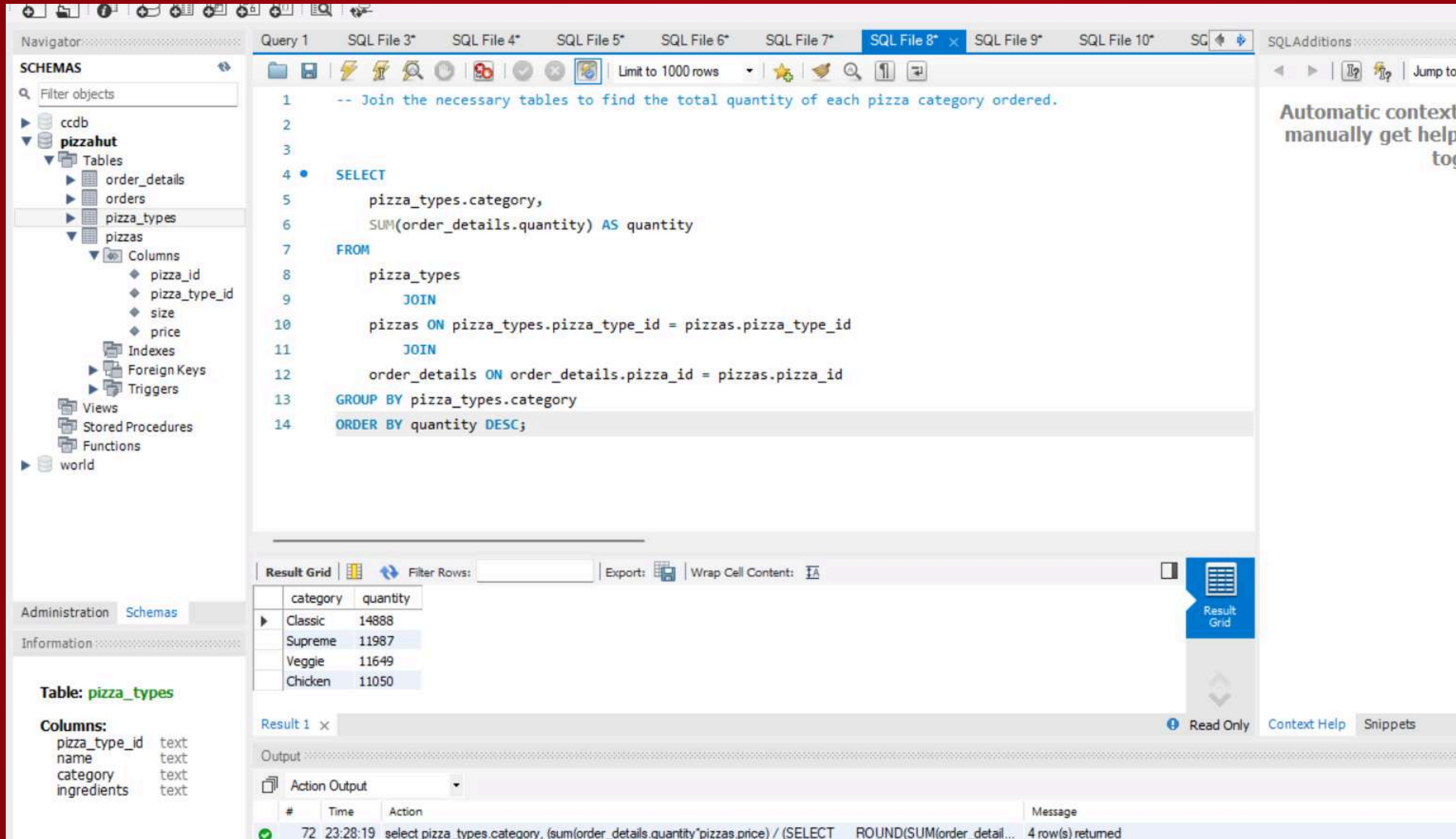
name	quantity
The Classic Deluxe Pizza	2453
The Barbecue Chicken Pizza	2432
The Hawaiian Pizza	2422
The Pepperoni Pizza	2418
The Thai Chicken Pizza	2371

Result 1 x

Output

Action Output

Q6) JOIN THE NECESSARY TABLES TO FIND THE TOTAL QUANTITY OF EACH PIZZA CATEGORY ORDERED.



