(T5)比較LinqToObject的Select、SelectMany  
CourseGUID: 5ba9a6fe-7475-4b0c-8b99-bbcf7f5e2e1c  
=======================================================================  
(T5)比較LinqToObject的Select、SelectMany  
=======================================================================  
0. Summary

-----------

1. New Project

1.1. Create New Project : Sample

-----------

2. Sample : Program.cs  
=======================================================================

0. Summary

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](https://msdn.microsoft.com/en-us/library/bb548891%28v=vs.110%29.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](https://msdn.microsoft.com/en-us/library/bb534336%28v=vs.110%29.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](https://msdn.microsoft.com/en-us/library/bb534631%28v=vs.110%29.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.

When using Sql like query which has 2 from clause,

the second from clause will use the result set

from the first from  clause as its source.

=============================================

1. New Project

1.1. Create New Project : Sample

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

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

Name: **Sample**

Graphical user interface, application, email

Description automatically generated



Graphical user interface, text, application, Excel

Description automatically generated

=============================================

2. Sample : Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using OnlineGame;

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

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

            //Select()

            Console.WriteLine("1. GetGamersId(); =========================== ");

            GetGamersId();

            //2. =======================================

            //Select()

            Console.WriteLine("2. GetGamersIdNameGender(); =========================== ");

            GetGamersIdNameGender();

            //3. =======================================

            //Select()

            Console.WriteLine("3. GetGamerScoreGreaterThan5k(); =========================== ");

            GetGamerScoreGreaterThan5k();

            //4. =======================================

            //SelectMany()

            Console.WriteLine("4. GetAllSkills(); =========================== ");

            GetAllSkills();

            //5. =======================================

            //SelectMany()

            Console.WriteLine("5. GetAllSkillsSqlLikeQuery(); =========================== ");

            GetAllSkillsSqlLikeQuery();

            //6. =======================================

            //SelectMany()

            Console.WriteLine("6. StrToCharEnumerable(); =========================== ");

            StrToCharEnumerable();

            //7. =======================================

            //SelectMany()

            Console.WriteLine("7. StrToCharEnumerableSqlLikeQuery(); =========================== ");

            StrToCharEnumerableSqlLikeQuery();

            //8. =======================================

            //SelectMany()

            Console.WriteLine("8. GetDistinctSkills(); =========================== ");

            GetDistinctSkills();

            //9. =======================================

            //SelectMany()

            Console.WriteLine("9. GetDistinctSkillsSqlLikeQuery(); =========================== ");

            GetDistinctSkillsSqlLikeQuery();

            //10. =======================================

            //Selects gamer name along with skils

            Console.WriteLine("10. GetGamerNameAndSkills(); =========================== ");

            GetGamerNameAndSkills();

            //11. =======================================

            //Selects gamer name along with skils

            Console.WriteLine("11. GetGamerNameAndSkillsSqlLikeQuery(); =========================== ");

            GetGamerNameAndSkillsSqlLikeQuery();

            //12. =======================================

            //Select() V.S. SelectMany()

            Console.WriteLine("12. GetGamerAndSkillsBySelect(); =========================== ");

            GetGamerAndSkillsBySelect();

            //13. =======================================

            //Select() V.S. SelectMany()

            Console.WriteLine("13. GetGamerAndSkillsBySelectMany(); =========================== ");

            GetGamerAndSkillsBySelectMany();

            Console.ReadLine();

        }

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

        //Select()

        static void GetGamersId()

        {

            ////Error

            //var gamerIds2 =

            //    GamerHelper.GetSampleGamers().SelectMany(g => g.Id);

            IEnumerable<int> gamerIds =

                GamerHelper.GetSampleGamers().Select(g => g.Id);

            foreach (int id in gamerIds)

            {

                Console.WriteLine(id);

            }

        }

        //1

        //2

        //3

        //4

        //2. =======================================

        //Select()

        static void GetGamersIdNameGender()

        {

            ////Error

            //var anonymousTypes2 = GamerHelper.GetSampleGamers()

