(T13)討論 GroupJoin、InnerJoin、LeftOutterJoin、CrossJoin
CourseGUID: 5ba9a6fe-7475-4b0c-8b99-bbcf7f5e2e1c
==========
(T13)討論 GroupJoin、InnerJoin、LeftOutterJoin、CrossJoin
==========

==========

0. Summary

0.
In this Tutorial, each Team have several Gamers.
Each Gamer can only have one Team.
This is One to Many Relationship.

1.
Select() and SelectMany() are projection operators
which can specify what properties to retrieve,
just like TSQL Select clause can specify what columns to retrieve.

1.0.
Select() V.S. SelectMany()
1.0.1.
If T1 has List<T2> as its property,
I assume there is a List<T1>.
When we use Select() method,
then it will return List of List<T2>.
Thus, we have to use 2 nested foreach loops to get all List of List<T2>
1.0.2.
SelectMany() flattens queries that return lists of lists into a single list.
Thus, we just need 1 foreach loops to get all List<T2>

1.1.
//Enumerable.Select<TSource, TResult>
//(this IEnumerable<TSource> source, Func<TSource, TResult> selector)
Reference:
https://msdn.microsoft.com/en-us/library/bb548891(v=vs.110).aspx
Projects each element of a sequence into a new form.

1.2.
//Enumerable.SelectMany<TSource, TResult>
//(this IEnumerable<TSource> source, Func<TSource, IEnumerable<TResult>> selector)
Reference:
https://msdn.microsoft.com/en-us/library/bb534336(v=vs.110).aspx

Projects each element of a sequence to an IEnumerable<T>
and flattens the resulting sequences into one sequence.

1.3.
//Enumerable.SelectMany<TSource, TCollection, TResult>
//(this IEnumerable<TSource> source ,
//Func<TSource, IEnumerable<TCollection>> collectionSelector,
//Func<TSource, TCollection, TResult> resultSelector)
Reference:
https://msdn.microsoft.com/en-us/library/bb534631(v=vs.110).aspx
Projects each element of a sequence to an IEnumerable<T>,
flattens the resulting sequences into one sequence,
and invokes a result selector function on each element therein.
TSource
The type of the elements of source.
TCollection
The type of the intermediate elements collected by collectionSelector.
TResult
The type of the elements of the resulting sequence.

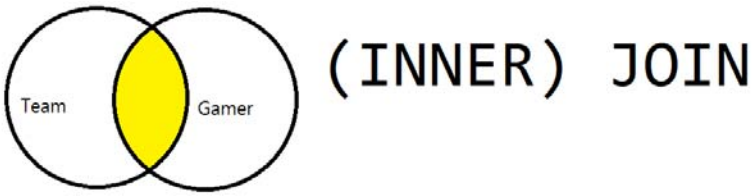
1.3.1.
E.g.
////Error!!
//var gamerNameAlongWithSkills2 = GamerHelper.GetSampleGamers()
// .SelectMany(
// (gamer, skill) => new { GamerName = gamer.Name, Skill = skill });

1.3.2.
//var gamerNameAlongWithSkills = GamerHelper.GetSampleGamers()
// .SelectMany(
// g => g.Skills,
// (gamer, skill) => new { GamerName = gamer.Name, Skill = skill });
//Console.WriteLine($"gamerNameAlongWithSkills.Count()=={gamerNameAlongWithSkills.Count()}");
//foreach (var gamerNameAlongWithSkillsItem in gamerNameAlongWithSkills)
//{
// Console.WriteLine($"GamerName=={gamerNameAlongWithSkillsItem.GamerName}, " +
// $"Skill=={gamerNameAlongWithSkillsItem.Skill}");
//}
If SelectMany want to project to anonymous type,
then it need the second parameter,
Func<TSource, IEnumerable<TCollection>> collectionSelector.
//g => g.Skills,
Firstly, invoke the one-to-many transform function collectionSelector on each source element.
//(gamer, skill) => new { GamerName = gamer.Name, Skill = skill });
The first parameter of (gamer, skill) represents each element from List<T>,
In this case, "gamer" means each gamer from List<Gamer> which is from GamerHelper.GetSampleGamers().
The second parameter of (gamer, skill) is from collectionSelector
which is the second parameter of SelectMany.
In this case, "skill" means each skill of "g.Skills".
And then mapping each of those to anonymous type properties.

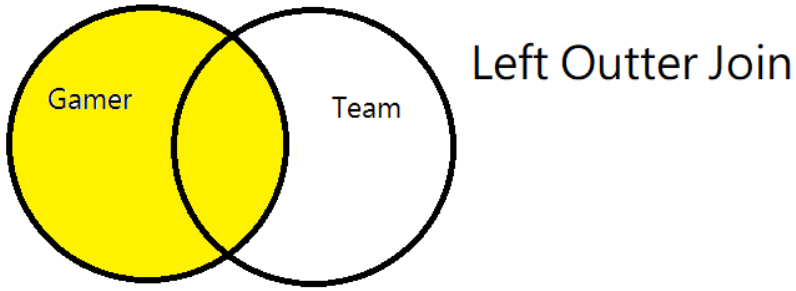
2.
//GroupJoin()

or
//from ... Join ...on ... Into ...
GroupJoin create hierarchical data structures
that each element from the first collection
is paired with several elements from the second collection.
E.g.
Each Team has several Gamers.
So you need 1st foreach to loop Teams
and 2nd foreach to loop Gamers.

3.
InnerJoin
//Join()
or
//from ... Join ... on ...
InnerJoin in Linq joins 2 collections into one collection,
just like InnerJoin in TSQL which only take the matching elements between 2 collections.



4.
LeftOutterJoin
//gamersList.GroupJoin(...teamsList...).SelectMany(...)
or
//From...Join...On...Into...(From...)Select...
LeftOutterJoin in Linq joins 2 collections into one collection,
just like LeftOutterJoin in TSQL which only take the matching elements between 2 collections,
plus non-matching elements from the left.



