# NoSQL 教程︰构建 DocumentDB C# 控制台应用程序

[AZURE.SELECTOR] - [.NET](/documentation/articles/documentdb-get-started) - [Node.js](/documentation/articles/documentdb-nodejs-get-started)

欢迎使用 DocumentDB .NET SDK 的 NoSQL 教程！ 学习本教程后，你将拥有一个可创建并查询 DocumentDB 资源的控制台应用程序。

我们将介绍：

* 创建并连接到 DocumentDB 帐户
* 配置 Visual Studio 解决方案
* 创建联机数据库
* 创建集合
* 创建 JSON 文档
* 查询集合
* 替换文档
* 删除文档
* 删除数据库

没有时间？ 不必担心！ 可在 [GitHub](https://github.com/Azure-Samples/documentdb-dotnet-getting-started) 上获取完整的解决方案。有关快速说明，请转到[获取完整解决方案部分](#GetSolution)。

然后，请使用位于本页顶部或底部的投票按钮向我们提供反馈。如果你希望我们直接与你联系，欢迎将你的电子邮件地址附在评论中。

现在，让我们开始吧！

## 先决条件

请确保你具有以下内容：

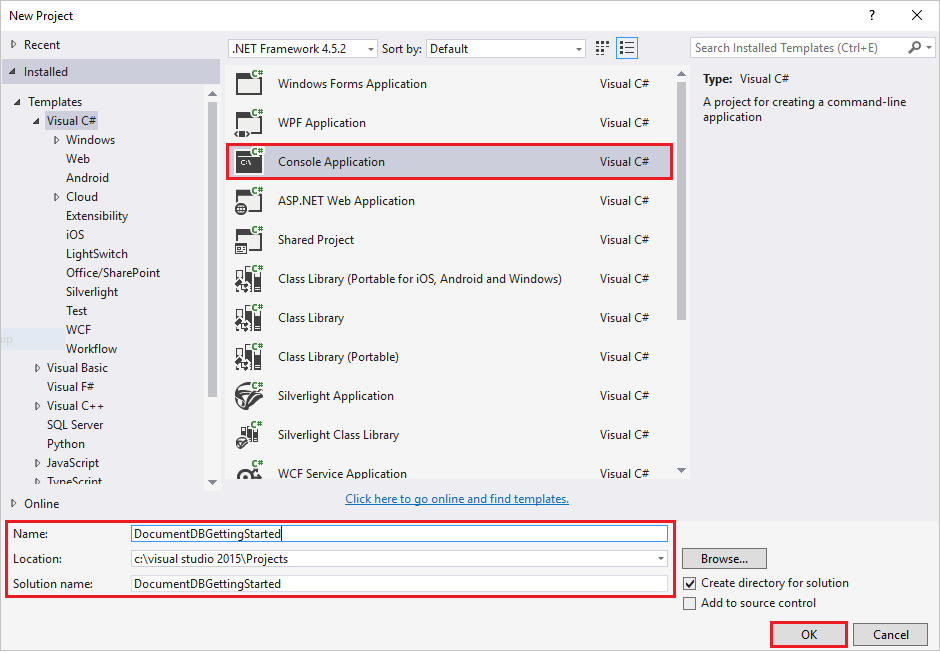
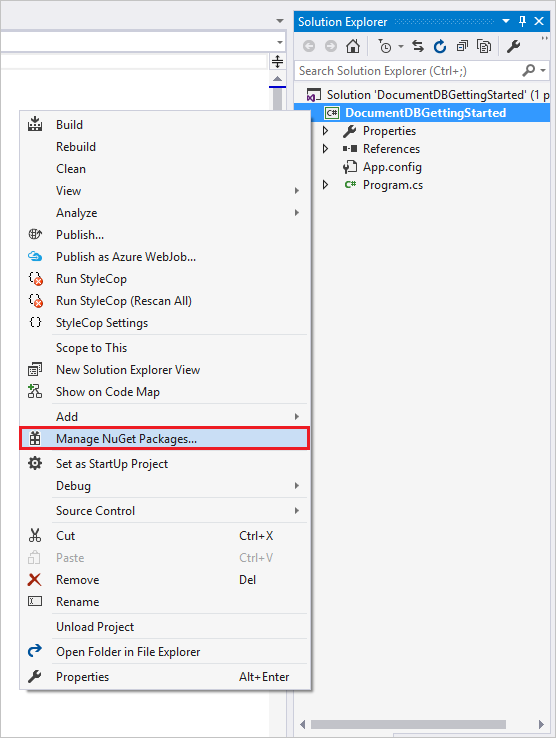
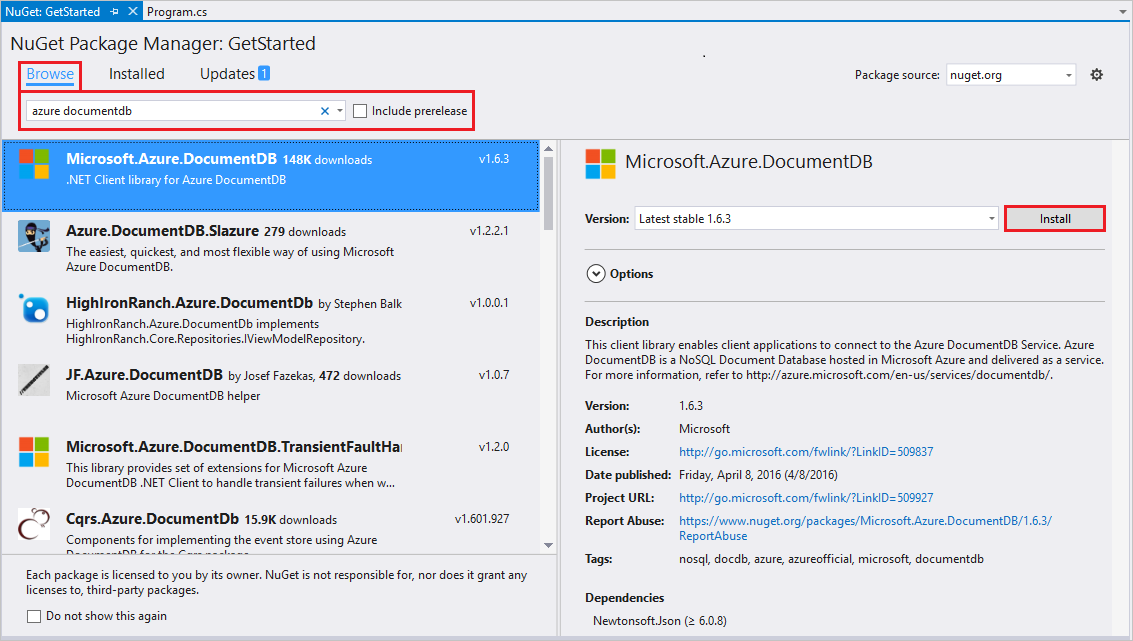
* 有效的 Azure 帐户。如果你没有订阅，可以注册[免费试用版](/pricing/free-trial/)。
* [Visual Studio 2013 / Visual Studio 2015](http://www.visualstudio.com/)。
* .NET Framework 4.6

