using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LinqAssignment

{

class Program

{

public static void CubeoftheNumber(int[] arr)

{

var cubeofNumberGreaterthan100andLessthan1000 = from num in arr

let cube = (num \* num \* num)

where cube > 100 && cube < 1000

orderby num ascending

select num;

Console.Write("Number list :");

foreach(var num in arr)

{

Console.Write(num + ", ");

}

Console.WriteLine();

foreach (var num in cubeofNumberGreaterthan100andLessthan1000)

Console.WriteLine("Number is {0} , its cube values is {1}", num, (num \* num \* num));

Console.WriteLine();

}

static void Main(string[] args)

{

#region 1

/\*

1.Given an array of numbers.Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ.

Change some of the array elements and execute the same query again.

\*/

Console.WriteLine("Given an array of numbers.Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ.");

int[] n = { 2, 3, 4, 5, 8, 10, 12, 23, 54, 43 };

CubeoftheNumber(n);

int[] n1 = { 2, 3, 4, 6, 7, 9, 10, 12, 23, 54, 43 };

CubeoftheNumber(n1);

Console.ReadLine();

#endregion

#region 2

/\*

\* 2.Given a list of participants for a tennis match.Split the list into 2 equal halves and display all the possible combination of matches possible between the participants in the two lists.

\* A condition is that no player should have an opponent who is from his own his own country.

\*

\*/

List <TennisPlayer> Group1 = new List<TennisPlayer>() { new TennisPlayer { PlayerName = "Feddrer", Country = "Swiss" },

new TennisPlayer { PlayerName = "Roger", Country = "Swiss" },

new TennisPlayer {PlayerName ="Andrew",Country="USA" },

new TennisPlayer {PlayerName ="DevenPort",Country="Sweden" },

new TennisPlayer {PlayerName ="Payes",Country="India" }

};

List<TennisPlayer> Group2 = new List<TennisPlayer>{ new TennisPlayer { PlayerName = "Rafel", Country = "Spanish" },

new TennisPlayer {PlayerName ="Agassi",Country="USA" },

new TennisPlayer {PlayerName ="Henmen",Country="Australia" },

new TennisPlayer {PlayerName ="Sales",Country="Sweden" },

new TennisPlayer {PlayerName ="Leyander",Country="India" }

};

Console.Clear();

var Fixtures = (from G1 in Group1

from G2 in Group2

where G1.Country != G2.Country

select G1.PlayerName + "(" + G1.Country + ")" + " vs " + G2.PlayerName + "(" + G2.Country + ")"

);

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

foreach (var fixture in Fixtures)

{

Console.WriteLine(fixture);

}

Console.ReadLine();

#endregion

#region 3

/\*

\* 3. Create an Order class that has order id, item name, order date and quantity.Create a collection of Order objects.

\* Display the data day wise from most recently ordered to least recently ordered and by quantity from highest to lowest.

\*

\*/

List<Order> Orderlst = new List<Order>() { new Order { OrderId = 1, ItemName = "Chair" , OrderDate = DateTime.Now.AddDays(10), Quantity = 10 , TotalPrice=0},

new Order { OrderId =2, ItemName = "Shoe" , OrderDate = DateTime.Now.AddDays(3), Quantity = 16 , TotalPrice=0},

new Order { OrderId = 3, ItemName = "Shirt" , OrderDate = DateTime.Now.AddDays(50), Quantity = 50 , TotalPrice=0},

new Order { OrderId = 4, ItemName = "Bike" , OrderDate = DateTime.Now.AddDays(-370), Quantity = 3 , TotalPrice=0 },

new Order { OrderId = 5, ItemName = "T-Shirt" , OrderDate = DateTime.Now.AddDays(-380), Quantity = 25 , TotalPrice=0 },

new Order { OrderId = 6, ItemName = "Watch" , OrderDate = DateTime.Now.AddDays(-330) , Quantity = 90 , TotalPrice=0 },

new Order { OrderId = 7, ItemName = "Mobile" , OrderDate = DateTime.Now.AddDays(12), Quantity = 3 , TotalPrice=0 },

new Order { OrderId = 8, ItemName = "HeadSet" , OrderDate = DateTime.Now.AddDays(32), Quantity = 4 , TotalPrice=0},

new Order { OrderId = 9, ItemName = "TV" , OrderDate = DateTime.Now.AddDays(15), Quantity = 1 , TotalPrice=0 },

new Order { OrderId = 10, ItemName = "Radio" , OrderDate = DateTime.Now.AddDays(8), Quantity = 4 , TotalPrice=0}

};

var OrderByDateWise = from i in Orderlst

orderby i.OrderDate descending, i.Quantity

select i;

var OrderByQuantityWise = from i in Orderlst

orderby i.Quantity descending

select i;

Console.Clear();

Console.WriteLine("Order Places from most recently ordered to least recently ordered");

Console.WriteLine(" OrderId ItemName OrderDate Quantity");

