1. For each payment, calculate the cumulative sum of the payment amounts ordered by customerNumber. Include all columns in the result. (hint, use only order by)

```
SELECT *,SUM(amount) OVER (PARTITION BY customerNumber ORDER BY paymentDate) AS
cumulativeSum
FROM payments
ORDER BY customerNumber, paymentDate;
```

2. For each payment, calculate the running total of the payment amounts within each customer's transactions, ordered by the payment amount.

```
SELECT *, SUM(amount) over(PARTITION BY customerNumber ORDER BY amount) as total
FROM `payments`;
```

3. For each payment, find the minimum and maximum payment amounts within each customer's transactions. Include all columns in the result, along with columns for the minimum and maximum amounts.

```
SELECT*,
  MIN(amount) OVER (PARTITION BY customerNumber) AS min,
  MAX(amount) OVER (PARTITION BY customerNumber) AS max
FROM payments;
```

4. For each order detail, find the minimum and maximum unit prices within each order. Include all columns in the result, along with columns for the minimum and maximum unit prices.

```
SELECT *,
  MIN(priceEach) OVER (PARTITION BY orderNumber) AS minPrice,
  MAX(priceEach) OVER (PARTITION BY orderNumber) AS maxPrice
```

```
FROM orderDetails;
```

5. Retrieve detailed information about each order detail, including the product name, product line, and vendor. Additionally, calculate the minimum and maximum unit prices for each order.

```
SELECT
  od.orderNumber,
  od.productCode,
  od.quantityOrdered,
  od.priceEach,
  p.productName,
  p.productLine,
  p.productVendor,
  MIN(od.priceEach) OVER (PARTITION BY od.orderNumber) AS minPrice,
  MAX(od.priceEach) OVER (PARTITION BY od.orderNumber) AS maxPrice
FROM orderDetails od
JOIN products p ON od.productCode = p.productCode
JOIN productLines pl ON p.productLine = pl.productLine;
```

6. Assign a unique row number to each order detail within each order, ordered by the quantity ordered. Display all columns along with the assigned row number.

```
SELECT *, ROW_NUMBER() OVER (PARTITION BY orderNumber ORDER BY quantityOrdered) AS
rowNumber
FROM orderDetails;
```

7. Retrieve detailed information about each order detail, including product name, product

line, and vendor. Additionally, assign a unique row number to each order detail within each order, ordered by the quantity ordered.

```
SELECT od.orderNumber, od.productCode, od.quantityOrdered, od.priceEach,
p.productName, p.productLine, p.productVendor,
ROW_NUMBER() OVER (PARTITION BY od.orderNumber ORDER BY od.quantityOrdered) AS rowNumber
FROM orderDetails od JOIN products p ON od.productCode = p.productCode;
```

8. Retrieve detailed information about each order detail, including product details, and assign a unique row number to each order detail within each order. Filter the results to include only orders with a specific product line 'Classic Cars'.

```
SELECT od.orderNumber, od.productCode, od.quantityOrdered, od.priceEach,
p.productName, p.productLine, p.productVendor,
ROW_NUMBER() OVER (PARTITION BY od.orderNumber ORDER BY od.quantityOrdered) AS rowNumber
FROM orderDetails od JOIN products p ON od.productCode = p.productCode
WHERE p.productLine = 'Classic Cars';
```

9. Retrieve detailed information about each order detail, including product details, and assign a unique row number to each order detail within each order. Filter the results to include only orders where the quantity ordered is greater than 10.

```
SELECT od.orderNumber, od.productCode, od.quantityOrdered, od.priceEach,
p.productName, p.productLine, p.productVendor,
ROW_NUMBER() OVER (PARTITION BY od.orderNumber ORDER BY od.quantityOrdered) AS rowNumber
FROM orderDetails od JOIN products p ON od.productCode = p.productCode
WHERE od.quantityOrdered > 10;
```

