1. Write a query that displays for each employee's first name, Last name & his office phone.

```
SELECT em.firstName , em.lastName , of.phone
FROM `employees` em JOIN offices of
ON em.officeCode = of.officeCode;
```

2. Write a query that returns for each customer his name, postal code & his order date.

```
SELECT c.customerName , c.postalCode , ord.orderDate
FROM customers c
JOIN orders ord ON c.customerNumber = ord.customerNumber;

================================================

SELECT c.customerName , c.postalCode , ord.orderDate
FROM `customers` c JOIN orders ord USING (customerNumber);
```

3. Retrieve and display the customer names and the shipping dates of their orders. This query utilizes a LEFT JOIN to ensure that even customers with no orders are included in the results.

```
SELECT c.customerName, o.shippedDate
FROM customers c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber;
```

4. retrieve and display the first names and email addresses of employees along with the second address lines of their corresponding offices. The query employs a RIGHT JOIN to ensure that all office addresses are included in the results, and it orders the results by country in ascending order.

```
SELECT em.firstName, em.email, o.addressLine2, o.country
FROM offices o
RIGHT JOIN employees em ON o.officeCode = em.officeCode
ORDER BY o.country ASC;
```

5. Retrieve the first and last names of customers along with the first and last names of the employees who are responsible for the customers. Display the results in a single table

```
SELECT c.customerName, e.firstName, e.lastName
FROM customers c
JOIN employees e ON c.salesRepEmployeeNumber = e.employeeNumber;

==========================================================

SELECT c.customerName ,concat(e.firstName, " " ,  e.lastName) as
Employee_full_name
FROM customers c
JOIN employees e ON c.salesRepEmployeeNumber = e.employeeNumber;
```

6. Retrieve the customer names, sales employee names, and office cities for all employees. Show results in a single table.

```
SELECT c.customerName, CONCAT(e.firstName, ' ', e.lastName) AS salesEmployee,
o.city AS officeCity
FROM customers c
JOIN employees e ON c.salesRepEmployeeNumber = e.employeeNumber
JOIN offices o ON e.officeCode = o.officeCode;
```

7. Retrieve a list of customers along with the product line and text description.

```
SELECT c.customerName , pl.productLine , pl.textDescription
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON od.orderNumber = o.orderNumber
JOIN products p ON p.productCode = od.productCode
JOIN productlines pl ON pl.productLine = p.productLine;
```

8. Retrieve a list of customers with their names and phone numbers, along with the status of their orders.

```
SELECT c.customerName , c.phone , o.status
FROM customers c
JOIN orders o  ON c.customerNumber = o.customerNumber;
```

9. Write a query that returns for each customer his first name , country and the quantity of the orders.

```
SELECT cus.contactFirstName , cus.country , ot.quantityOrdered
FROM customers cus
JOIN orders ord ON cus.customerNumber = ord.customerNumber
JOIN orderdetails ot ON ord.orderNumber = ot.orderNumber;


=========================================================

SELECT cus.contactFirstName , cus.country , ot.quantityOrdered
FROM customers cus
JOIN orders ord USING (customerNumber)
JOIN orderdetails ot USING (ordernumber);
```

10. Write a query that returns for each customer his first name , country and the quantity of the orders.

```
SELECT cus.contactFirstName , cus.country , ot.quantityOrdered
FROM customers cus
JOIN orders ord ON cus.customerNumber = ord.customerNumber
JOIN orderdetails ot ON ord.orderNumber = ot.orderNumber;


=========================================================

SELECT cus.contactFirstName , cus.country , ot.quantityOrdered
```

```
FROM customers cus
JOIN orders ord USING (customerNumber)
JOIN orderdetails ot USING (ordernumber);
```

11. Retrieve the customer names, order numbers, and product names for all products ordered by each customer. Display the results in a single table.

```
SELECT c.customerName, o.orderNumber, p.productName
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber
JOIN products p ON od.productCode = p.productCode;
```

12. Retrieve the customer names, order numbers, and product names for all products ordered by each customer, and show the results in a single table. Additionally, include the quantity ordered for each product.

```
SELECT c.customerName, o.orderNumber, p.productName, od.quantityOrdered
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber
JOIN products p ON od.productCode = p.productCode;
```

