SQL CASE STUDY

DATA IN MOTION
TINY SHOP SALES

DATA IN MOTION

OVERVIEW:

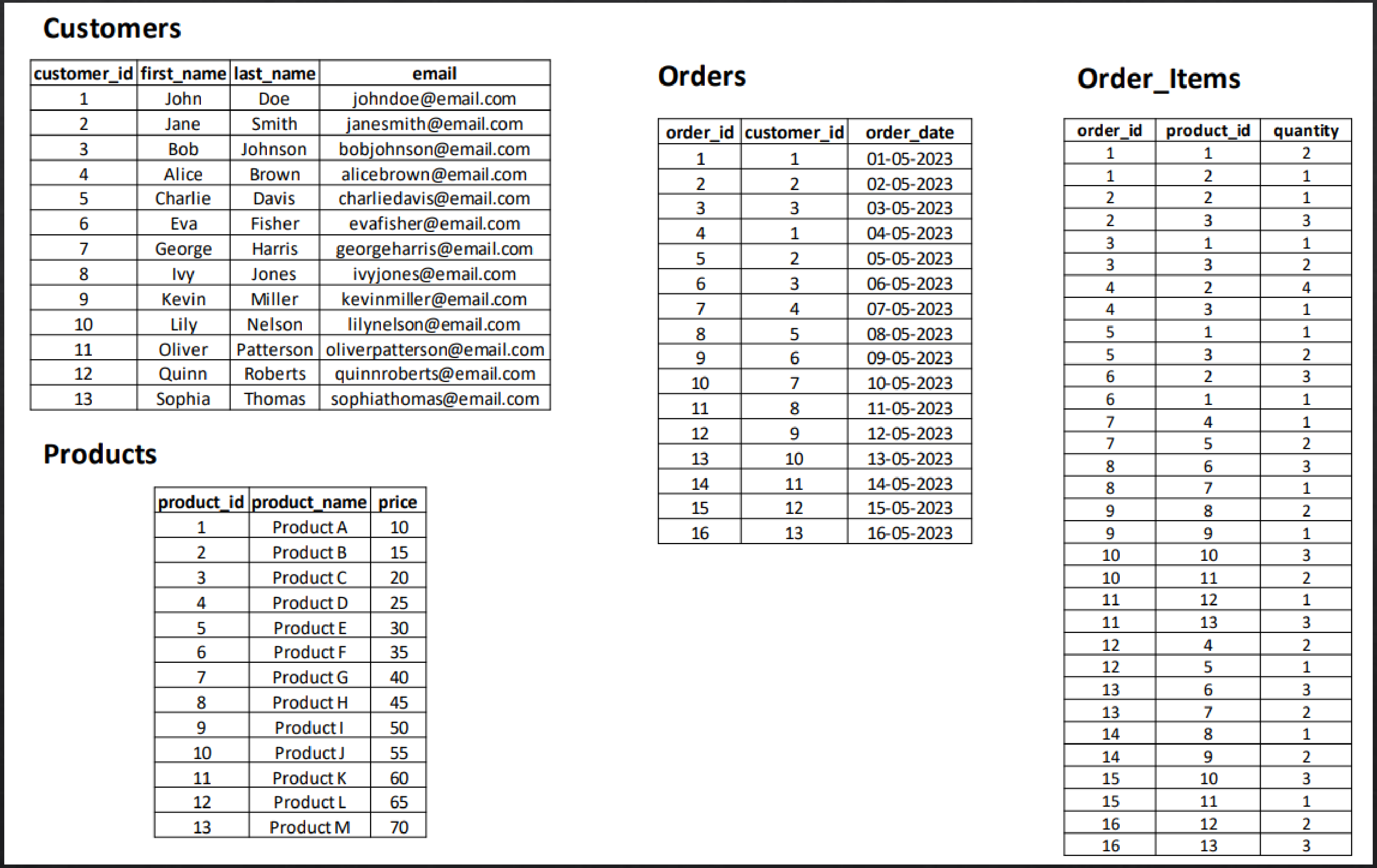
The Case study focusses on analyzing the sales of the Tiny Shop over time using SQL.
The database contains different tables that include:

- Products: Product level info like the names, prices of the items.
- Customers: Info related to customer – Id, Name, Email address.
- Orders: Order level info – Ids & Purchase date.
- Order_items: Item level info like quantity, basket value.