The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

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

SELECT
    pizza_types.category,
    SUM(order_details.quantity) AS quantity
FROM
    pizza_types
    JOIN
    pizzas ON pizza_types.pizza_type_id = pizzas.pizza_type_id
    JOIN
    order_details ON order_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.category
ORDER BY quantity DESC;
```

The results pane displays the following table:

category	quantity
Classic	14888
Supreme	11987
Veggie	11649
Chicken	11050

The interface also shows a schema browser on the left with the following structure:

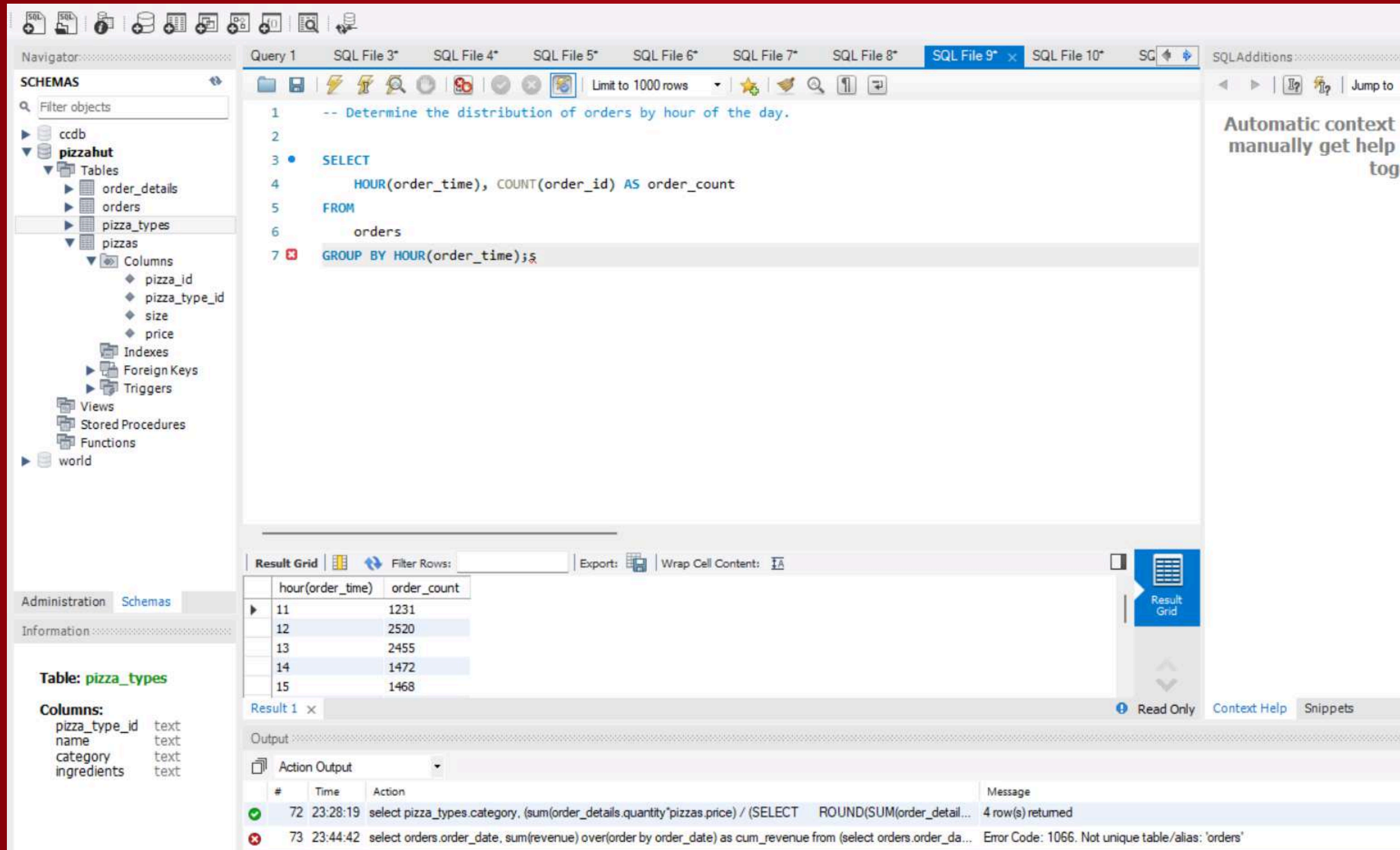
- ccdb
 - pizzahut
 - Tables
 - order_details
 - orders
 - pizza_types
 - pizzas
 - Columns
 - pizza_id
 - pizza_type_id
 - size
 - price
 - Indexes
 - Foreign Keys
 - Triggers
 - Views
 - Stored Procedures
 - Functions
 - world

The bottom of the interface shows a message pane with the following text:

