NHIBERNATE

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

NHibernate is an actively developed, fully featured, open source object-relational mapper for the .NET framework. It is used in thousands of successful projects. It's built on top of ADO.NET and the current version is NHibernate 4.0.4.

This tutorial will give you an idea of how to get started with NHibernate. The main goal is that after completing it, you will have a better understating of what NHibernate is and why you need NHibernate and of course learn how to add NHibernate to your project.

Audience

This tutorial will be extremely useful for developers whose objective is to understand the basics of Object Relational Mapping for .NET platform and implement it in practice. It is especially going to help the users who are mainly responsible for mapping an object-oriented domain model to a traditional relational database.

Prerequisites

It is an elementary tutorial and you can easily understand the concepts explained here with a basic knowledge of how to map .NET classes to database tables. However, it will be an added help if you have some prior exposure to databases and how to deal with object relational mapping solutions.

Copyright & Disclaimer

Table of Contents

1. NHibernate – Overview

In this chapter, we will discuss about what NHibernate is, which all platforms it can be implemented, what are its advantages and other aspects related to it.

What is NHibernate?

NHibernate is a mature, open source object-relational mapper for the .NET framework. It's actively developed, fully featured and used in thousands of successful projects. It's built on top of **ADO.NET** and the current version is **NHibernate 4.0.4**.

- NHibernate is an open-source .NET object-relational mapper and is distributed under the **GNU Lesser General Public License**.
- It is based on Hibernate which is a popular Java object-relational mapper and it has a very mature and active code base.
- It provides a framework for mapping an object-oriented domain model to a traditional relational database.
- NHibernate was started by **Tom Barrett** and this project has been around since February of 2003, which was their first commit.
- It's a big project and provides a lot of functionality.
- There is a **NuGet package** available, which makes it very easy to add to a project.

Why NHibernate?

Now the question is why do we need **object-relational mappers**? It is because there is a disconnect between the object world and the relational world.

- In the object world, everything is based around **objects**; we called objects those things which have our data.
- The relational world is all set-based and we are dealing with tables and rows which are different than the object world.
- In the object world, we have **unidirectional associations**. If a customer has a pointer to an order, it doesn't necessarily mean that an order has a pointer back to a customer, it might or might not.
- In the relational world, all associations are **bidirectional** and it can be done by a foreign key.
- All associations are inherently bidirectional, so when we are dealing with object-relational mapping, we also need to deal with this disconnect.

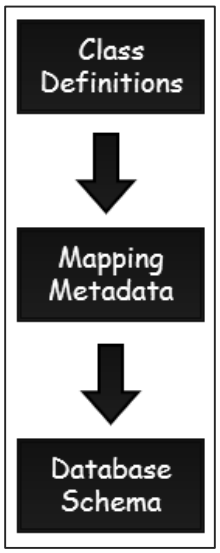
- In the object world, we are working with pointers that are unidirectional, whereas in with the relational world, we have foreign keys which are inherently bidirectional.

- The object world has this notion of inheritance, where a vehicle can have a number of different subclasses, so a car is a type of vehicle, a boat is a type of vehicle, and a sports car is a type of car, these types of inheritance relationships.

- The relational world doesn't have this notion of inheritance.

Mapping

So how do we map all these **disjoint relationships?** This concept of mapping comes from the object-relational mapper. There are mainly three things to understand as shown in the following diagram.



- In your application, you will need class definitions, which is typically C# code and its .NET code that represents our classes, such as Employee class, Customer class, Order class, etc.

- At the bottom, you can see a database schema, which is our **Data Definition Language** in a relational database that specifies what a customer table looks like, what an employee table looks like.

- In between these we have the mapping metadata that tells the object-relational mapper how to translate from the object world in C# to the database world in terms of rows and columns and foreign key relationships.

- This mapping metadata can be represented in a variety of different ways and we will be looking at a number of this different ways most typical in NHibernate application.

- It is represented by **HBM (Hibernate Mapping)** files, which are XML files.

Database Supported

NHibernate supports a wide variety of different databases. Any existing relational database out there can be accessed to NHibernate.

- SQL server is the primary supported database, that's what most developers are using during the development, it's probably the most common one.
- It also **works very well with Oracle**.
- It also supports DB2, the Firebird, MySQL, PostgreSQL, SQL Lite
- It also has **ODBC and OLEDB drivers**.

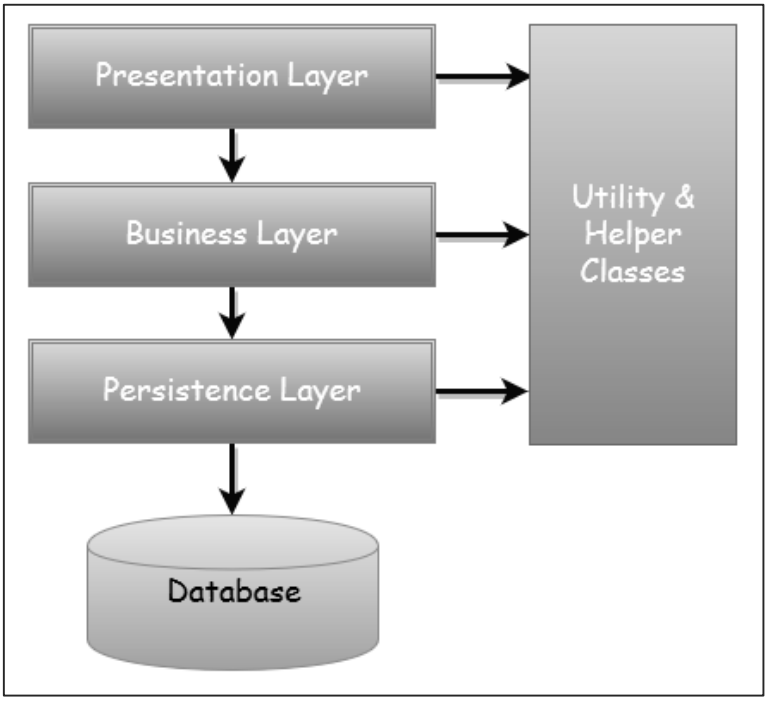
2. NHibernate – Architecture

Now-a-days, many systems are designed with layered architecture, NHibernate also has it and works perfectly well with that design.

Layered Architecture

A layered architecture divides a system into a number of groups, where each group contains code addressing a particular problem area and these groups are called layers. Most of the enterprise level applications use **high-level application architecture** that consist of three layers –

- The Presentation layer
- The Business layer
- The Persistence layer



For example, a user interface layer which is also known as the presentation layer might contain all the application code for building web pages and processing user input.

One major benefit of the layering approach is that you can often make changes to one layer without any significant disruption to the other layers, thus making the systems **lesser fragile and more maintainable**.

Presentation Layer

- It is the topmost layer, which contains the code responsible for drawing User Interface, pages, dialogs or screens, and collecting user input, and controlling navigation.

Business Layer

- The business layer is responsible for implementing any business rules or system requirements that users would understand as part of the problem domain.
- It also reuses the model defined by the persistence layer.

Persistence Layer

- The persistence layer consists of classes and components which are responsible for saving and retrieving application data.
- This layer also defines a mapping between the model class and the database. NHibernate is used primarily in this layer.

Database

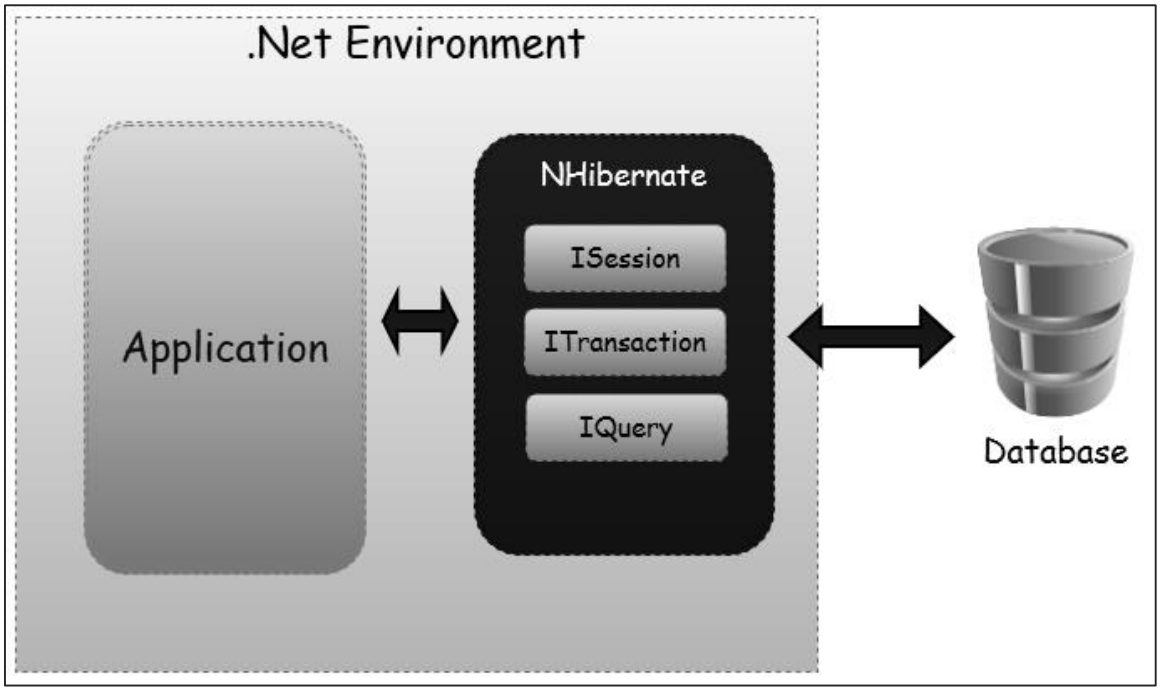
- The database exists outside the .NET application.
- It's the actual, persistent representation of the system state.
- If a SQL database is used, the database includes the relational schema and possibly stored procedures.

Helper/Utility Classes

- Every application has a set of helper or utility classes that support the other layers: for example, UI widgets, messaging classes, Exception classes, and logging utilities.
- These elements aren't considered to be layers, because they don't obey the rules for interlayer dependency in a layered architecture.

NHibernate Architecture

- It is a high-level view of the NHibernate application and you can also see the simple NHibernate architecture.



- The application code uses the NHibernate **ISession** and **IQuery** APIs for persistence operations and only has to manage database transactions, ideally using the NHibernate **ITransaction** API.

3. NHibernate – ORM

Before we can really start using NHibernate, we need to understand the foundation on which it is built. NHibernate is a persistence technology that is based on the idea of object relational mapping or ORM.

What is ORM?

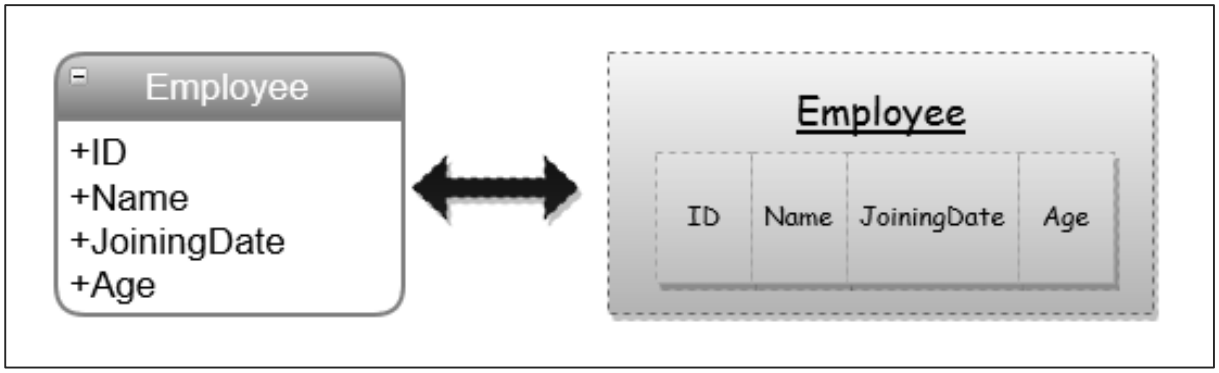
Object-Relational Mapping (ORM) is a **programming technique** for converting data between incompatible type systems in object-oriented programming languages. In other words, it is the concept of mapping an application's business objects to relational database tables, so that the data can be easily accessed and updated entirely through the object model of an application.

- As you already know that relational databases provide a good means of storing data, while object-oriented programming is a good approach to building complex applications.
- NHibernate and ORM in general are most relevant to applications with nontrivial business logic, the domain model and some sort of database.
- With ORM, it is very easy to create a translation layer that can easily transform objects into relational data and back again.
- The acronym ORM can also mean object role modeling, and this term was invented before object/relational mapping became relevant.
- It describes a method for information analysis, used in database modeling.

Why ORM?

ORM is a **framework** that enables you to map the world of objects found in object oriented languages to rows in relational tables found in relational databases

To understand this concept, let's have a look at the following diagram.



- In the above diagram, you can see that we have a table called Employee on the right side that contains columns with each piece of data associated with an individual employee.

- We have a column for an Id which uniquely identifies each employee.
- A column for the employee's name, another column for their joining date, and finally a column that has an age of an employee.
- If we wanted to write some code to store a new employee in the tables, it isn't so easy.
- In the above diagram, you can also see that we have an employee object that has fields for the Id, name, joining date and age.
- Without an ORM we have to translate this object into a few different SQL statements that will insert the employee data into the employee table.
- So writing code to create the SQL to do the above scenario is not that hard, but it is a bit tedious and pretty easy to get wrong.
- Using an ORM like NHibernate, we can declare how certain classes should be mapped to relational tables and let the ORM or NHibernate deal with the nasty job of creating the SQL to insert, update, delete, in query data in our employee table.
- This allows us to keep our code focused on using objects and have those objects automatically translated to relational tables.
- So really what an ORM does is it saves us from manually having to map objects to tables.

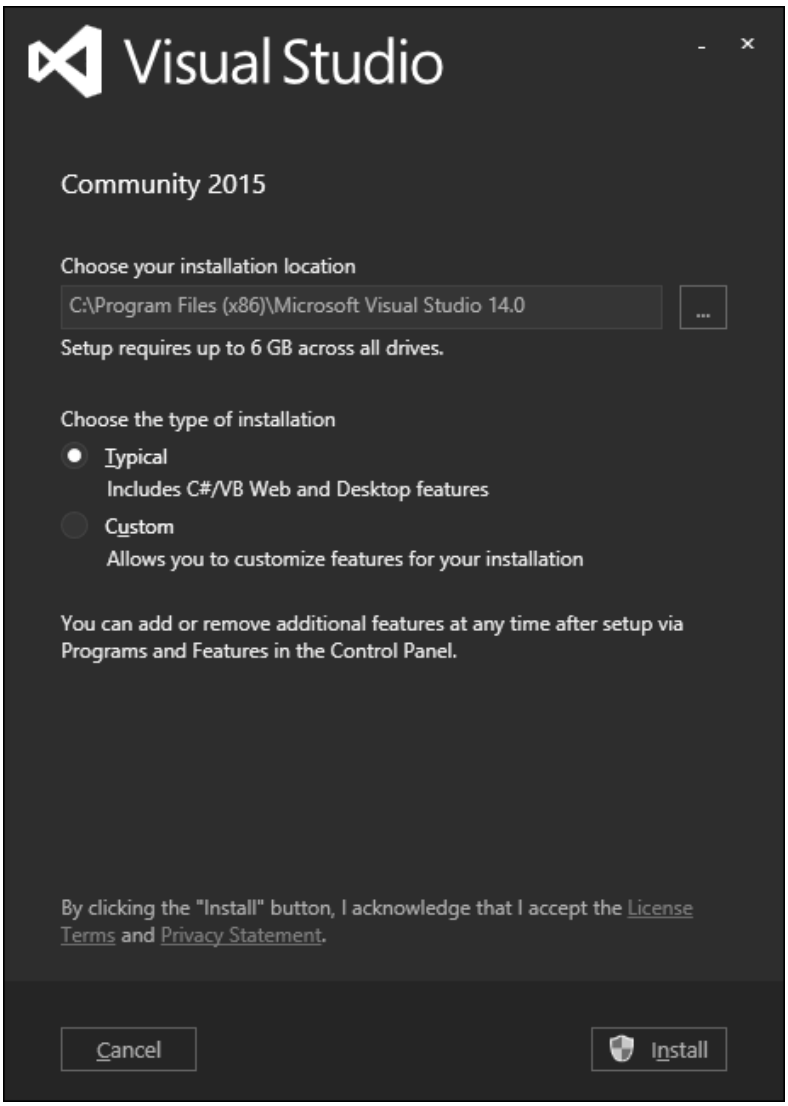
4. NHibernate – Environment Setup

To start working on NHibernate, we will need Visual Studio and the NHibernate package.