13. Retrieve the customer names, order numbers, product names, and product line descriptions for all products ordered by each customer, and show the results in a single table.

```
SELECT c.customerName, o.orderNumber, p.productName, pl.textDescription
FROM customers c
```

```
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber
JOIN products p ON od.productCode = p.productCode
JOIN productlines pl ON p.productLine = pl.productLine;
```

14. Retrieve the customer names, order numbers, and product codes for products ordered by customers, but only for orders where the total order price (quantity ordered * price each) is greater than $2,000. Show the results in a single table.

```
SELECT c.customerName, o.orderNumber, od.productCode , od.quantityOrdered *
p.buyPrice
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber
JOIN products p ON od.productCode = p.productCode
WHERE (od.quantityOrdered * p.buyPrice) > 2000;

============================================================

SELECT c.customerName, o.orderNumber, od.productCode, (od.quantityOrdered *
p.buyPrice) AS Total_order_price
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber
JOIN products p ON od.productCode = p.productCode
HAVING Total_order_price > 2000;
```

15. Write a query that returns the customer's first name, phone, the date and the status of his order for the first quarter of 2004’s year in descending order by the year.

```
SELECT c.contactFirstName, c.phone , o.orderDate , o.status
 FROM customers c JOIN orders o
 ON c.customernumber = o.customernumber
 WHERE o.orderDate BETWEEN "2004-01-01" AND "2004-03-31"
 ORDER BY o.orderDate DESC

=====================================================
```

```
SELECT c.contactFirstName, c.phone, o.orderDate, o.status
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
WHERE DATE(o.orderDate) BETWEEN '2004-01-01' AND '2004-03-31'
ORDER BY o.orderDate DESC;

==========================================================


SELECT c.contactFirstName, c.phone, o.orderDate, o.status
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
WHERE YEAR(o.orderDate) = 2004 AND QUARTER(o.orderDate) = 1
ORDER BY o.orderDate DESC;
```

16. Display customers name and their required date in ascending order.

```
SELECT c.customerName , o.requiredDate

FROM customers c

JOIN orders o ON c.customerNumber = o.customerNumber

Order by o.requiredDate;
```

17. Write a query that returns the customer's last name, postal code, the status of his order and the quantity of the order in descending alphabetical order.

```
SELECT cus.contactLastName, cus.postalCode, ord.status, ot.quantityOrdered
FROM `customers` cus
JOIN orders ord USING (customerNumber)
JOIN orderdetails ot USING (orderNumber)
ORDER BY cus.contactLastName;
```

18. Retrieve a list of employees and their first names, last names, along with the city and country where their offices are located. Display the employee information in alphabetical ascending order based on the employee's first name.

```
SELECT e.firstName , e.lastName , o.city , o.state
FROM employees e JOIN offices o
ON e.officeCode = o.officeCode
ORDER BY e.firstName;
```

19. Retrieve for each customer, his name, phone, and the status of his order, in alphabetical descending order based on the **customer**'s name

```
SELECT c.customerName , c.phone , o.status
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
ORDER BY customerName DESC;
```

20. Retrieve for each customer, his name, phone, and the status of his order where his quantity order is more than 40 in ascending order based on the quantity of the order.

```
SELECT c.customerName , c.phone , o.status , od.quantityOrdered
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber
WHERE od.quantityOrdered > 40
ORDER BY od.quantityOrdered;
```

21. Retrieve the customer names and the quantity ordered for each order

```
SELECT c.customerName , od.quantityOrdered
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber;
```

22. retrieve and display information about customers **name**, the products they've ordered, and the buying price of those products. Ensure that the results are ordered by the buying price in descending order.

```
SELECT c.customerName, p.productName, p.buyPrice
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN orderdetails od ON o.orderNumber = od.orderNumber
JOIN products p ON od.productCode = p.productCode
ORDER BY p.buyPrice DESC;
```

23. Retrieve the customer names, order numbers, and order dates for orders placed by customers whose contact last name contains 'son'. Calculate the number of days between the order date and the current date. Sort the results by the number of days in ascending order.

```
SELECT c.customerName, o.orderNumber, o.orderDate,
      DATEDIFF(CURDATE(), o.orderDate) AS daysBetween
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
WHERE c.contactLastName LIKE '%son%'
ORDER BY daysBetween;
```