            //    .SelectMany(g => new

            //    {

            //        Id = g.Id,

            //        Name = g.Name,

            //        Gender = g.Gender

            //    });

            var anonymousTypes = GamerHelper.GetSampleGamers()

                .Select(g => new

                {

                    Id = g.Id,

                    Name = g.Name,

                    Gender = g.Gender

                });

            foreach (var anonymousTypesItem in anonymousTypes)

            {

                Console.WriteLine($"Id=={anonymousTypesItem.Id}, " +

                                  $"Name=={anonymousTypesItem.Name}, " +

                                  $"Gender=={anonymousTypesItem.Gender}");

            }

        }

        //Id==1, Name==Name01, Gender==Male

        //Id==2, Name==Name02, Gender==Male

        //Id==3, Name==Name03, Gender==Female

        //Id==4, Name==Name04, Gender==Male

        //3. =======================================

        //Select()

        static void GetGamerScoreGreaterThan5k()

        {

            ////Error

            //var anonymousTypes2 = GamerHelper.GetSampleGamers()

            //    .Where(g => g.Score >= 5000)

            //    .SelectMany(g => new

            //    {

            //        NameAndGender = $"{g.Name}, {g.Gender}",

            //        Score = g.Score

            //    });

            var anonymousTypes = GamerHelper.GetSampleGamers()

                .Where(g => g.Score >= 5000)

                .Select(g => new

                {

                    NameAndGender = $"{g.Name}, {g.Gender}",

                    Score = g.Score

                });

            foreach (var anonymousTypesItem in anonymousTypes)

            {

                Console.WriteLine($"NameAndGender=={anonymousTypesItem.NameAndGender}, \n\r" +

                                  $"Score=={anonymousTypesItem.Score}");

            }

        }

        //NameAndGender==Name01, Male,

        //Score==6000

        //NameAndGender==Name04, Male,

        //Score==8000

        //4. =======================================

        //SelectMany()

        static void GetAllSkills()

        {

            //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](https://msdn.microsoft.com/en-us/library/bb534336%28v=vs.110%29.aspx)

            //Projects each element of a sequence to an IEnumerable<T>

            //and flattens the resulting sequences into one sequence.

            IEnumerable<string> allSkills = GamerHelper.GetSampleGamers()

                .SelectMany(g => g.Skills);

            foreach (string allSkillsItem in allSkills)

            {

                Console.WriteLine($"allSkillsItem=={allSkillsItem}");

            }

        }

        //allSkillsItem==SkilA

        //allSkillsItem==SkillB

        //allSkillsItem==SkillC

        //allSkillsItem==SkilA

        //allSkillsItem==SkillD

        //allSkillsItem==SkilC

        //allSkillsItem==SkillE

        //allSkillsItem==SkilA

        //allSkillsItem==SkillB

        //allSkillsItem==SkillC

        //allSkillsItem==SkillD

        //5. =======================================

        //SelectMany()

        static void GetAllSkillsSqlLikeQuery()

        {

            //2.

            //When using Sql like query which has 2 from clause,

            //the second from clause will use the result set

            //from the first from clause as its source.

            IEnumerable<string> allSkills = from gamer in GamerHelper.GetSampleGamers()

                                            from skills in gamer.Skills

                                            select skills;

            foreach (string allSkillsItem in allSkills)

            {

                Console.WriteLine($"allSkillsItem=={allSkillsItem}");

            }

        }

        //allSkillsItem==SkilA

        //allSkillsItem==SkillB

        //allSkillsItem==SkillC

        //allSkillsItem==SkilA

        //allSkillsItem==SkillD

        //allSkillsItem==SkilC

        //allSkillsItem==SkillE

        //allSkillsItem==SkilA

        //allSkillsItem==SkillB

        //allSkillsItem==SkillC

        //allSkillsItem==SkillD

        //6. =======================================

        //SelectMany()

        private static void StrToCharEnumerable()