5.
CrossJoin
// from ... from ...Select...
or
//SelectMany(...)
Returns Cartesian product of two collections
involved in the join
CrossJoin does not need ON

==========

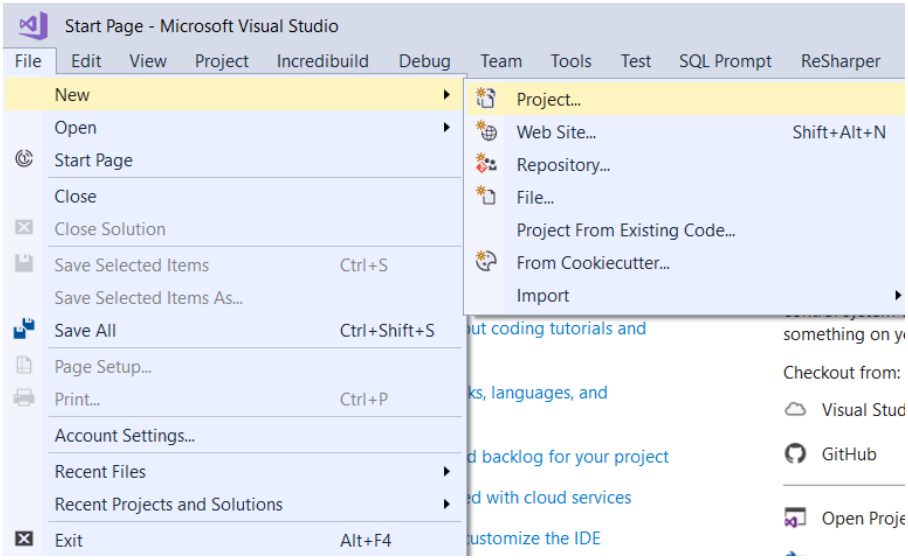
1. New Project

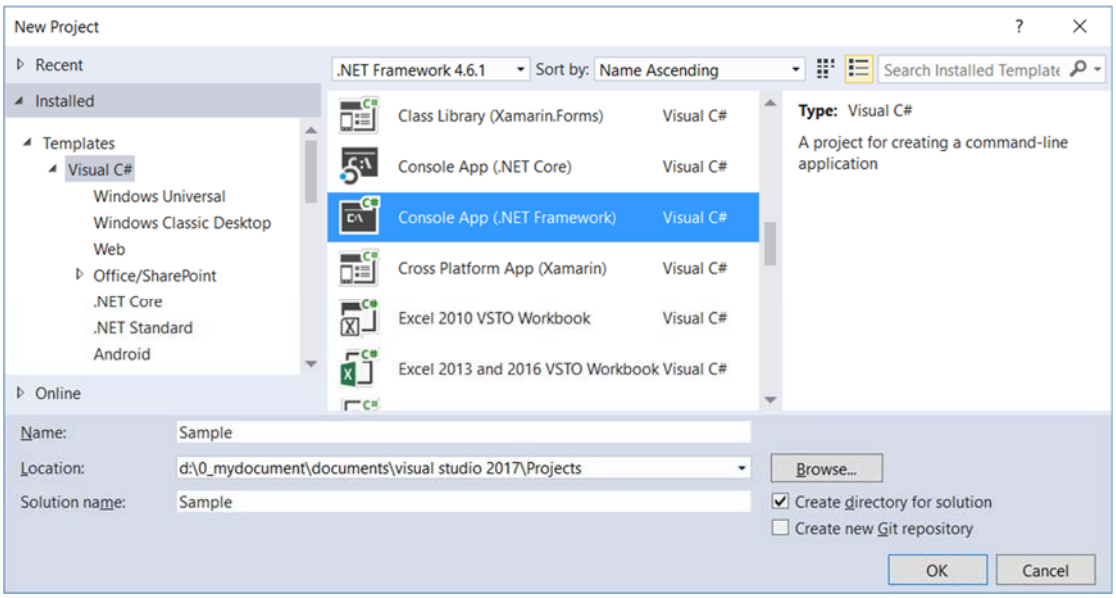
1.1. Create New Project : Sample

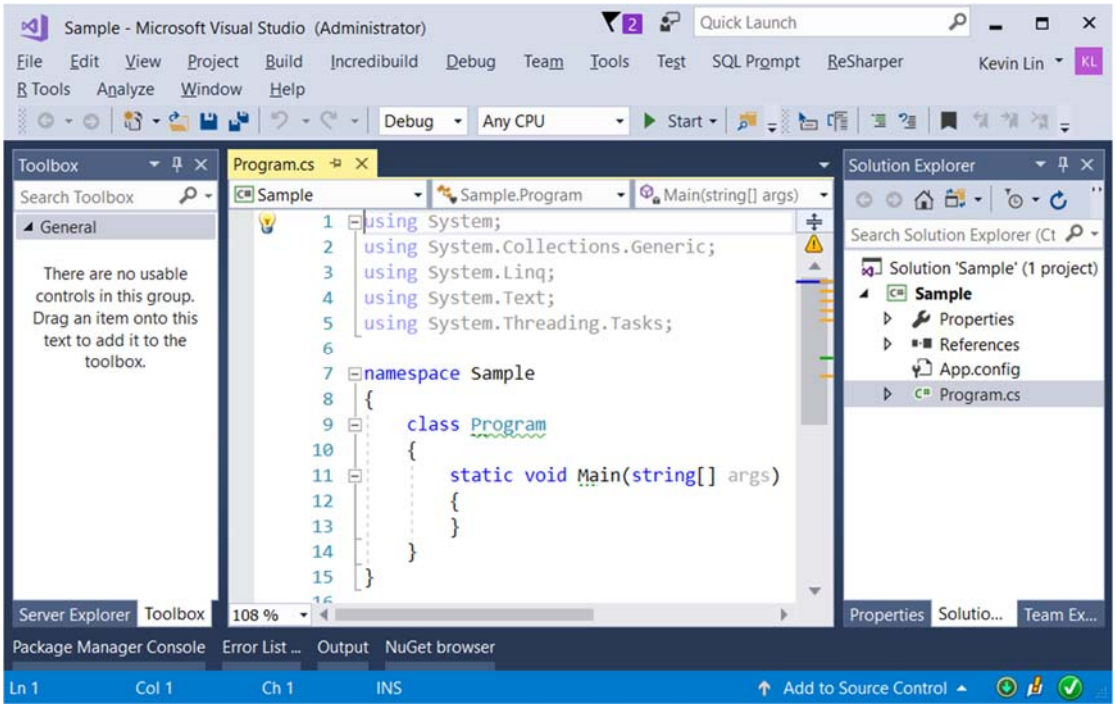
File --> New --> Project... -->

Visual C# --> **Console App (.Net Framework)** -->

Name: **Sample**







==========

2. Sample：Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using OnLieGame;
namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            List<Team> teamsList = TeamHelper.GetSampleTeam();
            List<Gamer> gamersList = GamerHelper.GetSampleGamer();
            //1. ==================================
            //TeamsListJoinGamersList(teamsList, gamersList)
            Console.WriteLine("1. TeamsListJoinGamersList(teamsList, gamersList) ======= ");
             TeamsListJoinGamersList(teamsList, gamersList);
            //2. ==================================
            //GamersListJoinTeamsList(teamsList, gamersList)
            Console.WriteLine("2. GamersListJoinTeamsList(teamsList, gamersList) ======= ");
             GamersListJoinTeamsList(teamsList, gamersList);
            //3. ==================================
            //GamersListCrossJoinTeamsList(teamsList, gamersList)
            Console.WriteLine("3. GamersListCrossJoinTeamsList(teamsList, gamersList) ======= ");
             GamersListCrossJoinTeamsList(teamsList, gamersList);
            Console.ReadLine();
        }


