**子查询**

描述：查询订单数超过5的顾客信息

查询句法：

|  |
| --- |
| var 子查询 = **from** c in ctx.Customers  **where**                         (**from** o in ctx.Orders **group** o **by** o.CustomerID **into** o **where** o.Count() > 5 **select** o.**Key**).**Contains**(c.CustomerID)  **select** c; |

**in 操作**

描述：查询指定城市中的客户

查询句法：

var in操作 = from c in ctx.Customers

wherenewstring[] { "Brandenburg", "Cowes", "Stavern" }.Contains(c.City)

select c;

**Join**

描述：内连接，没有分类的产品查询不到

查询句法：

|  |
| --- |
| var innerjoin = **from** p in ctx.Products                          join c in ctx.Categories  **on** p.CategoryID equals c.CategoryID  **select** p.ProductName; |

描述：外连接，没有分类的产品也能查询到

查询句法：

|  |
| --- |
| var leftjoin = **from** p in ctx.Products                         join c in ctx.Categories  **on** p.CategoryID equals c.CategoryID  **into** pro  **from** x in pro.DefaultIfEmpty()  **select** p.ProductName; |

|  |
| --- |
| 1.简单形式：  var q =      from p in db.Products      group p by p.CategoryID into g      select g;  语句描述：使用Group By按CategoryID划分产品。   说明：from p in db.Products 表示从表中将产品对象取出来。group p by p.CategoryID into g表示对p按CategoryID字段归类。其结果命名为g，一旦重新命名，p的作用域就结束了，所以，最后select时，只能select g。当然，也不必重新命名可以这样写：   var q =      from p in db.Products      group p by p.CategoryID;  我们用示意图表示：     如果想遍历某类别中所有记录，这样：   foreach (var gp in q)  {      if (gp.Key == 2)      {          foreach (var item in gp)          {              //do something          }      }  }  2.Select匿名类：  var q =      from p in db.Products      group p by p.CategoryID into g      select new { CategoryID = g.Key, g };   说明：在这句LINQ语句中，有2个property：CategoryID和g。这个匿名类，其实质是对返回结果集重新进行了包装。把g的property封装成一个完整的分组。如下图所示：     如果想遍历某匿名类中所有记录，要这么做：   foreach (var gp in q)  {      if (gp.CategoryID == 2)      {          foreach (var item in gp.g)          {              //do something          }      }  }  3.最大值  var q =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,          MaxPrice = g.Max(p => p.UnitPrice)      };  语句描述：使用Group By和Max查找每个CategoryID的最高单价。   说明：先按CategoryID归类，判断各个分类产品中单价最大的Products。取出CategoryID值，并把UnitPrice值赋给MaxPrice。   4.最小值  var q =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,          MinPrice = g.Min(p => p.UnitPrice)      };  语句描述：使用Group By和Min查找每个CategoryID的最低单价。   说明：先按CategoryID归类，判断各个分类产品中单价最小的Products。取出CategoryID值，并把UnitPrice值赋给MinPrice。   5.平均值  var q =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,          AveragePrice = g.Average(p => p.UnitPrice)      };  语句描述：使用Group By和Average得到每个CategoryID的平均单价。   说明：先按CategoryID归类，取出CategoryID值和各个分类产品中单价的平均值。   6.求和  var q =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,          TotalPrice = g.Sum(p => p.UnitPrice)      };  语句描述：使用Group By和Sum得到每个CategoryID 的单价总计。   说明：先按CategoryID归类，取出CategoryID值和各个分类产品中单价的总和。   7.计数  var q =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,          NumProducts = g.Count()      };  语句描述：使用Group By和Count得到每个CategoryID中产品的数量。   说明：先按CategoryID归类，取出CategoryID值和各个分类产品的数量。   8.