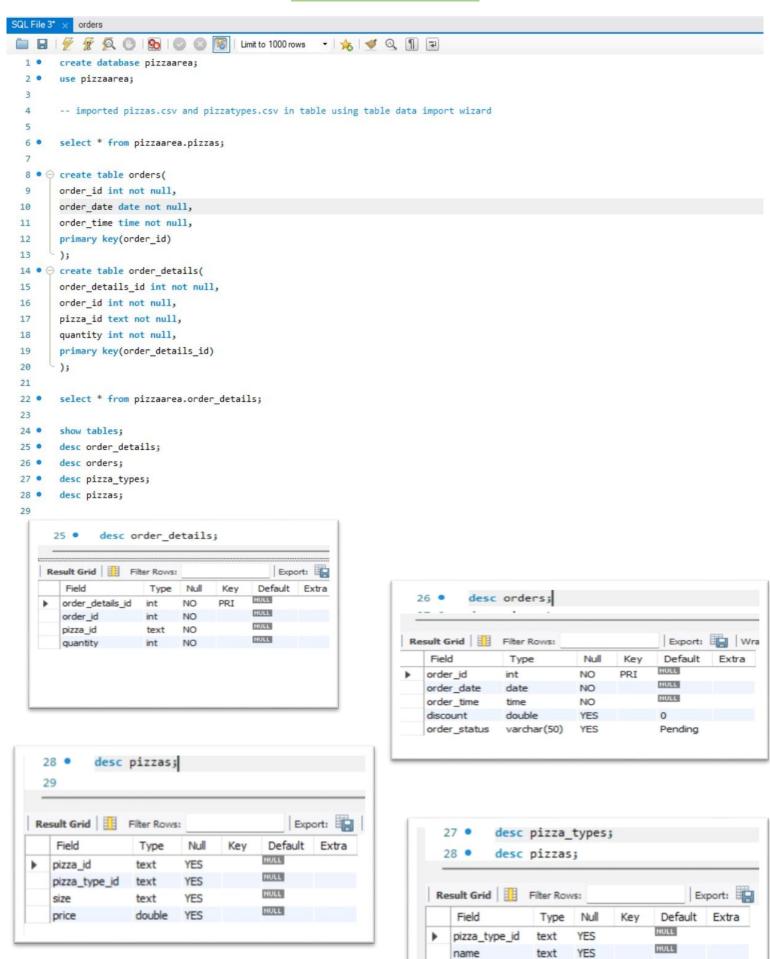
Pizza sales analysis



category

ingredients

text

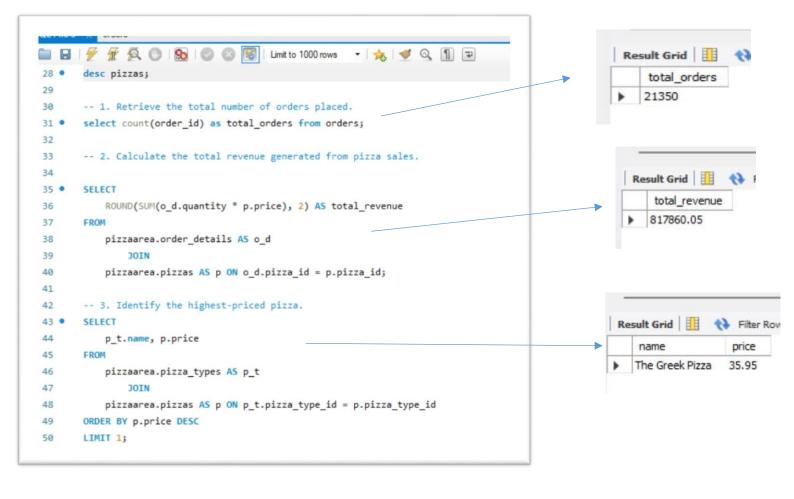
text

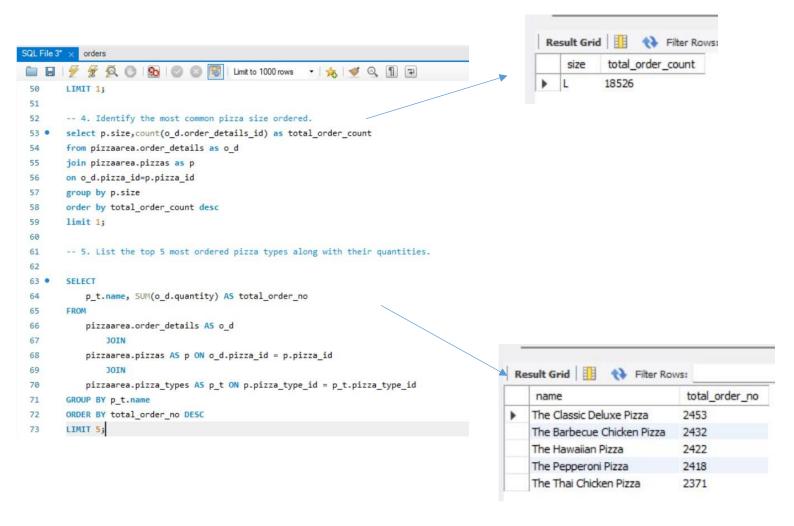
YES

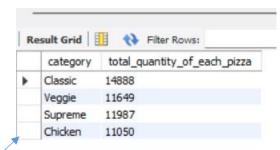
YES

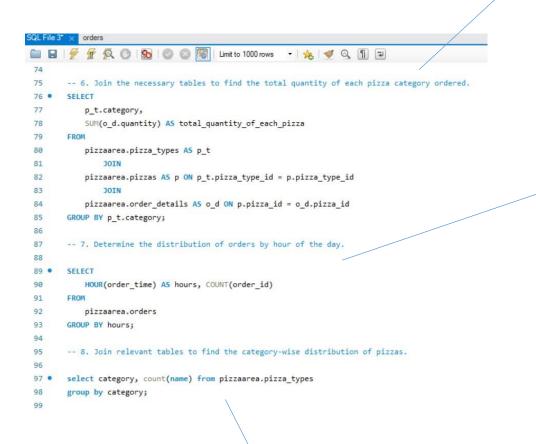
NULL

NULL