        {

            //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](https://msdn.microsoft.com/en-us/library/bb534336%28v=vs.110%29.aspx)

            //Projects each element of a sequence to an IEnumerable<T>

            //and flattens the resulting sequences into one sequence.

            string[] strArr =

            {

                "123",

                "456",

                "7890"

            };

            IEnumerable<char> charEnumerable = strArr.SelectMany(s => s);

            foreach (char charEnumerableItem in charEnumerable)

            {

                Console.Write($"[ {charEnumerableItem} ]  ");

            }

            Console.WriteLine($"\r\ncharEnumerable.Count()=={charEnumerable.Count()}");

        }

        //[ 1 ]  [ 2 ]  [ 3 ]  [ 4 ]  [ 5 ]  [ 6 ]  [ 7 ]  [ 8 ]  [ 9 ]  [ 0 ]

        //charEnumerable.Count()==10

        //7. =======================================

        //SelectMany()

        static void StrToCharEnumerableSqlLikeQuery()

        {

            //2.

            //When using Sql like query which has 2 from clause,

            //the second from clause will use the result set

            //from the first from clause as its source.

            string[] strArr =

            {

                "123",

                "456",

                "7890"

            };

            //IEnumerable<char> charEnumerable = strArr.SelectMany(s => s);

            IEnumerable<char> charEnumerable =

                from strArrItem in strArr

                from charItem in strArrItem

                select charItem;

            foreach (char charEnumerableItem in charEnumerable)

            {

                Console.Write($"[ {charEnumerableItem} ]  ");

            }

            Console.WriteLine($"\r\ncharEnumerable.Count()=={charEnumerable.Count()}");

        }

        //[ 1 ]  [ 2 ]  [ 3 ]  [ 4 ]  [ 5 ]  [ 6 ]  [ 7 ]  [ 8 ]  [ 9 ]  [ 0 ]

        //charEnumerable.Count()==10

        //8. =======================================

        //SelectMany()

        static void GetDistinctSkills()

        {

            //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](https://msdn.microsoft.com/en-us/library/bb534336%28v=vs.110%29.aspx)

            //Projects each element of a sequence to an IEnumerable<T>

            //and flattens the resulting sequences into one sequence.

            IEnumerable<string> allSkills =

                GamerHelper.GetSampleGamers()

                .SelectMany(s => s.Skills).Distinct();

            foreach (string allSkillsItem in allSkills)

            {

                Console.WriteLine($"allSkillsItem=={allSkillsItem}");

            }

        }

        //allSkillsItem==SkilA

        //allSkillsItem==SkillB

        //allSkillsItem==SkillC

        //allSkillsItem==SkillD

        //allSkillsItem==SkilC

        //allSkillsItem==SkillE

        //9. =======================================

        //SelectMany()

        static void GetDistinctSkillsSqlLikeQuery()

        {

            //2.

            //When using Sql like query which has 2 from clause,

            //the second from clause will use the result set

            //from the first from clause as its source.

            IEnumerable<string> allSkills =

                (from gamer in GamerHelper.GetSampleGamers()

                 from skills in gamer.Skills

                 select skills).Distinct();

            foreach (string allSkillsItem in allSkills)

            {

                Console.WriteLine($"allSkillsItem=={allSkillsItem}");

            }

        }

        //allSkillsItem==SkilA

        //allSkillsItem==SkillB

        //allSkillsItem==SkillC

        //allSkillsItem==SkillD

        //allSkillsItem==SkilC

        //allSkillsItem==SkillE

        //10. =======================================

        //Selects gamer name along with skils

        private static void GetGamerNameAndSkills()

        {

            //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](https://msdn.microsoft.com/en-us/library/bb534631%28v=vs.110%29.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.

            //10.1. ---------------------------------------------

            ////Error!!

            //var gamerNameAlongWithSkills2 = GamerHelper.GetSampleGamers()

            //    .SelectMany(

            //        (gamer, skill) => new { GamerName = gamer.Name, Skill = skill });

            //10.2. ---------------------------------------------

            //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, it 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.