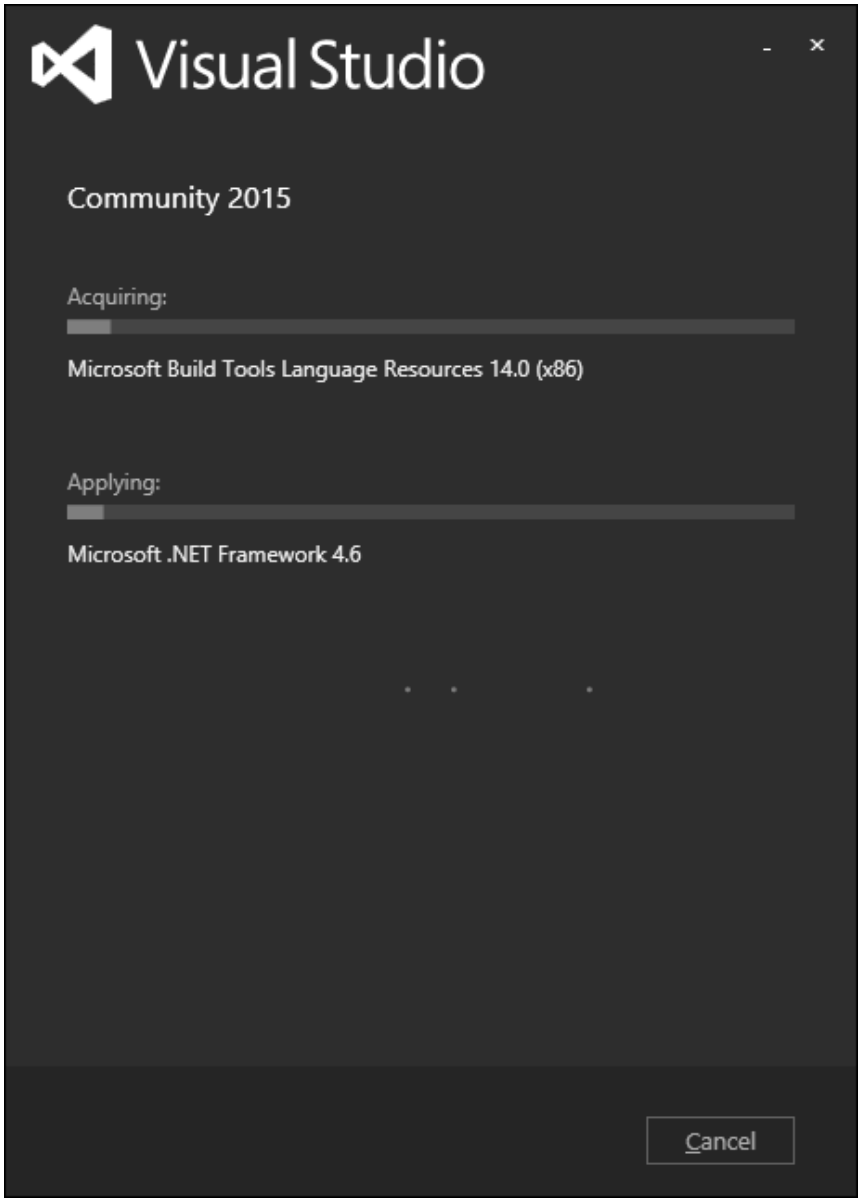
Visual Studio Installation

Microsoft provides a **free version** of Visual Studio, which also contains **SQL Server** and it can be downloaded from https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx. Following are the steps for the installation.

Step 1: Once the downloading is completed then run the installer, then the following dialog box will be displayed.



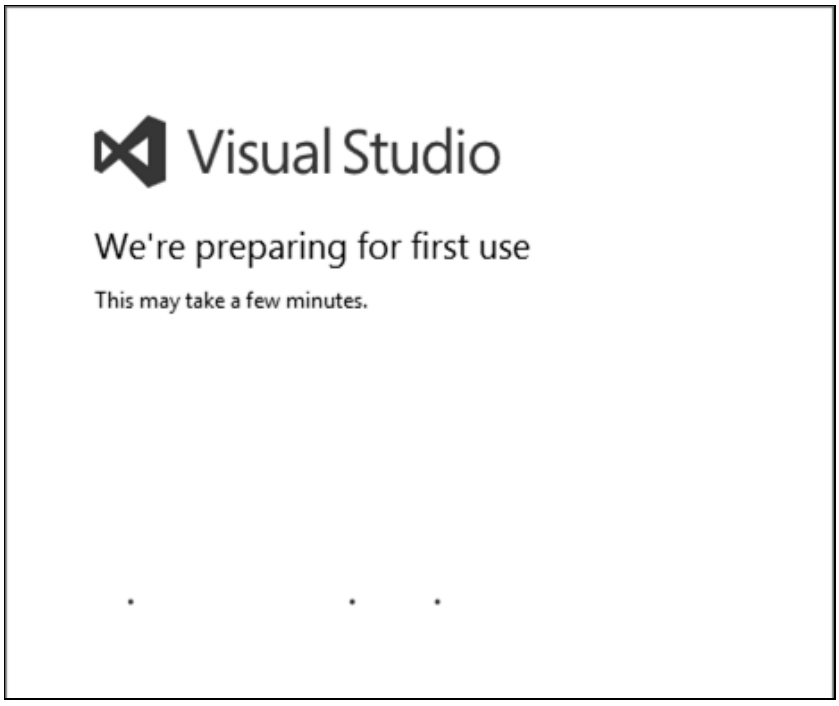
Step 2: Click on the Install button and it will start installation process.



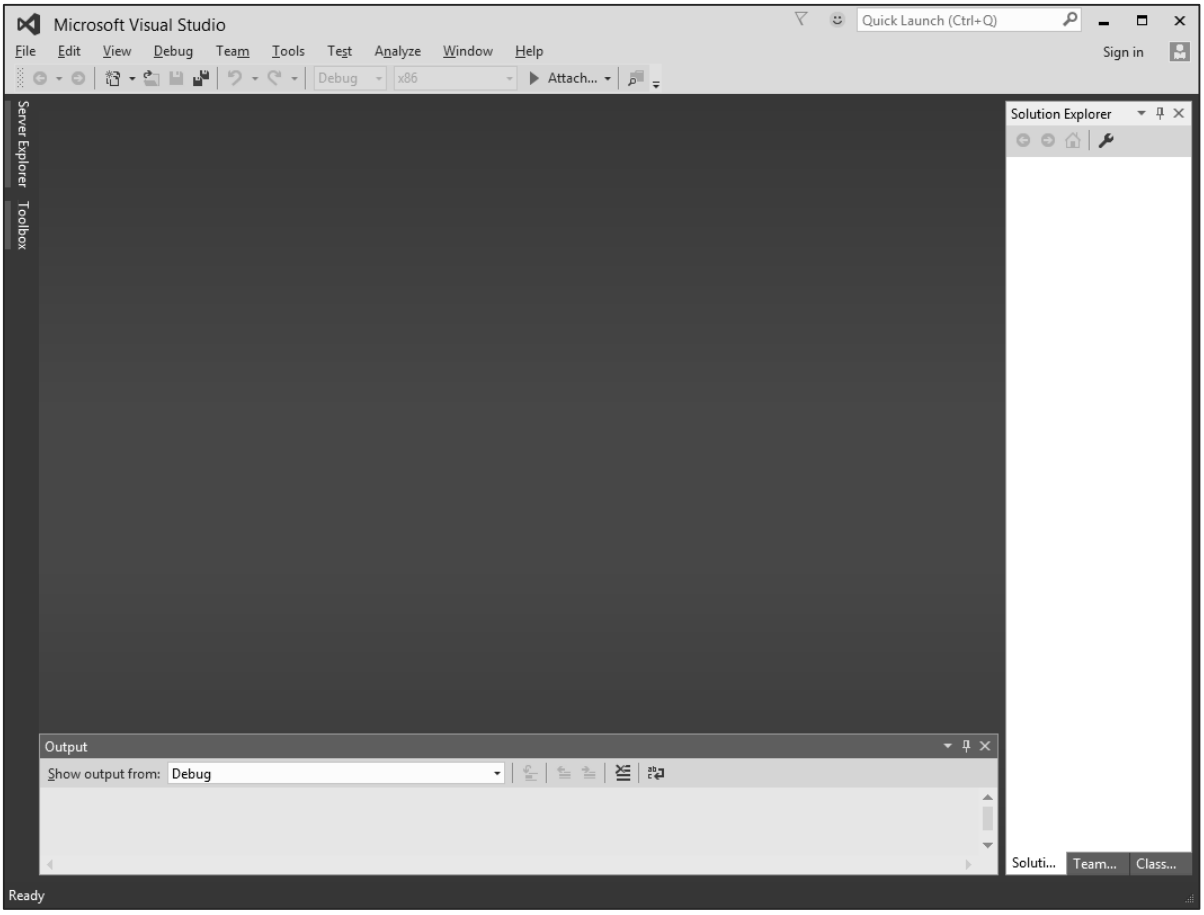
Step 3: Once the installation process is completed successfully, you will see the following dialog box.

Step 4: Close this dialog box and restart your computer if required.

Step 5: Now open Visual studio from the Start Menu which will open the following dialog. It will take some time for the first time for preparation.



Step 6: Once all this is done, you will see the main window of Visual Studio.



NHibernate Package Installation

NHibernate is a mature, open source object-relational mapper for the .NET framework. It is actively developed, fully featured and used in thousands of successful projects. You can install NHibernate package with the following methods.

Direct Download

- Download the zip from file from https://sourceforge.net/projects/nhibernate/files/NHibernate/4.0.4.GA/NHibernate-4.0.4.GA-bin.zip/download, which contains all the binaries that are required.

- Extract this zip file and include all these binaries in your project.

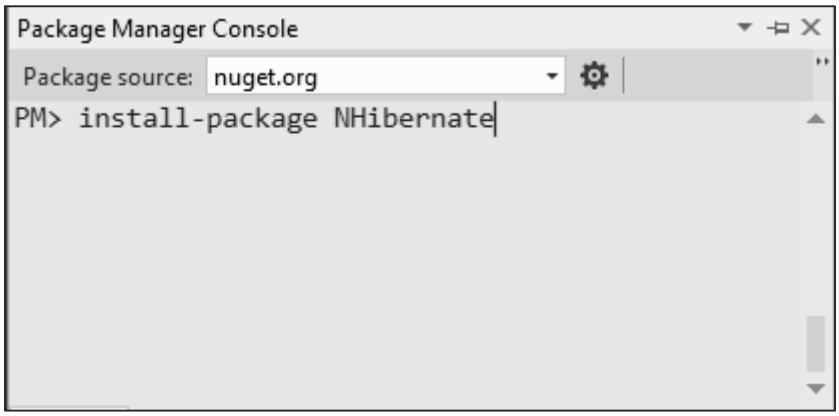
Install Using NuGet

- Another way of installing NHibernate is to use NuGet to install the NHibernate package, which is by far the easiest way to incorporate NHibernate into a project.

- It's going to download all the NHibernate dependencies and create references to all of the required assemblies.

- To install NHibernate, run the following command in the Package Manager Console.

```
install-package NHibernate
```



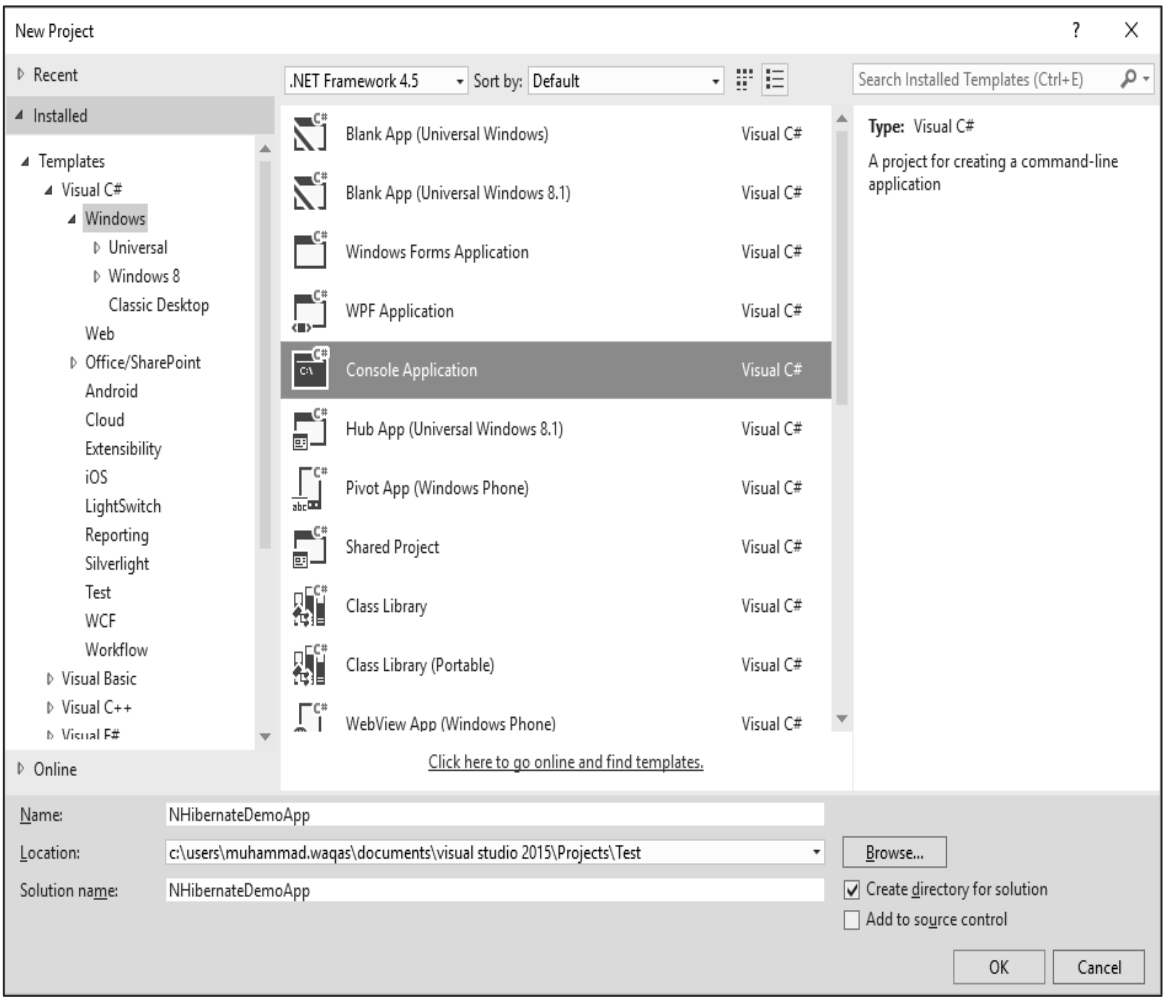
You are now ready to start your application.

5. NHibernate – Getting Started

In this chapter, we will look at how to start a simple example using NHibernate. We will be building a **simple console application**. To create a console application, we will use Visual Studio 2015, which contains all of the features you need to create, test your application using the NHibernate package.