      //1. ==================================
```

```
        //TeamsListGroupJoinGamersList(teamsList, gamersList)
        static void TeamsListJoinGamersList(List<Team> teamsList, List<Gamer> gamersList)
        {
            // 1.1. TeamsListGroupJoinGamersList -------------------------------
            //GroupJoin
            // teamsList GroupJoin GamersList by Lambda expression query.
            //1.1.1.
            //Each Team have several Gamers.
            //Each Gamer can only have one Team.
            //This is One to Many Relationship.
            //1.1.2.
            //GroupJoin create hierarchical data structures
            //that each element from the first collection
            //is paired with several elements from the second collection.
            //This will create outter "Team" collection, and
            //Each "Team" will contain inner "Gamers" collections.
            //1st foreach loop the Team, 2n foreach loop the Gamers.
            //1.1.3.
            //There are 3 Teams, and Team with Id==3 has no Gamer.
            //There are 6 Gamers, and Gamer with Id==6 has no Team.
            //GroupJoin will only take matching elements from both two collections.
            //Thus, Gamer with Id==6 ,which has no team, will not be returned.
            //It will return only 5 matching Gamers who has a Team.
            Console.WriteLine("1.1. TeamsListGroupJoinGamersList ------------- ");
            Console.WriteLine("1.1. teamsList GroupJoin GamersList by Lambda expression query ------------
- ");
            var teamsListGroupJoinGamersList =
             teamsList.GroupJoin(
                 gamersList,
                 t => t.Id,
                 g => g.TeamId,
                 (team, gamer) => new
                 {
                     Team = team,
                     Gamers = gamer
                 }
             );
            Console.WriteLine($"teamsListGroupJoinGamersList.Count(): {teamsListGroupJoinGamersList.Count(
)}");
            foreach (var teamsListGroupJoinGamersListItem in teamsListGroupJoinGamersList)
            {
                Console.WriteLine($"teamsListGroupJoinGamersListItem.Gamers.Count(): {teamsListGroupJoinGa
mersListItem.Gamers.Count()}");
                // Look the Type
                var item = teamsListGroupJoinGamersListItem;
                Team itemTeam = teamsListGroupJoinGamersListItem.Team;
                IEnumerable<Gamer> itemGamers = teamsListGroupJoinGamersListItem.Gamers;
                //Print Team
                Console.WriteLine($"Team.Name: {itemTeam} ----- ");
                //Print Gamer
                foreach (Gamer gamer in itemGamers)
                {
                    Console.WriteLine($"gamer: {gamer}");
                }
                Console.WriteLine();
            }
```

```
            // teamsListGroupJoinGamersList.Count(): 3
            // teamsListGroupJoinGamersListItem.Gamers.Count(): 3
            // Team.Name: TeamId==1,TeamName=Team1 -----
            // gamer: GamerId==1,GamerName=Name1,TeamId=1
            // gamer: GamerId==3,GamerName=Name3,TeamId=1
            // gamer: GamerId==4,GamerName=Name4,TeamId=1
            // teamsListGroupJoinGamersListItem.Gamers.Count(): 2
            // Team.Name: TeamId==2,TeamName=Team2 -----
            // gamer: GamerId==2,GamerName=Name2,TeamId=2
            // gamer: GamerId==5,GamerName=Name9,TeamId=2
            // teamsListGroupJoinGamersListItem.Gamers.Count(): 0
            // Team.Name: TeamId==3,TeamName=Team3 -----


           // 1.2. teamsListJoinGamersListIntoGroupSqlLikeQuery ----------------------------
            //GroupJoin
            // teamsList GroupJoin GamersList by Sql like query.
            //1.2.1.
            //Each Team have several Gamers.
            //Each Gamer can only have one Team.
            //This is One to Many Relationship.
            //1.2.2.
            //GroupJoin create hierarchical data structures
            //that each element from the first collection
            //is paired with several elements from the second collection.
            //This will create outter "Team" collection, and
            //Each "Team" will contain inner "Gamers" collections.
            //1st foreach loop the Team, and 2n foreach loop the Gamers.
            //1.2.3.
            //There are 3 Teams, and Team with Id==3 has no Gamer.
            //There are 6 Gamers, and Gamer with Id==6 has no Team.
            //GroupJoin will only take matching elements from both two collections.
            //Thus, Gamer with Id==6 ,which has no team, will not be returned.
            //It will return only 5 matching Gamers who has a Team.
            Console.WriteLine("1.2. teamsListJoinGamersListIntoGroupSqlLikeQuery ------------- ");
            Console.WriteLine("1.2. teamsList GroupJoin GamersList by Sql like query. ------------- ");
            var teamsListJoinGamersListIntoGroupSqlLikeQuery =
                from t in teamsList
                join g in gamersList
                on t.Id equals g.TeamId
                into gamerGroup     // Different Here
                select new
                {
                    Team = t,
                    Gamers = gamerGroup
                };
            Console.WriteLine($"teamsListJoinGamersListIntoGroupSqlLikeQuery.Count(): {teamsListJoinGamers
ListIntoGroupSqlLikeQuery.Count()}");
            foreach (var teamsListJoinGamersListIntoGroupSqlLikeQueryItem in teamsListJoinGamersListIntoGrou
pSqlLikeQuery)
            {
                Console.WriteLine($"teamsListJoinGamersListIntoGroupSqlLikeQueryItem.Gamers.Count(): {team
sListJoinGamersListIntoGroupSqlLikeQueryItem.Gamers.Count()}");
                // Look the Type
```