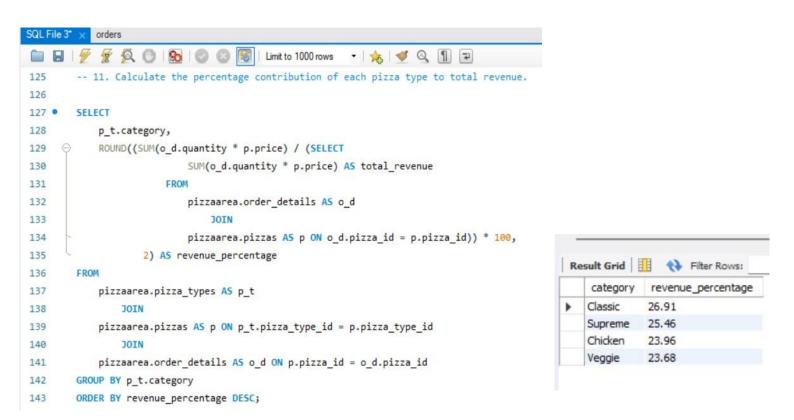


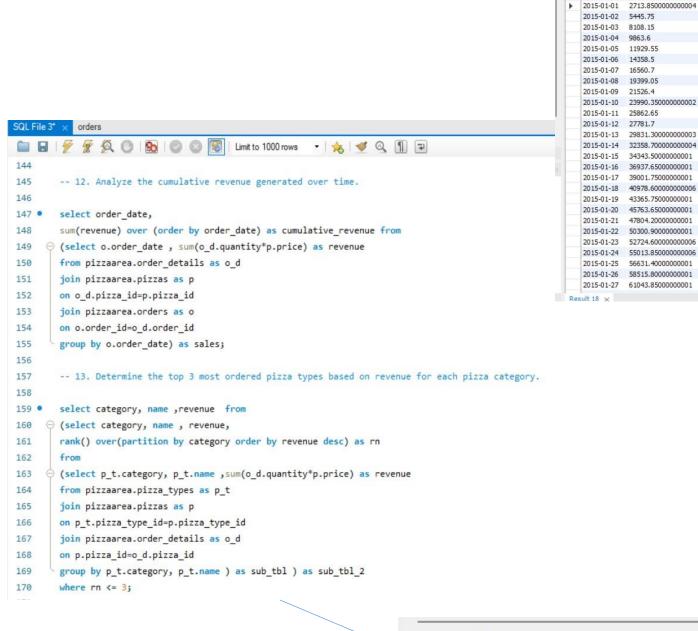


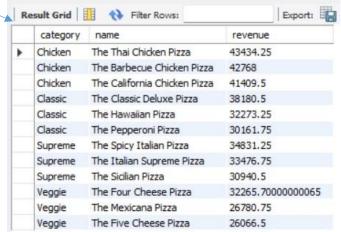
Re	esult Grid	Filter Row
	hours	COUNT(order_id)
•	11	1231
	12	2520
	13	2455
	14	1472
	15	1468
	16	1920
	17	2336
	18	2399
	19	2009
	20	1642
	21	1198
	22	663
	23	28
	10	8
	9	1