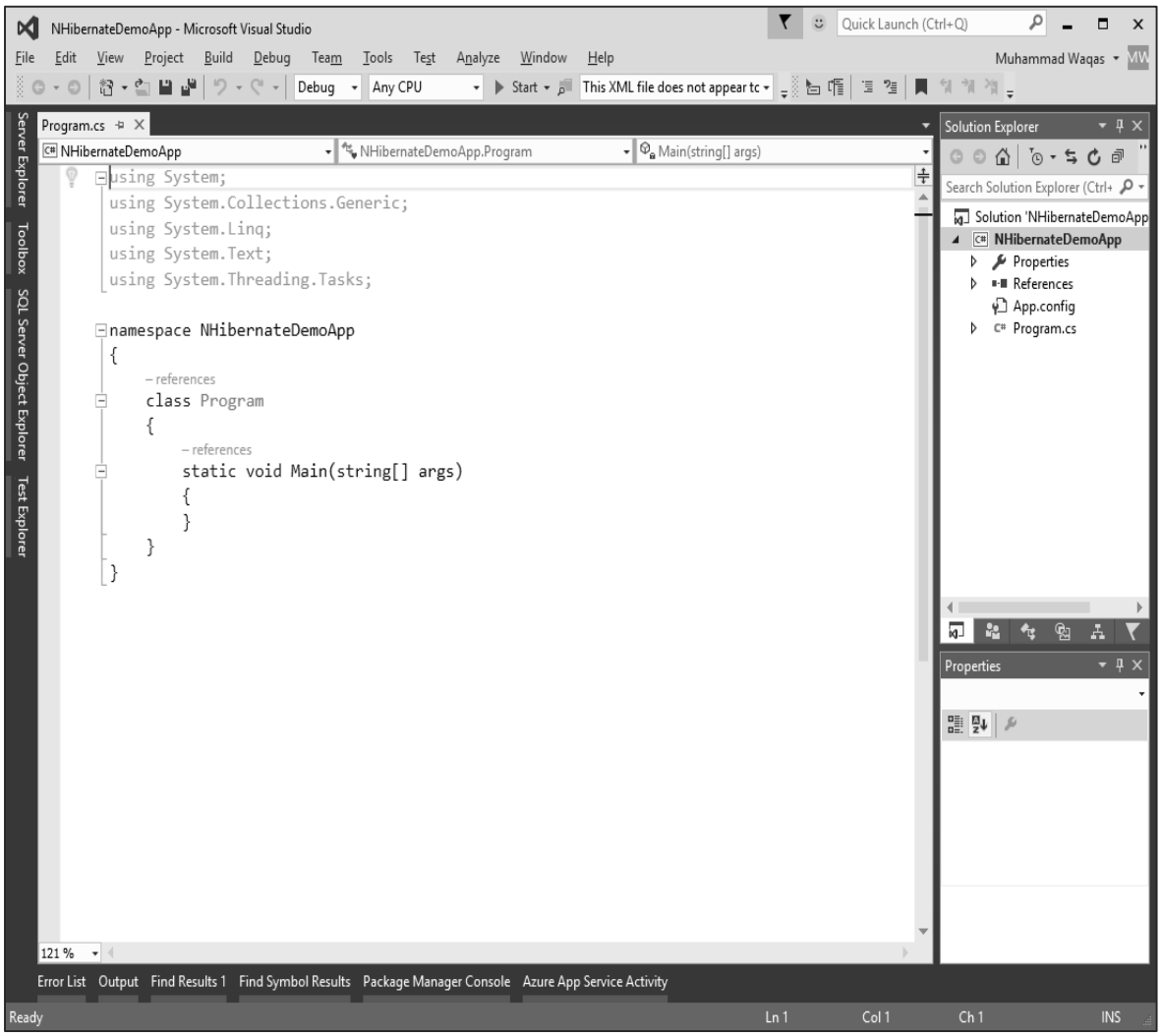
Following are the steps to create a project using project templates available in the Visual Studio.

1. Open the Visual studio and click File -> New -> Project menu option.
2. A new Project dialog opens.



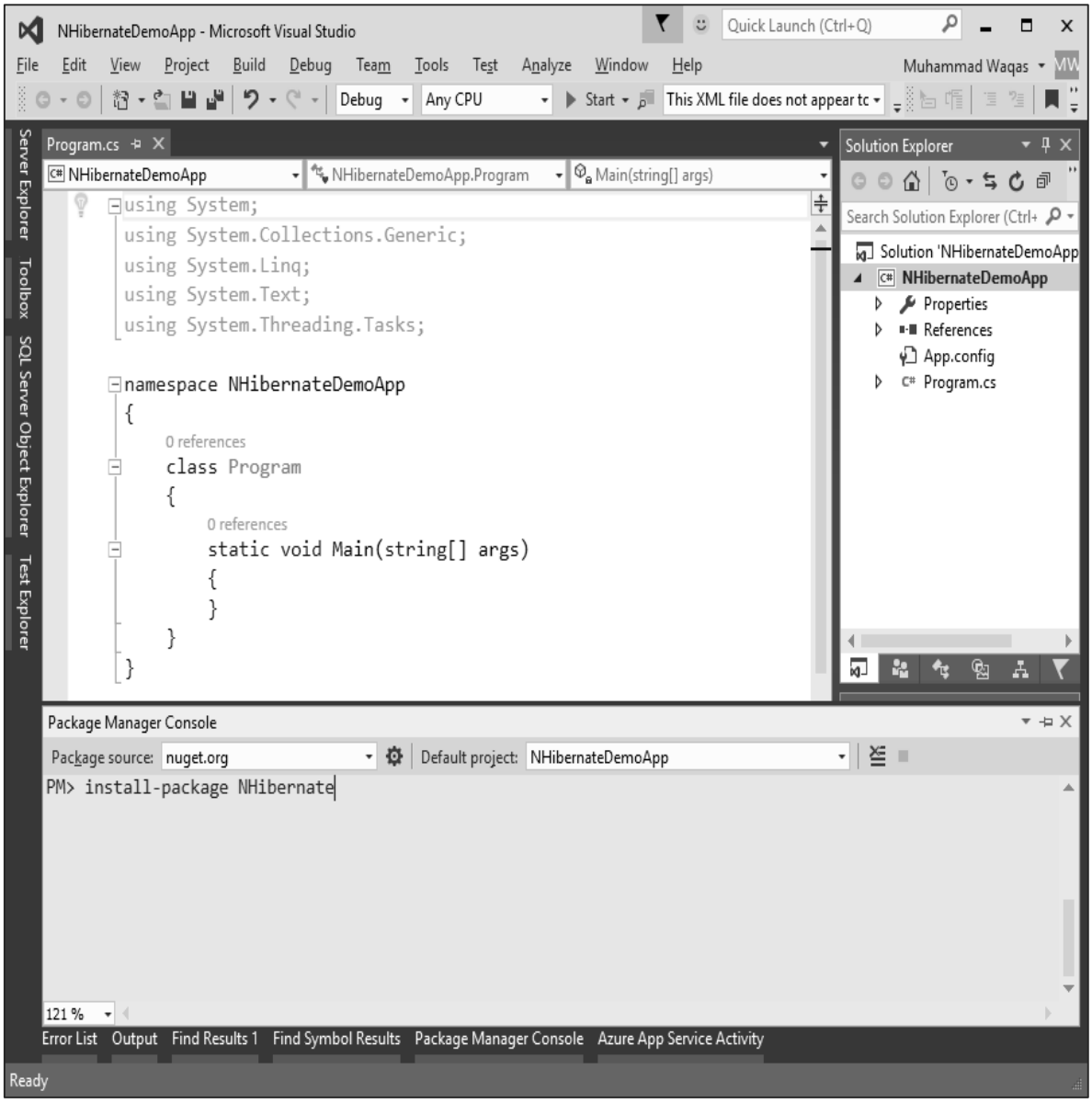
3. From the left pane, select Templates -> Visual C# -> Windows.
4. In the middle pane, select Console Application.
5. Enter the project name, 'NHibernateDemoApp', in the Name field and click Ok to continue.

6. Once the project is created by Visual Studio, you will see a number of files displayed in the Solution Explorer window.

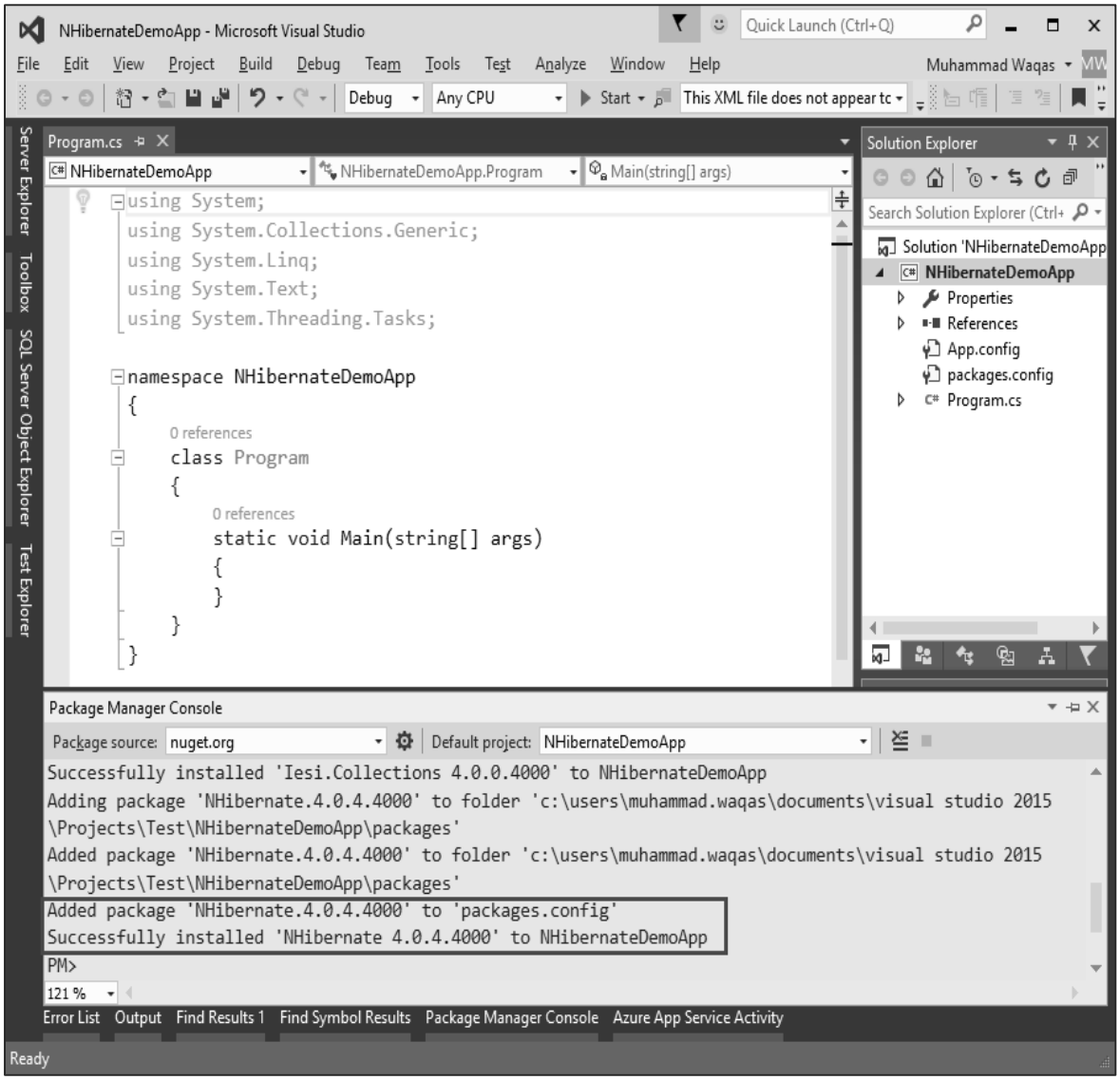


As you know that we have created a simple console application project, now we need to include the NHibernate package to our console project.

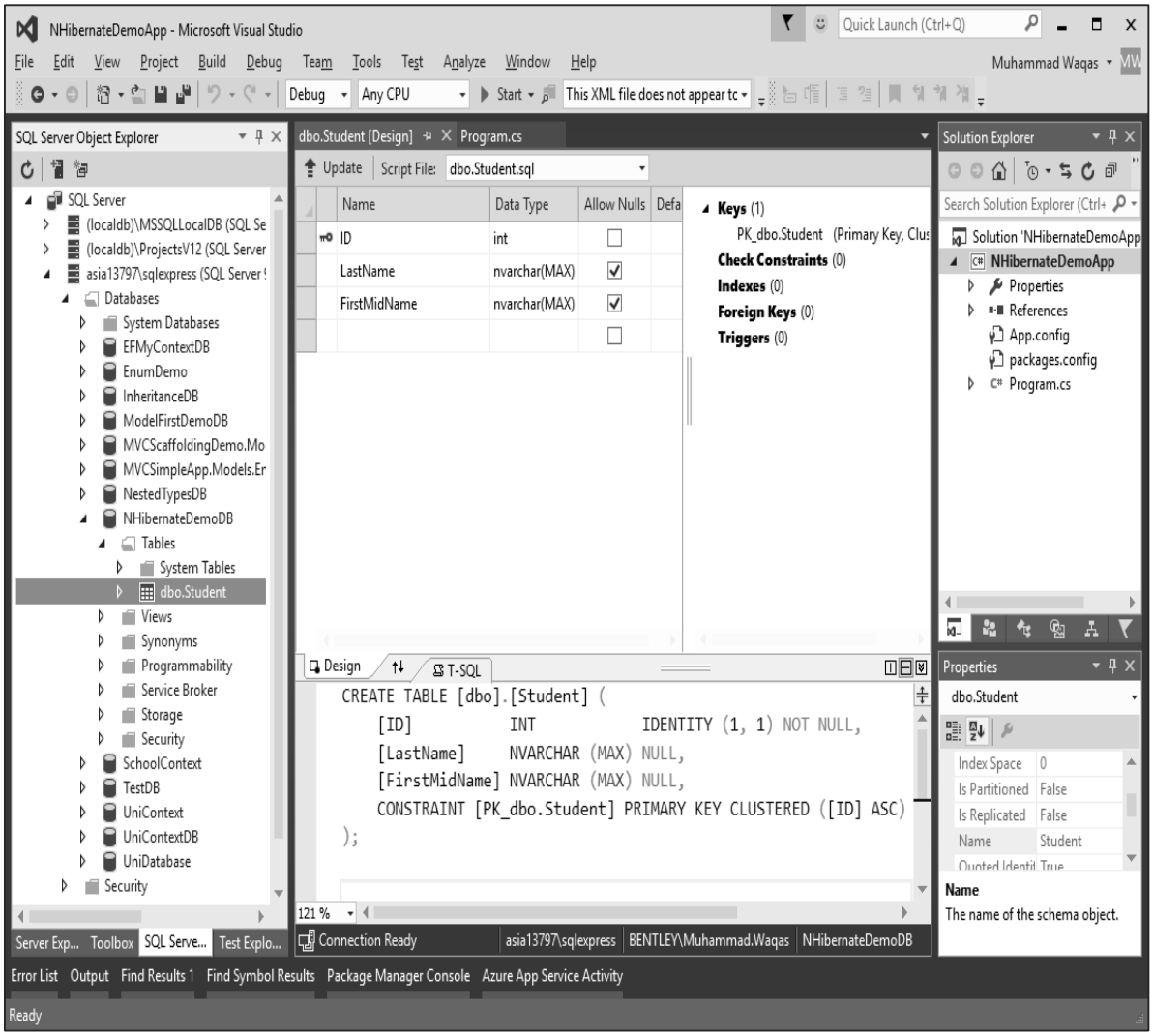
Go to the Tools menu and select NuGet Package Manager -> Package Manager Console, it will open the Package Manager Console window.