```
    var item = teamsListJoinGamersListIntoGroupSqlLikeQueryItem;
    Team itemTeam = teamsListJoinGamersListIntoGroupSqlLikeQueryItem.Team;
    IEnumerable<Gamer> itemGamers = teamsListJoinGamersListIntoGroupSqlLikeQueryItem.Gamers;
    // Print Team
    Console.WriteLine($"Team: {itemTeam} ---- ");
    //Print Gamer
    foreach (Gamer gamer in itemGamers)
    {
        Console.WriteLine($"gamer: {gamer}");
    }
    Console.WriteLine();
}
// teamsListJoinGamersListIntoGroupSqlLikeQuery.Count(): 3
// teamsListJoinGamersListIntoGroupSqlLikeQueryItem.Gamers.Count(): 3
// Team: TeamId==1,TeamName=Team1 ----
// gamer: GamerId==1,GamerName=Name1,TeamId=1
// gamer: GamerId==3,GamerName=Name3,TeamId=1
// gamer: GamerId==4,GamerName=Name4,TeamId=1
// teamsListJoinGamersListIntoGroupSqlLikeQueryItem.Gamers.Count(): 2
// Team: TeamId==2,TeamName=Team2 ----
// gamer: GamerId==2,GamerName=Name2,TeamId=2
// gamer: GamerId==5,GamerName=Name9,TeamId=2
// teamsListJoinGamersListIntoGroupSqlLikeQueryItem.Gamers.Count(): 0
// Team: TeamId==3,TeamName=Team3 ----


// 1.3. teamsListJoinGamersListSqlLikeQuery -----------------------------------
//InnerJoin
// teamsList InnerJoin GamersList by Sql like query.
//1.3.1.
//Each Team have several Gamers.
//Each Gamer can only have one Team.
//This is One to Many Relationship.
//1.3.2.
//"Teams" collection InnerJoin "Gamer" collection into one collection.
//Thus, it need only one foreach to loop all elements in collection.
//1.3.3.
//There are 3 Teams, and Team with Id==3 has no Gamer.
//There are 6 Gamers, and Gamer with Id==6 has no Team.
//InnerJoin will only take matching elements from both two collections.
//Thus, Gamer with Id==6 ,which has no team, will not be returned.
//It will return only 5 matching "Gamers" who has a "Team".
Console.WriteLine("1.3. teamsListJoinGamersListSqlLikeQuery ------------ ");
Console.WriteLine("1.3. teamsList InnerJoin GamersList by Sql like query. ------------ ");
var teamsListJoinGamersListSqlLikeQuery =
    from t in teamsList
    join g in gamersList
    on t.Id equals g.TeamId
    select new
    {
        Gamer = g,
        Team = t
    };
```

```
            Console.WriteLine($"teamsListJoinGamersListSqlLikeQuery.Count()=={teamsListJoinGamersListSqlLi
keQuery.Count()}");
            foreach (var teamsListJoinGamersListSqlLikeQueryItem in teamsListJoinGamersListSqlLikeQuery)
            {
                //Look the Type
                var item = teamsListJoinGamersListSqlLikeQueryItem;
                Team itemTeam = teamsListJoinGamersListSqlLikeQueryItem.Team;
                Gamer itemGamer = teamsListJoinGamersListSqlLikeQueryItem.Gamer;
                // Print Team and Gamer
                Console.WriteLine($"Gamer:{itemGamer}  ;  " +
                                  $"Team:{itemTeam}");
            }
            // teamsListJoinGamersListSqlLikeQuery.Count()==5
            // Gamer:GamerId==1,GamerName=Name1,TeamId=1  ;  Team:TeamId==1,TeamName=Team1
            // Gamer:GamerId==3,GamerName=Name3,TeamId=1  ;  Team:TeamId==1,TeamName=Team1
            // Gamer:GamerId==4,GamerName=Name4,TeamId=1  ;  Team:TeamId==1,TeamName=Team1
            // Gamer:GamerId==2,GamerName=Name2,TeamId=2  ;  Team:TeamId==2,TeamName=Team2
            // Gamer:GamerId==5,GamerName=Name9,TeamId=2  ;  Team:TeamId==2,TeamName=Team2


           // 1.4. teamsListJoinGamersList ----------------------------------
            //InnerJoin
            // teamsList InnerJoin GamersList by Lambda expression query.
            //1.4.1.
            //Each Team have several Gamers.
            //Each Gamer can only have one Team.
            //This is One to Many Relationship.
            //1.4.2.
            //"Teams" collection InnerJoin "Gamer" collection into one collection.
            //Thus, it need only one foreach to loop all elements in collection.
            //1.4.3.
            //There are 3 Teams, and Team with Id==3 has no Gamer.
            //There are 6 Gamers, and Gamer with Id==6 has no Team.
            //InnerJoin will only take matching elements from both two collections.
            //Thus, Gamer with Id==6 ,which has no team, will not be returned.
            //It will return only 5 matching "Gamers" who has a "Team".
            Console.WriteLine("1.4. teamsListJoinGamersList ------------- ");
            Console.WriteLine("1.4. teamsList InnerJoin GamersList by Lambda expression query. ----------
-- ");
            var teamsListJoinGamersList =
                 teamsList.Join(gamersList,
                 t => t.Id,
                 g => g.TeamId,
                 (team, gamer) => new
                 {
                     Gamer = gamer,
                     Team = team
                 });
            Console.WriteLine($"teamsListJoinGamersList.Count()=={teamsListJoinGamersList.Count()}");
            foreach (var teamsListJoinGamersListItem in teamsListJoinGamersList)
            {
                //Look the Type
                var item = teamsListJoinGamersListItem;
               Team itemTeam = teamsListJoinGamersListItem.Team;
               Gamer itemGamer = teamsListJoinGamersListItem.Gamer;
                // Print Team and Gamer
```

```
                Console.WriteLine($"Gamer:{itemGamer}  ;  " +
                                  $"Team:{itemTeam}");
            }
            // teamsListJoinGamersList.Count()==5
            // Gamer:GamerId==1,GamerName=Name1,TeamId=1  ;  Team:TeamId==1,TeamName=Team1
            // Gamer:GamerId==3,GamerName=Name3,TeamId=1  ;  Team:TeamId==1,TeamName=Team1
            // Gamer:GamerId==4,GamerName=Name4,TeamId=1  ;  Team:TeamId==1,TeamName=Team1
            // Gamer:GamerId==2,GamerName=Name2,TeamId=2  ;  Team:TeamId==2,TeamName=Team2
            // Gamer:GamerId==5,GamerName=Name9,TeamId=2  ;  Team:TeamId==2,TeamName=Team2
        }