TOPICS COVERED:

- Basic aggregations
- CASE WHEN statements
- Window Functions
- Joins
- Date time functions
- CTEs

TABLES:



Customers

customer_id	first_name	last_name	email
1	John	Doe	johndoe@email.com
2	Jane	Smith	janesmith@email.com
3	Bob	Johnson	bobjohnson@email.com
4	Alice	Brown	alicebrown@email.com
5	Charlie	Davis	charliedavis@email.com
6	Eva	Fisher	evafisher@email.com
7	George	Harris	georgeharris@email.com
8	Ivy	Jones	ivyjones@email.com
9	Kevin	Miller	kevinmiller@email.com
10	Lily	Nelson	lilynelson@email.com
11	Oliver	Patterson	oliverpatterson@email.com
12	Quinn	Roberts	quinnroberts@email.com
13	Sophia	Thomas	sophiathomas@email.com

Products

product_id	product_name	price
1	Product A	10
2	Product B	15
3	Product C	20
4	Product D	25
5	Product E	30
6	Product F	35
7	Product G	40
8	Product H	45
9	Product I	50
10	Product J	55
11	Product K	60
12	Product L	65
13	Product M	70

Orders

order_id	customer_id	order_date
1	1	01-05-2023
2	2	02-05-2023
3	3	03-05-2023
4	1	04-05-2023
5	2	05-05-2023
6	3	06-05-2023
7	4	07-05-2023
8	5	08-05-2023
9	6	09-05-2023
10	7	10-05-2023
11	8	11-05-2023
12	9	12-05-2023
13	10	13-05-2023
14	11	14-05-2023
15	12	15-05-2023
16	13	16-05-2023

Order_Items

order_id	product_id	quantity
1	1	2
1	2	1
2	2	1
2	3	3
3	1	1
3	3	2
4	2	4
4	3	1
5	1	1
5	3	2
6	2	3
6	1	1
7	4	1
7	5	2
8	6	3
8	7	1
9	8	2
9	9	1
10	10	3
10	11	2
11	12	1
11	13	3
12	4	2
12	5	1
13	6	3
13	7	2
14	8	1
14	9	2
15	10	3
15	11	1
16	12	2
16	13	3

SQL SCHEMA:

```sql
CREATE TABLE customers (
customer_id integer PRIMARY KEY,
first_name varchar(100),
last_name varchar(100),
email varchar(100)
);
CREATE TABLE products (
product_id integer PRIMARY KEY,
product_name varchar(100),
price decimal );
CREATE TABLE orders (
order_id integer PRIMARY KEY,
customer_id integer,
order_date date );
CREATE TABLE order_items (
order_id integer,
product_id integer,
quantity integer );
```

```
INSERT INTO customers (customer_id, first_name, last_name, email) VALUES
(1, 'John', 'Doe', 'johndoe@email.com’),
(2, 'Jane', 'Smith', 'janesmith@email.com’),
(3, 'Bob', 'Johnson', 'bobjohnson@email.com’),
(4, 'Alice', 'Brown', 'alicebrown@email.com’),
(5, 'Charlie', 'Davis', 'charliedavis@email.com’),
(6, 'Eva', 'Fisher', 'evafisher@email.com’),
(7, 'George', 'Harris', 'georgeharris@email.com’),
(8, 'Ivy', 'Jones', 'ivyjones@email.com’),
(9, 'Kevin', 'Miller', 'kevinmiller@email.com’),
(10, 'Lily', 'Nelson', 'lilynelson@email.com’),
(11, 'Oliver', 'Patterson', 'oliverpatterson@email.com’),
(12, 'Quinn', 'Roberts', 'quinnroberts@email.com’),
(13, 'Sophia', 'Thomas', 'sophiathomas@email.com’);

INSERT INTO products (product_id, product_name, price) VALUES
(1, 'Product A', 10.00),
(2, 'Product B', 15.00),
(3, 'Product C', 20.00),
(4, 'Product D', 25.00),
(5, 'Product E', 30.00),
(6, 'Product F', 35.00),
(7, 'Product G', 40.00),
```