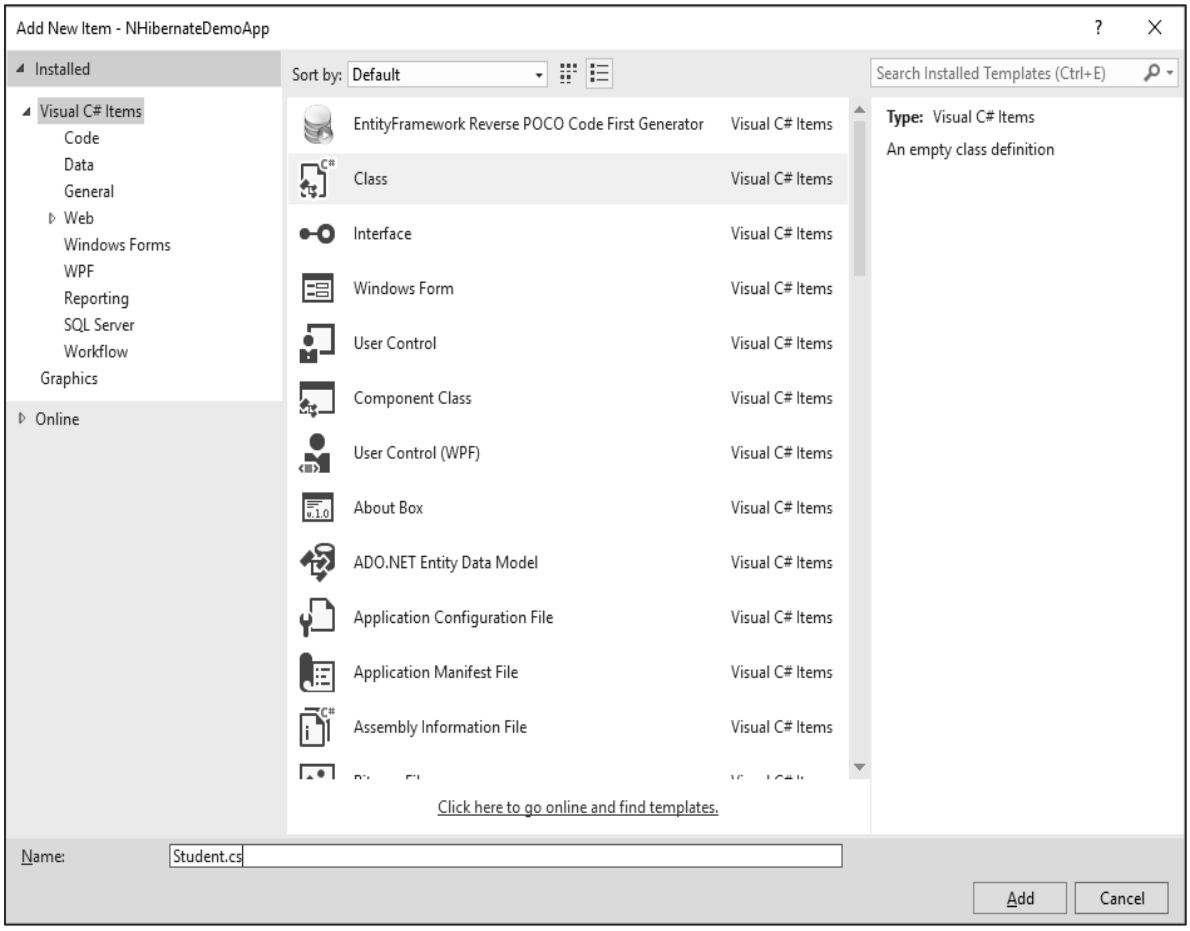
Specify the command shown in the above **Package Manager Console** window and press enter, it will download all the NHibernate dependencies and create references to all of the required assemblies. Once the installation is finished, you will see the message as shown in the following image.



Now that we have NHibernate added, we can now start implementation. So, we are going to start out by mapping a very simple **table** called **Student**, which simply has an integer primary key called ID and a FirstName and LastName column.



We need a class to represent this student, so let's create a new class called Student by right clicking on the project in the solution explorer and then select Add -> Class which will open the Add New Item dialog box.



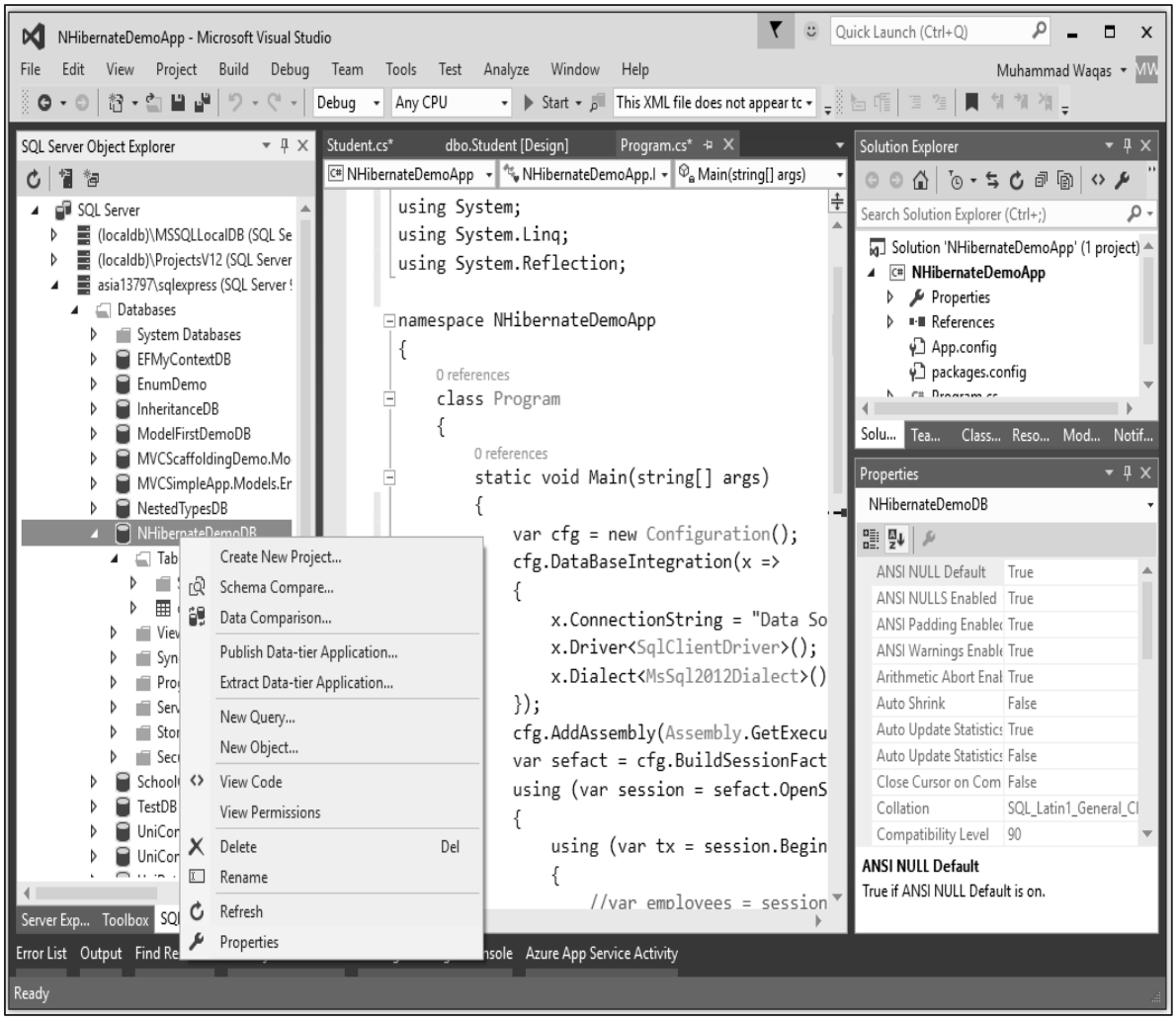
Enter **Student.cs** in the name field, click the Add button. In this Student class, we need to have our integer primary key called ID, and we need to create this string, **FirstName** and **LastName** fields as shown in the following complete implementation of Student class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NHibernateDemoApp
{
    class Student
    {
        public virtual int ID { get; set; }
        public virtual string LastName { get; set; }
        public virtual string FirstMidName { get; set; }
    }
}
```

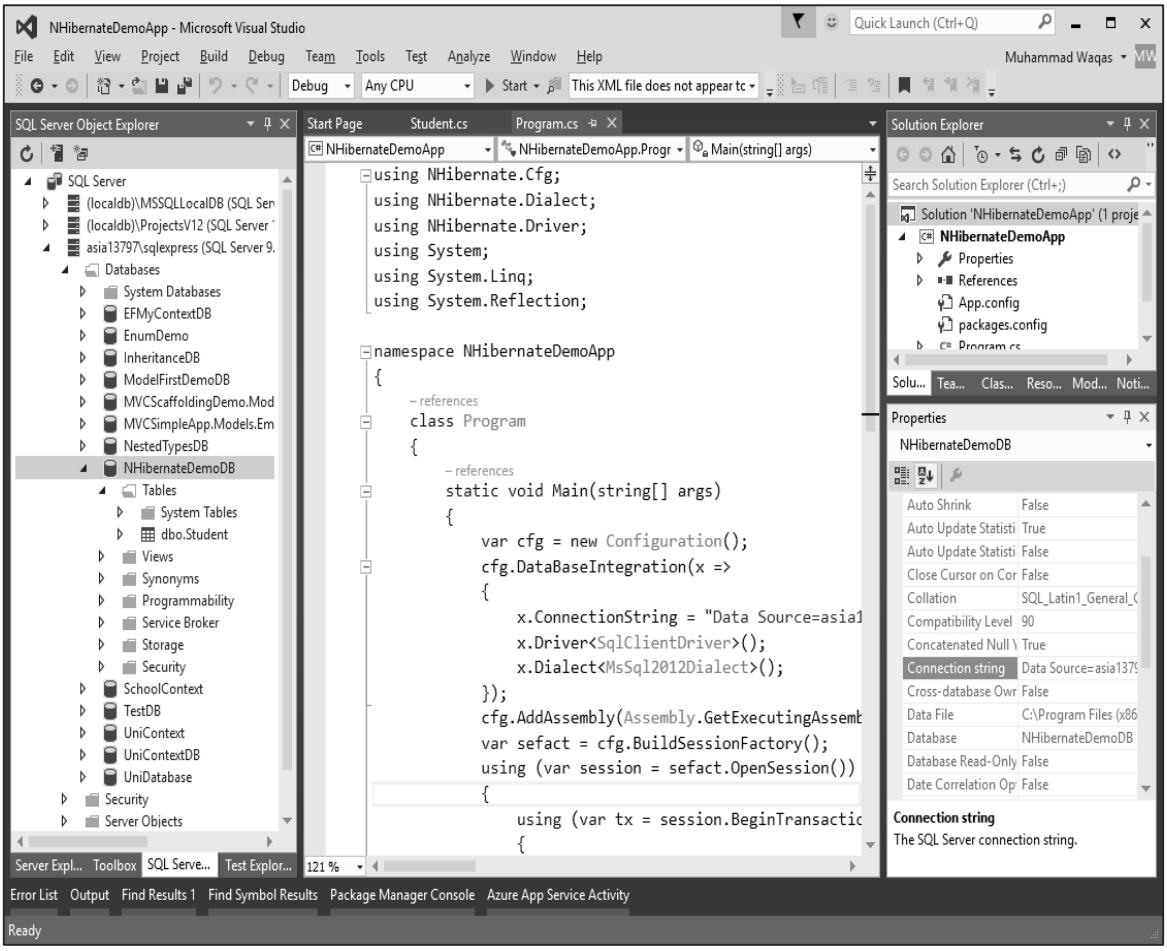
When dealing with models in NHibernate application, it is easiest to make all of your fields virtual. So this is our simple NHibernate model that we will use and will map this to the back end database.

Now let's go to the Main method in the Program class and create a new NHibernate configuration object.

The first thing that we need to provide is the **connection string**. This is a database specific connection string and the easiest way to find the connection string is that right click on the database in **SQL Server Object Explorer** and select Properties.



It will open the Properties Window, now scroll down and you will see the Connection String field in the Properties window.



Copy the Connection string and specify in your code. Following is the implementation of Main method in which we need configuration for NHibernate.

```
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            var cfg = new Configuration();
```

```
            cfg.DataBaseIntegration(x =>


            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
            });
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    //perform database logic
                    tx.Commit();
                }
                Console.ReadLine();
            }
        }
    }
}
```

After the connection string, we need to supply a driver, which is the **SQLClientDriver** and then we also need to provide it a dialect, which version of SQL Server, and we are going to use MS SQL 2008.

NHibernate now knows how to connect to the database. The other thing we need to do is to provide it a list of models that we will map.

We can do this by adding an assembly, so by specifying the **Assembly**.**GetExecutingAssembly** and this is where program will find mapping files. Mapping files tell NHibernate how to go from C# classes into database tables.

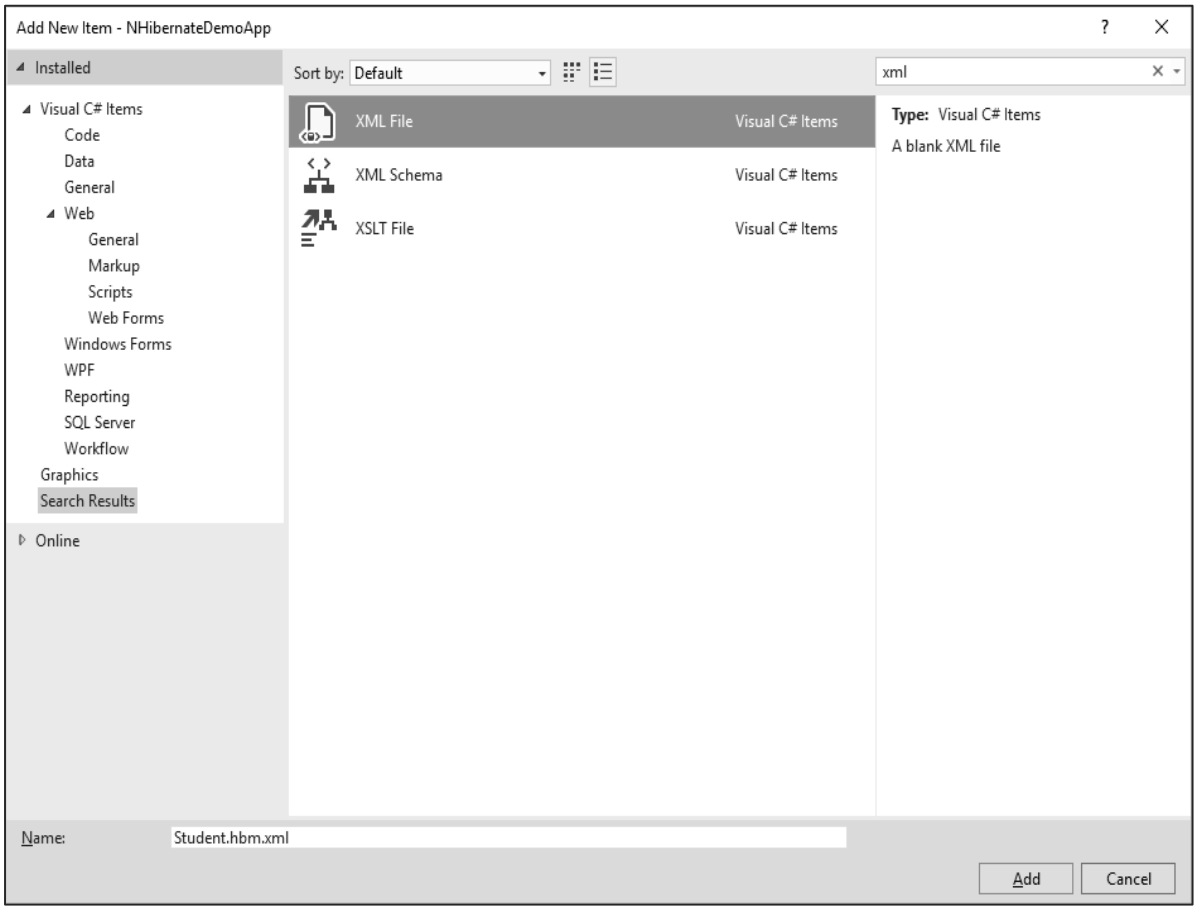
SessionFactory compiles all the metadata necessary for initializing NHibernate. SessionFactory can be used to build sessions, which are roughly analogous to database connections. So the appropriate way is to use it in the using block. I can say **var session** equals **sessionFactory**.**OpenSession** and I'm going to want to do this inside of its transaction.

Once the session is opened, we can tell the session to begin a new transaction and we can then perform some logic in here. So perform some database logic and finally commit that transaction.

6. NHibernate – Basic ORM

In this chapter, we will be covering some **basic mapping** and you know that from the last chapter that we have the database table as well as the C# class definition. We now need a mapping that explains how to translate from C# to the database and back again.

So let's go ahead and add a new XML file by right clicking on the project in the solution explorer and select Add -> New Item...



Enter **Student.hbm.xml** in the name field. We need to specify a default assembly which is going to be **NHibernateDemoApp** and also specify a default namespace. This just shortens up a lot of the other type definitions that we're going to making in this file.

Following is the implementation in the XML file:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemoApp"
                   namespace="NHibernateDemoApp">
  <class name="Student">
    <id name="ID">
```

```
      <generator class="native"/>
    </id>
    <property name="LastName"/>
    <property name="FirstMidName"/>
  </class>
</hibernate-mapping>
```

