**Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Linq\_Assignments

{

class Program

{

/\*

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.

Hint : use the logical operators of C# to combine the conditions

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.

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.

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.

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.

6. Do the previous exercise using anonymous types.

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.

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

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.

\*/

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.

\*/

int[] n = { 2, 3, 4, 5, 8, 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43 };

var r = from i in n

where (i \* i \* i) > 100 && (i \* i \* i) < 1000

orderby i ascending

select i;

Console.WriteLine("Given an array of numbers.Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ. Number list : 2, 3, 4, 5,8, 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43");

Console.WriteLine();

foreach (var k in r)

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

Console.WriteLine();

Console.WriteLine("Given an array of numbers.Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ. Number list : 2, 3, 4, 6,7, 9, 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43");

int[] n1 = { 2, 3, 4, 6, 7, 9, 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43 };

var r1 = from i in n1

where (i \* i \* i) > 100 && (i \* i \* i) < 1000

orderby i ascending

select i;

Console.WriteLine();

foreach (var k in r1)

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

Console.WriteLine();

Console.WriteLine("Given an array of numbers.Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ. Number list : 2, 3, 4, 5,6,7 , 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43");

int[] n2 = { 2, 3, 4, 5, 6, 7, 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43 };

var r2 = from i in n2

where (i \* i \* i) > 100 && (i \* i \* i) < 1000

orderby i ascending

select i;

Console.WriteLine();

foreach (var k in r2)

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

Console.WriteLine();

Console.WriteLine("Given an array of numbers.Find the cube of the numbers that are greater than 100 but less than 1000 using LINQ. Number list : 2, 3, 4, ,8, 9, 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43");

int[] n3 = { 2, 3, 4, 8, 9, 10, 12, 23, 34, 45, 56, 67, 78, 89, 98, 76, 65, 54, 43 };

var r3 = from i in n3

where (i \* i \* i) > 100 && (i \* i \* i) < 1000

orderby i ascending

select i;

Console.WriteLine();

foreach (var k in r3)

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

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 = "1.Feddrer", Country = "Swiss" },

new TennisPlayer { PlayerName = "3.Roger", Country = "Swiss" },

new TennisPlayer {PlayerName ="5.Andrew",Country="USA" },

new TennisPlayer {PlayerName ="7.DevenPort",Country="Sweden" },

new TennisPlayer {PlayerName ="9.Tim",Country="Australia" },

new TennisPlayer {PlayerName ="11.Payes",Country="India" }

};

//{ "1.Feddrer", "3.Roger", "5.Andrew", "Devenport" };

List<TennisPlayer> Group2 = new List<TennisPlayer>{ new TennisPlayer { PlayerName = "2.Rafel", Country = "Spanish" },

new TennisPlayer { PlayerName = "3.Roger", Country = "Swiss" },

new TennisPlayer {PlayerName ="5.Agassi",Country="USA" },

new TennisPlayer {PlayerName ="7.Henmen",Country="Australia" },

new TennisPlayer {PlayerName ="9.Sales",Country="Sweden" },

new TennisPlayer {PlayerName ="11.Leyander",Country="India" }

};

//{ "2.Rafel", "4.Samprass", "6.Agassi", "Staffy" };

//var res = (from l1 in Group1

// from l2 in Group1

// where l1 != l2

// select l1 + " vs " + l2

// );

//var res1 = (from l1 in Group2

// from l2 in Group2

// where l1 != l2

// select l1 + " vs " + l2

// );

Console.Clear();

var res3 = (from l1 in Group1

from l2 in Group2

where l1.Country != l2.Country

select l1.PlayerName + "(" + l1.Country + ")" + " vs " + l2.PlayerName + "(" + l2.Country + ")"

);

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

//foreach (var item in res)

//{

// Console.WriteLine(item);

//}

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

//foreach (var item in res1)

//{

// Console.WriteLine(item);

//}

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

foreach (var item in res3)

{

Console.WriteLine(item);

}

Console.ReadLine();

Console.Clear();

#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}

};

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 OrderByDateWise = from i in Orderlst

orderby i.OrderDate, i.Quantity

select i;

var OrderByQuantityWise = from i in Orderlst

orderby i.Quantity descending

select i;

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);

}

#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.