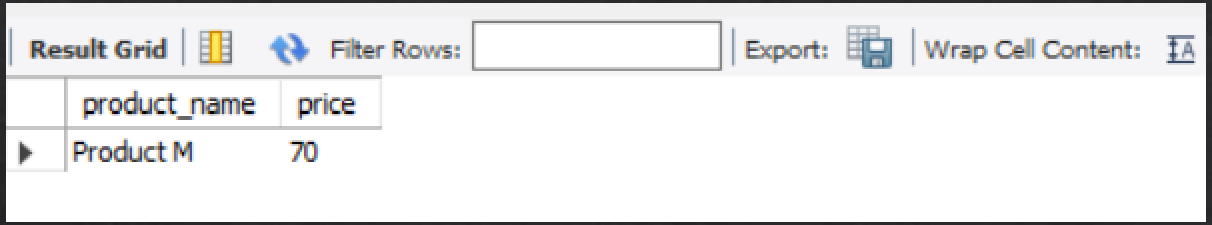
```sql
(8, 'Product H', 45.00),
(9, 'Product I', 50.00),
(10, 'Product J', 55.00),
(11, 'Product K', 60.00),
(12, 'Product L', 65.00),
(13, 'Product M', 70.00);

INSERT INTO orders (order_id, customer_id, order_date) VALUES
(1, 1, '2023-05-01’),
(2, 2, '2023-05-02’),
(3, 3, '2023-05-03’),
(4, 1, '2023-05-04’),
(5, 2, '2023-05-05’),
(6, 3, '2023-05-06’),
(7, 4, '2023-05-07’),
(8, 5, '2023-05-08’),
(9, 6, '2023-05-09’),
(10, 7, '2023-05-10’),
(11, 8, '2023-05-11’),
(12, 9, '2023-05-12’),
(13, 10, '2023-05-13’),
(14, 11, '2023-05-14’),
(15, 12, '2023-05-15’),
(16, 13, '2023-05-16’);
INSERT INTO order_items (order_id, product_id, quantity) VALUES
(1, 1, 2),
(1, 2, 1),
(2, 2, 1),
(2, 3, 3),
(3, 1, 1),
(3, 3, 2),
(4, 2, 4),
(4, 3, 1),
(5, 1, 1),
(5, 3, 2),
(6, 2, 3),
(6, 1, 1),
(7, 4, 1),
(7, 5, 2),
(8, 6, 3),
(8, 7, 1),
(9, 8, 2),
(9, 9, 1),
(10, 10, 3),
(10, 11, 2),
(11, 12, 1),
(11, 13, 3),
(12, 4, 2),
(12, 5, 1),
(13, 6, 3),
(13, 7, 2),
(14, 8, 1),
(14, 9, 2),
(15, 10, 3),
(15, 11, 1),
(16, 12, 2),
(16, 13, 3);
```

QUESTIONS:

1. Which product has the highest price? Only return a single row.
2. Which customer has made the most orders?
3. What’s the total revenue per product?
4. Find the day with the highest revenue.
5. Find the first order (by date) for each customer.
6. Find the top 3 customers who have ordered the most distinct products
7. Which product has been bought the least in terms of quantity?
8. What is the median order total?
9. For each order, determine if it was ‘Expensive’ (total over 300), ‘Affordable’ (total over 100), or ‘Cheap’.
10. Find customers who have ordered the product with the highest price.

1]Which product has the highest price? Only return a single row.

