**执行内部联接（C# 编程指南）**

按照关系数据库的说法，“内部联接”产生一个结果集，对于该结果集内第一个集合中的每个元素，只要在第二个集合中存在一个匹配元素，该元素就会出现一次。如果第一个集合中的某个元素没有匹配元素，则它不会出现在结果集内。 [Join](https://msdn.microsoft.com/zh-cn/library/system.linq.enumerable.join.aspx) 方法（通过 C# 中的 **join** 子句调用）可实现内联。

本主题演示如何执行内部联接的四种变体：

* 简单的内部联接，它基于一个简单的键将来自两个数据源的元素相互关联。
* 内部联接，它基于一个复合键将来自两个数据源的元素相互关联。使用复合键（即由多个值组成的键）可以基于多个属性将元素相互关联。
* 多联接，在其中连续的联接操作被相互拼接在一起。
* 通过使用分组联接实现的内部联接。

**简单键联接示例**

下面的示例创建了两个集合，其中分别包含以下两个用户定义类型的对象：Person 和 Pet。查询使用 C# 中的 **join** 子句来将 Person 对象与 Pet 对象（其 Owner 为该 Person）进行匹配。C# 中的 **select** 子句可定义生成的对象的外观。在此示例中，生成的对象是由主人的名字和宠物的名字组成的匿名类型。

class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

class Pet

{

public string Name { get; set; }

public Person Owner { get; set; }

}

/// <summary>

/// Simple inner join.

/// </summary>

public static void InnerJoinExample()

{

Person magnus = new Person { FirstName = "Magnus", LastName = "Hedlund" };

Person terry = new Person { FirstName = "Terry", LastName = "Adams" };

Person charlotte = new Person { FirstName = "Charlotte", LastName = "Weiss" };

Person arlene = new Person { FirstName = "Arlene", LastName = "Huff" };

Person rui = new Person { FirstName = "Rui", LastName = "Raposo" };

Pet barley = new Pet { Name = "Barley", Owner = terry };

Pet boots = new Pet { Name = "Boots", Owner = terry };

Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };

Pet bluemoon = new Pet { Name = "Blue Moon", Owner = rui };

Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

// Create two lists.

List<Person> people = new List<Person> { magnus, terry, charlotte, arlene, rui };

List<Pet> pets = new List<Pet> { barley, boots, whiskers, bluemoon, daisy };

// Create a collection of person-pet pairs. Each element in the collection

// is an anonymous type containing both the person's name and their pet's name.

var query = from person in people

join pet in pets on person equals pet.Owner

select new { OwnerName = person.FirstName, PetName = pet.Name };

foreach (var ownerAndPet in query)

{

Console.WriteLine("\"{0}\" is owned by {1}", ownerAndPet.PetName, ownerAndPet.OwnerName);

}

}

// This code produces the following output:

//

// "Daisy" is owned by Magnus

// "Barley" is owned by Terry

// "Boots" is owned by Terry

// "Whiskers" is owned by Charlotte

// "Blue Moon" is owned by Rui

请注意，其 LastName 为“Huff”的 Person 对象未出现在结果集内，因为不存在 Pet.Owner 等于该 Person 的 Pet 对象。

**复合键联接示例**

与仅仅基于一个属性将元素相互关联不同，使用复合键可基于多个属性来比较元素。为此，需要为每个集合指定键选择器函数，以便返回一个由要比较的属性组成的匿名类型。如果给属性加上了标签，则这些属性必须在每个键的匿名类型中都有相同的标签，而且还必须以相同顺序出现。

下面的示例使用一个 Employee 对象列表和一个 Student 对象列表来确定哪些雇员同时还是学生。这两个类型都具有 [String](https://msdn.microsoft.com/zh-cn/library/system.string.aspx) 类型的 FirstName 和 LastName 属性。能够从每个列表的元素创建联接键的函数可返回一个由每个元素的 FirstName 和 LastName 属性组成的匿名类型。联接操作比较这些复合键是否相等，并且从每个列表中返回名字和姓氏都匹配的对象对。

class Employee

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int EmployeeID { get; set; }

}

class Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int StudentID { get; set; }

}

/// <summary>

/// Performs a join operation using a composite key.