        //2. ==================================
        //GamersListJoinTeamsList(teamsList, gamersList)
        static void GamersListJoinTeamsList(List<Team> teamsList, List<Gamer> gamersList)
        {
            // 2.1. gamersListGroupJoinTeamsList ------------------------------------
            //LeftOutterJoin (Not the right way)
            //GamersList GroupJoin teamsList by Lambda expression query.
            //In fact, this is GamersList LeftOutterJoin teamsList by lambda expression Query.
            //But this is NOT the "right way" to do LeftOutterJoin
            //gamersList.GroupJoin(...teamsList...)...
            //2.1.1.
            //Each Team have several Gamers.
            //Each Gamer can only have one Team.
            //This is One to Many Relationship.
            //2.1.2.
            //"Gamer" collection LeftOutterJoin "Team" collection into one collection.
            //Thus, it need only one foreach to loop all elements in collection.
            //2.1.3.
            //There are 3 Teams, and Team with Id==3 has no Gamer.
            //There are 6 Gamers, and Gamer with Id==6 has no Team.
            //LeftOutterJoin will take the matching elements from both two collections,
            //plus the non-matching elements from the left, which is "Gamer".
            //Thus, Gamer with Id==6 ,which has no team, will still be returned.
            //It will return 5 "Gamers" who has a "Team", plus 1 "Gamer" who has no "Team".
            Console.WriteLine("2.1. gamersListGroupJoinTeamsList ------------- ");
            Console.WriteLine("2.1. GamersList LeftOutterJoin teamsList by lambda expression Query, but
NOT the right way to do 'LeftOutterJoin' ------ ");
            var gamersListGroupJoinTeamsList =
            gamersList.GroupJoin(
                teamsList,
                g => g.TeamId,
                t => t.Id,
                (gamer, teams) => new
                {
                    Teams = teams,
                    //teams is actually a collections, not single item.
                    //because Each Team can have many Gamers
                    //but each Gamer can only have one Team in this case.
                    Gamer = gamer
                }
            );
            Console.WriteLine($"gamersListGroupJoinTeamsList.Count()=={gamersListGroupJoinTeamsList.Count(
)}");
            foreach (var gamersListGroupJoinTeamsListItem in gamersListGroupJoinTeamsList)
```

```
{
    //Look the Type
    var item = gamersListGroupJoinTeamsListItem;
    Gamer itemGamer = gamersListGroupJoinTeamsListItem.Gamer;
    IEnumerable<Team> itemTeams = gamersListGroupJoinTeamsListItem.Teams;
    Team itemTeamsFirst = gamersListGroupJoinTeamsListItem.Teams.FirstOrDefault();
    //Print Gamer
    Console.WriteLine($"Gamer: {itemGamer}");
    //Print Team
    Console.WriteLine($"Teams: {itemTeams}");
    Console.WriteLine($"Teams.FirstOrDefault(): {itemTeamsFirst}");
    foreach (Team team in itemTeams)
    {
        Console.WriteLine($"team: {team}");
    }
    Console.WriteLine();
}
// gamersListGroupJoinTeamsList.Count()==6
// Gamer: GamerId==1,GamerName=Name1,TeamId=1
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
// team: TeamId==1,TeamName=Team1
// Gamer: GamerId==2,GamerName=Name2,TeamId=2
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==2,TeamName=Team2
// team: TeamId==2,TeamName=Team2
// Gamer: GamerId==3,GamerName=Name3,TeamId=1
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
// team: TeamId==1,TeamName=Team1
// Gamer: GamerId==4,GamerName=Name4,TeamId=1
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
// team: TeamId==1,TeamName=Team1
// Gamer: GamerId==5,GamerName=Name9,TeamId=2
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==2,TeamName=Team2
// team: TeamId==2,TeamName=Team2
// Gamer: GamerId==6,GamerName=Name10,TeamId=0
// Teams: OnLieGame.Team[]
// Teams.FirstOrDefault():


// 2.2. gamersListGroupJoinTeamsListSelectMany ---------------------------------------
//LeftOutterJoin
//GamersList GroupJoin teamsList by Lambda expression query.
//In fact, this is GamersList LeftOutterJoin teamsList by lambda expression Query.
//this is the "right way" to do LeftOutterJoin
//gamersList.GroupJoin(...teamsList...).SelectMany(...)
//2.2.1.
//Each Team have several Gamers.
//Each Gamer can only have one Team.
//This is One to Many Relationship.
//2.2.2.
```

```
            //"Gamer" collection LeftOutterJoin "Team" collection into one collection.
            //Thus, it need only one foreach to loop all elements in collection.
            //2.2.3.
            //There are 3 Teams, and Team with Id==3 has no Gamer.
            //There are 6 Gamers, and Gamer with Id==6 has no Team.
            //LeftOutterJoin will take the matching elements from both two collections,
            //plus the non-matching elements from the left, which is "Gamer".
            //Thus, Gamer with Id==6 ,which has no team, will still be returned.
            //It will return 5 "Gamers" who has a "Team", plus 1 "Gamer" who has no "Team".
            Console.WriteLine("2.2. gamersListGroupJoinTeamsListSelectMany ------------- ");
            Console.WriteLine("2.2. GamersList LeftOutterJoin teamsList by lambda expression Query, this
is the right way to do 'LeftOutterJoin' ------ ");
            var gamersListGroupJoinTeamsListSelectMany =
             gamersList.GroupJoin(
                 teamsList,
                 g => g.TeamId,
                 t => t.Id,
                 (gamer, teams) => new
                 {
                     Teams = teams,
                    //teams is actually a collections, not single item.
                    //because Each Team can have many Gamers
                    //but each Gamer can only have one Team in this case.
                     Gamer = gamer
                 }
             ).SelectMany(
                 item => item.Teams.DefaultIfEmpty(),
                 (item, itemTeam) => new
                 {
                     Gamer = item.Gamer,
                     Team = itemTeam
                    //"item" parameter means each item of "new{Teams,Gamer}"
                    //"itemTeam" parameter means each item of "item.Teams.DefaultIfEmpty()"
                 }
             );
            Console.WriteLine($"gamersListGroupJoinTeamsListSelectMany.Count()=={gamersListGroupJoinTeamsL
istSelectMany.Count()}");
            foreach (var gamersListGroupJoinTeamsListSelectManyItem in gamersListGroupJoinTeamsListSelectMan
y)
            {
                //Look the Type
                var item = gamersListGroupJoinTeamsListSelectManyItem;
                Gamer itemGamer = gamersListGroupJoinTeamsListSelectManyItem.Gamer;
                Team itemTeam = gamersListGroupJoinTeamsListSelectManyItem.Team;
                //Print Gamer
                Console.WriteLine($"Gamer: {itemGamer}");
                //Print Team
                Console.WriteLine($"Team: {itemTeam}");
                Console.WriteLine();
            }
            // gamersListGroupJoinTeamsListSelectMany.Count()==6
            // Gamer: GamerId==1,GamerName=Name1,TeamId=1
            // Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==2,GamerName=Name2,TeamId=2
            // Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==3,GamerName=Name3,TeamId=1
            // Team: TeamId==1,TeamName=Team1
```