\*

\*/

//var groupByOrderDateMonth = (from p in Orderlst

// group p by new { id = p.OrderId,month = p.OrderDate.Month, year = p.OrderDate.Year } into d

// select new { dt = string.Format("{0}/{1}", d.Key.month, d.Key.year), count = d.Count() }).OrderByDescending(g => g.dt);

Console.WriteLine();

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.Month, l1.OrderDate.Year) } 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.Month, p.OrderDate.Year) == item.dt

select p

);

foreach (var k in groupByOrderDateMonth)

{

Console.WriteLine(k.ItemName);

}

}

Console.WriteLine();

Console.WriteLine();

Console.ReadLine();

Console.Clear();

#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.

\*

\*/

var O2rderlst = (Orderlst

.Join(Itemlst, l1 => l1.ItemName, l2 => l2.ItemName,

(l1, l2) => new Order()

{

OrderId = l1.OrderId,

ItemName = l1.ItemName,

Quantity = l1.Quantity,

TotalPrice = (l1.Quantity \* l2.price),

OrderDate = l1.OrderDate,

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

}

))

.GroupBy(l2 => l2.month).ToList();

foreach (var item in O2rderlst)

{

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

#region ref

// Join between two list using query methods

//list1.Join(list2, l1 => l1.Sid, l2 => l2.Sid, (l1, l2) => new { l1.Sid, l1.Sname, l2.Cname });

// with Group by

// (list1.Join(list2, l1 => l1.Sid, l2 => l2.Sid, (l1, l2) => new { l1.Sid, l1.Sname, l2.Cname })).GroupBy(l2 => l2.Cname);

// https://docs.microsoft.com/en-us/dotnet/csharp/linq/perform-inner-joins

#endregion

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();

Console.Clear();

#endregion

#region 6

/\*

\*

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

\*

\*/

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.Month, l1.OrderDate.Year) } 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.Month, p.OrderDate.Year) == 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();

Console.Clear();

#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.

\*/

List<Order> seventh1 = (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();

Order seventh = (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.WriteLine();

Console.WriteLine();

if (seventh != null)

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

else

Console.WriteLine(" No Order is placed quantity >0 and it has larger quatity ");

Console.WriteLine();

Console.WriteLine();

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

foreach (var item in seventh1)

{

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

}

Console.ReadLine();

Console.Clear();

#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.

\*

\*/

List<Order> eightQry1 = (Orderlst

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

.OrderByDescending(p => p.Quantity)

.Select(s => s)

).ToList();

Order eightQry = (Orderlst

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

.OrderByDescending(p => p.Quantity)

.Select(s => s)

).Take(1).FirstOrDefault();

Console.WriteLine();

Console.WriteLine();

if (eightQry != null)

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

else

Console.WriteLine(" No Order is placed quantity >0 and it has larger quatity ");

Console.WriteLine();

Console.WriteLine();

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

foreach (var item in eightQry1)

{

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

}

Console.ReadLine();

Console.Clear();

#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.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;

int cnt = 0;

foreach (var item in evenNumbers)

{

Console.WriteLine("{0}.Even Numbers is {1}", (cnt + 1), item);

cnt++;

}

Console.Clear();

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 result = (from td in Orderlst

group td by td.ItemName into detail

select new

{

ItemName = detail.Key,

Qty = detail.Sum(x => x.Quantity)

}).Max(u => u.Qty);

var MaxinumOrder = overAll.Select(s => s).Where(u => u.qty.ToString() == result.ToString());

foreach (var item in overAll)

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

Console.WriteLine();

Console.WriteLine();

foreach (var item in MaxinumOrder)

Console.WriteLine(" overall maximum orders is {0} which is having quantity:{1}", item.qty, item.name);

#region Ref

//var results = from line in Lines

// group line by line.ProductCode into g

// select new ResultLine

// {

// ProductName = g.First().Name,

// Price = g.Sum(\_ => \_.Price).ToString(),

// Quantity = g.Count().ToStri ng(),

// };

//foreach (var line in Orderlst.GroupBy(info => info.ItemName)

// .Select(group => new {

// Metric = group.Key,

// Count = group.Count()

// })

// .OrderBy(x => x.Metric)

//{

// Console.WriteLine("{0} {1}", line.Metric, line.Count);

//}

#endregion

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

}

**Screen Shots:**

