Pizza sales analysis

SQL File 3* | orders

Limit to 1000 rows

```sql
create database pizzaarea;
use pizzaarea;

-- imported pizzas.csv and pizzatypes.csv in table using table data import wizard

select * from pizzaarea.pizzas;

create table orders(
order_id int not null,
order_date date not null,
order_time time not null,
primary key(order_id)
);
create table order_details(
order_details_id int not null,
order_id int not null,
pizza_id text not null,
quantity int not null,
primary key(order_details_id)
);

select * from pizzaarea.order_details;

show tables;
desc order_details;
desc orders;
desc pizza_types;
desc pizzas;
```

```sql
desc order_details;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
order_details_id	int	NO	PRI	NULL	
order_id	int	NO		NULL	
pizza_id	text	NO		NULL	
quantity	int	NO		NULL	

```sql
desc orders;
```

Result Grid | Filter Rows: | Export: | Wra

Field	Type	Null	Key	Default	Extra
order_id	int	NO	PRI	NULL	
order_date	date	NO		NULL	
order_time	time	NO		NULL	
discount	double	YES		0	
order_status	varchar(50)	YES		Pending	

```sql
desc pizzas;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
pizza_id	text	YES		NULL	
pizza_type_id	text	YES		NULL	
size	text	YES		NULL	
price	double	YES		NULL	

```sql
desc pizza_types;
desc pizzas;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
pizza_type_id	text	YES		NULL	
name	text	YES		NULL	
category	text	YES		NULL	
ingredients	text	YES		NULL	

```sql
desc pizzas;

-- 1. Retrieve the total number of orders placed.
select count(order_id) as total_orders from orders;

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

SELECT
    ROUND(SUM(o_d.quantity * p.price), 2) AS total_revenue
FROM
    pizzaarea.order_details AS o_d
        JOIN
    pizzaarea.pizzas AS p ON o_d.pizza_id = p.pizza_id;

-- 3. Identify the highest-priced pizza.
SELECT
    p_t.name, p.price
FROM
    pizzaarea.pizza_types AS p_t
        JOIN
    pizzaarea.pizzas AS p ON p_t.pizza_type_id = p.pizza_type_id
ORDER BY p.price DESC
LIMIT 1;
```

Result Grid

total_orders
21350

Result Grid

total_revenue
817860.05

Result Grid

name	price
The Greek Pizza	35.95

```sql
LIMIT 1;

-- 4. Identify the most common pizza size ordered.
select p.size,count(o_d.order_details_id) as total_order_count
from pizzaarea.order_details as o_d
join pizzaarea.pizzas as p
on o_d.pizza_id=p.pizza_id
group by p.size
order by total_order_count desc
limit 1;

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

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

Result Grid

size	total_order_count
L	18526

Result Grid

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

SQL File 3* | orders

Limit to 1000 rows

```sql

-- 6. Join the necessary tables to find the total quantity of each pizza category ordered.
SELECT
    p_t.category,
    SUM(o_d.quantity) AS total_quantity_of_each_pizza
FROM
    pizzaarea.pizza_types AS p_t
        JOIN
    pizzaarea.pizzas AS p ON p_t.pizza_type_id = p.pizza_type_id
        JOIN
    pizzaarea.order_details AS o_d ON p.pizza_id = o_d.pizza_id
GROUP BY p_t.category;

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

SELECT
    HOUR(order_time) AS hours, COUNT(order_id)
FROM
    pizzaarea.orders
GROUP BY hours;

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

select category, count(name) from pizzaarea.pizza_types
group by category;

```

Result Grid | Filter Rows:

category	total_quantity_of_each_pizza
Classic	14888
Veggie	11649
Supreme	11987
Chicken	11050

Result Grid | Filter Rows:

hours	COUNT(order_id)
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

Result Grid | Filter Ro

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

SQL File 3* | orders

Limit to 1000 rows

```sql
group by category;

-- 9. Group the orders by date and calculate the average number of pizzas ordered per day.
SELECT
    ROUND(AVG(total_orders), 0)
FROM
    (SELECT
        o.order_date, SUM(o_d.quantity) AS total_orders
    FROM
        pizzaarea.orders AS o
    JOIN pizzaarea.order_details AS o_d ON o.order_id = o_d.order_id
    GROUP BY o.order_date) AS average_number_of_pizzas_ordered_per_day;

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

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

Result Grid | Filter Rows:

ROUND(AVG(total_orders), 0)
138

Result Grid | Filter Rows:

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

SQL File 3* | orders

Limit to 1000 rows

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

SELECT
    p_t.category,
    ROUND((SUM(o_d.quantity * p.price) / (SELECT
                    SUM(o_d.quantity * p.price) AS total_revenue
                FROM
                    pizzaarea.order_details AS o_d
                        JOIN
                    pizzaarea.pizzas AS p ON o_d.pizza_id = p.pizza_id)) * 100,
            2) AS revenue_percentage
FROM
    pizzaarea.pizza_types AS p_t
        JOIN
    pizzaarea.pizzas AS p ON p_t.pizza_type_id = p.pizza_type_id
        JOIN
    pizzaarea.order_details AS o_d ON p.pizza_id = o_d.pizza_id
GROUP BY p_t.category
ORDER BY revenue_percentage DESC;
```