```
72 23:28:19 select pizza_types.category, (sum(order_details.quantity*pizzas.price) / (SELECT ROUND(SUM(order_detail...
```

4 row(s) returned

Q7) DETERMINE THE DISTRIBUTION OF ORDERS BY HOUR OF THE DAY.



The screenshot displays a SQL IDE interface with the following components:

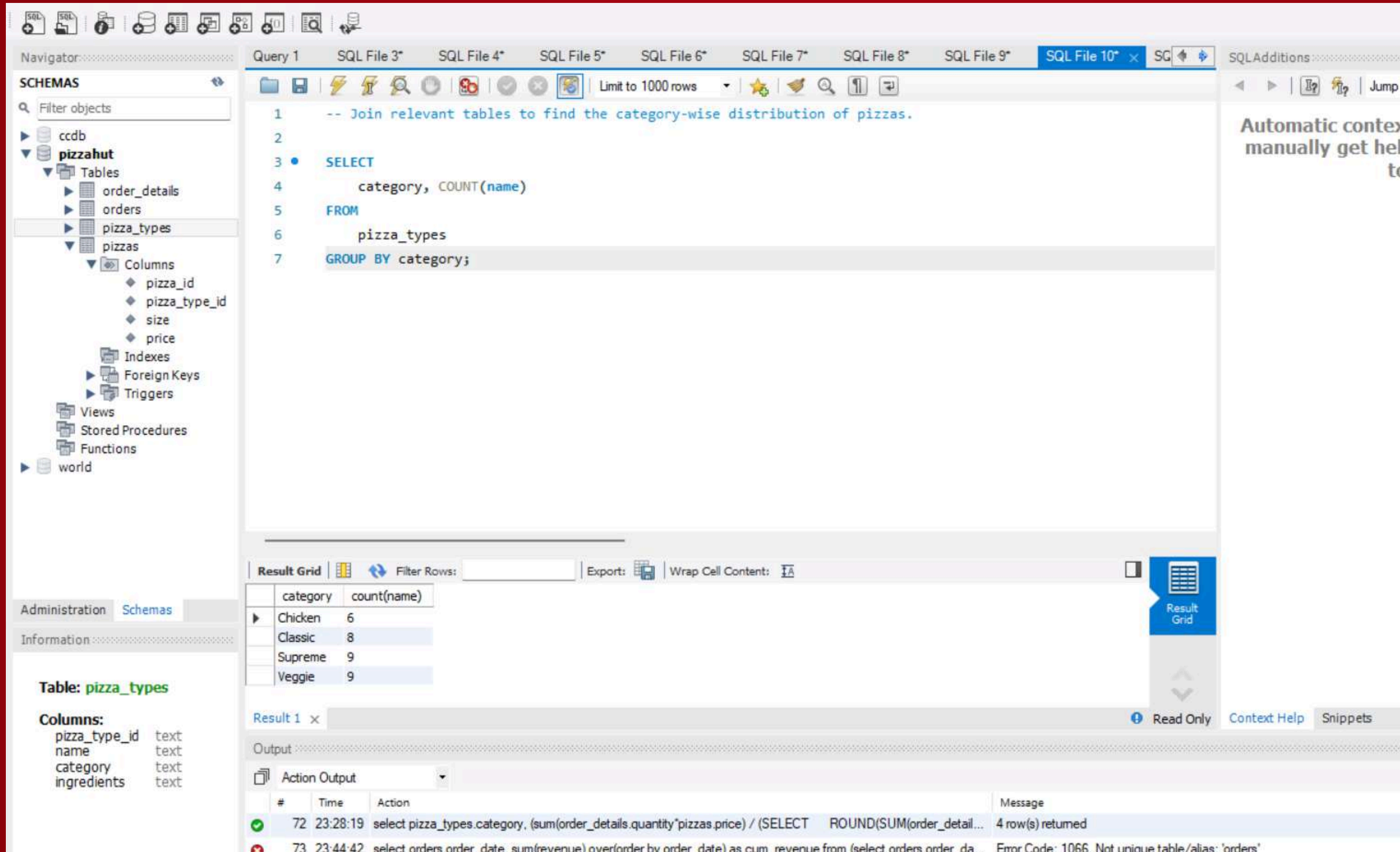
- Navigator:** Shows a database schema for 'pizzahut' with tables: order_details, orders, pizza_types, and pizzas. The 'pizzas' table is expanded, showing columns: pizza_id, pizza_type_id, size, price, and indexes.
- Query Editor:** Contains the following SQL query:

```
-- Determine the distribution of orders by hour of the day.

SELECT
    HOUR(order_time), COUNT(order_id) AS order_count
FROM
    orders
GROUP BY HOUR(order_time);
```
- Result Grid:** Displays the results of the query, showing the hour of the day and the corresponding order count.

hour(order_time)	order_count
11	1231
12	2520
13	2455
14	1472
15	1468
- Output Panel:** Shows the execution log with two entries:
 - 72 23:28:19 select pizza_types.category, (sum(order_details.quantity*pizzas.price) / (SELECT ROUND(SUM(order_detail... 4 row(s) returned
 - 73 23:44:42 select orders.order_date, sum(revenue) over(order by order_date) as cum_revenue from (select orders.order_da... Error Code: 1066. Not unique table/alias: 'orders'

Q8) JOIN RELEVANT TABLES TO FIND THE CATEGORY-WISE DISTRIBUTION OF PIZZAS.



Query 1 SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* x SG

Limit to 1000 rows