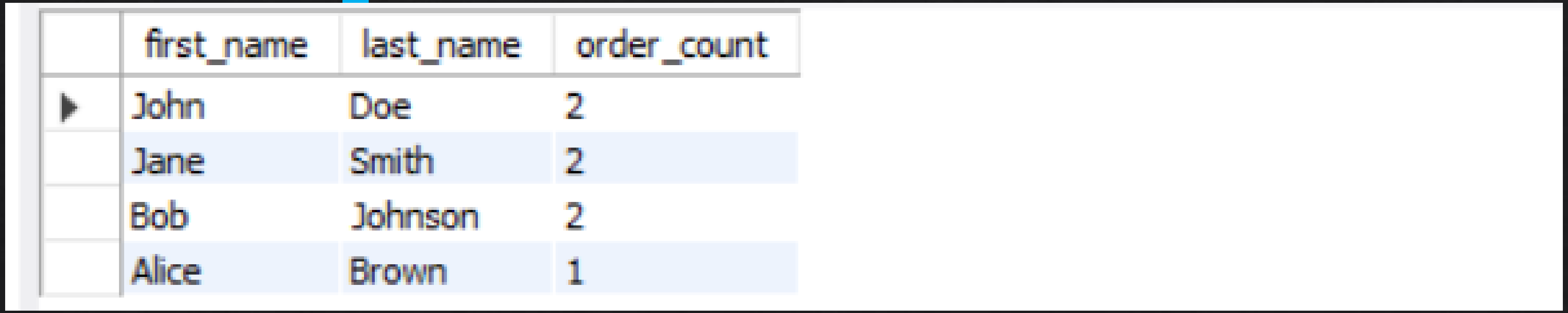
```
SELECT product_name, price FROM products
WHERE price = (SELECT MAX(price) from products);
```



product_name	price
Product M	70

2]Which customer has made the most orders?

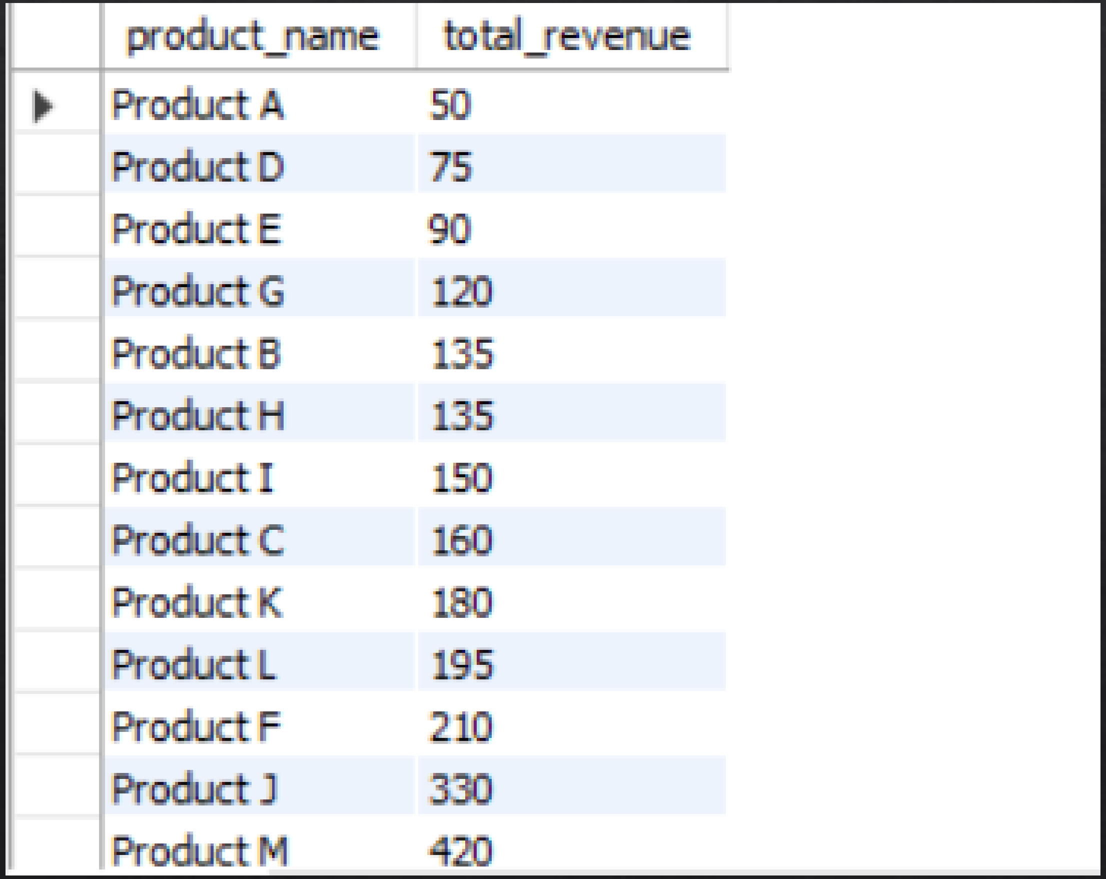
```
SELECT customers.first_name, customers.last_name, COUNT(orders.order_id) AS order_count FROM
customers
JOIN orders ON customers.customer_id = orders.customer_idGROUP BY customers.first_name,
customers.last_name
ORDER BY order_count DESC LIMIT 1;
```