## 第 1 步：创建 DocumentDB 帐户

让我们创建一个 DocumentDB 帐户。如果已经有一个想要使用的帐户，可以跳到[设置 Visual Studio 解决方案](#SetupVS)。

[AZURE.INCLUDE [documentdb-create-dbaccount](../includes/documentdb-create-dbaccount.md)]

## 第 2 步：设置 Visual Studio 解决方案

1. 在计算机上打开 **Visual Studio 2015**。
2. 在**“文件”**菜单中，选择**“新建”**，然后选择**“项目”**。
3. 在**“新建项目”**对话框中，选择**“模板”** / **“Visual C#”** / **“控制台应用程序”**，为项目命名，然后单击**“确定”**。
4. 在**“解决方案资源管理器”**中，右键单击 Visual Studio 解决方案下方的新控制台应用程序。
5. 接下来，无需离开菜单，单击**“管理 NuGet 程序包…”** 
6. 在“Nuget”选项卡上，单击“浏览”，然后在搜索框中输入“azure documentdb”。
7. 在结果中，找到“Microsoft.Azure.DocumentDB”，然后单击“安装”。DocumentDB 客户端库的程序包 ID 是 [Microsoft.Azure.DocumentDB](https://www.nuget.org/packages/Microsoft.Azure.DocumentDB) 

很好！ 现在，我们已完成安装，让我们开始编写一些代码。可以在 [GitHub](https://github.com/Azure-Samples/documentdb-dotnet-getting-started/blob/master/src/Program.cs) 上找到本教程的完整代码项目。

## 第 3 步：连接到 DocumentDB 帐户

首先，在 Program.cs 文件中 C# 应用程序的开始位置添加这些引用︰

using System;  
using System.Linq;  
using System.Threading.Tasks;  
  