/// </summary>

public static void CompositeKeyJoinExample()

{

// Create a list of employees.

List<Employee> employees = new List<Employee> {

new Employee { FirstName = "Terry", LastName = "Adams", EmployeeID = 522459 },

new Employee { FirstName = "Charlotte", LastName = "Weiss", EmployeeID = 204467 },

new Employee { FirstName = "Magnus", LastName = "Hedland", EmployeeID = 866200 },

new Employee { FirstName = "Vernette", LastName = "Price", EmployeeID = 437139 } };

// Create a list of students.

List<Student> students = new List<Student> {

new Student { FirstName = "Vernette", LastName = "Price", StudentID = 9562 },

new Student { FirstName = "Terry", LastName = "Earls", StudentID = 9870 },

new Student { FirstName = "Terry", LastName = "Adams", StudentID = 9913 } };

// Join the two data sources based on a composite key consisting of first and last name,

// to determine which employees are also students.

IEnumerable<string> query = from employee in employees

join student in students

on new { employee.FirstName, employee.LastName }

equals new { student.FirstName, student.LastName }

select employee.FirstName + " " + employee.LastName;

Console.WriteLine("The following people are both employees and students:");

foreach (string name in query)

Console.WriteLine(name);

}

// This code produces the following output:

//

// The following people are both employees and students:

// Terry Adams

// Vernette Price

**多联接示例**

可以将任意数量的联接操作拼接在一起以执行多联接。C# 中的每一个 **join** 子句都可将指定的数据源与前一个联接的结果相互关联。

下面的示例创建了三个集合：一个 Person 对象列表、一个 Cat 对象列表以及一个 Dog 对象列表。

C# 中的第一个 **join** 子句将基于匹配 Cat.Owner 的 Person 对象对主人和猫进行匹配。并返回包含 Person 对象和 Cat.Name 的匿名类型的序列。

C# 中的第二个 **join** 子句基于一个组合键将第一个联接返回的匿名类型与所提供的犬列表中的 Dog 对象相互关联，该组合键由类型为 Person 的 Owner 属性和动物名字的首字母组成。该子句返回一个匿名类型序列，这些类型包含每个匹配对中的 Cat.Name 和 Dog.Name 属性。由于这是一个内部联接，因此仅返回第一个数据源中那些在第二个数据源中具有匹配对象的对象。

class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

class Pet

{

public string Name { get; set; }

public Person Owner { get; set; }

}

class Cat : Pet

{ }

class Dog : Pet

{ }

public static void MultipleJoinExample()

{

Person magnus = new Person { FirstName = "Magnus", LastName = "Hedlund" };

Person terry = new Person { FirstName = "Terry", LastName = "Adams" };

Person charlotte = new Person { FirstName = "Charlotte", LastName = "Weiss" };

Person arlene = new Person { FirstName = "Arlene", LastName = "Huff" };

Person rui = new Person { FirstName = "Rui", LastName = "Raposo" };

Person phyllis = new Person { FirstName = "Phyllis", LastName = "Harris" };

Cat barley = new Cat { Name = "Barley", Owner = terry };

Cat boots = new Cat { Name = "Boots", Owner = terry };

Cat whiskers = new Cat { Name = "Whiskers", Owner = charlotte };

Cat bluemoon = new Cat { Name = "Blue Moon", Owner = rui };

Cat daisy = new Cat { Name = "Daisy", Owner = magnus };

Dog fourwheeldrive = new Dog { Name = "Four Wheel Drive", Owner = phyllis };

Dog duke = new Dog { Name = "Duke", Owner = magnus };

Dog denim = new Dog { Name = "Denim", Owner = terry };

Dog wiley = new Dog { Name = "Wiley", Owner = charlotte };

Dog snoopy = new Dog { Name = "Snoopy", Owner = rui };

Dog snickers = new Dog { Name = "Snickers", Owner = arlene };

// Create three lists.

