**Use northwind database.**

**Exercise 01:**

Create a stored procedure named **GetOrderDetails** that takes an order ID as a parameter and returns the details of the products in that order, including product name, quantity, and unit price.

**Exercise 02:**

Create a stored procedure named **GetCategoryProducts** that takes a category ID as a parameter and returns the list of products within that category, including product name, unit price, and quantity available in stock.

**Exercise 03:**

Create a stored procedure named **GetEmployeeOrders** that takes an employee ID as a parameter and returns the list of orders placed by that employee, including order ID, order date, and total order amount.

**Exercise 04:**

Create a stored procedure named **GetCustomerOrders** that takes a customer ID as a parameter and returns the list of orders placed by that customer, including order ID, order date, and total order amount.

**Exercise 05:**

Create a stored procedure named **GetTopSellingProducts** that takes a start date and an end date as parameters and returns the top N selling products within that date range, where N is another parameter.

**Exercise 06:**

Create a stored procedure named **GetEmployeeSalesSummary** that takes an employee ID and a date range (start date and end date) as parameters. The procedure should return a summary of the sales made by that employee during the specified date range. Include information such as the total number of orders, total revenue, and average order value.

**Exercise 07:**

Create a stored procedure named **GetCategorySalesAnalysis** that takes a category ID and a year as parameters. The procedure should return a detailed analysis of sales for that category during the specified year. Include information such as the total revenue, total quantity sold, average unit price, and the month with the highest sales.

**Exercise 08:**

Create a stored procedure named **GetCustomerOrderFrequency** that takes a customer ID as a parameter. The procedure should return the frequency of orders placed by that customer, categorized by the day of the week. Include information such as the day of the week, the total number of orders, and the average order value for each day.

**Exercise 09:**

Create a stored procedure named **GetEmployeeSalesSummaryWithOutParam** that takes an employee ID and a date range (start date and end date) as parameters. The procedure should calculate and return the total revenue and average order value for that employee during the specified date range. Use an **OUT** parameter to return the average order value.

**Exercise 10:**

Create a stored procedure named **GetCustomerOrderStatistics** that takes a customer ID as a parameter and returns statistics about the customer's orders. The procedure should calculate and return the total number of orders, the total revenue, and the average order value. Use **OUT** parameters for each of these statistics.

**Exercise 11:**

Create a stored procedure named **UpdateEmployeeSalary** that takes an employee ID as an IN parameter and a percentage increase as an INOUT parameter. The procedure should update the salary of the specified employee by applying the percentage increase.

**Exercise 12**

Create a stored procedure named **ProcessOrder** that takes an order ID as an IN parameter and an INOUT parameter representing the order status. The procedure should update the status of the specified order and provide feedback about the processing.

**Exercise 13:**

Create a stored procedure named **UpdateProductStock** that takes a product ID as an IN parameter, an INOUT parameter representing the quantity to add to the stock, and an OUT parameter representing the updated stock level. The procedure should update the stock level of the specified product and provide feedback about the update.

**Exercise 14:**

Create a stored procedure named **CalculateDiscountedPrice** that takes a product ID as an IN parameter, an INOUT parameter representing the original price, and an OUT parameter representing the discounted price. The procedure should calculate and provide feedback about the discounted price.

**Exercise 15:**

Create a stored procedure named **ProcessOrderDetails** that takes an order ID as an IN parameter, an INOUT parameter representing the order status, and an OUT parameter representing the total order value. The procedure should perform the following tasks:

* Update the order status to 'Processing'.
* Calculate the total order value by summing the product of quantity and unit price for each item in the order.
* Update the INOUT parameter to reflect the new order status.
* Provide feedback about the order processing.