Console.WriteLine("----------------------------------------------------------------------------------------");

foreach (var item in OrderByDateWise)

{

Console.WriteLine(" {0} | {1} | {2} | {3} ", item.OrderId, item.ItemName, item.OrderDate.ToShortDateString(), item.Quantity);

}

Console.WriteLine();

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Order Places by quantity from highest to lowest");

Console.WriteLine(" OrderId ItemName OrderDate Quantity");

Console.WriteLine("----------------------------------------------------------------------------------------");

foreach (var item in OrderByQuantityWise)

{

Console.WriteLine(" {0} | {1} | {2} | {3} ", item.OrderId, item.ItemName, item.OrderDate.ToShortDateString(), item.Quantity);

}

Console.ReadLine();

#endregion

#region 4

/\*

\* 4. For the previous exercise, write a LINQ query that displays the details grouped by the month in the descending order of the order date.

\*

\*/

Console.Clear();

Console.WriteLine("Displays the details grouped by the month in the descending order of the order date");

Console.WriteLine();

var rr1 = (from l1 in Orderlst

group new { itemname = l1.ItemName, id = l1.OrderId, month = l1.OrderDate.Month, year = l1.OrderDate.Year } by new { month = string.Format("{0}/{1}", l1.OrderDate.Year, l1.OrderDate.Month) } into d

select new { dt = d.Key.month, count = d.Count() }).OrderByDescending(g => g.dt);

foreach (var item in rr1)

{

Console.WriteLine("Orders Placed in the month {0} ", item.dt);

var groupByOrderDateMonth = (from p in Orderlst

where string.Format("{0}/{1}", p.OrderDate.Year, p.OrderDate.Month) == item.dt

select p

);

foreach (var k in groupByOrderDateMonth)

{

Console.WriteLine(k.ItemName);

}

}

Console.WriteLine();

Console.WriteLine();

Console.ReadLine();

#endregion

#region 5

/\*

\* 5. You have created Order class in the previous exercise and that has order id, item name, order date and quantity.

\* Create another class called Item that has item name and price.

\* Write a LINQ query such that it returns order id, item name, order date

\* and the total price(price\* quantity) grouped by the month in the descending order of the order date.

\*

\*/

List<Item> Itemlst = new List<Item>() { new Item { price = 450, ItemName = "Chair" },

new Item { price =200, ItemName = "Shoe" },

new Item { price = 340, ItemName = "Shirt" },

new Item { price = 40000, ItemName = "Bike" },

new Item { price = 549.99, ItemName = "T-Shirt" },

new Item { price = 699.99, ItemName = "Watch" },

new Item { price = 7000, ItemName = "Mobile" },

new Item { price = 800, ItemName = "HeadSet" },

new Item { price = 9000, ItemName = "TV" },

new Item { price = 155, ItemName = "Radio" }

};

var OrderListWithPrice = (from order in Orderlst

join item in Itemlst on order.ItemName equals item.ItemName

orderby order.OrderDate descending

select new Order()

{

OrderId = order.OrderId,

ItemName = order.ItemName,

Quantity = order.Quantity,

TotalPrice = (order.Quantity \* item.price),

OrderDate = order.OrderDate,

month = string.Format("{0}/{1}", order.OrderDate.Year, order.OrderDate.Month)

}).GroupBy(o=> o.month).ToList();

Console.Clear();

foreach (var item in OrderListWithPrice)

{

Console.WriteLine("Orders Placed in the month {0} ", item.Key);

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

foreach (var item1 in item)

{

Console.WriteLine("OrderID : {0} ItemName: {1} OrderDate: {2} TotalPrice: {3}", item1.OrderId, item1.ItemName, item1.OrderDate, item1.TotalPrice);

}

Console.WriteLine();

Console.WriteLine();

}

Console.ReadLine();

#endregion

#region 6

/\*

\*

\* 6. Do the previous exercise using anonymous types.

\*

\*/

Console.Clear();

Console.WriteLine("Using anonymous types");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

var rr2 = (from l1 in Orderlst

group new { itemname = l1.ItemName, id = l1.OrderId, month = l1.OrderDate.Month, year = l1.OrderDate.Year }

by new { month = string.Format("{0}/{1}", l1.OrderDate.Year, l1.OrderDate.Month) } into d

select new { dt = d.Key.month, count = d.Count() }).OrderByDescending(g => g.dt);

foreach (var item in rr2)

{

Console.WriteLine("Orders Placed in the month {0} ", item.dt);

var groupByOrderDateMonthWithPrice = (from p in Orderlst

join d1 in Itemlst on p.ItemName equals d1.ItemName

where string.Format("{0}/{1}", p.OrderDate.Year, p.OrderDate.Month) == item.dt

select new { p.OrderId, p.ItemName, p.OrderDate, totalPrice = (p.Quantity \* d1.price) }

);

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

foreach (var k in groupByOrderDateMonthWithPrice)

{

Console.WriteLine("OrderID : {0} ItemName: {1} OrderDate: {2} TotalPrice: {3}", k.OrderId, k.ItemName, k.OrderDate, k.totalPrice);

}

Console.WriteLine();

Console.WriteLine();

}

Console.ReadLine();

#endregion

#region 7

/\*

\* 7.Check if all the quantities in the Order collection is > 0.