```
1 -- Join relevant tables to find the category-wise distribution of pizzas.
2
3 SELECT
4     category, COUNT(name)
5 FROM
6     pizza_types
7 GROUP BY category;
```

Result Grid

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

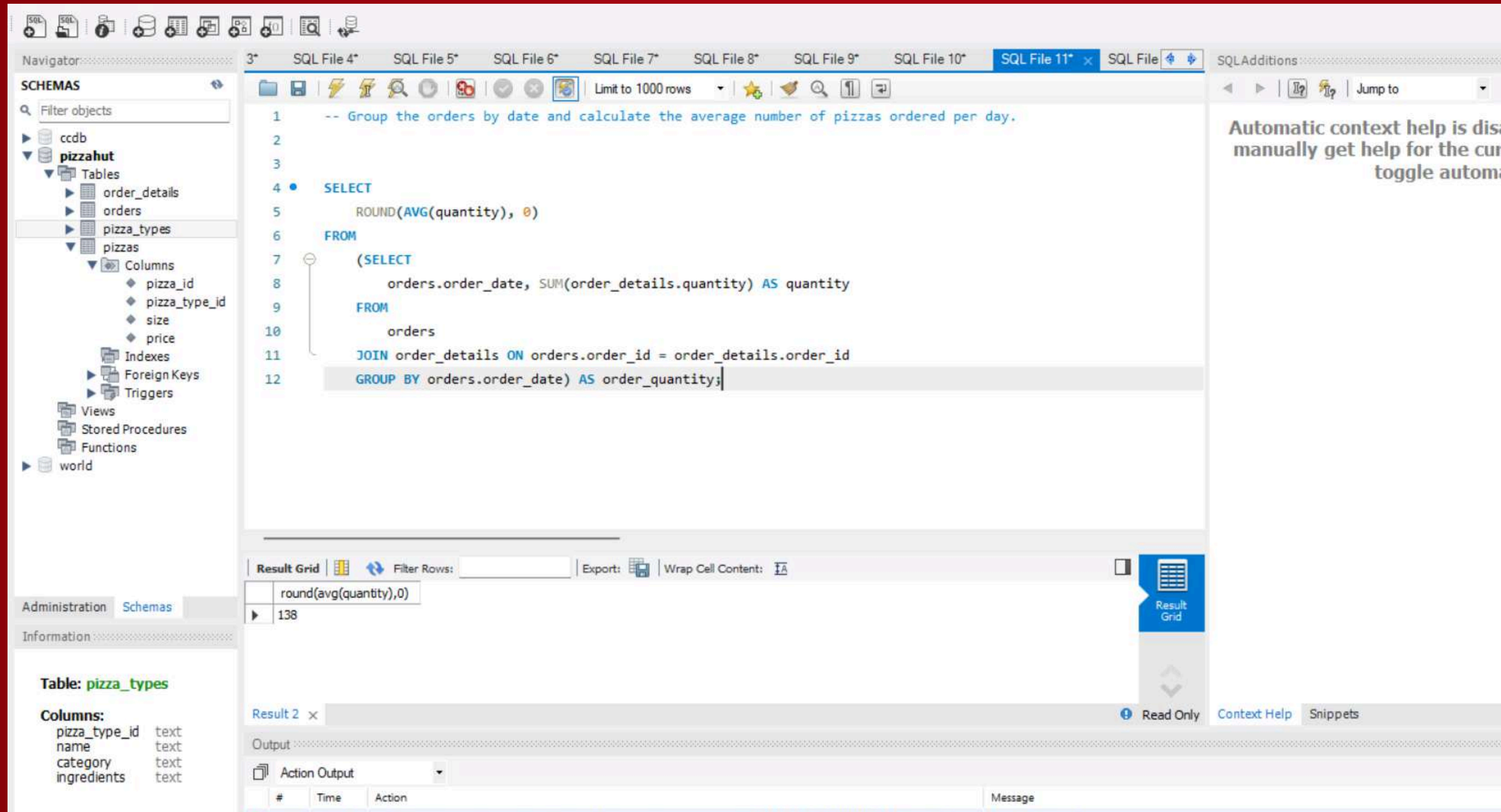
Result 1 x

Output

Action Output

#	Time	Action	Message
72	23:28:19	select pizza_types.category, (sum(order_details.quantity*pizzas.price) / (SELECT ROUND(SUM(order_detail...	4 row(s) returned
73	23:44:42	select orders.order_date, sum(revenue) over(order by order_date) as cum_revenue from (select orders.order_da...	Error Code: 1066. Not unique table/alias: 'orders'

Q9) GROUP THE ORDERS BY DATE AND CALCULATE THE AVERAGE NUMBER OF PIZZAS ORDERED PER DAY.



The screenshot shows a SQL IDE interface with a query editor, a schema browser, and a result grid. The query editor contains the following SQL code:

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

SELECT
    ROUND(AVG(quantity), 0)
FROM
    (SELECT
        orders.order_date, SUM(order_details.quantity) AS quantity
    FROM
        orders
    JOIN order_details ON orders.order_id = order_details.order_id
    GROUP BY orders.order_date) AS order_quantity;
```