            //And then mapping each of those to anonymous type properties.

            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}");

            }

        }

        //gamerNameAlongWithSkills.Count()==11

        //GamerName==Name01, Skill==SkilA

        //GamerName==Name01, Skill==SkillB

        //GamerName==Name01, Skill==SkillC

        //GamerName==Name02, Skill==SkilA

        //GamerName==Name02, Skill==SkillD

        //GamerName==Name03, Skill==SkilC

        //GamerName==Name03, Skill==SkillE

        //GamerName==Name04, Skill==SkilA

        //GamerName==Name04, Skill==SkillB

        //GamerName==Name04, Skill==SkillC

        //GamerName==Name04, Skill==SkillD

        //11. =======================================

        //Selects gamer name along with skils

        static void GetGamerNameAndSkillsSqlLikeQuery()

        {

            var gamerNameAlongWithSkills = from gamer in GamerHelper.GetSampleGamers()

                                           from skill in gamer.Skills

                                           select new { GamerName = gamer.Name, Skill = skill };

            foreach (var gamerNameAlongWithSkillsItem in gamerNameAlongWithSkills)

            {

                Console.WriteLine($"GamerName=={gamerNameAlongWithSkillsItem.GamerName}, " +

                                  $"Skill=={gamerNameAlongWithSkillsItem.Skill}");

            }

        }

        //GamerName==Name01, Skill==SkilA

        //GamerName==Name01, Skill==SkillB

        //GamerName==Name01, Skill==SkillC

        //GamerName==Name02, Skill==SkilA

        //GamerName==Name02, Skill==SkillD

        //GamerName==Name03, Skill==SkilC

        //GamerName==Name03, Skill==SkillE

        //GamerName==Name04, Skill==SkilA

        //GamerName==Name04, Skill==SkillB

        //GamerName==Name04, Skill==SkillC

        //GamerName==Name04, Skill==SkillD

       //12. =======================================

        //Select() V.S. SelectMany()

        static void GetGamerAndSkillsBySelect()

        {

            IEnumerable<List<string>> allGamers = GamerHelper.GetSampleGamers().Select(g => g.Skills);

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

            foreach (List<string> skills in allGamers)

            {

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

                foreach (string skillsItem in skills)

                {

                    Console.WriteLine(skillsItem);

                }

            }

        }

        //allGamers.Count():4

        //skills.Count():3

        //SkilA

        //SkillB

        //SkillC

        //skills.Count():2

        //SkilA

        //SkillD

        //skills.Count():2

        //SkilC

        //SkillE

        //skills.Count():4

        //SkilA

        //SkillB

        //SkillC

        //SkillD

        //13. =======================================

        //Select() V.S. SelectMany()

        private static void GetGamerAndSkillsBySelectMany()

        {

            IEnumerable<string> skills = GamerHelper.GetSampleGamers().SelectMany(g => g.Skills);

            Console.WriteLine($"skills.Count()=={skills.Count()}");

            foreach (string skillsItem in skills)

            {

                Console.WriteLine(skillsItem);

            }

        }

        //skills.Count()==11

        //SkilA

        //SkillB

        //SkillC

        //SkilA

        //SkillD

        //SkilC

        //SkillE

        //SkilA

        //SkillB

        //SkillC

        //SkillD

    }

}

namespace OnlineGame

{

    public class Gamer

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public string Gender { get; set; }

        public int Score { get; set; }

        public List<string> Skills { get; set; }

    }

    public class GamerHelper

    {

        public static List<Gamer> GetSampleGamers()

        {

            return new List<Gamer>

            {

                new Gamer{Id=1,Name="Name01",Gender="Male", Score =6000,

                    Skills = new List<string>{"SkilA", "SkillB", "SkillC"}},

                new Gamer{Id=2,Name="Name02",Gender="Male", Score =3000,

                    Skills = new List<string>{"SkilA", "SkillD"}},

                new Gamer{Id=3,Name="Name03",Gender="Female", Score =4500,

                    Skills = new List<string>{"SkilC", "SkillE"}},

                new Gamer{Id=4,Name="Name04",Gender="Male", Score =8000,

                    Skills = new List<string>{"SkilA", "SkillB", "SkillC", "SkillD"}},

            };

        }

    }

}

/\*

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](https://msdn.microsoft.com/en-us/library/bb548891%28v=vs.110%29.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](https://msdn.microsoft.com/en-us/library/bb534336%28v=vs.110%29.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](https://msdn.microsoft.com/en-us/library/bb534631%28v=vs.110%29.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.