List<Person> people =

new List<Person> { magnus, terry, charlotte, arlene, rui, phyllis };

List<Cat> cats =

new List<Cat> { barley, boots, whiskers, bluemoon, daisy };

List<Dog> dogs =

new List<Dog> { fourwheeldrive, duke, denim, wiley, snoopy, snickers };

// The first join matches Person and Cat.Owner from the list of people and

// cats, based on a common Person. The second join matches dogs whose names start

// with the same letter as the cats that have the same owner.

var query = from person in people

join cat in cats on person equals cat.Owner

join dog in dogs on

new { Owner = person, Letter = cat.Name.Substring(0, 1) }

equals new { dog.Owner, Letter = dog.Name.Substring(0, 1) }

select new { CatName = cat.Name, DogName = dog.Name };

foreach (var obj in query)

{

Console.WriteLine(

"The cat \"{0}\" shares a house, and the first letter of their name, with \"{1}\".",

obj.CatName, obj.DogName);

}

}

// This code produces the following output:

//

// The cat "Daisy" shares a house, and the first letter of their name, with "Duke".

// The cat "Whiskers" shares a house, and the first letter of their name, with "Wiley".

**使用分组联接实现内部联接的示例**

下面的示例演示如何使用分组联接来实现内部联接。

在 query1 中，Person 对象列表基于与 Pet.Owner 属性匹配的 Person 分组联接到 Pet 对象列表。分组联接创建了一个中间组集合，该集合中的每个组都由一个 Person 对象和匹配的 Pet 对象序列组成。

通过向查询中添加另一个 **from** 子句，此序列的序列被组合（或展平）为一个较长的序列。最终序列的元素类型由 **select** 子句指定。在此示例中，该类型是由每个匹配对的 Person.FirstName 和 Pet.Name 属性组成的匿名类型。

query1 的结果等效于使用 **join** 子句所获得的结果集，而没有 **into** 子句执行内联。 query2 变量演示了这一等效查询。

class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

class Pet

{

public string Name { get; set; }

public Person Owner { get; set; }

}

/// <summary>

/// Performs an inner join by using GroupJoin().

/// </summary>

public static void InnerGroupJoinExample()

{

Person magnus = new Person { FirstName = "Magnus", LastName = "Hedlund" };

Person terry = new Person { FirstName = "Terry", LastName = "Adams" };

Person charlotte = new Person { FirstName = "Charlotte", LastName = "Weiss" };

Person arlene = new Person { FirstName = "Arlene", LastName = "Huff" };

Pet barley = new Pet { Name = "Barley", Owner = terry };

Pet boots = new Pet { Name = "Boots", Owner = terry };

Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };

Pet bluemoon = new Pet { Name = "Blue Moon", Owner = terry };

Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

// Create two lists.

List<Person> people = new List<Person> { magnus, terry, charlotte, arlene };

List<Pet> pets = new List<Pet> { barley, boots, whiskers, bluemoon, daisy };

var query1 = from person in people

join pet in pets on person equals pet.Owner into gj

from subpet in gj

select new { OwnerName = person.FirstName, PetName = subpet.Name };

Console.WriteLine("Inner join using GroupJoin():");

foreach (var v in query1)

{

Console.WriteLine("{0} - {1}", v.OwnerName, v.PetName);

}

var query2 = from person in people

join pet in pets on person equals pet.Owner

select new { OwnerName = person.FirstName, PetName = pet.Name };

Console.WriteLine("\nThe equivalent operation using Join():");

foreach (var v in query2)

Console.WriteLine("{0} - {1}", v.OwnerName, v.PetName);

}

// This code produces the following output:

//

// Inner join using GroupJoin():

// Magnus - Daisy

// Terry - Barley

// Terry - Boots

// Terry - Blue Moon

// Charlotte - Whiskers

//

// The equivalent operation using Join():

// Magnus - Daisy

// Terry - Barley

// Terry - Boots

// Terry - Blue Moon

// Charlotte - Whiskers