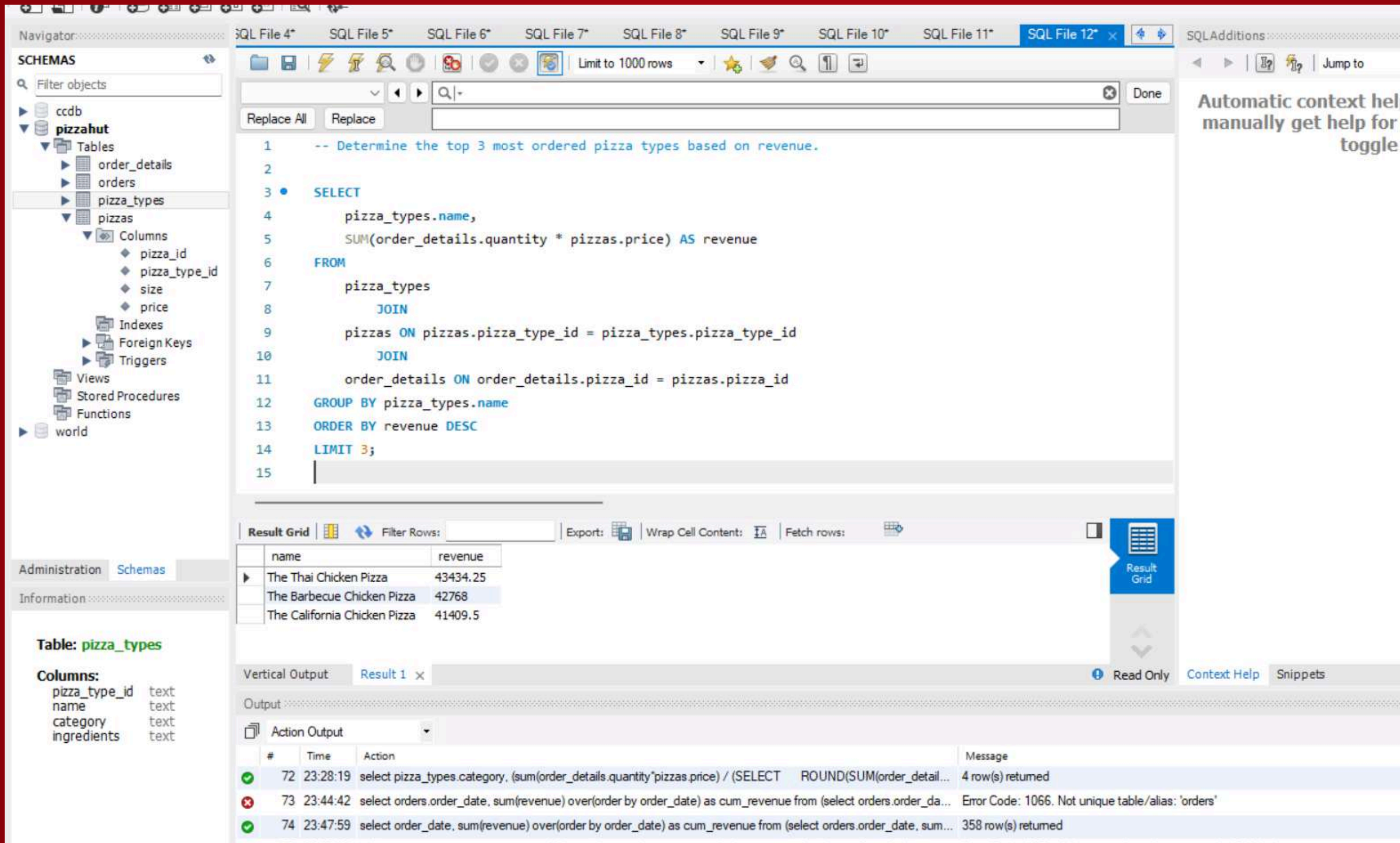
The schema browser on the left shows the database structure, including the **pizzahut** database with tables **order_details**, **orders**, **pizza_types**, and **pizzas**. The **pizzas** table has columns **pizza_id**, **pizza_type_id**, **size**, and **price**.

The result grid shows the output of the query, which is a single row with the value 138.

round(avg(quantity),0)
138

The bottom section of the IDE shows the **Table: pizza_types** with columns **pizza_type_id**, **name**, **category**, and **ingredients**.

Q10) DETERMINE THE TOP 3 MOST ORDERED PIZZA TYPES BASED ON REVENUE.