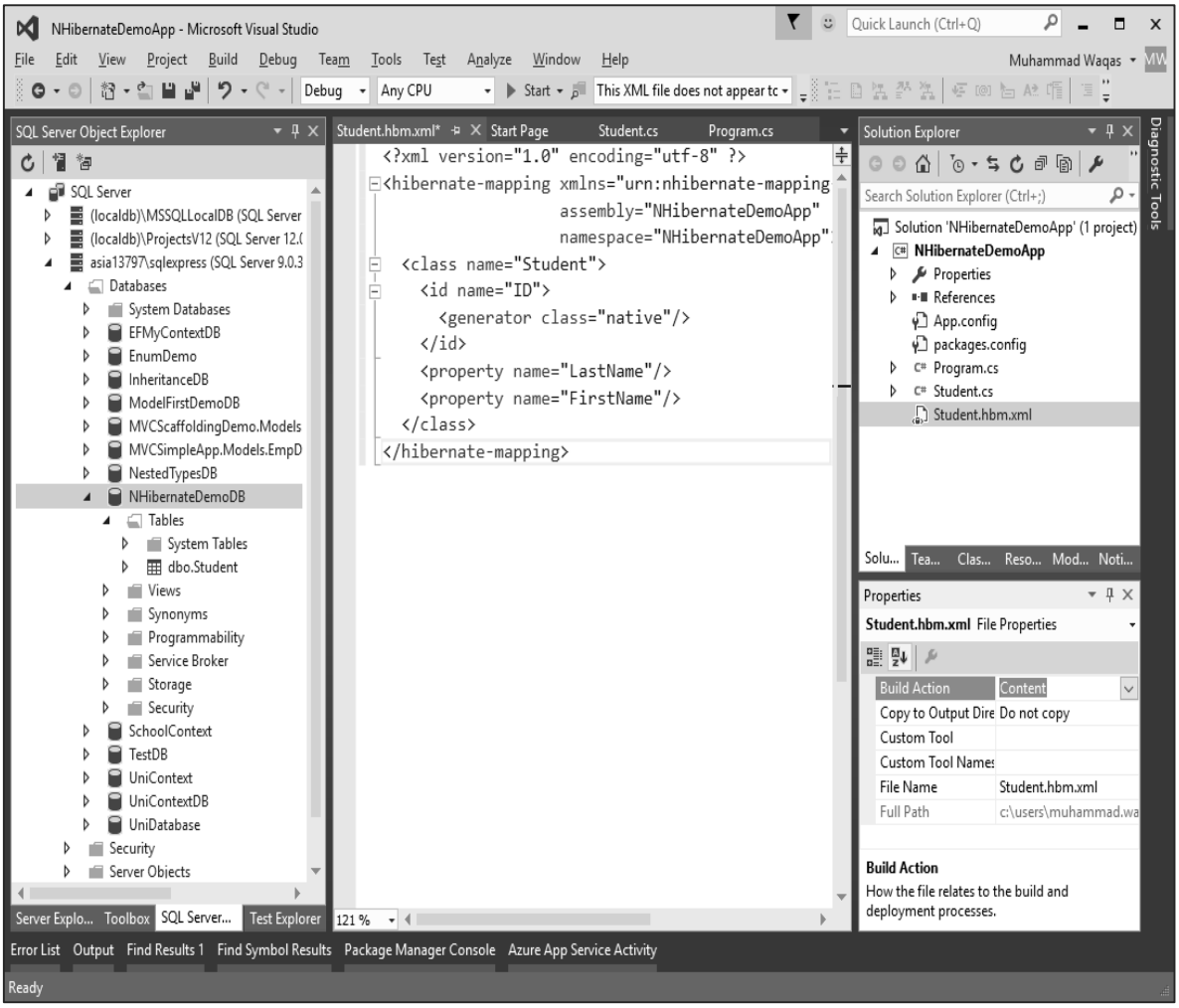
The next thing we need to define a class; this class is going to be our **Student class**. Next, we need to tell NHibernate the name of the id, which is ID and I also have to tell NHibernate how to generate ID's, so our generator is going to be of type native.

The native type generator means that in a database like SQL Server, it's going to use the identity column, the identity type.

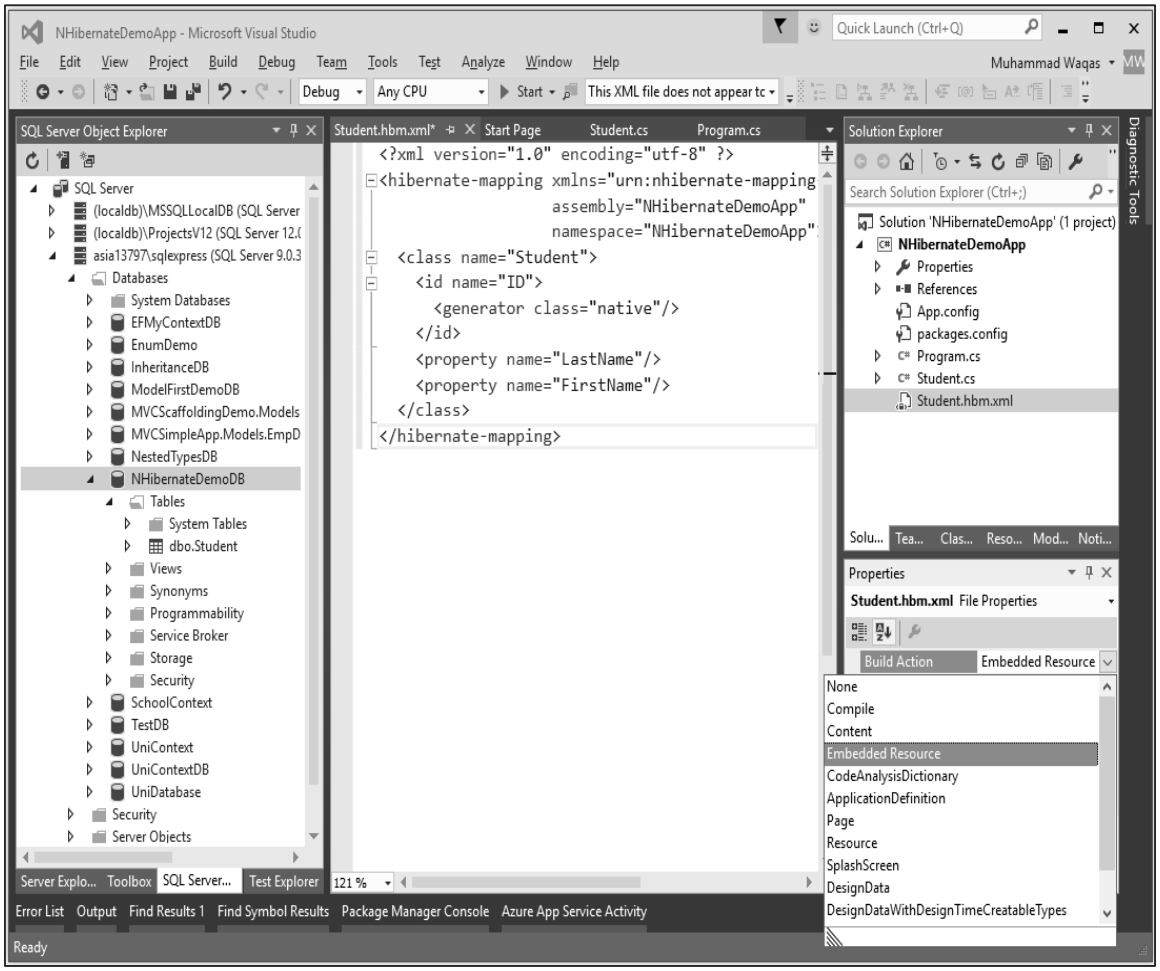
The next thing we have to do is to give the names of the properties. So, add two more properties for the FirstName, and LastName.

Now, we are reading these mapping files from the assembly. So the preferred way of doing this is to have these **HBM files** baked into your assembly. We can do this by simply setting a property.

Now right click on the project in the solution explorer and select Properties, you will see the **Build Action field** in which the Content is selected by default.



Select the embedded resource from the dropdown list.



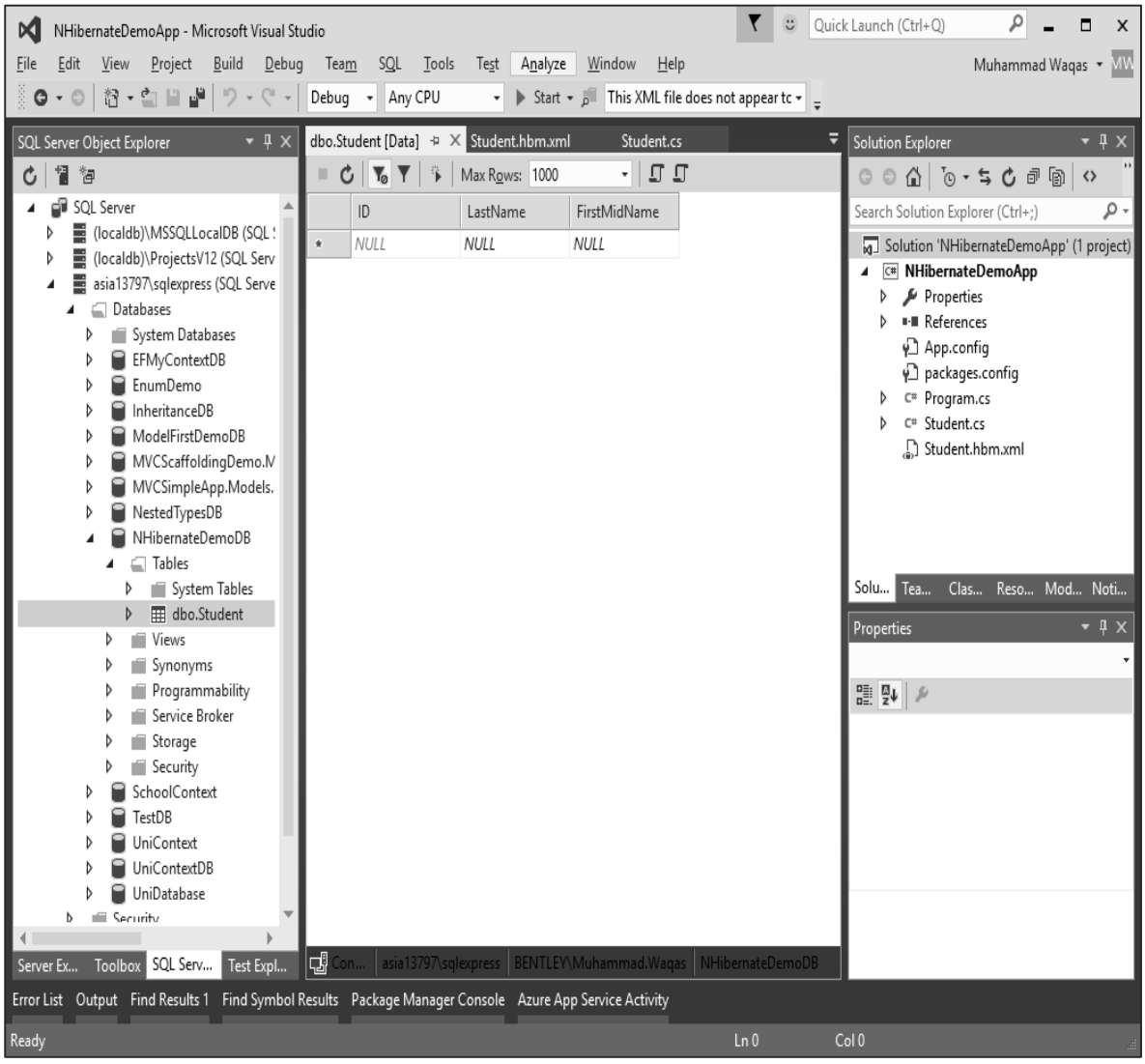
So this actually embeds that XML file inside of the **NHibernateDemoApp** assembly.

7. NHibernate – Basic CRUD Operations

In this chapter, we will cover the basic **CRUD operations**. Now that our system is ready to start, as we have successfully implemented our domain Student class, we have also defined the mapping files and configured NHibernate. We can now use some queries to perform CRUD operations.

Create Data

As you can see that we have no data in our Student table in **NHibernateDemoDB** database.