-------------------------------

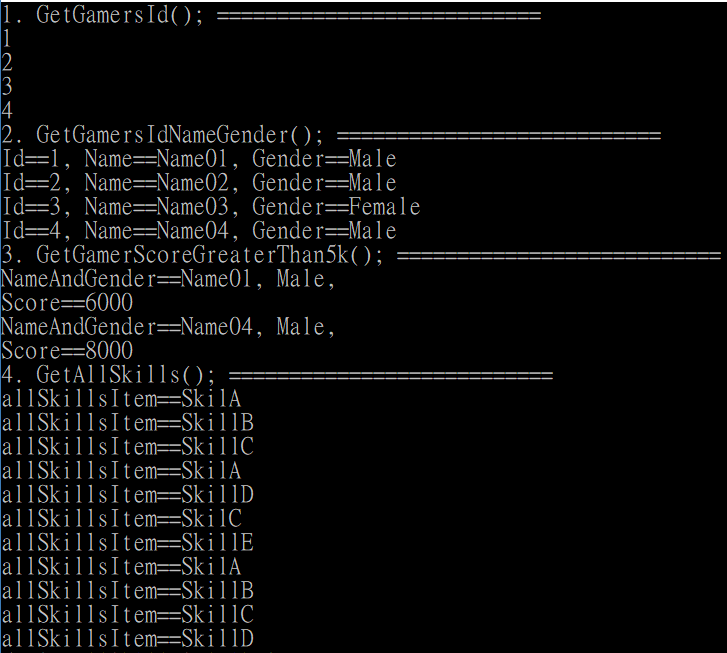
2.

When using Sql like query which has 2 from clause,

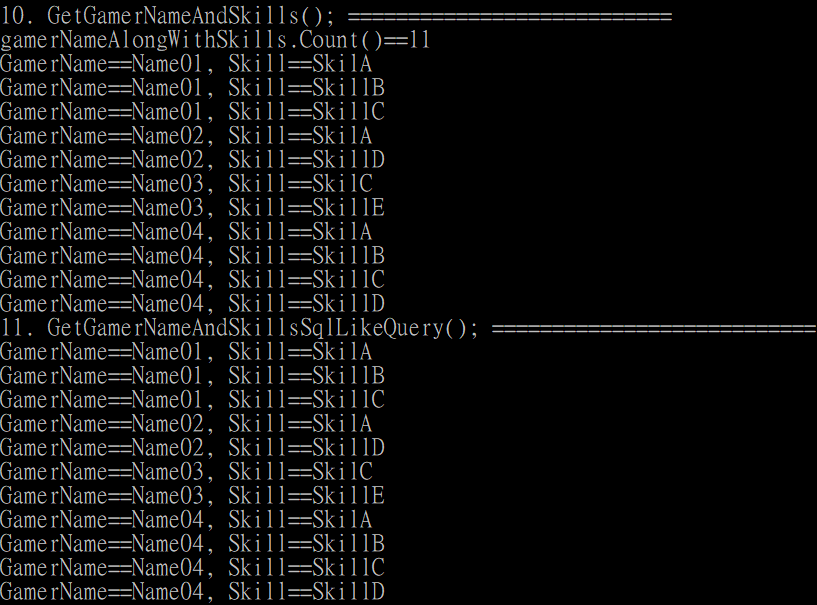
the second from clause will use the result set

from the first from  clause as its source.

\*/







Text

Description automatically generated