The screenshot shows a SQL IDE interface with a query editor, a schema browser, and a results pane. The query is as follows:

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

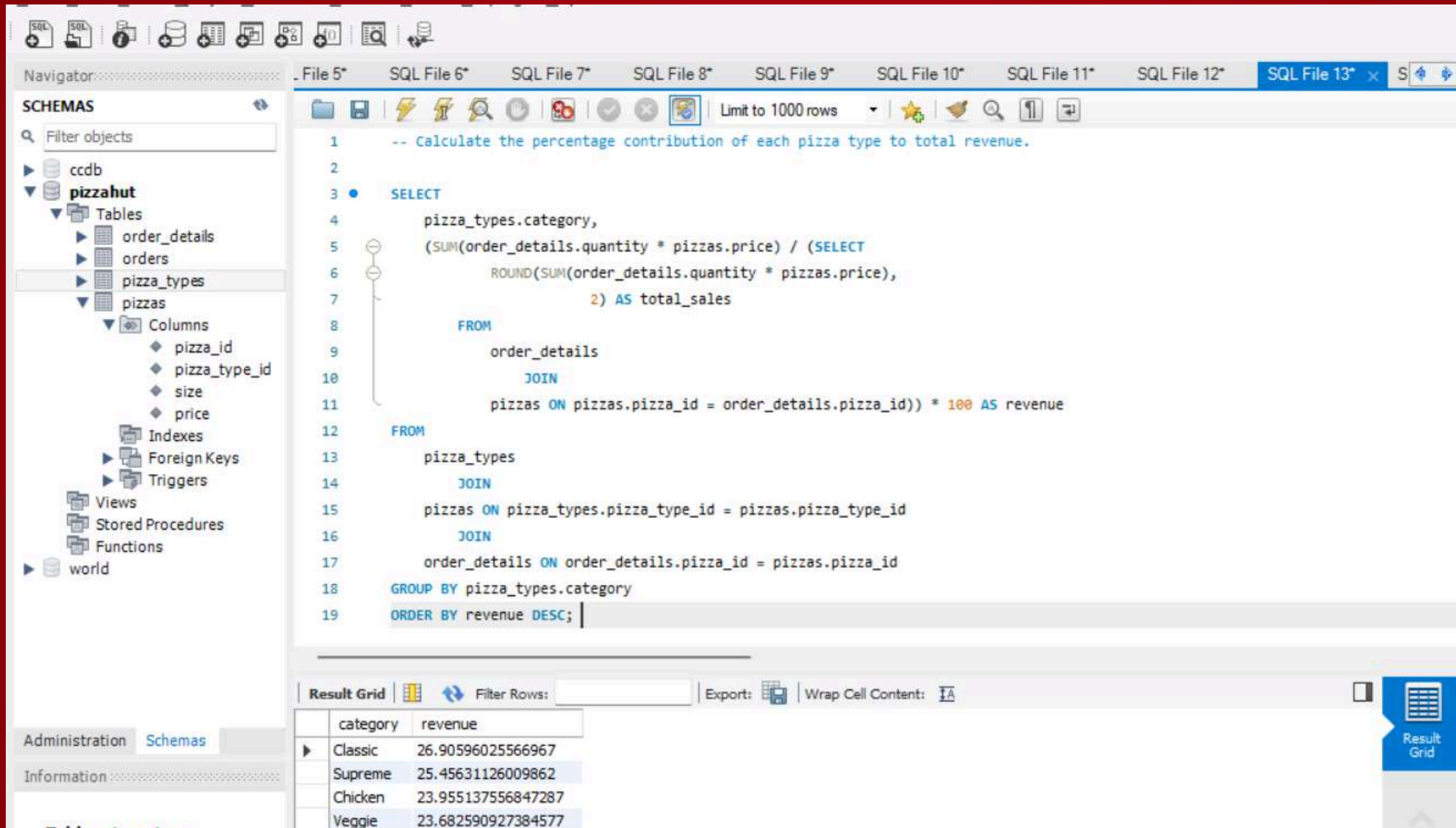
SELECT
    pizza_types.name,
    SUM(order_details.quantity * pizzas.price) AS revenue
FROM
    pizza_types
JOIN
    pizzas ON pizzas.pizza_type_id = pizza_types.pizza_type_id
JOIN
    order_details ON order_details.pizza_id = pizzas.pizza_id
GROUP BY pizza_types.name
ORDER BY revenue DESC
LIMIT 3;
```

The results are displayed in a grid:

name	revenue
The Thai Chicken Pizza	43434.25
The Barbecue Chicken Pizza	42768
The California Chicken Pizza	41409.5