So to add some data, we need to perform the **Add/Create** operation as shown below.

```
using (var session = sefact.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var student1 = new Student
        {
            ID = 1,
            FirstMidName = "Allan",
            LastName = "Bommer"
        };
        var student2 = new Student
        {
            ID = 2,
            FirstMidName = "Jerry",
            LastName = "Lewis"
        };
        session.Save(student1);
        session.Save(student2);
        tx.Commit();
    }
    Console.ReadLine();
}
```

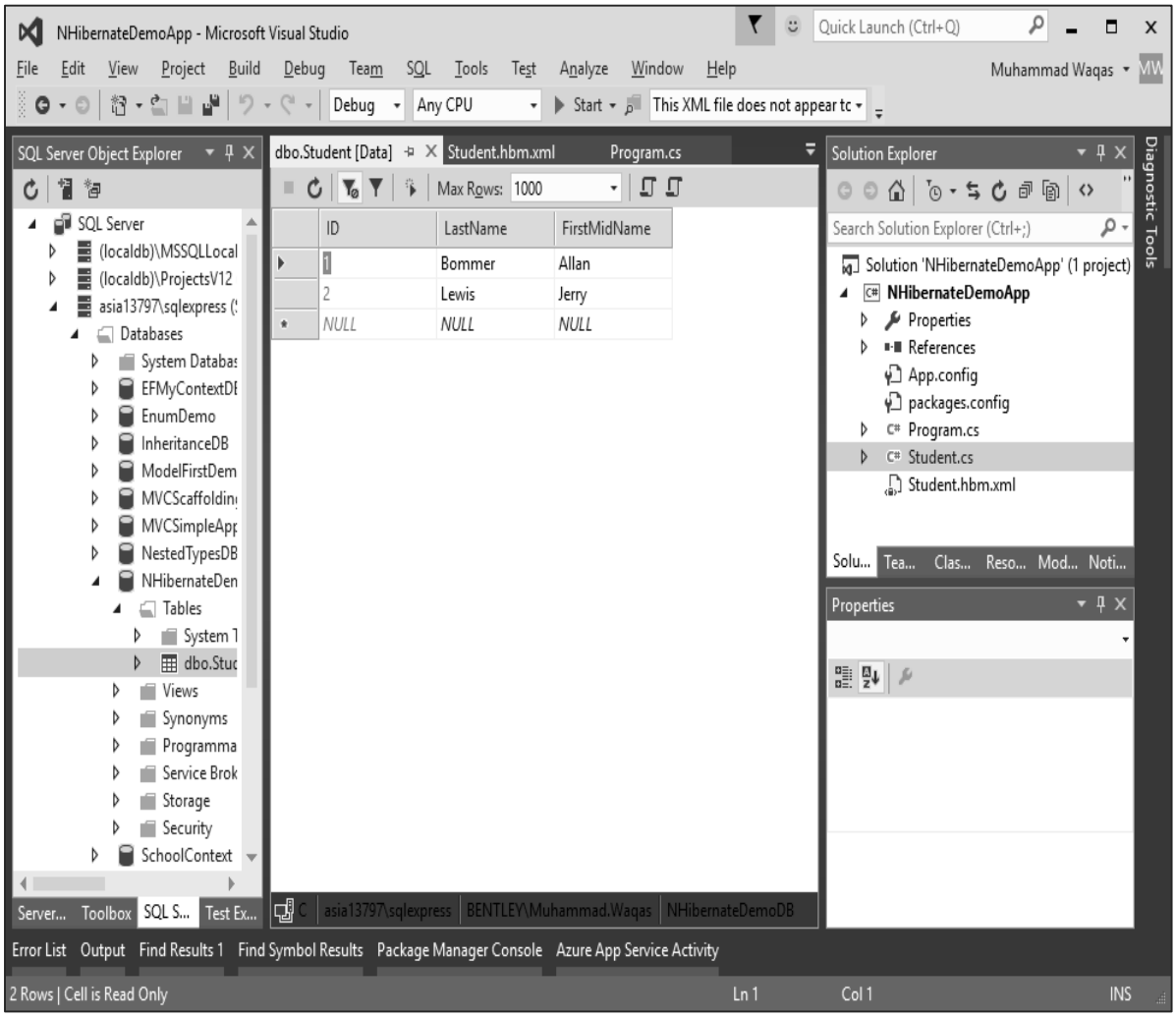
As you can see that we have created two students and then call the Save() method of the **OpenSession** and then call the Commit() of the **BeginTransaction**. Here is the complete implementation in **Program.cs** file

```
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;
```

```
namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
            });
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    var student1 = new Student
                    {
                        ID = 1,
                        FirstMidName = "Allan",
                        LastName = "Bommer"
                    };
                    var student2 = new Student
                    {
                        ID = 2,
                        FirstMidName = "Jerry",
                        LastName = "Lewis"
                    };
                    session.Save(student1);
                    session.Save(student2);
                    tx.Commit();
```

```
            }
                    Console.ReadLine();
                }
            }
        }
    }
```

Now let's run this application and then go to the SQL Server Object Explorer and refresh your database. You will see that the above two students are now added to the Student table in NHibernateDemoDB database.



Read Data from Student Table

You can see that now we have two records in our student table. To read these records from the table, we need to call the **CreateCriteria()** of OpenSession as shown in the following code.

```
using (var session = sefact.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var students = session.CreateCriteria<Student>().List<Student>();
        foreach (var student in students)
        {
            Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
        }
        tx.Commit();
    }
    Console.ReadLine();
}
```

So if you want the list of record then we can just simply say list of type Student.

Now use the **foreach** through all of the students and say print the ID, **FirstMidName** and **LastName** on the console. Now, let's run this application again and you will see the following output on the console window.

```
1       Allan   Bommer
2       Jerry   Lewis
```

You can also retrieve any record by specifying the ID in the **Get()** method of OpenSession using the following code.

```
using (var session = sefact.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var students = session.CreateCriteria<Student>().List<Student>();
        foreach (var student in students)
{
            Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
        }
```

```
        var stdnt = session.Get<Student>(1);
        Console.WriteLine("Retrieved by ID");
        Console.WriteLine("{0} \t{1} \t{2}", stdnt.ID, stdnt.FirstMidName,
stdnt.LastName);
        tx.Commit();

    }
    Console.ReadLine();
}
```

Now when you run your application, you will see the following output.

```
1       Allan   Bommer
2       Jerry   Lewis
Retrieved by ID
1       Allan   Bommer
```

Update Record

To update record in the table, we need to first fetch that particular record and then update that record by calling the **Update()** method of OpenSession as shown in the following code.

```
using (var session = sefact.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var students = session.CreateCriteria<Student>().List<Student>();
        foreach (var student in students)
        {
            Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
        }

        var stdnt = session.Get<Student>(1);
        Console.WriteLine("Retrieved by ID");
        Console.WriteLine("{0} \t{1} \t{2}", stdnt.ID, stdnt.FirstMidName,
stdnt.LastName);
```

```
Console.WriteLine("Update the last name of ID = {0}", stdnt.ID);
        stdnt.LastName = "Donald";
        session.Update(stdnt);
        Console.WriteLine("\nFetch the complete list again\n");
        foreach (var student in students)
        {
             Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
        }
        tx.Commit();

    }
    Console.ReadLine();
}
```

Now when you run your application, you will see the following output.

```
1       Allan   Bommer
2       Jerry   Lewis
Retrieved by ID
1       Allan   Bommer
Update the last name of ID = 1

Fetch the complete list again

1       Allan   Donald
2       Jerry   Lewis
```

As you can see, LastName of ID equal 1 is updated from Bommer to Donald.

Delete Record

To delete any record from the table, we need to first fetch that particular record and then delete that record by calling the **Delete()** method of OpenSession as shown in the following code.

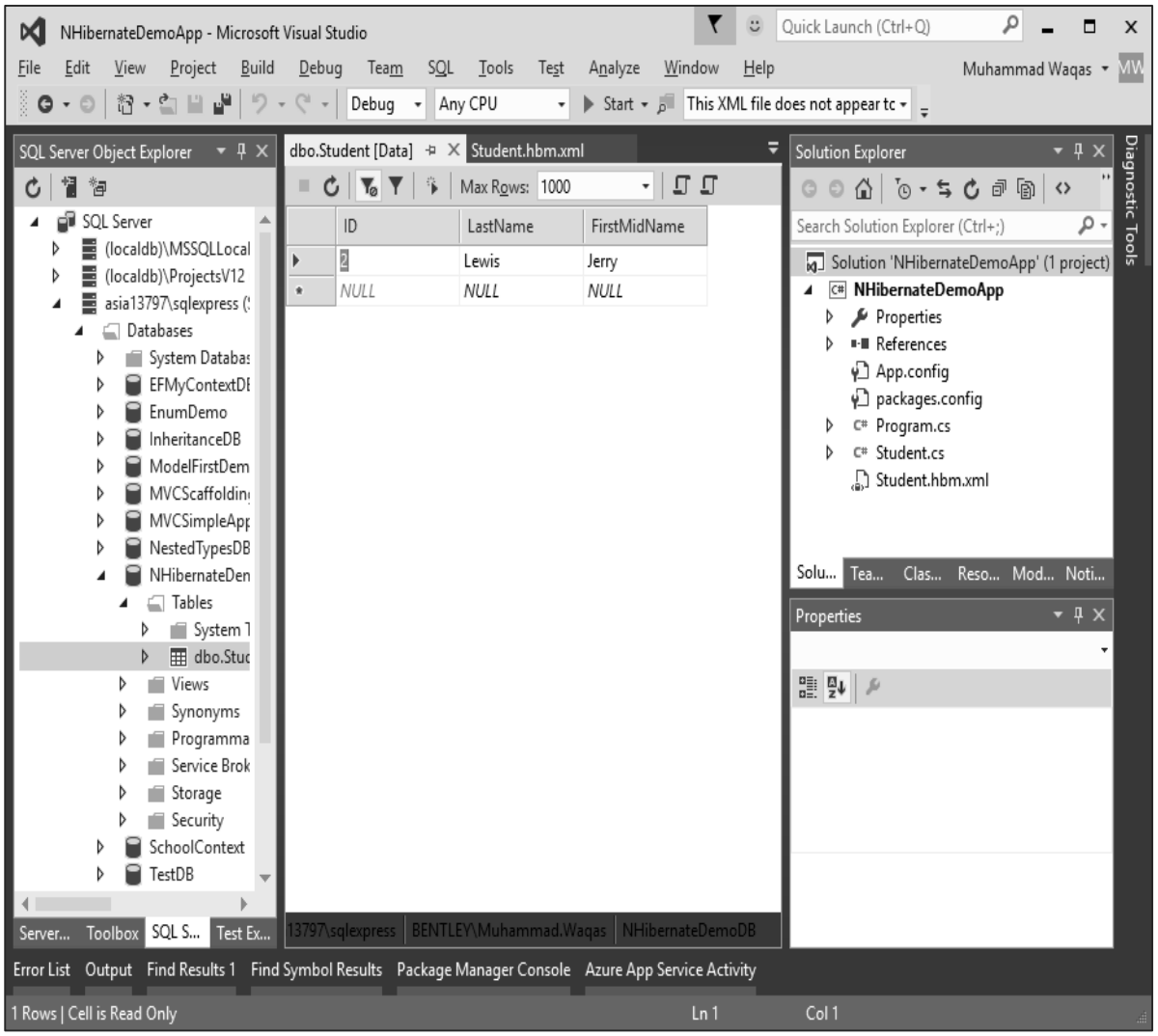
```
using (var session = sefact.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var students = session.CreateCriteria<Student>().List<Student>();
```

```
 foreach (var student in students)
        {
            Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
        }

        var stdnt = session.Get<Student>(1);
        Console.WriteLine("Retrieved by ID");
        Console.WriteLine("{0} \t{1} \t{2}", stdnt.ID, stdnt.FirstMidName,
stdnt.LastName);

        Console.WriteLine("Delete the record which has ID = {0}", stdnt.ID);
        session.Delete(stdnt);

        Console.WriteLine("\nFetch the complete list again\n");
        foreach (var student in students)
        {
             Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
        }
        tx.Commit();

    }
    Console.ReadLine();
}
```

Now when you run your application, you will see the following output.

```
1       Allan   Donald
2       Jerry   Lewis
Retrieved by ID
1       Allan   Bommer
Delete the record which has ID = 1
Fetch the complete list again
2       Jerry   Lewis
```

As you can see that the record which has ID equal to 1 is no longer available in the database. You can also see the database in the SQL Server Object Explorer.



8. NHibernate – NHibernate Profiler

In this chapter, we will be understanding how all the records from the database are **retrieved, updated, created, and deleted** and how exactly these queries are performed?

To understand all these, we can simply add an option into our configuration, which logs the SQL in the console. Here is the simple statement that will log the SQL query –

```
x.LogSqlInConsole = true;
```

Now, we have two records in our student table in the NHibernateDemoDB database. Let's retrieve all the records from the database as shown in the following code.

```
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
                x.LogSqlInConsole = true;
            });
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
```

```
     using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    Console.WriteLine("\nFetch the complete list again\n");
                    var students =
session.CreateCriteria<Student>().List<Student>();
                    foreach (var student in students)
                    {
                        Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
                    }
                    tx.Commit();
                }
                Console.ReadLine();
            }
        }
    }
}
```

So let's go ahead and run this application again, and you will see the following output:

```
NHibernate: SELECT this_.ID as ID0_0_, this_.LastName as LastName0_0_,
this_.FirstMidName as FirstMid3_0_0_ FROM Student this_


Fetch the complete list again


3       Allan   Bommer
4       Jerry   Lewis
```

As you can see, the **select clause** being sent to the database, it is actually like clause which will retrieve the ID, FirstMidName and LastName. So all this is being sent to the database and processed there rather than having a lot of records brought back to your server and processed on the server side.

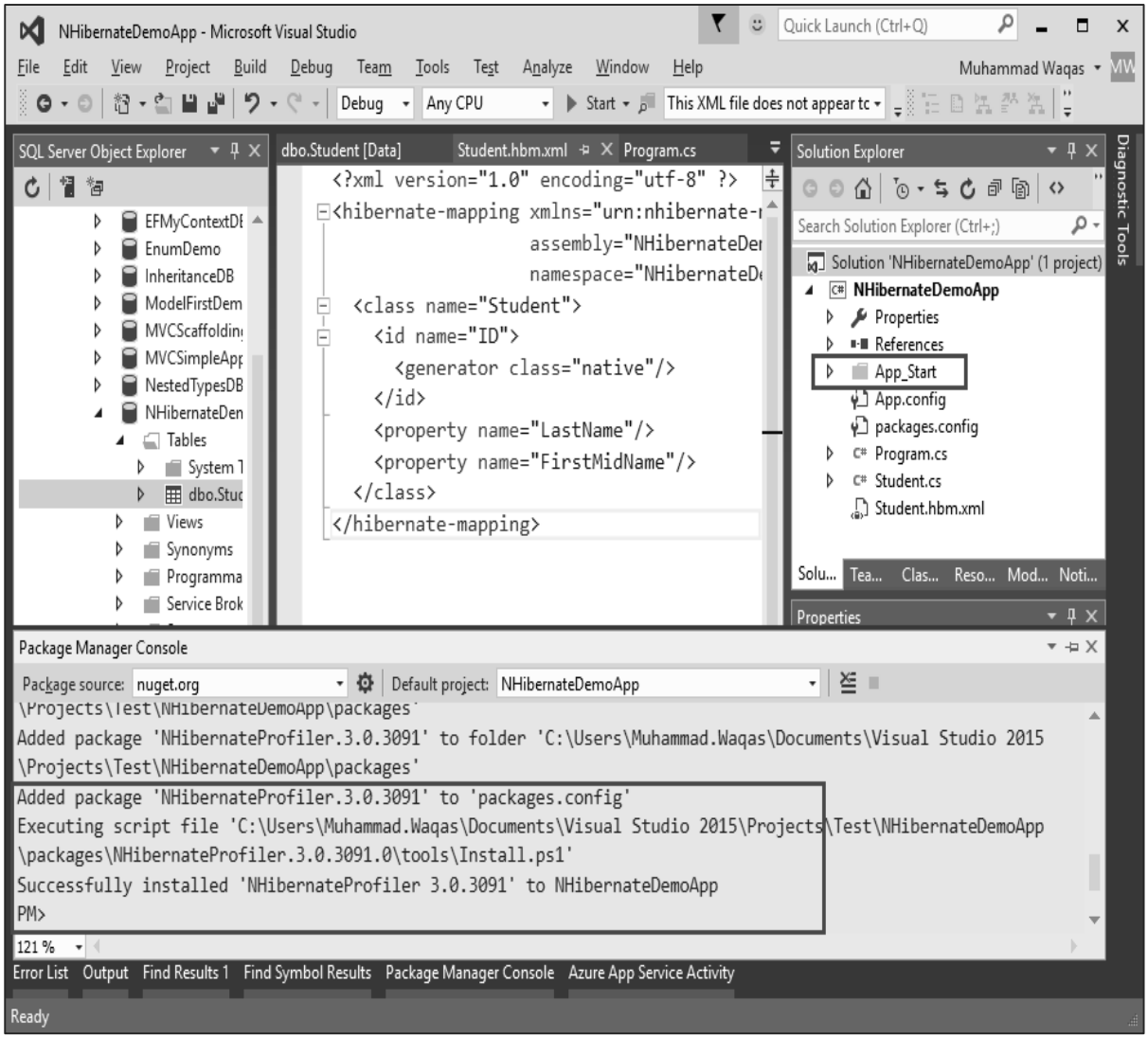
NHibernate Profiler

Another way to look at these results is to use NHibernate Profiler. NHibernate Profiler is a commercial tool, but is it very useful for working with NHibernate applications. You can easily install the NHibernate Profiler into your application from NuGet.

Let's go to the NuGet Manager console from the Tools menu by selecting the NuGet Package Manager -> Package Manager Console. It will open the Package Manager Console window. Enter the following command and press enter.

```
PM> install-package NHibernateProfiler
```

It will install all the required binaries for the NHibernate Profiler, once it is successfully installed you will see the following message.