\* Get the name of the item that was ordered in largest quantity in a single order. (Hint: use LINQ methods to sort)

\* Find if there are any orders placed before Jan of this year.

\*/

Order OrdersWithLargeQuantity = (from p in Orderlst

join d1 in Itemlst on p.ItemName equals d1.ItemName

where p.Quantity > 0

orderby p.Quantity descending

select p

).Take(1).SingleOrDefault();

Console.Clear();

Console.WriteLine();

Console.WriteLine();

Console.WriteLine(" Order which is checked quantity >0 and it has most larger in quantity order . The ItemName: {0} ", OrdersWithLargeQuantity.ItemName);

List<Order> OrderPlacedBeforeJanThisYear = (from p in Orderlst

join d1 in Itemlst on p.ItemName equals d1.ItemName

where p.OrderDate.Year < DateTime.Now.Year

orderby p.Quantity descending

select p).ToList();

Console.WriteLine();

Console.WriteLine();

Console.WriteLine(" Orders placed before Jan of this year: ");

foreach (var item in OrderPlacedBeforeJanThisYear)

{

Console.WriteLine(" ItemName: {0} ", item.ItemName);

}

Console.ReadLine();

#endregion

#region 8

/\*

\* 8. Rewrite the last two example of that used Count using LINQ query methods entirely.

\* 7.a. Get the name of the item that was ordered in largest quantity in a single order. (Hint: use LINQ methods to sort)

\* 7.b. Find if there are any orders placed before Jan of this year.

\*

\*/

Console.Clear();

Order OrdersWithLargeQuantityQry = (Orderlst

.Where(s => s.Quantity > 0)

.OrderByDescending(p => p.Quantity)

.Select(s => s)

).Take(1).FirstOrDefault();

Console.WriteLine();

Console.WriteLine();

Console.WriteLine(" QueryMethods:(using Lambda) Order which is checked quantity >0 and it has most larger in quantity order . The ItemName: {0} ", OrdersWithLargeQuantityQry.ItemName);

List<Order> OrderPlacedBeforeJanThisYearQry = (Orderlst

.Where(o => o.OrderDate.Year < DateTime.Now.Year)

.OrderByDescending(p => p.Quantity)

.Select(s => s)

).ToList();

Console.WriteLine();

Console.WriteLine();

Console.WriteLine(" QueryMethods:(using Lambda) Orders placed before Jan of this year: ");

foreach (var item in OrderPlacedBeforeJanThisYearQry)

{

Console.WriteLine(" ItemName: {0} ", item.ItemName);

}

Console.ReadLine();

#endregion

#region 9

/\*

\* 9. Given the array of numbers.Count and display even numbers.

\* Write LINQ to get the sum of quantities for each item and also find out and display the item that has overall maximum orders.

\*/

Console.Clear();

Console.WriteLine();

Console.WriteLine();

int[] NumberLst = new int[100];

for (int i = 0; i < 100; i++)

{

NumberLst[i] = i+1;

}

var evenNumbers = from i in NumberLst

where i % 2 == 0

orderby i ascending

select i;

Console.WriteLine("Even Numbers between 1 to 100");

foreach (var item in evenNumbers)

{

Console.Write(item + ", ");

}

Console.WriteLine();

Console.WriteLine();

var overAll = (from i in Orderlst

group i by new { i.ItemName } into res

select new

{

count = res.Count().ToString(),

qty = res.Sum(t => t.Quantity).ToString(),

name = res.Key.ToString()

});

var MaximumOrder = overAll.OrderByDescending(a => a.qty).Take(1).ToList();

Console.WriteLine("Products and over all order quantity");

foreach (var item in overAll)

Console.WriteLine(" Item: {1} OrderQuantity:{0} ", item.qty, item.name);

Console.WriteLine();

Console.WriteLine();

foreach (var item in MaximumOrder)

Console.WriteLine("Overall maximum orders is {0} for the product:{1}", item.qty, item.name);

Console.ReadLine();

Console.Clear();

#endregion

}

}

#region Class and Members

class TennisPlayer

{

public string PlayerName { get; set; }

public string Country { get; set; }

}

class Order

{

//order id, item name, order date and quantity

public int OrderId { get; set; }

public string ItemName { get; set; }

public DateTime OrderDate { get; set; }

public int Quantity { get; set; }

public string month { get; set; }

public double TotalPrice { get; set; }

}

class Item

{

public string ItemName { get; set; }

public double price { get; set; }

}

#endregion

}

**Output ScreenShot:**

