带条件计数  var q =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,          NumProducts = g.Count(p => p.Discontinued)      };  语句描述：使用Group By和Count得到每个CategoryID中断货产品的数量。   说明：先按CategoryID归类，取出CategoryID值和各个分类产品的断货数量。 Count函数里，使用了Lambda表达式，Lambda表达式中的p，代表这个组里的一个元素或对象，即某一个产品。   9.Where限制  var q =      from p in db.Products      group p by p.CategoryID into g      where g.Count() >= 10      select new {          g.Key,          ProductCount = g.Count()      };  语句描述：根据产品的―ID分组，查询产品数量大于10的ID和产品数量。这个示例在Group By子句后使用Where子句查找所有至少有10种产品的类别。   说明：在翻译成SQL语句时，在最外层嵌套了Where条件。   10.多列(Multiple Columns)  var categories =      from p in db.Products      group p by new      {          p.CategoryID,          p.SupplierID      }          into g          select new              {                  g.Key,                  g              };  语句描述：使用Group By按CategoryID和SupplierID将产品分组。   说明：既按产品的分类，又按供应商分类。在by后面，new出来一个匿名类。这里，Key其实质是一个类的对象，Key包含两个Property：CategoryID、SupplierID。用g.Key.CategoryID可以遍历CategoryID的值。   11.表达式(Expression)  var categories =      from p in db.Products      group p by new { Criterion = p.UnitPrice > 10 } into g      select g;  语句描述：使用Group By返回两个产品序列。第一个序列包含单价大于10的产品。第二个序列包含单价小于或等于10的产品。   说明：按产品单价是否大于10分类。其结果分为两类，大于的是一类，小于及等于为另一类。   Exists/In/Any/All/Contains操作符  适用场景：用于判断集合中元素，进一步缩小范围。   Any  说明：用于判断集合中是否有元素满足某一条件；不延迟。（若条件为空，则集合只要不为空就返回True，否则为False）。有2种形式，分别为简单形式和带条件形式。   1.简单形式：  仅返回没有订单的客户：   var q =      from c in db.Customers      where !c.Orders.Any()      select c;  生成SQL语句为：   SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],  [t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region],  [t0].[PostalCode], [t0].[Country], [t0].[Phone], [t0].[Fax]  FROM [dbo].[Customers] AS [t0]  WHERE NOT (EXISTS(      SELECT NULL AS [EMPTY] FROM [dbo].[Orders] AS [t1]      WHERE [t1].[CustomerID] = [t0].[CustomerID]     ))  2.带条件形式：  仅返回至少有一种产品断货的类别：   var q =      from c in db.Categories      where c.Products.Any(p => p.Discontinued)      select c;  生成SQL语句为：   SELECT [t0].[CategoryID], [t0].[CategoryName], [t0].[Description],  [t0].[Picture] FROM [dbo].[Categories] AS [t0]  WHERE EXISTS(      SELECT NULL AS [EMPTY] FROM [dbo].[Products] AS [t1]      WHERE ([t1].[Discontinued] = 1) AND       ([t1].[CategoryID] = [t0].[CategoryID])      )  All  说明：用于判断集合中所有元素是否都满足某一条件；不延迟   1.带条件形式  var q =      from c in db.Customers      where c.Orders.All(o => o.ShipCity == c.City)      select c;  语句描述：这个例子返回所有订单都运往其所在城市的客户或未下订单的客户。   Contains  说明：用于判断集合中是否包含有某一元素；不延迟。它是对两个序列进行连接操作的。   string[] customerID\_Set =      new string[] { "AROUT", "BOLID", "FISSA" };  var q = (      from o in db.Orders      where customerID\_Set.Contains(o.CustomerID)      select o).ToList();  语句描述：查找"AROUT", "BOLID" 和 "FISSA" 这三个客户的订单。先定义了一个数组，在LINQ to SQL中使用Contains，数组中包含了所有的CustomerID，即返回结果中，所有的CustomerID都在这个集合内。也就是in。 你也可以把数组的定义放在LINQ to SQL语句里。比如：   var q = (      from o in db.Orders      where (      new string[] { "AROUT", "BOLID", "FISSA" })      .Contains(o.CustomerID)      select o).ToList();  Not Contains则取反：   var q = (      from o in db.Orders      where !