// ADD THIS PART TO YOUR CODE  
using System.Net;  
using Microsoft.Azure.Documents;  
using Microsoft.Azure.Documents.Client;  
using Newtonsoft.Json;

[AZURE.IMPORTANT] 为了完成该 NoSQL 教程，请确保添加以上依赖关系。

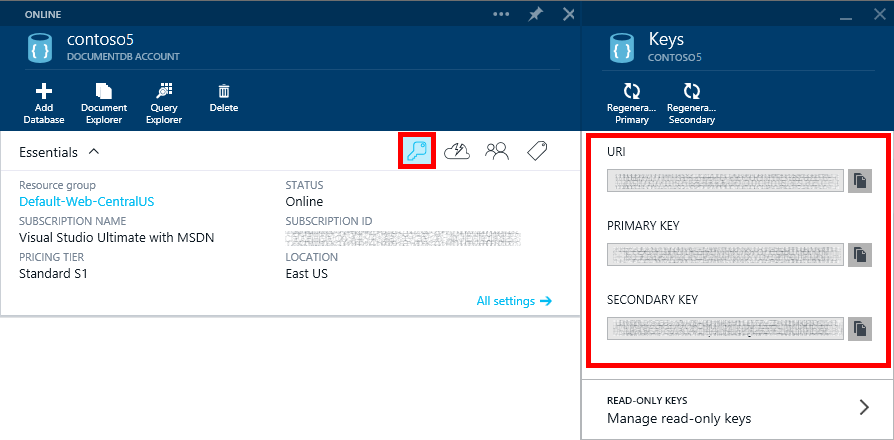
现在，在公共类 Program 下添加这两个常量和客户端变量。

public class Program  
{  
 // ADD THIS PART TO YOUR CODE  
 private const string EndpointUri = "<your endpoint URI>";  
 private const string PrimaryKey = "<your key>";  
 private DocumentClient client;

接下来，转到 [Azure 门户](https://portal.azure.cn)来检索 URI 和主键。DocumentDB URI 和主键是必需的，可让应用程序知道要连接的对象，让 DocumentDB 信任应用程序的连接。

在 Azure 门户中，从第 1 步导航到 DocumentDB 帐户。

单击“Essentials”栏上的“键”图标。 复制该 URI，并用在程序中复制的 URI 替换 <your endpoint URI>。 复制该主键，并用在程序中复制的键替换 <your key>。



NoSQL 教程用于创建 C# 控制台应用程序的 Azure 门户的屏幕截图。显示 DocumentDB 帐户，在“DocumentDB 帐户”边栏选项卡上突出显示“ACTIVE”中心、“键”按钮，在“键”边栏选项卡上突出显示 URI、主键、辅键的值

我们首先通过创建一个新的 **DocumentClient** 实例来开始创建应用程序。

在 **Main** 方法中，添加名为 **GetStartedDemo** 的这个新异步任务，它将实例化我们的新 **DocumentClient**。

static void Main(string[] args)  
{  
}  
  
// ADD THIS PART TO YOUR CODE  
private async Task GetStartedDemo()  
{  
 this.client = new DocumentClient(new Uri(EndpointUri), PrimaryKey);  
}

添加以下代码以从 **Main** 方法中运行异步任务。**Main** 方法将捕获异常并将它们写到控制台上。

static void Main(string[] args)  
{  
 // ADD THIS PART TO YOUR CODE  
 try  
 {  
 Program p = new Program();  
 p.GetStartedDemo().Wait();  
 }  
 catch (DocumentClientException de)  
 {  
 Exception baseException = de.GetBaseException();  
 Console.WriteLine("{0} error occurred: {1}, Message: {2}", de.StatusCode, de.Message, baseException.Message);  
 }  
 catch (Exception e)  
 {  
 Exception baseException = e.GetBaseException();  
 Console.WriteLine("Error: {0}, Message: {1}", e.Message, baseException.Message);  
 }  
 finally  
 {  
 Console.WriteLine("End of demo, press any key to exit.");  
 Console.ReadKey();  
 }