first_name	last_name	order_count
John	Doe	2
Jane	Smith	2
Bob	Johnson	2
Alice	Brown	1

3]What’s the total revenue per product?

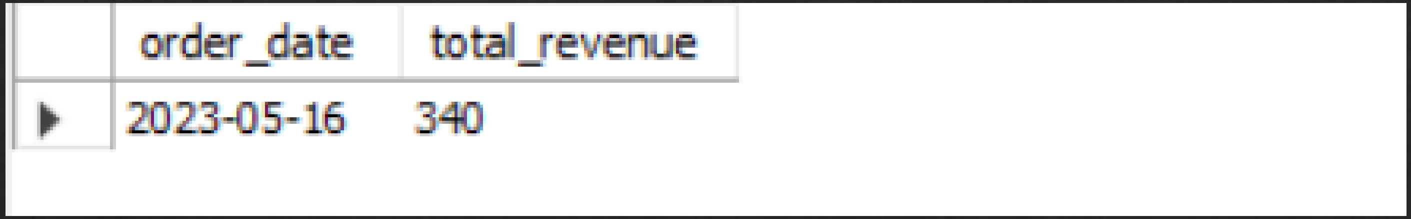
```
SELECT product_name, SUM(products.price * order_items.quantity) AS total_revenue FROM products
JOIN order_items ON products.product_id = order_items.product_id
GROUP BY product_name
ORDER BY total_revenue ASC;
```



product_name	total_revenue
Product A	50
Product D	75
Product E	90
Product G	120
Product B	135
Product H	135
Product I	150
Product C	160
Product K	180
Product L	195
Product F	210
Product J	330
Product M	420

4]Find the day with the highest revenue.

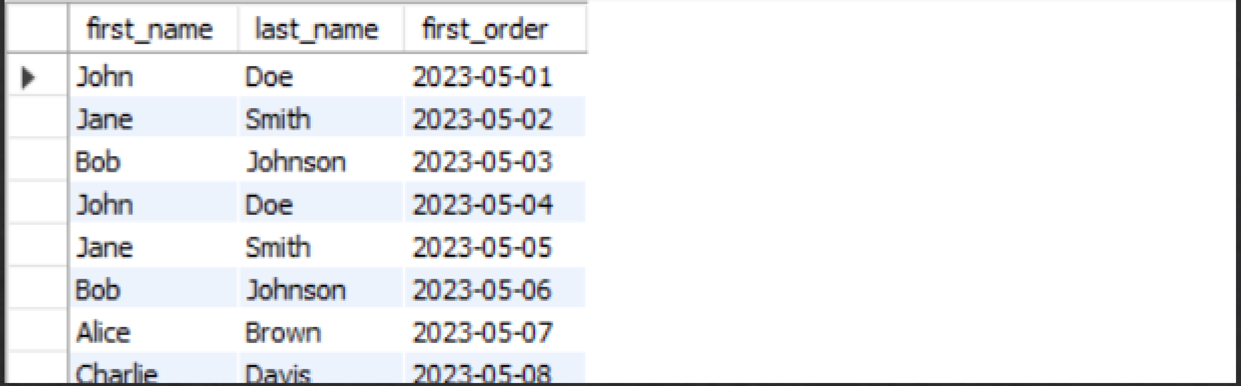
```
SELECT  ord.order_date,  SUM(pro.price * items.quantity) AS total_revenue FROM products prod
JOIN order_items items ON prod.product_id = items.product_id
JOIN orders ord ON items.order_id = ord.order_id
GROUP BY ord.order_date
ORDER BY total_revenue DESCLIMIT 1;
```



	order_date	total_revenue
▶	2023-05-16	340

5]Find the first order (by date) for each customer.