(      new string[] { "AROUT", "BOLID", "FISSA" })      .Contains(o.CustomerID)      select o).ToList();  1.包含一个对象：  var order = (from o in db.Orders               where o.OrderID == 10248               select o).First();  var q = db.Customers.Where(p => p.Orders.Contains(order)).ToList();  foreach (var cust in q)  {      foreach (var ord in cust.Orders)      {          //do something      }  }  语句描述：这个例子使用Contain查找哪个客户包含OrderID为10248的订单。   2.包含多个值：  string[] cities =       new string[] { "Seattle", "London", "Vancouver", "Paris" };  var q = db.Customers.Where(p=>cities.Contains(p.City)).ToList();  语句描述：这个例子使用Contains查找其所在城市为西雅图、伦敦、巴黎或温哥华的客户。   总结一下这篇我们说明了以下语句：   Group By/Having    分组数据；延迟  Any    用于判断集合中是否有元素满足某一条件；不延迟  All    用于判断集合中所有元素是否都满足某一条件；不延迟  Contains    用于判断集合中是否包含有某一元素；不延迟        适用场景：统计数据吧，比如统计一些数据的个数，求和，最小值，最大值，平均数。   Count  说明：返回集合中的元素个数，返回INT类型；不延迟。生成SQL语句为：SELECT COUNT(\*) FROM   1.简单形式：  得到数据库中客户的数量：   var q = db.Customers.Count();  2.带条件形式：  得到数据库中未断货产品的数量：   var q = db.Products.Count(p => !p.Discontinued);  LongCount  说明：返回集合中的元素个数，返回LONG类型；不延迟。对于元素个数较多的集合可视情况可以选用LongCount来统计元素个数，它返回long类型，比较精确。生成SQL语句为：SELECT COUNT\_BIG(\*) FROM   var q = db.Customers.LongCount();  Sum  说明：返回集合中数值类型元素之和，集合应为INT类型集合；不延迟。生成SQL语句为：SELECT SUM(…) FROM   1.简单形式：  得到所有订单的总运费：   var q = db.Orders.Select(o => o.Freight).Sum();  2.映射形式：  得到所有产品的订货总数：   var q = db.Products.Sum(p => p.UnitsOnOrder);  Min  说明：返回集合中元素的最小值；不延迟。生成SQL语句为：SELECT MIN(…) FROM   1.简单形式：  查找任意产品的最低单价：   var q = db.Products.Select(p => p.UnitPrice).Min();  2.映射形式：  查找任意订单的最低运费：   var q = db.Orders.Min(o => o.Freight);  3.元素：  查找每个类别中单价最低的产品：   var categories =      from p in db.Products      group p by p.CategoryID into g      select new {          CategoryID = g.Key,          CheapestProducts =              from p2 in g              where p2.UnitPrice == g.Min(p3 => p3.UnitPrice)              select p2      };  Max  说明：返回集合中元素的最大值；不延迟。生成SQL语句为：SELECT MAX(…) FROM   1.简单形式：  查找任意雇员的最近雇用日期：   var q = db.Employees.Select(e => e.HireDate).Max();  2.映射形式：  查找任意产品的最大库存量：   var q = db.Products.Max(p => p.UnitsInStock);  3.元素：  查找每个类别中单价最高的产品：   var categories =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,          MostExpensiveProducts =              from p2 in g              where p2.UnitPrice == g.Max(p3 => p3.UnitPrice)              select p2      };  Average  说明：返回集合中的数值类型元素的平均值。集合应为数字类型集合，其返回值类型为double；不延迟。生成SQL语句为：SELECT AVG(…) FROM   1.简单形式：  得到所有订单的平均运费：   var q = db.Orders.Select(o => o.Freight).Average();  2.映射形式：  得到所有产品的平均单价：   var q = db.Products.Average(p => p.UnitPrice);  3.元素：  查找每个类别中单价高于该类别平均单价的产品：   var categories =      from p in db.Products      group p by p.CategoryID into g      select new {          g.Key,           ExpensiveProducts =              from p2 in g              where p2.UnitPrice > g.Average(p3 => p3.UnitPrice)              select p2      }; |

http://www.cnblogs.com/ingstyle/p/5711403.html