```
            // Gamer: GamerId==4,GamerName=Name4,TeamId=1
            // Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==5,GamerName=Name9,TeamId=2
            // Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==6,GamerName=Name10,TeamId=0
            // Team:



            // 2.3. gamersListJoinTeamsListIntoGroupSqlLikeQuery ----------------------------------------
            //LeftOutterJoin (Not the right way)
            //GamersList GroupJoin teamsList by Sql Like Query
            //In fact, this is GamersList LeftOutterJoin teamsList by Sql Like Query.
            //But this is NOT the "right way" to do LeftOutterJoin
            //From...Join...On...Into...Select...
            //2.3.1.
            //Each Team have several Gamers.
            //Each Gamer can only have one Team.
            //This is One to Many Relationship.
            //2.3.2.
            //"Gamer" collection LeftOutterJoin "Team" collection into one collection.
            //Thus, it need only one foreach to loop all elements in collection.
            //2.3.3.
            //There are 3 Teams, and Team with Id==3 has no Gamer.
            //There are 6 Gamers, and Gamer with Id==6 has no Team.
            //LeftOutterJoin will take the matching elements from both two collections,
            //plus the non-matching elements from the left, which is "Gamer".
            //Thus, Gamer with Id==6 ,which has no team, will still be returned.
            //It will return 5 "Gamers" who has a "Team", plus 1 "Gamer" who has no "Team".
            Console.WriteLine("2.3. gamersListJoinTeamsListIntoGroupSqlLikeQuery ------------- ");
            Console.WriteLine("2.3. GamersList LeftOutterJoin teamsList by Sql Like Query, but NOT the
right way to do 'LeftOutterJoin' ------ ");
            var gamersListJoinTeamsListIntoGroupSqlLikeQuery =
                from g in gamersList
                join t in teamsList
                on g.TeamId equals t.Id
                into teampGroup     // Different Here
                select new
                {
                    Gamer = g,
                    Teams = teampGroup
                };
            Console.WriteLine($"gamersListJoinTeamsListIntoGroupSqlLikeQuery.Count()=={gamersListJoinTeams
ListIntoGroupSqlLikeQuery.Count()}");
            foreach (var gamersListJointeamsListIntoGroupSqlLikeQueryItem in gamersListJoinTeamsListIntoGrou
pSqlLikeQuery)
            {
                // Look the Type
                var item = gamersListJointeamsListIntoGroupSqlLikeQueryItem;
                IEnumerable<Team> itemTeams = gamersListJointeamsListIntoGroupSqlLikeQueryItem.Teams;
                Team itemTeamsFirst =
gamersListJointeamsListIntoGroupSqlLikeQueryItem.Teams.FirstOrDefault();
               Gamer itemGamer = gamersListJointeamsListIntoGroupSqlLikeQueryItem.Gamer;
                //Print Gamer
                Console.WriteLine($"Gamer: {itemGamer}");
```

```
        // Print Team
        Console.WriteLine($"Teams: {itemTeams}");
        Console.WriteLine($"Teams.FirstOrDefault(): {itemTeamsFirst}");
        foreach (Team team in itemTeams)
        {
            Console.WriteLine($"team: {team}");
        }
        Console.WriteLine();
    }
    // gamersListJoinTeamsListIntoGroupSqlLikeQuery.Count()==6
    // Gamer: GamerId==1,GamerName=Name1,TeamId=1
    // Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
    // Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
    // team: TeamId==1,TeamName=Team1
    // Gamer: GamerId==2,GamerName=Name2,TeamId=2
    // Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
    // Teams.FirstOrDefault(): TeamId==2,TeamName=Team2
    // team: TeamId==2,TeamName=Team2
    // Gamer: GamerId==3,GamerName=Name3,TeamId=1
    // Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
    // Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
    // team: TeamId==1,TeamName=Team1
    // Gamer: GamerId==4,GamerName=Name4,TeamId=1
    // Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
    // Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
    // team: TeamId==1,TeamName=Team1
    // Gamer: GamerId==5,GamerName=Name9,TeamId=2
    // Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
    // Teams.FirstOrDefault(): TeamId==2,TeamName=Team2
    // team: TeamId==2,TeamName=Team2
    // Gamer: GamerId==6,GamerName=Name10,TeamId=0
    // Teams: OnLieGame.Team[]
    // Teams.FirstOrDefault():



    // 2.4. gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQuery --------------------------------
    //LeftOutterJoin
    //GamersList GroupJoin teamsList by Sql Like Query
    //In fact, this is GamersList LeftOutterJoin teamsList by Sql Like Query.
    //and this is the "right way" to do 'LeftOutterJoin'
    //From...Join...On...Into...(From...)Select...
    //2.4.1.
    //Each Team have several Gamers.
    //Each Gamer can only have one Team.
    //This is One to Many Relationship.
    //2.4.2.
    //"Gamer" collection LeftOutterJoin "Team" collection into one collection.
    //Thus, it need only one foreach to loop all elements in collection.
    //2.4.3.
    //There are 3 Teams, and Team with Id==3 has no Gamer.
    //There are 6 Gamers, and Gamer with Id==6 has no Team.
    //LeftOutterJoin will take the matching elements from both two collections,
```