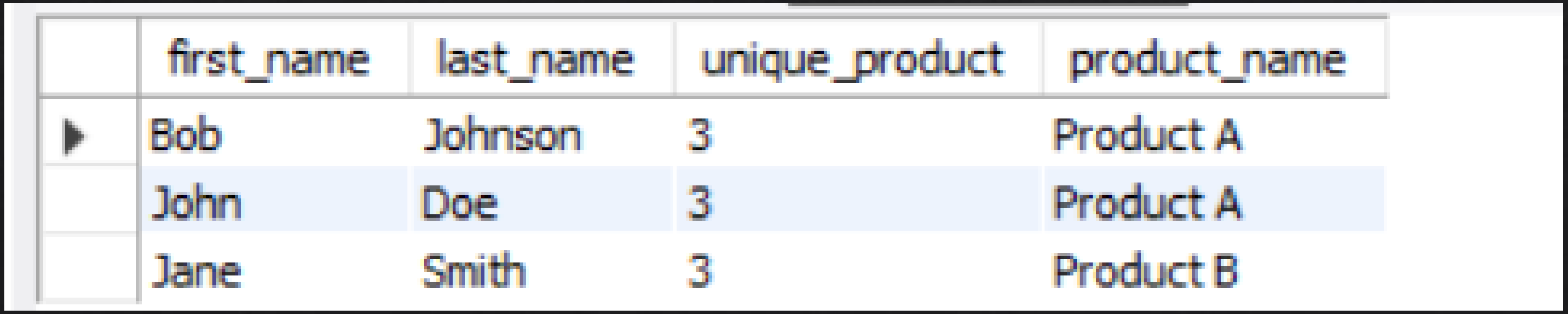
```
SELECT  cus.first_name, cus.last_name, min(ord.order_date) first_order FROM customers cus
JOIN orders ord ON cus.customer_id = ord.customer_id
GROUP  BY cus.first_name, cus.last_name, ord.order_date
ORDER BY first_order;
```



first_name	last_name	first_order
John	Doe	2023-05-01
Jane	Smith	2023-05-02
Bob	Johnson	2023-05-03
John	Doe	2023-05-04
Jane	Smith	2023-05-05
Bob	Johnson	2023-05-06
Alice	Brown	2023-05-07
Charlie	Davis	2023-05-08

6]Find the top 3 customers who have ordered the most distinct products

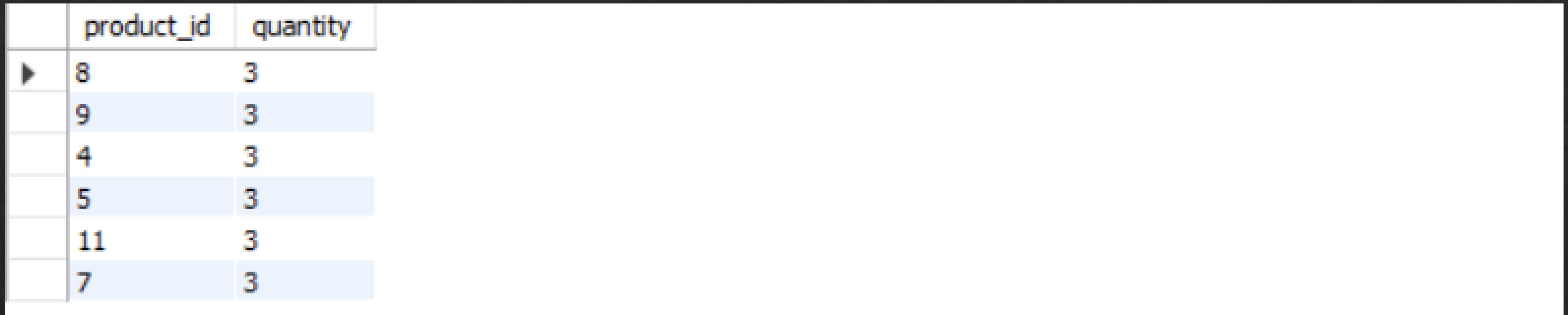
```
SELECT cus.first_name, cus.last_name, COUNT(DISTINCT product_name) AS unique_product,
product_name FROM customers cus
JOIN orders ON cus.customer_id = orders.customer_idJOIN order_items items ON orders.order_id =
items.order_id
JOIN products ON products.product_id = items.product_id
GROUP BY cus.first_name, cus.last_name
ORDER BY unique_product DESC LIMIT 3;
```



first_name	last_name	unique_product	product_name
Bob	Johnson	3	Product A
John	Doe	3	Product A
Jane	Smith	3	Product B

