# **Practical 9**

**Aim:** LINQ Operations – Querying and Manipulating In-Memory

Collections

Scenario

You are working as a software developer for a data analytics company.

Your manager has asked you to create a prototype that can perform

various operations on in-memory collections (like lists of products or

employees) to filter, sort, group, and summarize the data.

Instead of writing traditional loops and conditionals, you are required to

use LINQ (Language Integrated Query) to query and manipulate data

using clean, concise, and readable syntax.

This practical will help you understand how LINQ works with in-

memory collections like List<T> using both query syntax and method

syntax.

Tasks:

● Task 1: Create a New Console Application in C# Named

● Task 2: Define a Class Product with the appropriate Properties:

(Product ID, Name, Category, Price, Stock Quantity)

● Task 3: Create a List of Products and Populate it with at Least 6

Sample Records

● Task 4: Perform the Following LINQ Operations Using Method

Syntax:

○ Display all products in the list

○ Filter products where Price > 500

○ Get products that belong to a specific Category (e.g.,

"Electronics")

○ Sort products by Name in ascending order

○ Calculate the total stock quantity of all products

○ Find the most expensive product

● Task 5: Perform the Following LINQ Operations Using Query

Syntax:

○ Select only product names from the list

○ Group products by Category

○ Count the number of products in each category

○ Find average price of all products

● Task 6: Display the Output of Each Query in the Console with

Proper Labels

**CODE:**

#**Product.cs**

public class Product

{

public int ProductID { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public decimal Price { get; set; }

public int StockQuantity { get; set; }

}

**#Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

static void Main()

{

// Step 3: Create List of Products

List<Product> products = new List<Product>

{

new Product { ProductID = 1, Name = "Laptop", Category = "Electronics", Price = 75000, StockQuantity = 10 },

new Product { ProductID = 2, Name = "Mobile", Category = "Electronics", Price = 30000, StockQuantity = 25 },

new Product { ProductID = 3, Name = "Desk Chair", Category = "Furniture", Price = 5000, StockQuantity = 15 },

new Product { ProductID = 4, Name = "Notebook", Category = "Stationery", Price = 100, StockQuantity = 200 },

new Product { ProductID = 5, Name = "Headphones", Category = "Electronics", Price = 2000, StockQuantity = 50 },

new Product { ProductID = 6, Name = "Table", Category = "Furniture", Price = 7000, StockQuantity = 8 }

};

// Step 4: Method Syntax Operations

Console.WriteLine("\n--- Method Syntax Queries ---");

// (your method syntax queries go here...)

// Step 5: Query Syntax Operations

Console.WriteLine("\n--- Query Syntax Queries ---");

// 1. Select only product names

var productNames = from p in products

select p.Name;

Console.WriteLine("\nProduct Names:");

foreach (var name in productNames)

Console.WriteLine(name);

// 2. Group products by Category

var groupedByCategory = from p in products

group p by p.Category;

Console.WriteLine("\nProducts Grouped by Category:");

foreach (var group in groupedByCategory)

{

Console.WriteLine($"Category: {group.Key}");

foreach (var item in group)

Console.WriteLine($" {item.Name}");

}

// 3. Count products in each category

var categoryCounts = from p in products

group p by p.Category into g

select new { Category = g.Key, Count = g.Count() };

Console.WriteLine("\nProduct Count by Category:");

foreach (var c in categoryCounts)

Console.WriteLine($"{c.Category}: {c.Count}");

// 4. Find average price

var avgPrice = (from p in products select p.Price).Average();

Console.WriteLine($"\nAverage Price of Products: {avgPrice}");

}

}

**OUTPUT:**

****