Result Grid | Filter Rows:

category	revenue_percentage
Classic	26.91
Supreme	25.46
Chicken	23.96
Veggie	23.68

SQL File 3* | orders

Limit to 1000 rows

```sql

-- 12. Analyze the cumulative revenue generated over time.

select order_date,
sum(revenue) over (order by order_date) as cumulative_revenue from
(select o.order_date , sum(o_d.quantity*p.price) as revenue
from pizzaarea.order_details as o_d
join pizzaarea.pizzas as p
on o_d.pizza_id=p.pizza_id
join pizzaarea.orders as o
on o.order_id=o_d.order_id
group by o.order_date) as sales;

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

select category, name ,revenue  from
(select category, name , revenue,
rank() over(partition by category order by revenue desc) as rn
from
(select p_t.category, p_t.name ,sum(o_d.quantity*p.price) as revenue
from pizzaarea.pizza_types as p_t
join pizzaarea.pizzas as p
on p_t.pizza_type_id=p.pizza_type_id
join pizzaarea.order_details as o_d
on p.pizza_id=o_d.pizza_id
group by p_t.category, p_t.name ) as sub_tbl ) as sub_tbl_2
where rn <= 3;
```

Result Grid | Filter Rows:

order_date	cumulative_revenue
2015-01-01	2713.8500000000004
2015-01-02	5445.75
2015-01-03	8108.15
2015-01-04	9863.6
2015-01-05	11929.55
2015-01-06	14358.5
2015-01-07	16560.7
2015-01-08	19399.05
2015-01-09	21526.4
2015-01-10	23990.350000000002
2015-01-11	25862.65
2015-01-12	27781.7
2015-01-13	29831.300000000003
2015-01-14	32358.700000000004
2015-01-15	34343.50000000001
2015-01-16	36937.65000000001
2015-01-17	39001.75000000001
2015-01-18	40978.600000000006
2015-01-19	43365.75000000001
2015-01-20	45763.65000000001
2015-01-21	47804.20000000001
2015-01-22	50300.90000000001
2015-01-23	52724.600000000006
2015-01-24	55013.850000000006
2015-01-25	56631.40000000001
2015-01-26	58515.80000000001
2015-01-27	61043.85000000001

Result 18

Result Grid | Filter Rows: | Export:

category	name	revenue
Chicken	The Thai Chicken Pizza	43434.25
Chicken	The Barbecue Chicken Pizza	42768
Chicken	The California Chicken Pizza	41409.5
Classic	The Classic Deluxe Pizza	38180.5
Classic	The Hawaiian Pizza	32273.25
Classic	The Pepperoni Pizza	30161.75
Supreme	The Spicy Italian Pizza	34831.25
Supreme	The Italian Supreme Pizza	33476.75
Supreme	The Sicilian Pizza	30940.5
Veggie	The Four Cheese Pizza	32265.70000000065
Veggie	The Mexicana Pizza	26780.75
Veggie	The Five Cheese Pizza	26066.5

SQL File 3* | orders

Limit to 1000 rows

```sql
-- 14. Create a Backup Table for Order Details
-- Ensure the new table copies both the structure and the data from the original table.
-- Add a mechanism to refresh this backup table periodically by syncing it with the latest data from order_details.

CREATE TABLE order_details_backup AS
SELECT *
FROM order_details;

select * from order_details_backup;

-- procedure to refreshing the backup table
DELIMITER //
CREATE PROCEDURE refresh_order_details_backup()
BEGIN
    -- Step 1: Remove old backup data
    DELETE FROM order_details_backup;

    -- Step 2: Insert the latest data from the original table
    INSERT INTO order_details_backup
    SELECT *
    FROM order_details;
END //
DELIMITER ;

call refresh_order_details_backup();
```

SQL File 3* | orders

Limit to 1000 rows

```sql
-- 15. Automating Discount Calculation for Loyal Customers
-- selects orders placed more than 5 times in the last 7 days.
-- Apply a 10% discount to order_details

-- ading a discount column

ALTER TABLE orders ADD COLUMN discount DOUBLE DEFAULT 0;

-- procedure for applying discount to eligible customers

DELIMITER //

CREATE PROCEDURE apply_discount()
BEGIN
    -- Declare a cursor to get order_ids with more than 5 orders in the last 7 days
    DECLARE done INT DEFAULT 0;
    DECLARE v_order_id INT;

    -- Declare the cursor to fetch orders that have more than 5 occurrences
    DECLARE cur CURSOR FOR
        SELECT o.order_id
        FROM orders as o
        JOIN order_details as o_d ON o.order_id = o_d.order_id
        WHERE o.order_date < CURDATE() - INTERVAL 7 DAY
        GROUP BY o.order_id
        HAVING COUNT(o.order_id) > 5;

    -- Declare CONTINUE HANDLER for cursor fetch completion
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

SQL File 3* | orders

Limit to 1000 rows

```
    -- Declare CONTINUE HANDLER for cursor fetch completion
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open the cursor
    OPEN cur;

    -- Loop through the cursor and apply discount
    read_loop: LOOP
        FETCH cur INTO v_order_id;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Apply discount to orders for the selected order_id
        UPDATE orders
        SET discount = 0.1  -- 10% discount
        WHERE order_id = v_order_id;

    END LOOP;

    CLOSE cur;
END //

DELIMITER ;

CALL apply_discount();
```