You will also see that the NHibernate Profiler is launched, once it is installed. It will require a license to use it, but for demo purposes, we can use the 30-days trial version of NHibernate Profiler.

Now, NHibernate Profiler is optimized to work with web applications and you will see that it has added **App_Start folder** in the solution explorer. To keep all these simple, delete the App_Start folder and also you will observe that one statement is added at the start of the Main method in Program class.

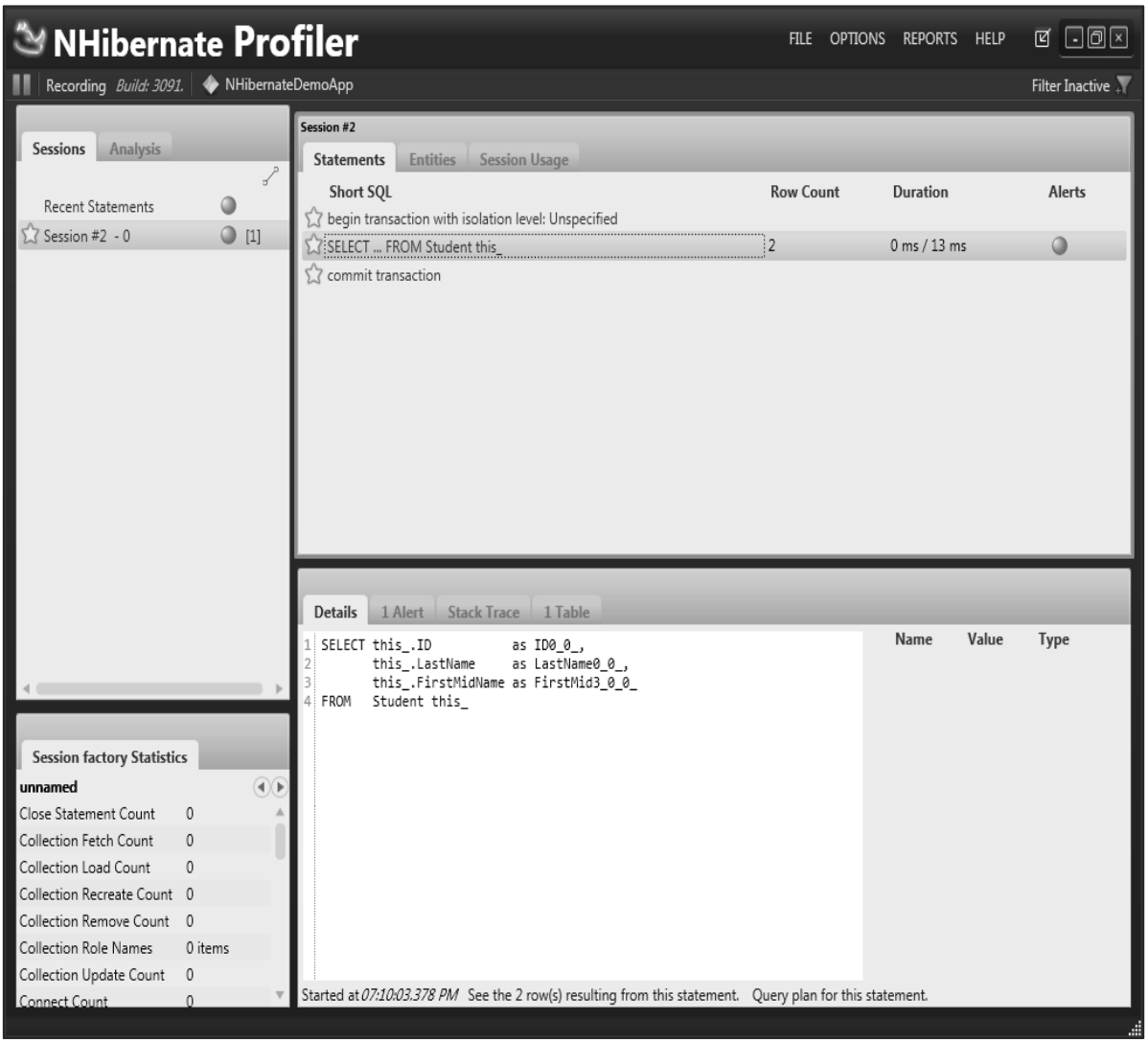
```
App_Start.NHibernateProfilerBootstrapper.PreStart();
```

Remove this statement as well and just add a simple call **NHibernateProfiler.Initialize** as shown in the following code.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
                x.LogSqlInConsole = true;
            });
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())

                {
                    var students =
session.CreateCriteria<Student>().List<Student>();
                    Console.WriteLine("\nFetch the complete list again\n");
```