按 **F5** 运行应用程序。

祝贺你！ 已成功连接到 DocumentDB 帐户了，现在让我们看看如何使用 DocumentDB 资源。

## 第 4 步：创建数据库

在添加创建数据库的代码之前，添加一个用于向控制台写入的帮助器方法。

复制并粘贴 **GetStartedDemo** 方法下的 **WriteToConsoleAndPromptToContinue** 方法。

// ADD THIS PART TO YOUR CODE  
private void WriteToConsoleAndPromptToContinue(string format, params object[] args)  
{  
 Console.WriteLine(format, args);  
 Console.WriteLine("Press any key to continue ...");  
 Console.ReadKey();  
}

可以通过使用 **DocumentClient** 类的 [CreateDatabaseAsync](https://msdn.microsoft.com/library/microsoft.azure.documents.client.documentclient.createdatabaseasync.aspx) 方法来创建 DocumentDB [数据库](documentdb-resources.md#databases)。数据库是跨集合分区的 JSON 文档存储的逻辑容器。

复制并粘贴 **WriteToConsoleAndPromptToContinue** 方法下的 **CreateDatabaseIfNotExists** 方法。

// ADD THIS PART TO YOUR CODE  
private async Task CreateDatabaseIfNotExists(string databaseName)  
{  
 // Check to verify a database with the id=FamilyDB does not exist  
 try  
 {  
 await this.client.ReadDatabaseAsync(UriFactory.CreateDatabaseUri(databaseName));  
 this.WriteToConsoleAndPromptToContinue("Found {0}", databaseName);  
 }  
 catch (DocumentClientException de)  
 {  
 // If the database does not exist, create a new database  
 if (de.StatusCode == HttpStatusCode.NotFound)  
 {  
 await this.client.CreateDatabaseAsync(new Database { Id = databaseName });  
 this.WriteToConsoleAndPromptToContinue("Created {0}", databaseName);  
 }  
 else  
 {  
 throw;  
 }  
 }  
}

将以下代码复制并粘贴到客户端创建下方的 **GetStartedDemo** 方法。这将创建一个名为 FamilyDB 的数据库。

private async Task GetStartedDemo()  
{  
 this.client = new DocumentClient(new Uri(EndpointUri), PrimaryKey);  
  
 // ADD THIS PART TO YOUR CODE  
 await this.CreateDatabaseIfNotExists("FamilyDB");

按 **F5** 运行应用程序。

祝贺你！ 你已成功创建 DocumentDB 数据库。

## 第 5 步：创建集合

[AZURE.WARNING] **CreateDocumentCollectionAsync** 将创建一个具有保留吞吐量的新集合，它牵涉定价。有关详细信息，请访问我们的[定价页](/pricing/details/documentdb/)。

可以通过使用 **DocumentClient** 类的 [CreateDocumentCollectionAsync](https://msdn.microsoft.com/library/microsoft.azure.documents.client.documentclient.createdocumentcollectionasync.aspx) 方法来创建[集合](documentdb-resources.md#collections)。集合是 JSON 文档和相关联的 JavaScript 应用程序逻辑的容器。

复制并粘贴 **CreateDatabaseIfNotExists** 方法下的 **CreateDocumentCollectionIfNotExists** 方法。

// ADD THIS PART TO YOUR CODE  
private async Task CreateDocumentCollectionIfNotExists(string databaseName, string collectionName)  
{  
 try  
 {  
 await this.client.ReadDocumentCollectionAsync(UriFactory.CreateDocumentCollectionUri(databaseName, collectionName));  
 this.WriteToConsoleAndPromptToContinue("Found {0}", collectionName);  
 }  
 catch (DocumentClientException de)  
 {  
 // If the document collection does not exist, create a new collection  
 if (de.StatusCode == HttpStatusCode.NotFound)  
 {  
 DocumentCollection collectionInfo = new DocumentCollection();  
 collectionInfo.Id = collectionName;  
  
 // Configure collections for maximum query flexibility including string range queries.  
 collectionInfo.IndexingPolicy = new IndexingPolicy(new RangeIndex(DataType.String) { Precision = -1 });  
  
 // Here we create a collection with 400 RU/s.  
 await this.client.CreateDocumentCollectionAsync(  
 UriFactory.CreateDatabaseUri(databaseName),  
 new DocumentCollection { Id = collectionName },  
 new RequestOptions { OfferThroughput = 400 });  
  
 this.WriteToConsoleAndPromptToContinue("Created {0}", collectionName);  
 }  
 else  
 {  
 throw;  
 }  
 }  
}

将以下代码复制并粘贴到数据库创建下方的 **GetStartedDemo** 方法。这将创建一个名为 FamilyCollection 的文档集合。

this.client = new DocumentClient(new Uri(EndpointUri), PrimaryKey);  
  
 await this.CreateDatabaseIfNotExists("FamilyDB");  
  