The schema browser on the left shows the database structure, including tables like `order_details`, `orders`, `pizza_types`, and `pizzas`. The results pane at the bottom shows the execution log with messages indicating the number of rows returned for each query step.

Q 11) CALCULATE THE PERCENTAGE CONTRIBUTION OF EACH PIZZA TYPE TO TOTAL REVENUE.



The screenshot displays a SQL IDE interface with a query editor and a result grid. The query is designed to calculate the percentage contribution of each pizza type to the total revenue.

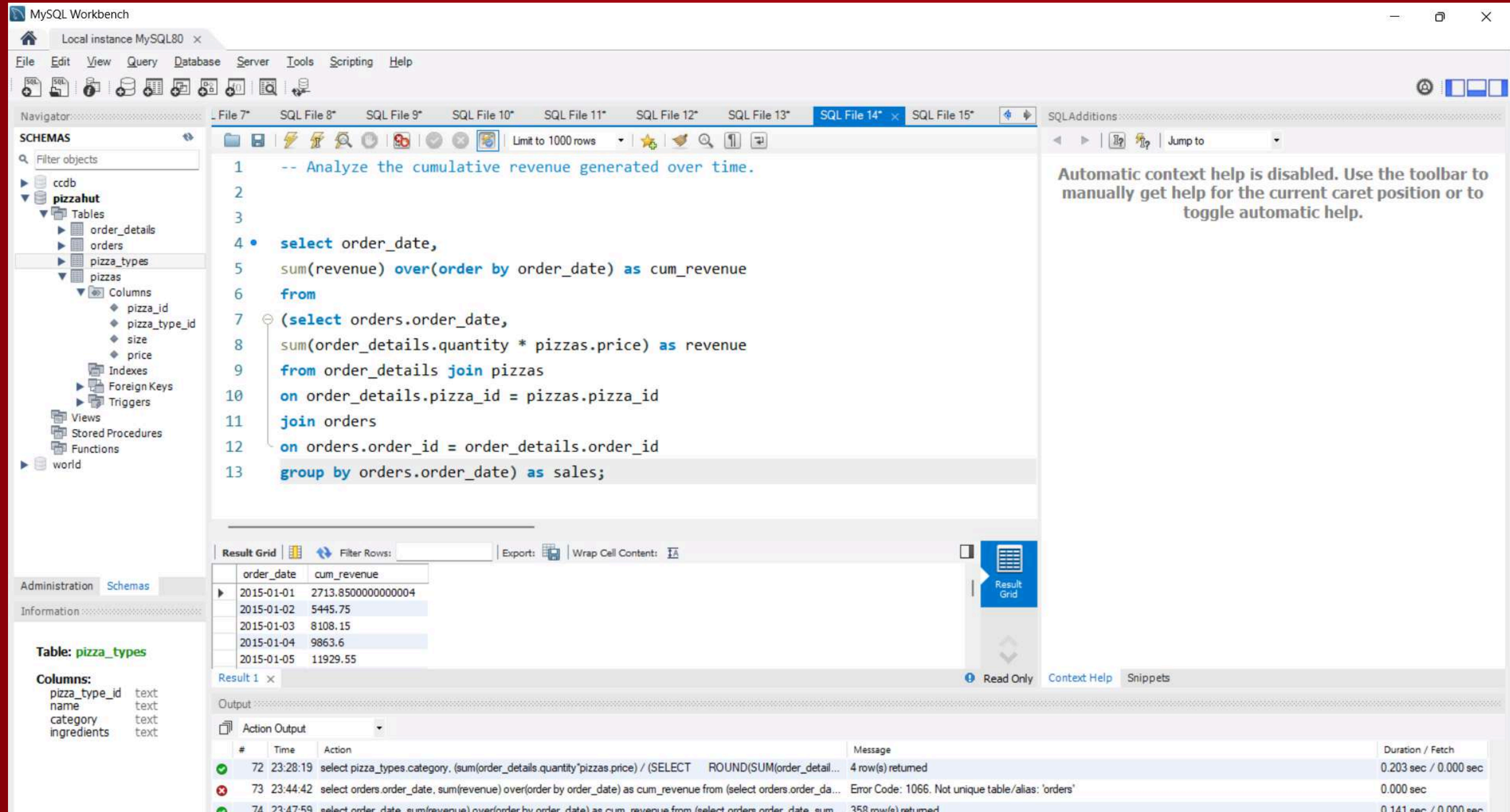
```
-- Calculate the percentage contribution of each pizza type to total revenue.

