Below are simplified SQL commands illustrating the relationships between the tables based on the provided schema and also These queries demonstrate the relationships between the tables through JOIN operations, linking tables based on their foreign key and primary key relationships.

(1)

In summary, the code retrieves information about the top 8 products with the highest quantity in stock by performing an inner join between the "products" and "inventory" tables and ordering the result set accordingly.

```
SELECT *

FROM products

INNER JOIN inventory ON products.ProductID = inventory.ProductID

ORDER BY inventory.QuantityInStock DESC

LIMIT 8
```

(2)

```
SELECT *

FROM sales_orders

INNER JOIN order_details ON sales_orders.SalesOrderID = order_details.OrderID

INNER JOIN products ON order_details.ProductID = products.ProductID

LIMIT 8;
```

This SQL code pulls together information from three tables: sales_orders, order_details, and products, to show details about the first 8 sales orders. It gives specifics about each order, like how many products were bought and details about those products—such as their names, ratings, weights, prices, categories, and colors. By focusing on the initial 8 orders, the query gives a quick look at what products were sold and their details in the early days of the sales system.It's like peeking at the first few pages of a sales diary to understand what started at the beginning

(3)

```
SELECT *

FROM customers

INNER JOIN sales_orders ON customers.CustomerID = sales_orders.CustomerID

LIMIT 6
```

The SQL query is designed to link each sales order with its corresponding customer, providing a consolidated view of customer-related sales transactions. It achieves this by combining relevant

columns from the 'customers' and 'sales_orders' tables through an inner join based on the shared 'CustomerID.' The result is a concise output displaying details about customers and their associated sales orders. The use of 'LIMIT 6' ensures that only the initial six rows of this combined information are presented, facilitating a focused preview of the data.

(4)

SELECT *

FROM employees

INNER JOIN sales_orders ON employees.EmployeeID = sales_orders.EmployeeID

LIMIT 8;

This SQL query retrieves information by performing an INNER JOIN between the 'employees' and 'sales_orders' tables based on matching 'EmployeeID' values. It selects all columns from the result set, providing details about employees and their associated sales orders. The LIMIT clause restricts the output to the first 4 rows, offering a concise view of employee-sales order associations. This query helps in examining a small subset of data for quick analysis

(5)

SELECT products.ProductID, products.ProductName, inventory.QuantityInStock

FROM products

JOIN inventory ON products.ProductID = inventory.ProductID

WHERE inventory.QuantityInStock < 10

LIMIT 8;

This SQL query extracts vital product information, specifically focusing on ProductID, ProductName, and QuantityInStock. The JOIN operation harmonizes data from the 'products' and 'inventory' tables based on matching 'ProductID' values. The WHERE clause acts as a filter, spotlighting products with a QuantityInStock of less than 10. The concluding LIMIT 8 ensures a concise result set, emphasizing the first eight entries that meet the criteria. This query effectively unveils products facing potential stock shortages, providing valuable insights for inventory management.

(6)

SELECT c.CustomerID, c.CustomerName, SUM(p.ProductPrice * od.Quantity) AS TotalAmountSpent

FROM customers c

JOIN sales_orders so ON c.CustomerID = so.CustomerID

JOIN order_details od ON so.SalesOrderID = od.OrderID

```
JOIN products p ON od.ProductID = p.ProductID

GROUP BY c.CustomerID, c.CustomerName

ORDER BY TotalAmountSpent DESC

LIMIT 5;
```

The query identifies the top 5 customers based on their total spending. It achieves this by combining information from the 'customers,' 'sales_orders,' 'order_details,' and 'products' tables. The results are grouped by CustomerID and CustomerName, and the total amount spent by each customer is calculated using the product price and quantity ordered. The final output is then ordered in descending order based on the total amount spent, and the result is limited to the top 5 customers. In essence, this query provides a concise overview of the highest-spending customers in terms of total product purchases.

(7)

```
SELECT *

FROM inventory

WHERE QuantityInStock > 50

LIMIT 8
```

This SQL query extracts details about products in the inventory with a quantity exceeding 50 units. By selecting all columns from the 'inventory' table, filtering for items where the 'QuantityInStock' surpasses 50, and limiting the results to the first 8 rows, the query offers a concise overview of well-stocked products in the inventory.

(8)

```
SELECT *

FROM products

WHERE ProductRating = 'Good'

LIMIT 8;
```

The query retrieves detailed information about products with a 'Good' rating from the products table, presenting up to 8 rows that meet the specified condition. This type of query is beneficial for extracting specific data subsets when dealing with large datasets