2. Retrieve customer details and assign a unique row number to each customer, regardless of any specific ordering. Display the customer name, country, and the assigned row number.

```
SELECT customername,country,customerNumber,
row_number() over(order by customerNumber) as rn
FROM `customers`;
```

3. Retrieve product details and assign a unique row number to each product, regardless of any specific ordering. Display the product name, product code, quantity in stock, and the assigned row number.

```
SELECT p.productName, p.productCode, p.quantityInStock,
row_number() over() as rn
FROM `products` p;
```

4. 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 rn
FROM `orderdetails`;
```

5. 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 o.orderNumber,o.productCode,o.priceEach,
rank() over(order by o.priceEach) as rn
FROM `orderdetails` o;
```

6. 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 o.orderNumber,o.productCode,o.priceEach,
rank() over(order by o.priceEach) as rn1,
row_number() over(order by o.priceEach) as rn2,
dense_rank() over(order by o.priceEach) as rn3
FROM `orderdetails` o;
```

7. Retrieve product details and assign a unique row number to each product based on the ascending order of the quantity in stock. Display the product name, product code, quantity in stock, and the assigned row number.

```
SELECT p.productName,p.productCode,p.quantityInStock,
row_number() over(order by p.quantityInStock ASC) as rn
FROM `products` p;
```

8. Retrieve product details and assign a unique row number within each product line. Display the product name, product line, product code, quantity in stock, and the assigned row number within the respective product line.

```
SELECT p.productName,p.productLine,p.productCode,p.quantityInStock,
row_number() over(PARTITION by p.productLine order by p.quantityInStock) as rn
FROM `products` p;
```

9. Retrieve product details and assign a unique row number within each product line based on the ascending order of product code. Display the product name, product line, product code, quantity in stock, and the assigned row number within the respective product line.

```
SELECT p.productName,p.productLine,p.productCode,p.quantityInStock,
row_number() over(PARTITION by p.productLine order by p.productCode ASC) as rn
FROM `products` p;
```

10.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 p.productName,p.productLine,p.productVendor,
row_number() over(PARTITION by p.productCode order by p.quantityInStock) as rn
FROM `products` p;
```

11. You are tasked with extracting information about orders containing products related to the 'ships' product line. Write a SQL query to achieve the following:

* Retrieve all columns from the 'orderdetails' table along with additional product information.
* Include columns from the 'products' table such as productName, productLine, and productVendor.
* Assign a unique row number to each record within its order, based on the ascending order of the quantity ordered.
* Only include records where the product line in the 'products' table is 'ships'.

```
SELECT od.*,p.productName,p.productLine,p.productVendor,
row_number() over(PARTITION by od.orderNumber order by od.quantityOrdered ASC) as rn
FROM `orderdetails` od
join products p on od.productCode = p.productCode
WHERE p.productLine = 'ships';
```

12.You are working with a product database, and your task is to analyze the inventory of products within each product line. Write a SQL query to achieve the following:

* Retrieve the product name, product line, product code, quantity in stock, and a unique row number assigned to each product within its product line.
* Order the results in ascending order based on the quantity in stock.

```
SELECT p.productName,p.productLine,p.quantityInStock,
row_number() over(PARTITION by p.productLine order by p.quantityInStock asc) as rn
FROM `products` p
ORDER BY p.quantityInStock ASC;
```

13.Consider the 'payments' table in the sample database. Write a SQL query that selects all columns and rows from the 'payments' table but limits the results to the first two payment records for each year.

```
SELECT *
FROM (SELECT payments.*,
ROW_NUMBER() OVER (PARTITION BY YEAR(paymentDate) ORDER BY paymentDate) AS row
FROM payments) AS rn
WHERE row <= 2;
```

14.Imagine you are working on a project to analyze the pricing structure of products in the 'orderdetails' table. Write a SQL query to achieve the following tasks:

* Retrieve all columns from the 'orderdetails' table.
* Assign a rank to each record based on the 'priceEach' column, ordering the records in ascending order of prices.
* Display the result with an additional column named 'RNK' representing the assigned rank.

```
SELECT * ,
rank()over(ORDER BY priceEach ASC) as RNK
FROM `orderdetails`;
```

15.As part of a pricing analysis project for your company, you are tasked with examining the pricing structure of products in the 'orderdetails' table. Write a SQL query to achieve the following:

* Retrieve all columns from the 'orderdetails' table.

* Assign a dense rank to each record based on the 'priceEach' column, ordering the records in ascending order of prices.
* Display the result with an additional column named 'RNK' representing the assigned dense rank.

```
SELECT * ,
dense_rank()over(ORDER BY priceEach ASC) as RNK
FROM `orderdetails`;
```

16.Write a SQL query to achieve the following:
* Retrieve all columns from the 'orderdetails' table.
* Assign a unique row number to each record based on the ascending order of the 'quantityOrdered' column.
* Assign a rank to each record based on the ascending order of the 'quantityOrdered' column.
* Assign a dense rank to each record based on the ascending order of the 'quantityOrdered' column.

```
SELECT * ,
ROW_number()over(ORDER BY quantityOrdered ASC) as row,
Rank()over(ORDER BY quantityOrdered ASC) as RNK,
Dense_rank()over(ORDER BY quantityOrdered ASC) as dense_RNK
FROM `orderdetails`;
```

17.You are tasked with extracting valuable information about customers and their payments for a comprehensive financial report. Write a SQL query to achieve the following:

* Retrieve the customer name along with all columns from the payments table.

* Include only the first two payment records for each year, ordered by the payment date.

```
SELECT p.*, c.customerName
FROM ( SELECT paymentDate, customerNumber, checkNumber,amount,
ROW_NUMBER() OVER (PARTITION BY YEAR(paymentDate) ORDER BY paymentDate) AS row_num
FROM payments) AS p
JOIN customers c ON p.customerNumber = c.customerNumber
WHERE row_num <= 2;
```

18.As part of a pricing analysis project, you are tasked with examining orders containing products with a price each of $50 or more. Write a SQL query to achieve the following:

* Retrieve all columns from the 'orderdetails' table along with the product name.

* Assign a dense rank to each product within its order, based on the ascending order of the quantity ordered.

* Only include records where the price each in the 'orderdetails' table is greater than or equal to $50.

* Order the results by the price each in ascending order.

```
SELECT od.*,p.productName,
dense_rank()over(ORDER BY quantityOrdered ASC) rnk
FROM `orderdetails` od
join products p ON od.productCode = p.productCode
WHERE od.priceEach >=50
ORDER BY od.priceEach ASC;
```

19.Your task is to analyze orders with products where the quantity ordered is less than or equal to 20. Write a SQL query to achieve the following:

* Retrieve all columns from the 'orderdetails' table along with additional product information.

* Include columns from the 'products' table such as productName, productLine, and productVendor.

* Assign a unique row number to each record within its order, based on the descending order of the quantity ordered.

* Only include records where the quantity ordered in the 'orderdetails' table is less than or equal to 20.

* Order the results by the quantity ordered in descending order.

```
SELECT od.*,p.productName,p.productLine,p.productVendor,
row_number()over(PARTITION BY od.orderNumber ORDER BY quantityOrdered DESC) rnk
FROM `orderdetails` od
join products p ON od.productCode = p.productCode
WHERE od.quantityOrdered <=20
ORDER BY od.quantityOrdered DESC;
```