 // ADD THIS PART TO YOUR CODE  
 await this.CreateDocumentCollectionIfNotExists("FamilyDB", "FamilyCollection");

按 **F5** 运行应用程序。

祝贺你！ 你已成功创建 DocumentDB 文档集合。

## 第 6 步：创建 JSON 文档

可以通过使用 **DocumentClient** 类的 [CreateDocumentAsync](https://msdn.microsoft.com/library/microsoft.azure.documents.client.documentclient.createdocumentasync.aspx) 方法创建[文档](/documentation/articles/documentdb-resources#documents)。文档为用户定义的（任意）JSON 内容。现在，我们可以插入一个或多个文档。如果已经有想要在数据库中存储的数据，则可以使用 DocumentDB 的[数据迁移工具](/documentation/articles/documentdb-import-data)。

在本例中，我们首先需要创建 **Family** 类来表示存储在 DocumentDB 中的对象。我们还将创建 **Family** 中用到的 **Parent**、**Child**、**Pet** 和 **Address** 子类。请注意，文档必须将 **Id** 属性序列化为 JSON 格式的 **id**。通过在 **GetStartedDemo** 方法后添加以下内部子类来创建这些类。

复制 **Family**、**Parent**、**Child**、**Pet** 和 **Address** 类并粘贴到 **WriteToConsoleAndPromptToContinue** 方法下。

private void WriteToConsoleAndPromptToContinue(string format, params object[] args)  
{  
 Console.WriteLine(format, args);  
 Console.WriteLine("Press any key to continue ...");  
 Console.ReadKey();  
}  
  
// ADD THIS PART TO YOUR CODE  
public class Family  
{  
 [JsonProperty(PropertyName = "id")]  
 public string Id { get; set; }  
 public string LastName { get; set; }  
 public Parent[] Parents { get; set; }  
 public Child[] Children { get; set; }  
 public Address Address { get; set; }  
 public bool IsRegistered { get; set; }  
 public override string ToString()  
 {  
 return JsonConvert.SerializeObject(this);  
 }  
}  
  
public class Parent  
{  
 public string FamilyName { get; set; }  
 public string FirstName { get; set; }  
}  
  
public class Child  
{  
 public string FamilyName { get; set; }  
 public string FirstName { get; set; }  
 public string Gender { get; set; }  
 public int Grade { get; set; }  
 public Pet[] Pets { get; set; }  
}  
  
public class Pet  
{  
 public string GivenName { get; set; }  
}  
  
public class Address  
{  
 public string State { get; set; }  
 public string County { get; set; }  
 public string City { get; set; }  
}

复制并粘贴 **CreateDocumentCollectionIfNotExists** 方法下的 **CreateFamilyDocumentIfNotExists** 方法。

// ADD THIS PART TO YOUR CODE  
private async Task CreateFamilyDocumentIfNotExists(string databaseName, string collectionName, Family family)  
{  
 try  
 {  
 await this.client.ReadDocumentAsync(UriFactory.CreateDocumentUri(databaseName, collectionName, family.Id));  
 this.WriteToConsoleAndPromptToContinue("Found {0}", family.Id);  
 }  
 catch (DocumentClientException de)  
 {  
 if (de.StatusCode == HttpStatusCode.NotFound)  
 {  
 await this.client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(databaseName, collectionName), family);  
 this.WriteToConsoleAndPromptToContinue("Created Family {0}", family.Id);  
 }  
 else  
 {  
 throw;  
 }  
 }  
}

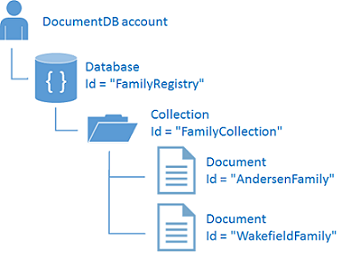
然后插入两个文档，Andersen Family 和 Wakefield Family 各一个。

将以下代码复制并粘贴到文档集合创建下的 **GetStartedDemo** 方法。

await this.CreateDatabaseIfNotExists("FamilyDB");  
  
await this.CreateDocumentCollectionIfNotExists("FamilyDB", "FamilyCollection");  
  