10. Retrieve detailed information about each order detail, including the assigned rank based on the quantity ordered.

```
select productCode , orderNumber , quantityOrdered , rank() over (order BY quantityOrdered DESC) as
Rank
FROM orderdetails;
```

11. Retrieve detailed information about each order detail, including the assigned dense rank

based on the quantity ordered.

```
select productCode , orderNumber , quantityOrdered , DENSE_RANK() over (order BY quantityOrdered
DESC) as DENSE_RANK
FROM orderdetails;
```

12. Assign a unique row number to each order detail based on the quantity ordered.

Retrieve detailed information about each order detail, including the assigned row

number.

```
select productCode , orderNumber , quantityOrdered ,    ROW_NUMBER() OVER (ORDER BY
quantityOrdered) AS row_number
FROM orderdetails;
```

13. Assign a unique row number to each order detail within each order based on the quantity

ordered. Retrieve detailed information about each order detail, including the assigned

row number within each order.

```
select productCode , orderNumber , quantityOrdered ,    ROW_NUMBER() OVER (PARTITION BY
orderNumber ORDER BY quantityOrdered) AS row_number
FROM orderdetails;
```

14. Calculate the cumulative sum of the quantity ordered for each order detail, ordered by

the unit price. Retrieve detailed information about each order detail, including the cumulative sum of quantity ordered.

```
select productCode , orderNumber , quantityOrdered , SUM(quantityOrdered) OVER (PARTITION BY
orderNumber ORDER BY priceEach) AS cumulative_sum

FROM orderdetails;
```

15. For each order detail, calculate the cumulative sum of the quantity ordered within each order, ordered by the order number. Retrieve information about the order number, product code, quantity ordered, and the cumulative sum of quantity ordered. Display the results in ascending order based on the order number.

```
select orderNumber, productCode , quantityOrdered , SUM(quantityOrdered) OVER (PARTITION BY
orderNumber ORDER BY priceEach) AS cumulative_sum

FROM orderdetails

ORDER BY orderNumber ASC;
```

16. For each order detail, assign a rank based on the unit price of the product. Retrieve information about the order number, product code, unit price, and the assigned rank.

```
select orderNumber, productCode , priceEach , rank() over (PARTITION by orderNumber order BY
priceEach) as Rank

FROM orderdetails
```

17. For each order detail, assign a dense rank based on the unit price of the product. Retrieve information about the order number, product code, unit price, and the assigned dense rank.

```
select orderNumber, productCode , priceEach , DENSE_RANK() over (order BY priceEach) as
DENSE_RANK

FROM orderdetails;
```

18. For each order detail, calculate and display the following ranking metrics based on the unit price of the product: row number, rank, and dense rank. Retrieve information about the order number, product code, unit price, row number, rank, and dense rank.

```
select orderNumber, productCode , priceEach ,
  ROW_NUMBER() OVER (PARTITION BY orderNumber ORDER BY priceEach) AS row_number,
  RANK() OVER (PARTITION BY orderNumber ORDER BY priceEach) AS rank,
  DENSE_RANK() OVER (PARTITION BY orderNumber ORDER BY priceEach) AS dense_rank
FROM orderdetails;
```

19. Assign a dense rank to each order detail based on the descending unit price of the product. Retrieve information about the order number, product code, unit price, and the assigned dense rank. Sort the results by the unit price in descending order.

```
select orderNumber, productCode , priceEach ,
  DENSE_RANK() OVER (PARTITION BY orderNumber ORDER BY priceEach) AS dense_rank
FROM orderdetails
ORDER BY priceEach DESC;
```

20. For each order detail with a unit price less than 100, assign a dense rank based on the descending unit price of the product. Retrieve information about the order number, product code, unit price, and the assigned dense rank. Sort the results by the unit price in descending order.

```
select orderNumber, productCode , priceEach ,
  DENSE_RANK() OVER (ORDER BY priceEach DESC) AS dense_rank
FROM orderdetails
WHERE priceEach < 100
ORDER BY priceEach DESC;
```