24. Retrieve the employee numbers, first names, and last names of employees working in offices located in cities starting with the letter 'S'. Sort the results by office city in ascending order

```
SELECT e.employeeNumber, e.firstName, e.lastName, o.city
FROM employees e
JOIN offices o ON e.officeCode = o.officeCode
WHERE o.city LIKE 'S%'
ORDER BY o.city;
```

25. Retrieve the customer names, sales employee names, payment dates, and the difference in days between the payment date and the order date for each payment. Include only payments made more than 10 days after the order date. Sort the results by the difference in days in descending order.

```
SELECT c.customerName, CONCAT(e.firstName, ' ', e.lastName) AS salesEmployee,
p.paymentDate,
      DATEDIFF(p.paymentDate, o.orderDate) AS daysDifference
FROM customers c
JOIN employees e ON c.salesRepEmployeeNumber = e.employeeNumber
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN payments p ON c.customerNumber = p.customerNumber
WHERE DATEDIFF(p.paymentDate, o.orderDate) > 10
ORDER BY daysDifference DESC;

============================================================

SELECT c.customerName, CONCAT(e.firstName, ' ', e.lastName) AS salesEmployee,
p.paymentDate,
      DATEDIFF(p.paymentDate, o.orderDate) AS daysDifference
FROM customers c
JOIN employees e ON c.salesRepEmployeeNumber = e.employeeNumber
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN payments p ON c.customerNumber = p.customerNumber
having daysDifference > 10
ORDER BY daysDifference DESC;
```

26. Retrieve the customer names, sales employee names, and the total payments made by customers whose contact last name contains 'son'. Calculate the total payments as the sum of all payment amounts. Sort the results by total payments in descending order.

```
SELECT c.customerName, CONCAT(e.firstName, ' ', e.lastName) AS salesEmployee,
SUM(p.amount) AS totalPayments , c.contactLastName
FROM customers c
JOIN employees e ON c.salesRepEmployeeNumber = e.employeeNumber
JOIN payments p ON c.customerNumber = p.customerNumber
WHERE c.contactLastName LIKE '%son%'
GROUP BY c.customerName, salesEmployee
ORDER BY totalPayments DESC;
```

27. Retrieve the order numbers, product names, and the total order amounts for each order. Calculate the total order amount by multiplying the quantity ordered by the price each. Include only orders with a total amount greater than $2,000. Sort the results by order amount in descending order.

```
SELECT o.orderNumber, p.productName, SUM(od.quantityOrdered * od.priceEach)
AS totalOrderAmount
FROM orders o
JOIN orderdetails od ON o.orderNumber = od.orderNumber
JOIN products p ON od.productCode = p.productCode
GROUP BY o.orderNumber, p.productName
HAVING totalOrderAmount > 2000
ORDER BY totalOrderAmount DESC;
```

28. Retrieve the customer names, order numbers, employee names, and payment dates for orders placed by customers. Calculate the number of days between the order date and the payment date for each payment. Include only payments made more than 30 days after the order date. Sort the results by the number of days in ascending order.

```
SELECT c.customerName, o.orderNumber, CONCAT(e.firstName, ' ', e.lastName) AS
employeeName, p.paymentDate,
      DATEDIFF(p.paymentDate, o.orderDate) AS daysDifference
FROM customers c
JOIN orders o ON c.customerNumber = o.customerNumber
JOIN employees e ON c.salesRepEmployeeNumber = e.employeeNumber
JOIN payments p ON c.customerNumber = p.customerNumber
WHERE DATEDIFF(p.paymentDate, o.orderDate) > 30
ORDER BY daysDifference;
```

29. retrieve and display information about products, including their names, Manufacturer's Suggested Retail Price (MSRP), discounted prices (10% off MSRP), and the text descriptions of their product lines. Ensure that the results are ordered by the discounted price in ascending order.

```
SELECT p.productName, p.MSRP, (p.MSRP - (p.MSRP * 0.10)) AS discountedPrice,
pl.textDescription
FROM products p
JOIN productLines pl ON p.productLine = pl.productLine
ORDER BY discountedPrice;
```