// ADD THIS PART TO YOUR CODE  
Family andersenFamily = new Family  
{  
 Id = "Andersen.1",  
 LastName = "Andersen",  
 Parents = new Parent[]  
 {  
 new Parent { FirstName = "Thomas" },  
 new Parent { FirstName = "Mary Kay" }  
 },  
 Children = new Child[]  
 {  
 new Child  
 {  
 FirstName = "Henriette Thaulow",  
 Gender = "female",  
 Grade = 5,  
 Pets = new Pet[]  
 {  
 new Pet { GivenName = "Fluffy" }  
 }  
 }  
 },  
 Address = new Address { State = "WA", County = "King", City = "Seattle" },  
 IsRegistered = true  
};  
  
await this.CreateFamilyDocumentIfNotExists("FamilyDB", "FamilyCollection", andersenFamily);  
  
Family wakefieldFamily = new Family  
{  
 Id = "Wakefield.7",  
 LastName = "Wakefield",  
 Parents = new Parent[]  
 {  
 new Parent { FamilyName = "Wakefield", FirstName = "Robin" },  
 new Parent { FamilyName = "Miller", FirstName = "Ben" }  
 },  
 Children = new Child[]  
 {  
 new Child  
 {  
 FamilyName = "Merriam",  
 FirstName = "Jesse",  
 Gender = "female",  
 Grade = 8,  
 Pets = new Pet[]  
 {  
 new Pet { GivenName = "Goofy" },  
 new Pet { GivenName = "Shadow" }  
 }  
 },  
 new Child  
 {  
 FamilyName = "Miller",  
 FirstName = "Lisa",  
 Gender = "female",  
 Grade = 1  
 }  
 },  
 Address = new Address { State = "NY", County = "Manhattan", City = "NY" },  
 IsRegistered = false  
};  
  
await this.CreateFamilyDocumentIfNotExists("FamilyDB", "FamilyCollection", wakefieldFamily);

按 **F5** 运行应用程序。

祝贺你！ 你已成功创建了两个 DocumentDB 文档。



说明 NoSQL 教程创建 C# 控制台应用程序所用帐户、联机数据库、集合和文档的层次关系的图表。

## 第 7 步：查询 DocumentDB 资源

DocumentDB 对存储在每个集合中的 JSON 文档都支持丰富[查询](/documentation/articles/documentdb-sql-query)。下面的示例代码演示了各种查询（使用 DocumentDB SQL 语法以及 LINQ），我们可以针对上一步中插入的文档执行查询。

复制 **ExecuteSimpleQuery** 方法并粘贴到 **CreateFamilyDocumentIfNotExists** 方法下。

// ADD THIS PART TO YOUR CODE  
private void ExecuteSimpleQuery(string databaseName, string collectionName)  
{  
 // Set some common query options  
 FeedOptions queryOptions = new FeedOptions { MaxItemCount = -1 };  
  
 // Here we find the Andersen family via its LastName  
 IQueryable<Family> familyQuery = this.client.CreateDocumentQuery<Family>(  
 UriFactory.CreateDocumentCollectionUri(databaseName, collectionName), queryOptions)  
 .Where(f => f.LastName == "Andersen");  
  
 // The query is executed synchronously here, but can also be executed asynchronously via the IDocumentQuery<T> interface  
 Console.WriteLine("Running LINQ query...");  
 foreach (Family family in familyQuery)  
 {  
 Console.WriteLine("\tRead {0}", family);  
 }  
  
 // Now execute the same query via direct SQL  
 IQueryable<Family> familyQueryInSql = this.client.CreateDocumentQuery<Family>(  
 UriFactory.CreateDocumentCollectionUri(databaseName, collectionName),  
 "SELECT \* FROM Family WHERE Family.lastName = 'Andersen'",  
 queryOptions);  
  
 Console.WriteLine("Running direct SQL query...");  
 foreach (Family family in familyQueryInSql)  
 {  
 Console.WriteLine("\tRead {0}", family);  
 }  
  
 Console.WriteLine("Press any key to continue ...");  
 Console.ReadKey();  
}

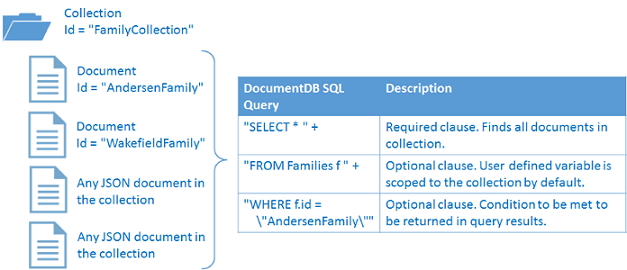
将以下代码复制并粘贴到第二次文档创建下的 **GetStartedDemo** 方法。

await this.CreateFamilyDocumentIfNotExists("FamilyDB", "FamilyCollection", wakefieldFamily);  
  