R	esult Grid	☐ () Filter
	category	count(name)
•	Chicken	6
	Classic	8
	Supreme	9
	Veggie	9

```
SQL File 3
                                         Limit to 1000 rows
        group by category;
99
        -- 9. Group the orders by date and calculate the average number of pizzas ordered per day.
100
        SELECT
101 .
            ROUND(AVG(total orders), 0)
102
                                                                                           FROM
103
104
            (SELECT
                                                                                                 ROUND(AVG(total_orders), 0)
                o.order_date, SUM(o_d.quantity) AS total_orders
105
                                                                                                138
106
            FROM
107
                pizzaarea.orders AS o
            JOIN pizzaarea.order_details AS o_d ON o.order_id = o_d.order_id
108
            GROUP BY o.order_date) AS average_number_of_pizzas_ordered_per_day;
109
110
111
        -- 10. Determine the top 3 most ordered pizza types based on revenue.
112
113
        SELECT
114
            p_t.name, SUM(o_d.quantity * p.price) AS revenue
115
        FROM
            pizzaarea.pizza_types AS p_t
116
117
                JOIN
            pizzaarea.pizzas AS p ON p_t.pizza_type_id = p.pizza_type_id
118
119
            pizzaarea.order_details AS o_d ON p.pizza_id = o_d.pizza_id
120
                                                                                        Result Grid
                                                                                                         Filter Rows:
        GROUP BY p_t.name
121
                                                                                           name
                                                                                                                      revenue
        ORDER BY revenue DESC
122
                                                                                           The Thai Chicken Pizza
                                                                                                                      43434.25
123
        LIMIT 3;
                                                                                           The Barbecue Chicken Pizza
                                                                                                                      42768
                                                                                           The California Chicken Pizza
                                                                                                                      41409.5
```







order_date

cumulative_revenue

```
SQL File 3" × orders
🚞 🔒 | 🦩 🙀 👰 🔘 | 🚱 | 💿 🔞 | Limit to 1000 rows
                                                         - | 🏂 | 🦪 🔍 👖 🖃
        -- 14. Create a Backup Table for Order Details
        -- Ensure the new table copies both the structure and the data from the original table.
173
        -- Add a mechanism to refresh this backup table periodically by syncing it with the latest data from order_details.
174
175
176 •
        CREATE TABLE order_details_backup AS
        SELECT *
177
178
        FROM order details;
179
        select * from order_details_backup;
180 •
181
        -- procedure to refreshing the backup table
182
        DELIMITER //
183
        CREATE PROCEDURE refresh_order_details_backup()
184 •
185
            -- Step 1: Remove old backup data
186
187
            DELETE FROM order details backup;
188
189
            -- Step 2: Insert the latest data from the original table
            INSERT INTO order details backup
190
191
            SELECT *
192
            FROM order_details;
        END //
193
        DELIMITER ;
194
195
196 •
        call refresh_order_details_backup();
107
```

```
SQL File 3" × orders
             ₹ 🙊 🔘 I 😘 I 🐷 🚳 🐻
                                          Limit to 1000 rows
                                                           - | 🏡 | 🥩 🔍 🗻 🖃
    198
         -- 15. Automating Discount Calculation for Loyal Customers
199
         -- selects orders placed more than 5 times in the last 7 days.
200
         -- Apply a 10% discount to order_details
201
202
         -- ading a discount column
203
         ALTER TABLE orders ADD COLUMN discount DOUBLE DEFAULT 0;
204 •
205
         -- procedure for applying discount to eligible customers
206
207
208
         DELIMITER //
209
210 •
         CREATE PROCEDURE apply_discount()
211
              - Declare a cursor to get order_ids with more than 5 orders in the last 7 days
212
213
             DECLARE done INT DEFAULT 0;
            DECLARE v_order_id INT;
214
215
216
             -- Declare the cursor to fetch orders that have more than 5 occurrences
217
             DECLARE cur CURSOR FOR
218
                 SELECT o.order_id
219
                 FROM orders as o
220
                 JOIN order_details as o_d ON o.order_id = o_d.order_id
221
                WHERE o.order_date < CURDATE() - INTERVAL 7 DAY
222
                 GROUP BY o.order_id
223
                 HAVING COUNT(o.order_id) > 5;
224
225
             -- Declare CONTINUE HANDLER for cursor fetch completion
226
             DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
227
```

```
SQL File 3* ×
           orders
            1 Limit to 1000 rows
                                                     - | 🏡 | 🥩 🔍 🗻 🖃
225
            -- Declare CONTINUE HANDLER for cursor fetch completion
            DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
226
227
228
           -- Open the cursor
229
           OPEN cur;
230
231
            -- Loop through the cursor and apply discount
232
           read_loop: LOOP
233
               FETCH cur INTO v_order_id;
               IF done THEN
234
235
                   LEAVE read_loop;
               END IF;
236
237
238
                -- Apply discount to orders for the selected order_id
               UPDATE orders
239
240
               SET discount = 0.1 -- 10% discount
241
               WHERE order_id = v_order_id;
242
243
      END LOOP;
244
245
           CLOSE cur;
246
      END //
247
        DELIMITER;
248
249
250 • CALL apply_discount();
251
252
```