7]Which product has been bought the least in terms of quantity?

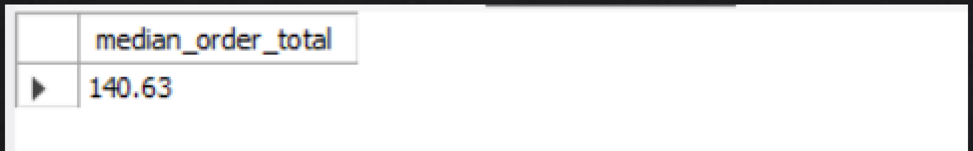
```
SELECT products.product_id, SUM(order_items.quantity) as quantity FROM products
JOIN order_items ON products.product_id = order_items.product_id
GROUP BY products.product_id
ORDER BY quantity LIMIT 6;
```



product_id	quantity
8	3
9	3
4	3
5	3
11	3
7	3

8]What is the median order total?

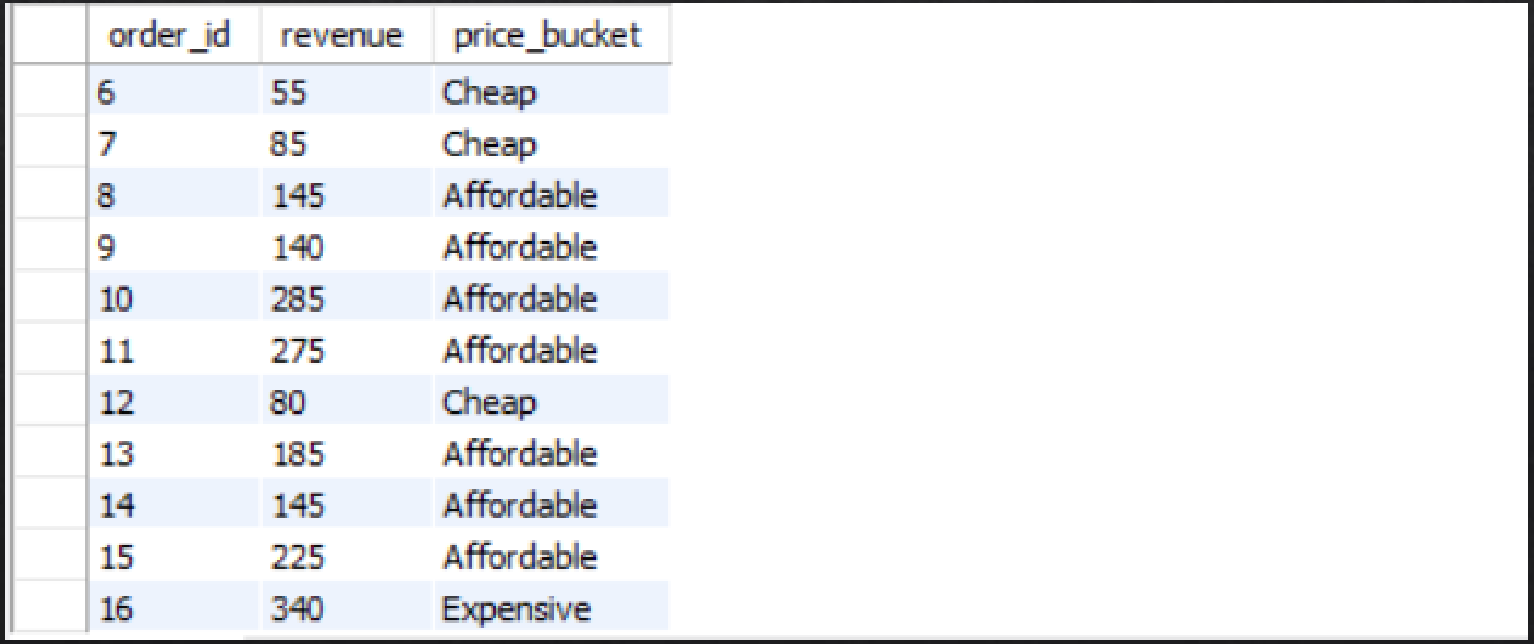
```
SELECT ROUND(AVG(total),2) AS median_order_total FROM
(SELECT ord.order_id, SUM(prod.price * items.quantity) AS total FROM orders ord
JOIN order_items items ON ord.order_id = items.order_id
JOIN products prod ON items.product_id = prod.product_id
GROUP BY ord.order_id) result
```