// ADD THIS PART TO YOUR CODE  
this.ExecuteSimpleQuery("FamilyDB", "FamilyCollection");

按 **F5** 运行应用程序。

祝贺你！ 你已成功完成对 DocumentDB 集合的查询。

下图说明了如何对创建的集合调用 DocumentDB SQL 查询语法，相同的逻辑也适用于 LINQ 查询。



说明 NoSQL 教程创建 C# 控制台应用程序所用查询的范围和意义的图表。

[FROM](/documentation/articles/documentdb-sql-query#from-clause) 关键字在查询中是可选的，因为 DocumentDB 查询的范围已限制为单个集合。因此，“FROM Families f”可与“FROM root r”或者任何其他所选变量名进行交换。默认情况下，DocumentDB 将推断你选择的 Families、root 或变量名，并默认引用当前集合。

## 第 8 步：替换 JSON 文档

DocumentDB 支持替换 JSON 文档。

复制并粘贴 **ExecuteSimpleQuery** 方法下的 **ReplaceFamilyDocument**。

// ADD THIS PART TO YOUR CODE  
private async Task ReplaceFamilyDocument(string databaseName, string collectionName, string familyName, Family updatedFamily)  
{  
 try  
 {  
 await this.client.ReplaceDocumentAsync(UriFactory.CreateDocumentUri(databaseName, collectionName, familyName), updatedFamily);  
 this.WriteToConsoleAndPromptToContinue("Replaced Family {0}", familyName);  
 }  
 catch (DocumentClientException de)  
 {  
 throw de;  
 }  
}

将以下代码复制并粘贴到查询执行下的 **GetStartedDemo** 方法。替换文档之后，将再次运行相同查询以查看更改后的文档。

await this.CreateFamilyDocumentIfNotExists("FamilyDB", "FamilyCollection", wakefieldFamily);  
  
this.ExecuteSimpleQuery("FamilyDB", "FamilyCollection");  
  
// ADD THIS PART TO YOUR CODE  
// Update the Grade of the Andersen Family child  
andersenFamily.Children[0].Grade = 6;  
  
await this.ReplaceFamilyDocument("FamilyDB", "FamilyCollection", "Andersen.1", andersenFamily);  
  
this.ExecuteSimpleQuery("FamilyDB", "FamilyCollection");

按 **F5** 运行应用程序。

祝贺你！ 你已成功替换 DocumentDB 文档。

## 第 9 步：删除 JSON 文档

DocumentDB 支持删除 JSON 文档。

复制并粘贴 **ReplaceFamilyDocument** 方法下的 **DeleteFamilyDocument** 方法。

// ADD THIS PART TO YOUR CODE  
private async Task DeleteFamilyDocument(string databaseName, string collectionName, string documentName)  
{  
 try  
 {  
 await this.client.DeleteDocumentAsync(UriFactory.CreateDocumentUri(databaseName, collectionName, documentName));  
 Console.WriteLine("Deleted Family {0}", documentName);  
 }  
 catch (DocumentClientException de)  
 {  
 throw de;  
 }  
}

将以下代码复制并粘贴到第二次查询执行下的 **GetStartedDemo** 方法。

await this.ReplaceFamilyDocument("FamilyDB", "FamilyCollection", "Andersen.1", andersenFamily);  
  
this.ExecuteSimpleQuery("FamilyDB", "FamilyCollection");  
  
// ADD THIS PART TO CODE  
await this.DeleteFamilyDocument("FamilyDB", "FamilyCollection", "Andersen.1");

按 **F5** 运行应用程序。

祝贺你！ 你已成功删除 DocumentDB 文档。

## 第 10 步：删除数据库

删除已创建的数据库将删除该数据库及其所有子资源（集合、文档等）。

将以下代码复制并粘贴到文档删除下的 **GetStartedDemo** 方法，以删除整个数据库和所有子资源。

this.ExecuteSimpleQuery("FamilyDB", "FamilyCollection");  
  
await this.DeleteFamilyDocument("FamilyDB", "FamilyCollection", "Andersen.1");  
  
// ADD THIS PART TO CODE  
// Clean up/delete the database  
await this.client.DeleteDatabaseAsync(UriFactory.CreateDatabaseUri("FamilyDB"));