```
            //plus the non-matching elements from the left, which is "Gamer".
            //Thus, Gamer with Id==6 ,which has no team, will still be returned.
            //It will return 5 "Gamers" who has a "Team", plus 1 "Gamer" who has no "Team".
            Console.WriteLine("2.4. gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQuery ------------
");
            Console.WriteLine("2.4. GamersList LeftOutterJoin teamsList by Sql Like Query, and this is the
right way to do 'LeftOutterJoin' ------ ");
            var gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQuery =
                from g in gamersList
                join t in teamsList
                on g.TeamId equals t.Id
                into teampGroup     // Different Here
                from team in teampGroup.DefaultIfEmpty()
                    // Different Here
                    //it means for each team from teampGroup, teampGroup is a collection
                select new
                {
                    Gamer = g,
                    Teams = teampGroup,
                    //Team = team == null ?
                    //    new Team { Id = 0, Name = "NoTeam" } :
                    //    team
                    Team = team ?? new Team { Id = 0, Name = "NoTeam" }
                };
            Console.WriteLine($"gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQuery.Count()=={gamersList
JoinTeamsListIntoGroupFromGroupSqlLikeQuery.Count()}");
            foreach (var gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQueryItem in gamersListJoinTeamsLis
tIntoGroupFromGroupSqlLikeQuery)
            {
                // Look the Type
                var item = gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQueryItem;
                Gamer itemGamer = gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQueryItem.Gamer;
                IEnumerable<Team> itemTeams =
gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQueryItem.Teams;
                // itemTeams is a collection
                Team itemTeamsFirst =
gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQueryItem.Teams.FirstOrDefault();
                Team itemTeam = gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQueryItem.Team;
                //Print Gamer
                Console.WriteLine($"Gamer: {itemGamer}");
                // Print Team
                Console.WriteLine($"Teams: {itemTeams}");
                Console.WriteLine($"Teams.FirstOrDefault(): {itemTeamsFirst}");
                Console.WriteLine($"Team: {itemTeam}");
                foreach (Team team in itemTeams)
                {
                    Console.WriteLine($"team: {team}");
                }
                Console.WriteLine();
            }
            // gamersListJoinTeamsListIntoGroupFromGroupSqlLikeQuery.Count()==6
            // Gamer: GamerId==1,GamerName=Name1,TeamId=1
            // Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
            // Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
            // Team: TeamId==1,TeamName=Team1
            // team: TeamId==1,TeamName=Team1
```

```
// Gamer: GamerId==2,GamerName=Name2,TeamId=2
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==2,TeamName=Team2
// Team: TeamId==2,TeamName=Team2
// team: TeamId==2,TeamName=Team2
// Gamer: GamerId==3,GamerName=Name3,TeamId=1
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
// Team: TeamId==1,TeamName=Team1
// team: TeamId==1,TeamName=Team1
// Gamer: GamerId==4,GamerName=Name4,TeamId=1
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==1,TeamName=Team1
// Team: TeamId==1,TeamName=Team1
// team: TeamId==1,TeamName=Team1
// Gamer: GamerId==5,GamerName=Name9,TeamId=2
// Teams: System.Linq.Lookup`2+Grouping[System.Int32,OnLieGame.Team]
// Teams.FirstOrDefault(): TeamId==2,TeamName=Team2
// Team: TeamId==2,TeamName=Team2
// team: TeamId==2,TeamName=Team2
// Gamer: GamerId==6,GamerName=Name10,TeamId=0
// Teams: OnLieGame.Team[]
// Teams.FirstOrDefault():
// Team: TeamId==0,TeamName=NoTeam



// 2.5. gamersListJoinTeamsListSqlLikeQuery ---------------------------------------
//InnerJoin
//GamersList InnerJoin teamsList by Sql Like Query
//2.5.1.
//Each Team have several Gamers.
//Each Gamer can only have one Team.
//This is One to Many Relationship.
//2.5.2.
//"Gamer" collection InnerJoin "Team" collection into one collection.
//Thus, it need only one foreach to loop all elements in collection.
//2.5.3.
//There are 3 Teams, and Team with Id==3 has no Gamer.
//There are 6 Gamers, and Gamer with Id==6 has no Team.
//InnerJoin will only take matching elements from both two collections.
//Thus, Gamer with Id==6 ,which has no team, will not be returned.
//It will return only 5 matching "Gamers" who has a "Team".
Console.WriteLine("2.5. gamersListJoinTeamsListSqlLikeQuery ------------- ");
var gamersListJoinTeamsListSqlLikeQuery =
    from g in gamersList
    join t in teamsList
    on g.TeamId equals t.Id
    select new
    {
        Team = t,
        Gamer = g
    };
```

```
            Console.WriteLine($"gamersListJoinTeamsListSqlLikeQuery.Count()=={gamersListJoinTeamsListSqlLi
keQuery.Count()}");
            foreach (var gamersListJoinTeamsListSqlLikeQueryItem in gamersListJoinTeamsListSqlLikeQuery)
            {
                //Look the Type
                var item = gamersListJoinTeamsListSqlLikeQueryItem;
                Gamer itemGamer = gamersListJoinTeamsListSqlLikeQueryItem.Gamer;
                Team itemTeam = gamersListJoinTeamsListSqlLikeQueryItem.Team;
                //Print Gamer and Team
                Console.WriteLine($"Gamer: {itemGamer}  ;  " +
                                  $"Team: {itemTeam}");
            }
            // gamersListJoinTeamsListSqlLikeQuery.Count()==5
            // Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==2,TeamName=Team2


           // 2.6. gamersListJoinTeamsList -------------
            //InnerJoin
            //GamersList InnerJoin teamsList by Lambda expression query.
            //2.6.1.
            //Each Team have several Gamers.
            //Each Gamer can only have one Team.
            //This is One to Many Relationship.
            //2.6.2.
            //"Gamer" collection InnerJoin "Team" collection into one collection.
            //Thus, it need only one foreach to loop all elements in collection.
            //2.6.3.
            //There are 3 Teams, and Team with Id==3 has no Gamer.
            //There are 6 Gamers, and Gamer with Id==6 has no Team.
            //InnerJoin will only take matching elements from both two collections.
            //Thus, Gamer with Id==6 ,which has no team, will not be returned.
            //It will return only 5 matching "Gamers" who has a "Team".
            Console.WriteLine("2.6. gamersListJoinTeamsList ------------- ");
            var gamersListJoinTeamsList =
                gamersList.Join(
                    teamsList,
                    g => g.TeamId,
                    t => t.Id,
                    (gamer, team) => new
                    {
                        Gamer = gamer,
                        Team = team
                    });
            Console.WriteLine($"gamersListJoinTeamsList.Count()=={gamersListJoinTeamsList.Count()}");
            foreach (var gamersListJoinTeamsListItem in gamersListJoinTeamsList)
            {
                //Look the Type
                var item = gamersListJoinTeamsListItem;
                Gamer itemGamer = gamersListJoinTeamsListItem.Gamer;
                Team itemTeam = gamersListJoinTeamsListItem.Team;
                //Print Gamer and Team
```

```
            Console.WriteLine($"Gamer: {itemGamer}  ;  " +
                              $"Team: {itemTeam}");
        }
        // gamersListJoinTeamsList.Count()==5
        // Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
        // Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==2,TeamName=Team2
        // Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
        // Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
        // Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==2,TeamName=Team2
    }