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

The result grid shows the following data:

category	revenue
Classic	26.90596025566967
Supreme	25.45631126009862
Chicken	23.955137556847287
Veggie	23.682590927384577

Q12) ANALYZE THE CUMULATIVE REVENUE GENERATED OVER TIME.



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with the 'pizzahut' database selected, containing tables like 'order_details', 'orders', 'pizza_types', and 'pizzas'. The main editor window shows a SQL query in 'SQL File 14':

```
-- 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(order_details.quantity * pizzas.price) as revenue
from order_details join pizzas
on order_details.pizza_id = pizzas.pizza_id
join orders
on orders.order_id = order_details.order_id
group by orders.order_date) as sales;
```

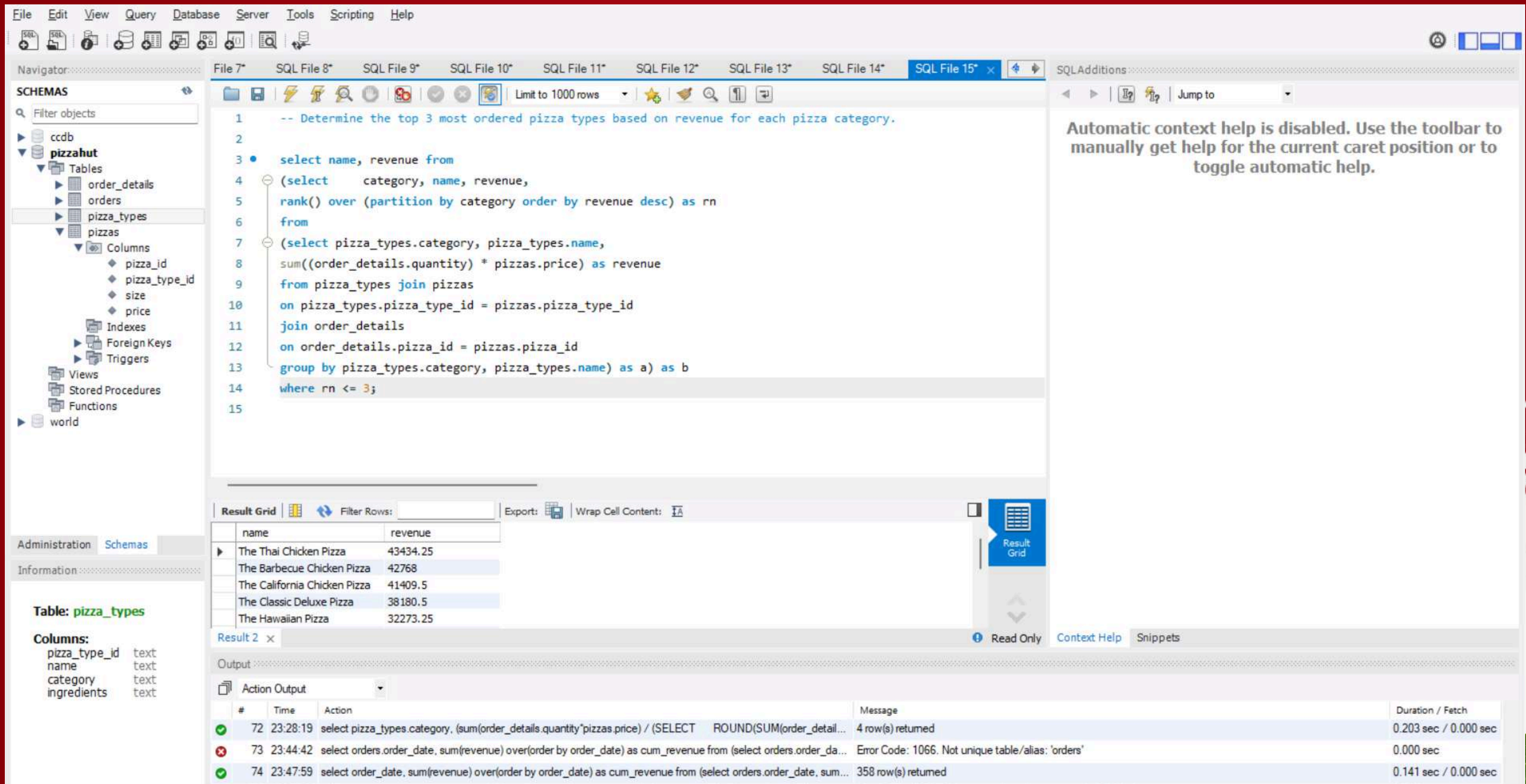
The 'Result Grid' at the bottom shows the output of the query:

order_date	cum_revenue
2015-01-01	2713.85000000000004
2015-01-02	5445.75
2015-01-03	8108.15
2015-01-04	9863.6
2015-01-05	11929.55

The bottom status bar shows the 'Output' tab with a table of execution logs:

#	Time	Action	Message	Duration / Fetch
72	23:28:19	select pizza_types.category, (sum(order_details.quantity*pizzas.price) / (SELECT ROUND(SUM(order_detail...	4 row(s) returned	0.203 sec / 0.000 sec
73	23:44:42	select orders.order_date, sum(revenue) over(order by order_date) as cum_revenue from (select orders.order_da...	Error Code: 1066. Not unique table/alias: 'orders'	0.000 sec
74	23:47:59	select order_date, sum(revenue) over(order by order_date) as cum_revenue from (select orders.order_date, sum...	358 row(s) returned	0.141 sec / 0.000 sec

Q13) DETERMINE THE TOP 3 MOST ORDERED PIZZA TYPES BASED ON REVENUE FOR EACH PIZZA CATEGORY.



The screenshot shows a SQL IDE interface with a query editor, a schema browser, and a results grid. The query is as follows:

```
-- 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((order_details.quantity) * pizzas.price) as revenue
from pizza_types join pizzas
on pizza_types.pizza_type_id = pizzas.pizza_type_id
join order_details
on order_details.pizza_id = pizzas.pizza_id
group by pizza_types.category, pizza_types.name) as a) as b
where rn <= 3;
```

The results grid shows the following data:

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 schema browser shows the following structure:

- ccdb
 - pizzahut
 - Tables
 - order_details
 - orders
 - pizza_types
 - Columns
 - pizza_id
 - pizza_type_id
 - size
 - price
 - Indexes
 - Foreign Keys
 - Triggers
 - Views
 - Stored Procedures
 - Functions
 - world



THANK YOU!