按 **F5** 运行应用程序。

祝贺你！ 你已成功删除 DocumentDB 数据库。

## 第 11 步：一起运行 C# 控制台应用程序！

在 Visual Studio 中按 F5，即可在调试模式下构建应用程序。

你应该看到已启动应用的输出。输出会显示我们所添加的查询的结果，并且应与下面的示例文本相匹配。

Created FamilyDB  
Press any key to continue ...  
Created FamilyCollection  
Press any key to continue ...  
Created Family Andersen.1  
Press any key to continue ...  
Created Family Wakefield.7  
Press any key to continue ...  
Running LINQ query...  
 Read {"id":"Andersen.1","LastName":"Andersen","District":"WA5","Parents":[{"FamilyName":null,"FirstName":"Thomas"},{"FamilyName":null,"FirstName":"Mary Kay"}],"Children":[{"FamilyName":null,"FirstName":"Henriette Thaulow","Gender":"female","Grade":5,"Pets":[{"GivenName":"Fluffy"}]}],"Address":{"State":"WA","County":"King","City":"Seattle"},"IsRegistered":true}  
Running direct SQL query...  
 Read {"id":"Andersen.1","LastName":"Andersen","District":"WA5","Parents":[{"FamilyName":null,"FirstName":"Thomas"},{"FamilyName":null,"FirstName":"Mary Kay"}],"Children":[{"FamilyName":null,"FirstName":"Henriette Thaulow","Gender":"female","Grade":5,"Pets":[{"GivenName":"Fluffy"}]}],"Address":{"State":"WA","County":"King","City":"Seattle"},"IsRegistered":true}  
Replaced Family Andersen.1  
Press any key to continue ...  
Running LINQ query...  
 Read {"id":"Andersen.1","LastName":"Andersen","District":"WA5","Parents":[{"FamilyName":null,"FirstName":"Thomas"},{"FamilyName":null,"FirstName":"Mary Kay"}],"Children":[{"FamilyName":null,"FirstName":"Henriette Thaulow","Gender":"female","Grade":6,"Pets":[{"GivenName":"Fluffy"}]}],"Address":{"State":"WA","County":"King","City":"Seattle"},"IsRegistered":true}  
Running direct SQL query...  
 Read {"id":"Andersen.1","LastName":"Andersen","District":"WA5","Parents":[{"FamilyName":null,"FirstName":"Thomas"},{"FamilyName":null,"FirstName":"Mary Kay"}],"Children":[{"FamilyName":null,"FirstName":"Henriette Thaulow","Gender":"female","Grade":6,"Pets":[{"GivenName":"Fluffy"}]}],"Address":{"State":"WA","County":"King","City":"Seattle"},"IsRegistered":true}  
Deleted Family Andersen.1  
End of demo, press any key to exit.

祝贺你！ 你已经完成该 NoSQL 教程，并且获得了一个正常工作的 C# 控制台应用程序！

## 获取完整的 NoSQL 教程解决方案

若要生成包含本文所有示例的 GetStarted 解决方案，你将需要以下内容：

* 有效的 Azure 帐户。如果你没有订阅，可以注册[免费帐户](/free/)。
* [DocumentDB 帐户](/documentation/articles/documentdb-create-account)。
* GitHub 上可用的 [GetStarted](https://github.com/Azure-Samples/documentdb-dotnet-getting-started) 解决方案。

若要在 Visual Studio 中还原 DocumentDB.NET SDK 引用，请在“解决方案资源管理器”中右键单击 **GetStarted** 解决方案，然后单击**“启用 NuGet 程序包还原”**。接下来，按照[连接到一个 DocumentDB 帐户](#Connect)中所述的方法在 App.config 文件中更新 EndpointUrl 和 AuthorizationKey 值。

## 后续步骤

* 需要更复杂的 ASP.NET MVC NoSQL 教程？ 请参阅[使用 DocumentDB 构建具有 ASP.NET MVC 的 Web 应用程序](/documentation/articles/documentdb-dotnet-application)。
* 了解如何[监视 DocumentDB 帐户](/documentation/articles/documentdb-monitor-accounts)。
* 在 [Query Playground（查询板块）](https://www.documentdb.com/sql/demo)中对示例数据集运行查询。
* 了解有关编程模型的详细信息，请参阅 [DocumentDB 文档页面](/documentation/services/documentdb/)的开发部分。