median_order_total
140.63

9]For each order, determine if it was ‘Expensive’ (total over 300), ‘Affordable’ (total over 100), or ‘Cheap’.

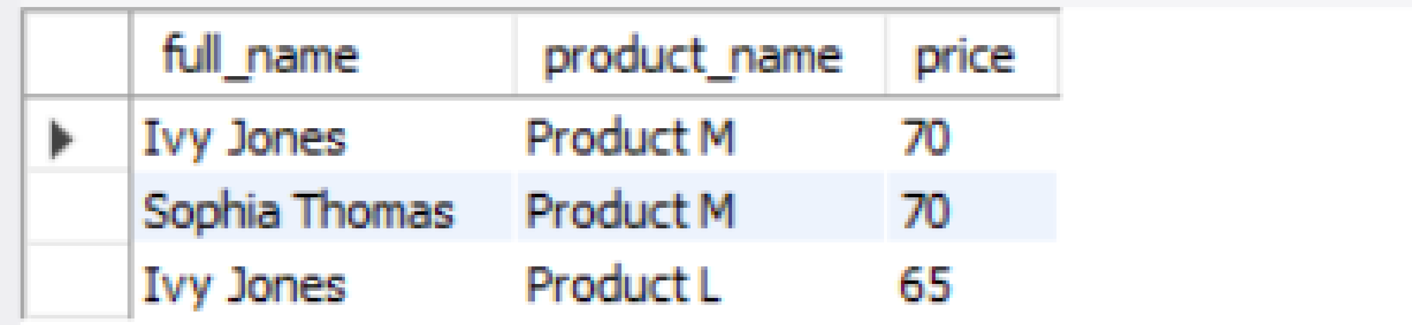
```
SELECT items.order_id, SUM(prod.price * items.quantity) AS revenue,
CASE
    WHEN SUM(prod.price * items.quantity) > 300 THEN 'Expensive’
     WHEN SUM(prod.price * items.quantity) > 100 THEN 'Affordable’
      ELSE 'Cheap’
END AS price_bucket
FROM products prod
JOIN order_items items ON items.product_id = prod.product_id
GROUP BY items.order_id;
```



order_id	revenue	price_bucket
6	55	Cheap
7	85	Cheap
8	145	Affordable
9	140	Affordable
10	285	Affordable
11	275	Affordable
12	80	Cheap
13	185	Affordable
14	145	Affordable
15	225	Affordable
16	340	Expensive

10]Find customers who have ordered the product with the highest price.

```
SELECT CONCAT(cus.first_name, ' ', cus.last_name) AS full_name, prod.product_name, prod.price FROM
customers cus
JOIN orders ord ON cus.customer_id = ord.customer_id
JOIN order_items items ON items.order_id = ord.order_id
JOIN products prod ON items.product_id = prod.product_id
WHERE prod.price = (SELECT MAX(prod.price) FROM products)
ORDER BY prod.price DESC
LIMIT 3;
```



full_name	product_name	price
Ivy Jones	Product M	70
Sophia Thomas	Product M	70
Ivy Jones	Product L	65

CONCLUSION:

This case study highlights the practical application of MySQL and SQL functionalities in analyzing sales data for Tiny Shop. By utilizing various SQL techniques, we can gain valuable insights into product pricing, customer behavior, revenue analysis, and order categorization.