    //3. ==================================
    //GamersListCrossJoinTeamsList(teamsList, gamersList)
    static void GamersListCrossJoinTeamsList(List<Team> teamsList, List<Gamer> gamersList)
    {
        // 3.1. teamsListCrossJoinGamersList -------------------------------
        //CrossJoin
        //TeamsList CrossJoin GamersList by Sql like query.
        //3.1.1.
        //Each Team have several Gamers.
        //Each Gamer can only have one Team.
        //This is One to Many Relationship.
        //3.1.2.
        //CrossJoin
        ////from ... from ...Select...
        //or
        ////SelectMany(...)
        //Returns Cartesian product of two collections
        //involved in the join
        //CrossJoin does not need ON keyword.
        //3.1.3.
        //"Team" collection CrossJoin "Gamer" collection into one collection.
        //Thus, it need only one foreach to loop all elements in collection.
        //3.1.4.
        //There are 3 Teams,
        //There are 6 Gamers,
        //CrossJoin will return 3*6=18 elements.
        Console.WriteLine("3.1. teamsListCrossJoinGamersList ------------- ");
        var teamsListCrossJoinGamersList =
            from t in teamsList
            from g in gamersList
            select new { Gamer = g, Team = t };
        Console.WriteLine($"teamsListCrossJoinGamersList.Count()=={teamsListCrossJoinGamersList.Count(
)}");
        foreach (var teamsListCrossJoinGamersListItem in teamsListCrossJoinGamersList)
        {
            //Look the Type
            var item = teamsListCrossJoinGamersListItem;
            Gamer itemGamer = teamsListCrossJoinGamersListItem.Gamer;
            Team itemTeam = teamsListCrossJoinGamersListItem.Team;
            //Print Gamer and Team
            Console.WriteLine($"Gamer: {itemGamer}  ;  " +
```

```
                    $"Team: {itemTeam}");
}
// teamsListCrossJoinGamersList.Count()==18
// Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
// Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==1,TeamName=Team1
// Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
// Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
// Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==1,TeamName=Team1
// Gamer: GamerId==6,GamerName=Name10,TeamId=0  ;  Team: TeamId==1,TeamName=Team1
// Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==2,TeamName=Team2
// Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==2,TeamName=Team2
// Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==2,TeamName=Team2
// Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==2,TeamName=Team2
// Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==2,TeamName=Team2
// Gamer: GamerId==6,GamerName=Name10,TeamId=0  ;  Team: TeamId==2,TeamName=Team2
// Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==3,TeamName=Team3
// Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==3,TeamName=Team3
// Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==3,TeamName=Team3
// Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==3,TeamName=Team3
// Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==3,TeamName=Team3
// Gamer: GamerId==6,GamerName=Name10,TeamId=0  ;  Team: TeamId==3,TeamName=Team3



// 3.2. GamersListCrossJoinTeamsList --------------------------------
//CrossJoin
//GamersList CrossJoin TeamsList by Lambda expression query.
//3.2.1.
//Each Team have several Gamers.
//Each Gamer can only have one Team.
//This is One to Many Relationship.
//3.2.2.
//CrossJoin
////from ... from ...Select...
//or
////SelectMany(...)
//Returns Cartesian product of two collections
//involved in the join
//CrossJoin does not need ON keyword.
//3.2.3.
//"Gamer" collection CrossJoin "Team" collection into one collection.
//Thus, it need only one foreach to loop all elements in collection.
//3.2.4.
//There are 3 Teams,
//There are 6 Gamers,
//CrossJoin will return 3*6=18 elements.
Console.WriteLine("3.2. GamersListCrossJoinTeamsList ------------- ");
var GamersListCrossJoinTeamsList = gamersList.SelectMany(
                g => teamsList,
                (g, t) => new
                {
                    Gamer = g,
                    Team = t
                   //"g" parameter means each item of gamersList
```

```
                                //"t" parameter mean each item of teamsList
                            }
                        );
            Console.WriteLine($"GamersListCrossJoinTeamsList.Count()=={GamersListCrossJoinTeamsList.Count(
)}");
            foreach (var GamersListCrossJoinTeamsListItem in GamersListCrossJoinTeamsList)
            {
                //Look the Type
                var item = GamersListCrossJoinTeamsListItem;
                Gamer itemGamer = GamersListCrossJoinTeamsListItem.Gamer;
                Team itemTeam = GamersListCrossJoinTeamsListItem.Team;
                //Print Gamer and Team
                Console.WriteLine($"Gamer: {itemGamer}  ;  " +
                                  $"Team: {itemTeam}");
            }
            // GamersListCrossJoinTeamsList.Count()==18
            // Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==1,GamerName=Name1,TeamId=1  ;  Team: TeamId==3,TeamName=Team3
            // Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==2,GamerName=Name2,TeamId=2  ;  Team: TeamId==3,TeamName=Team3
            // Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==3,GamerName=Name3,TeamId=1  ;  Team: TeamId==3,TeamName=Team3
            // Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==4,GamerName=Name4,TeamId=1  ;  Team: TeamId==3,TeamName=Team3
            // Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==5,GamerName=Name9,TeamId=2  ;  Team: TeamId==3,TeamName=Team3
            // Gamer: GamerId==6,GamerName=Name10,TeamId=0  ;  Team: TeamId==1,TeamName=Team1
            // Gamer: GamerId==6,GamerName=Name10,TeamId=0  ;  Team: TeamId==2,TeamName=Team2
            // Gamer: GamerId==6,GamerName=Name10,TeamId=0  ;  Team: TeamId==3,TeamName=Team3
        }
    }
}

namespace OnLieGame
{
    public class Team
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public override string ToString()
        {
            return $"TeamId=={Id},TeamName={Name}";
        }
    }
    public class TeamHelper
    {
        public static List<Team> GetSampleTeam()
        {
            return new List<Team>
            {
                new Team { Id = 1, Name = "Team1"},
```

```
                new Team { Id = 2, Name = "Team2"},
                new Team { Id = 3, Name = "Team3"},
            };
        }
    }
    public class Gamer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int TeamId { get; set; }
        public override string ToString()
        {
            return $"GamerId=={Id},GamerName={Name},TeamId={TeamId}";
        }
    }
    public class GamerHelper
    {
        public static List<Gamer> GetSampleGamer()
        {
            return new List<Gamer>
            {
                new Gamer { Id = 1, Name = "Name1", TeamId = 1 },
                new Gamer { Id = 2, Name = "Name2", TeamId = 2 },
                new Gamer { Id = 3, Name = "Name3", TeamId = 1 },
                new Gamer { Id = 4, Name = "Name4", TeamId = 1 },
                new Gamer { Id = 5, Name = "Name9", TeamId = 2 },
                new Gamer { Id = 6, Name = "Name10"}
            };
        }
    }
}
```