```
                    foreach (var student in students)
                {
                    Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstMidName, student.LastName);
                }
                tx.Commit();
            }
            Console.ReadLine();
        }
    }
}
```

Now when you run the application, it's going to send data over to the NHibernate Profiler application.



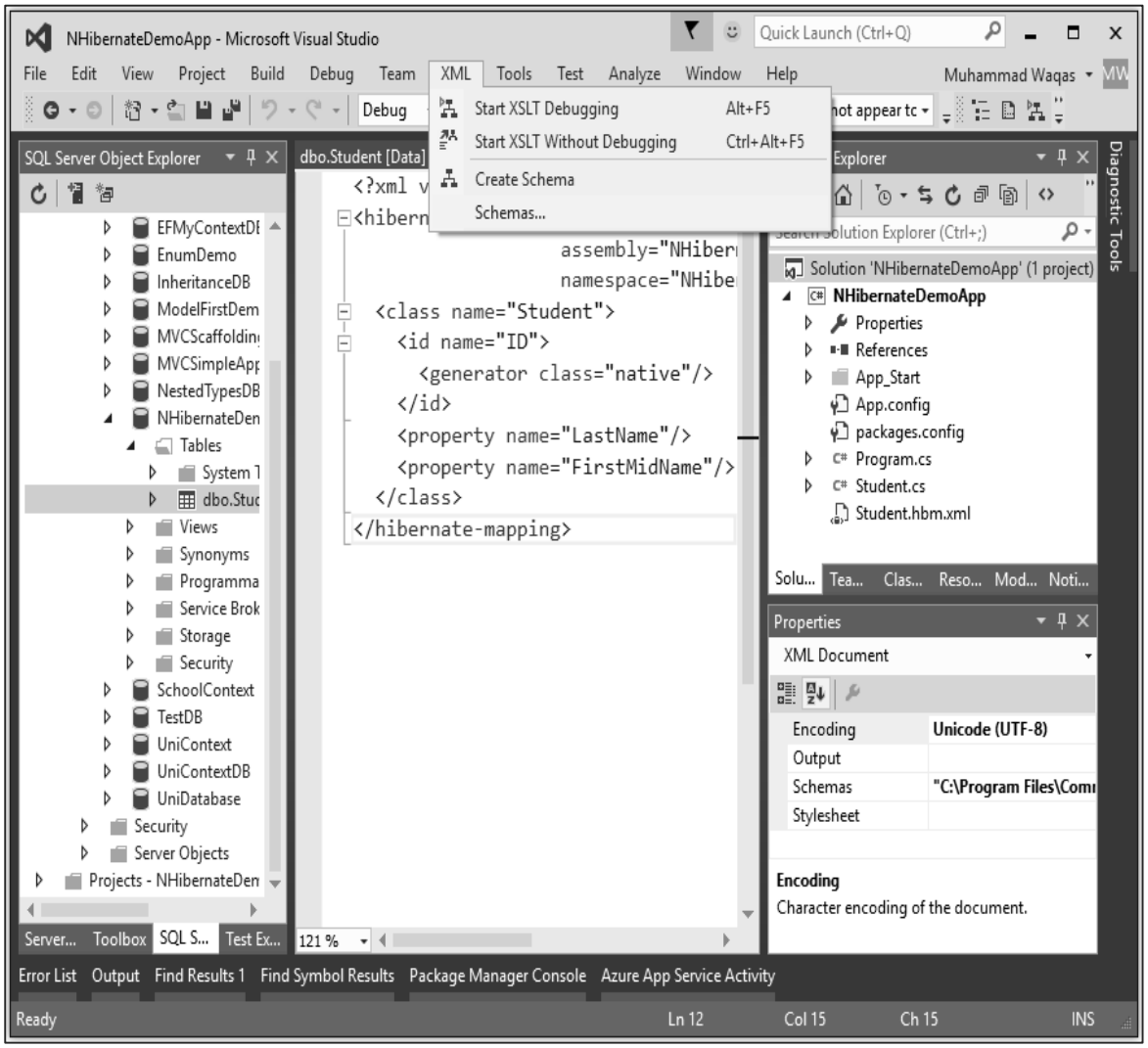
You can see here, we've got a nice display that shows that we've started the transaction, what the SQL is doing to the database in a nice format.

So this is very useful for determining what exactly is happening inside of your NHibernate application. It becomes incredibly useful once the application gets to a certain level of complexity, where you need something more like a SQL Profiler, but with the knowledge of NHibernate.

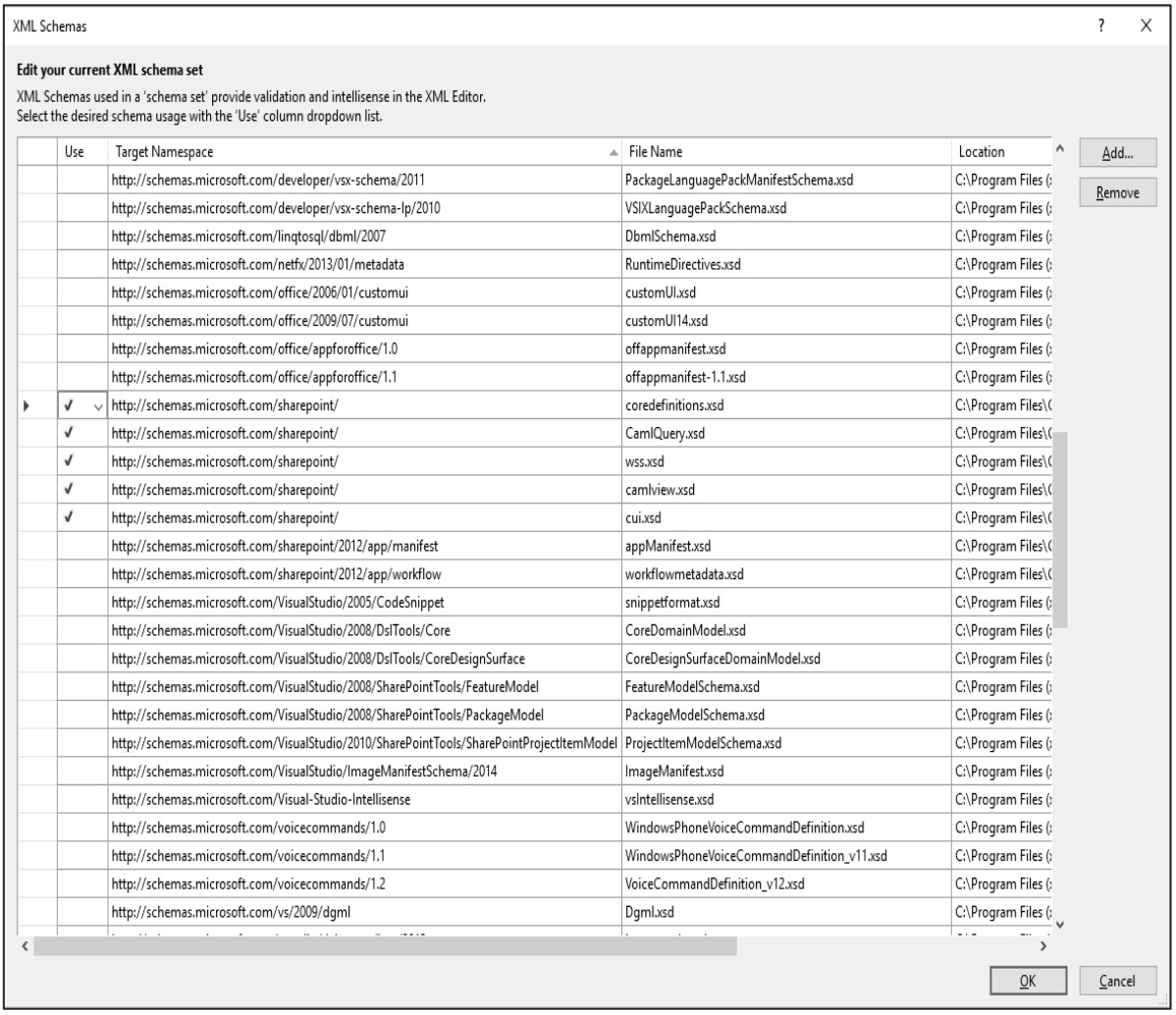
9. NHibernate – Add IntelliSense to Mapping File

In this chapter, we will add **IntelliSense** to our NHibernate mapping files **(*.hbm.xml files)**. As you have observed while mapping the domain Student class that currently we don't have IntelliSense available. It's very useful to have the **XML schemas** available. So in this chapter you will understand how to add IntelliSense in Visual Studio for these NHibernate XML files.

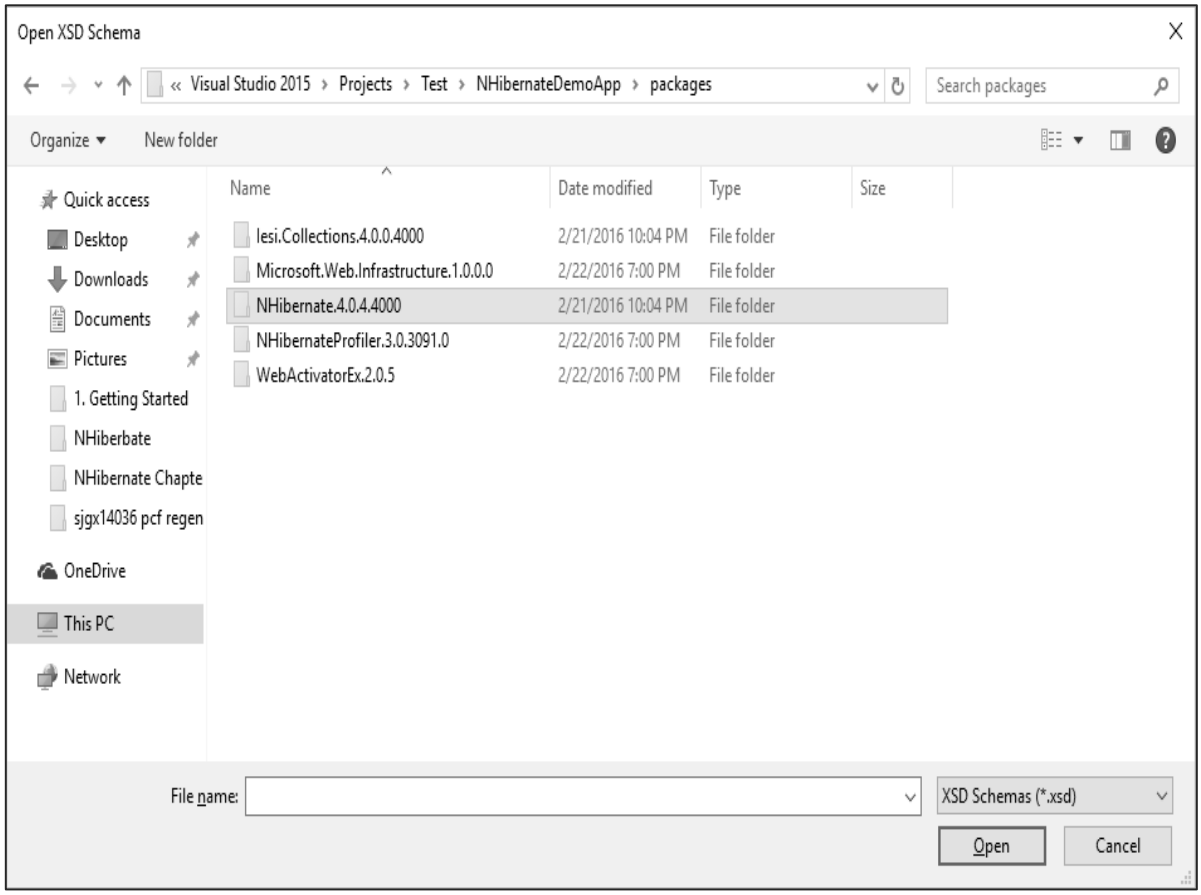
Open the mapping file and you will see that XML menu option appears in the main menu.



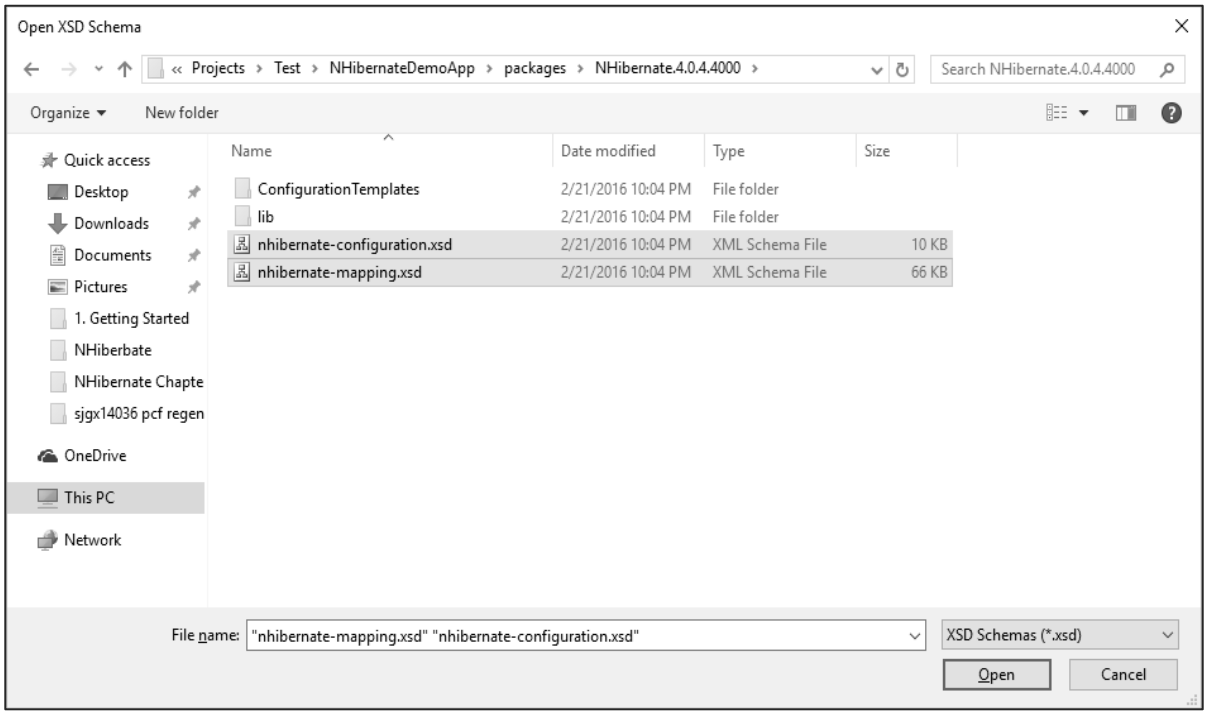
Select the XML -> Schemas… menu option and it will display the XML Schemas dialog box.



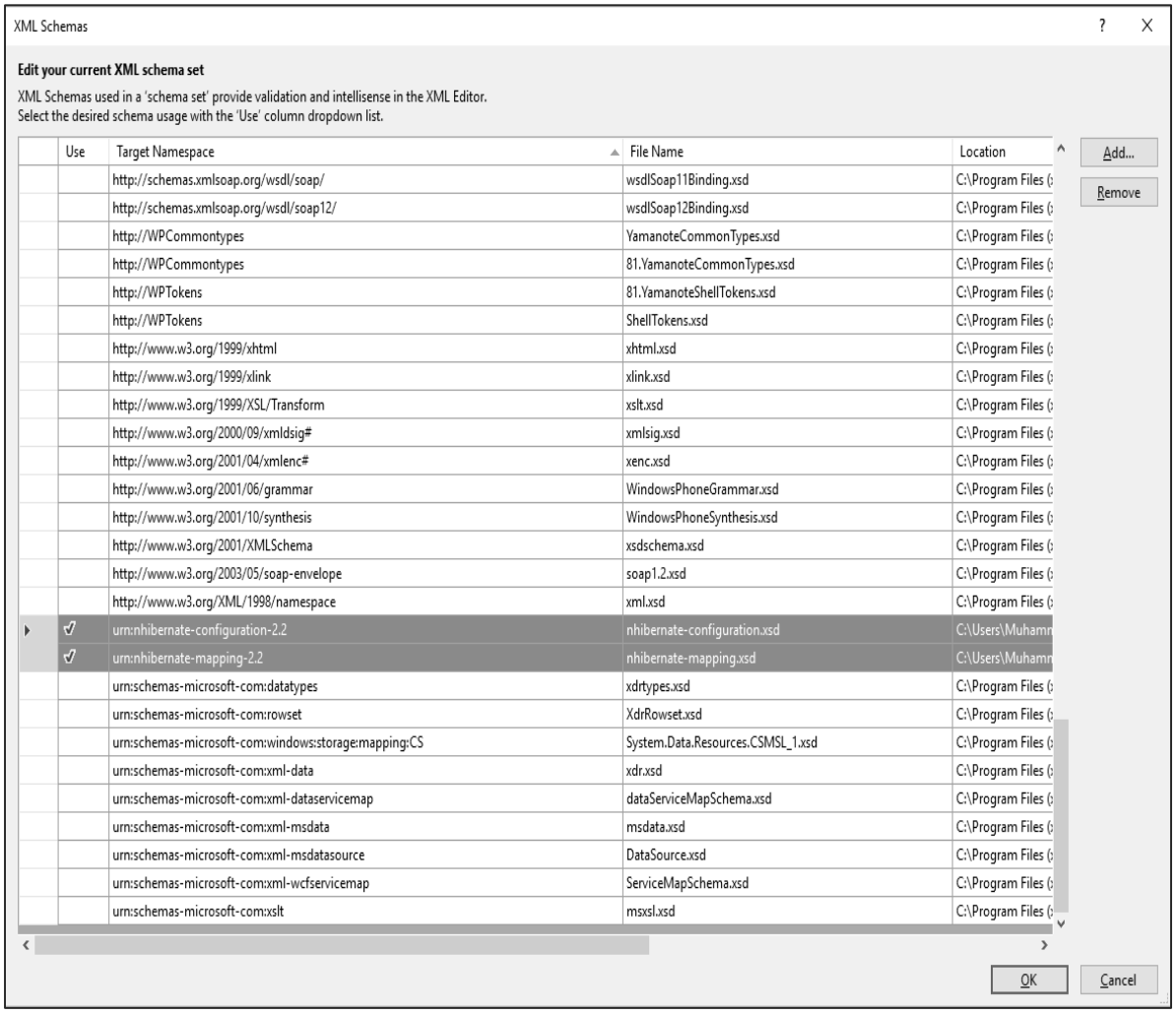
Select the Add… button which is on the top right of the dialog box, which opens the file dialog. Now go to the **packages folder,** which is in the Solution folder of your project and you will see the different packages included in your project.



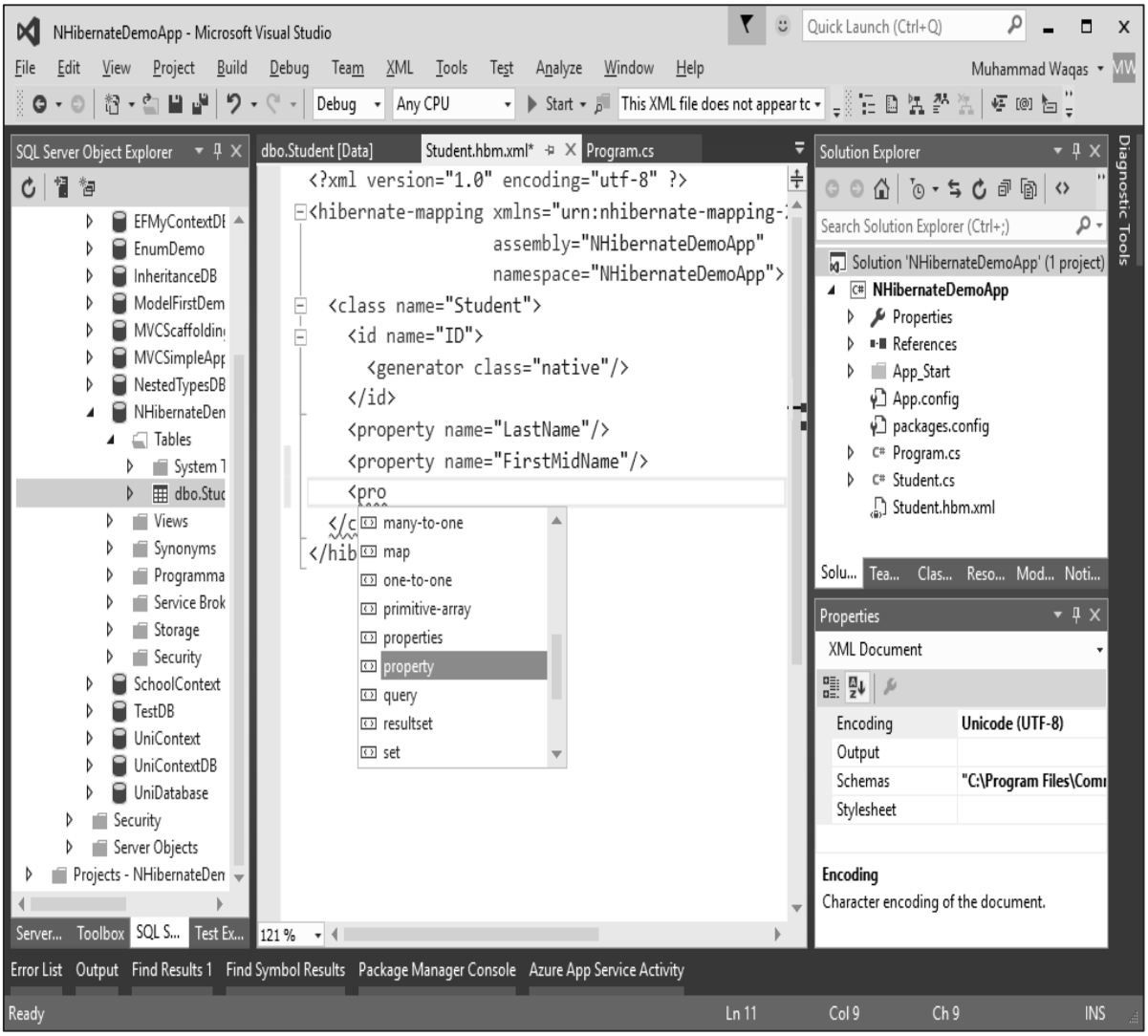
Now, double-click on **NHibernate.4.*** folder** and you will see the two schemas (*.xsd) files or XML schema definition files that define the NHibernate configuration and mapping.



Select these two schema files and click Open button.



You can see that the NHibernate schemas are added into the XML Schemas dialog. Click the OK button. Now, let's start a new property tag and you will see that we've got full IntelliSense on here.



IntelliSense is now available for you which saves a lot of time during object-relational mapping.

10. NHibernate – Data Types Mapping

In this chapter, we will be covering mapping data types. Mapping entities is straightforward, entity classes are always mapped to database tables using **<class>, <subclass>, and <joined-subclass>** mapping elements. Value types need something more, which is where mapping types are required.

NHibernate is able to map a wide variety of data types. Here is the list of the most common data types which are supported.

Mapping type	.NET type	System.Data.DbType
Int16	System.Int16	DbType.Int16
Int32	System.Int32	DbType.Int32
Int64	System.Int64	DbType.Int64
Single	System.Single	DbType.Single
Double	System.Double	DbType.Double
Decimal	System.Decimal	DbType.Decimal
String	System.String	DbType.String
AnsiString	System.String	DbType.AnsiString
Byte	System.Byte	DbType.Byte
Char	System.Char	DbType.StringFixedLength—one character
AnsiChar	System.Char	DbType.AnsiStringFixedLength—one character
Boolean	System.Boolean	DbType.Boolean
Guid	System.Guid	DbType.Guid
PersistentEnum	System.Enum(an enumeration)	DbType for the underlying value
TrueFalse	System.Boolean	DbType.AnsiStringFixedLength—either 'T' or 'F'
YesNo	System.Boolean	DbType.AnsiStringFixedLength—either 'Y' or 'N'
DateTime	System.DateTime	DbType.DateTime—ignores milliseconds
Ticks	System.DateTime	DbType.Int64
TimeSpan	System.TimeSpan	DbType.Int64

Timestamp	System.DateTime	DbType.DateTime—as specific as the database supports
Binary	System.Byte[]	DbType.Binary
BinaryBlob	System.Byte[]	DbType.Binary
StringClob	System.String	DbType.String
Serializable	Any System.Object marked with SerializableAttribute	DbType.Binary
CultureInfo	System.Globalization.CultureInfo	DbType.String—five characters for culture
Type	System.Type	DbType.String holding the Assembly Qualified Name

The above given table explains in detail the below mentioned pointers.

- Everything from simple numeric types to strings, which can be mapped in a variety of ways using **varchars, chars** etc. as well as string blobs and all the variety of types that databases support.

- It is also able to map **Booleans**, both to fields using zeros and ones, character fields that contain true, false or T and F.

- There's a wide variety of ways of defining how that maps to the back end, boolean values in the database.

- We can handle the mapping of **DateTime**, both including and excluding time zone offsets, daylight savings time, etc.

- We can also map **enumerations**; we can map these to strings or to their underlying numeric values.

Let’s have a look into a simple example in which we have the same property names both in the database as well as in the Student class.

Now let’s change the FirstMidName to FirstName in the student class, where we will not change the FirstMidName column, but we will see how to tell NHibernate know to carry out this conversion. Following is the updated student class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace NHibernateDemoApp
```

```
{
    class Student
    {
        public virtual int ID { get; set; }
        public virtual string LastName { get; set; }
        public virtual string FirstName { get; set; }
    }
}
```

Here is the implementation of NHibernate mapping file.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemoApp"
                   namespace="NHibernateDemoApp">
  <class name="Student">
    <id name="ID">
      <generator class="native"/>
    </id>
    <property name="LastName"/>
    <property name="FirstName"
              column="FirstMidName"
              type="String"/>
  </class>
</hibernate-mapping>
```

In this example, assume that the FirstName field is a .NET string, and the FirstMidName column is a **SQL nvarchar**. Now to tell NHibernate how to carry out this conversion, set the name equal to **FirstName** and column equal to **FirstMidName** and specify the mapping type equal to String, which is appropriate for this particular conversion.

The following is a **Program.cs** file implementation.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;
```

```
namespace NHibernateDemoApp
{    class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
                x.LogSqlInConsole = true;
            });
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    var students =
session.CreateCriteria<Student>().List<Student>();
                    Console.WriteLine("\nFetch the complete list again\n");
                    foreach (var student in students)
                    {
                        Console.WriteLine("{0} \t{1} \t{2}", student.ID,
student.FirstName, student.LastName);                    }
                    tx.Commit();
                }
                Console.ReadLine();            }
        }
    }

}
```

Now when you run your application, you will see the following output.

```
NHibernate: SELECT this_.ID as ID0_0_, this_.LastName as LastName0_0_,
this_.FirstMidName as FirstMid3_0_0_ FROM Student this_

Fetch the complete list again

3       Allan   Bommer

4       Jerry   Lewis
```

As you can see that it has mapped the different property name to column name in the database.

Let's take a look at another example in which we will add another property in the Student class of **enum** type. Here is the Student class implementation.

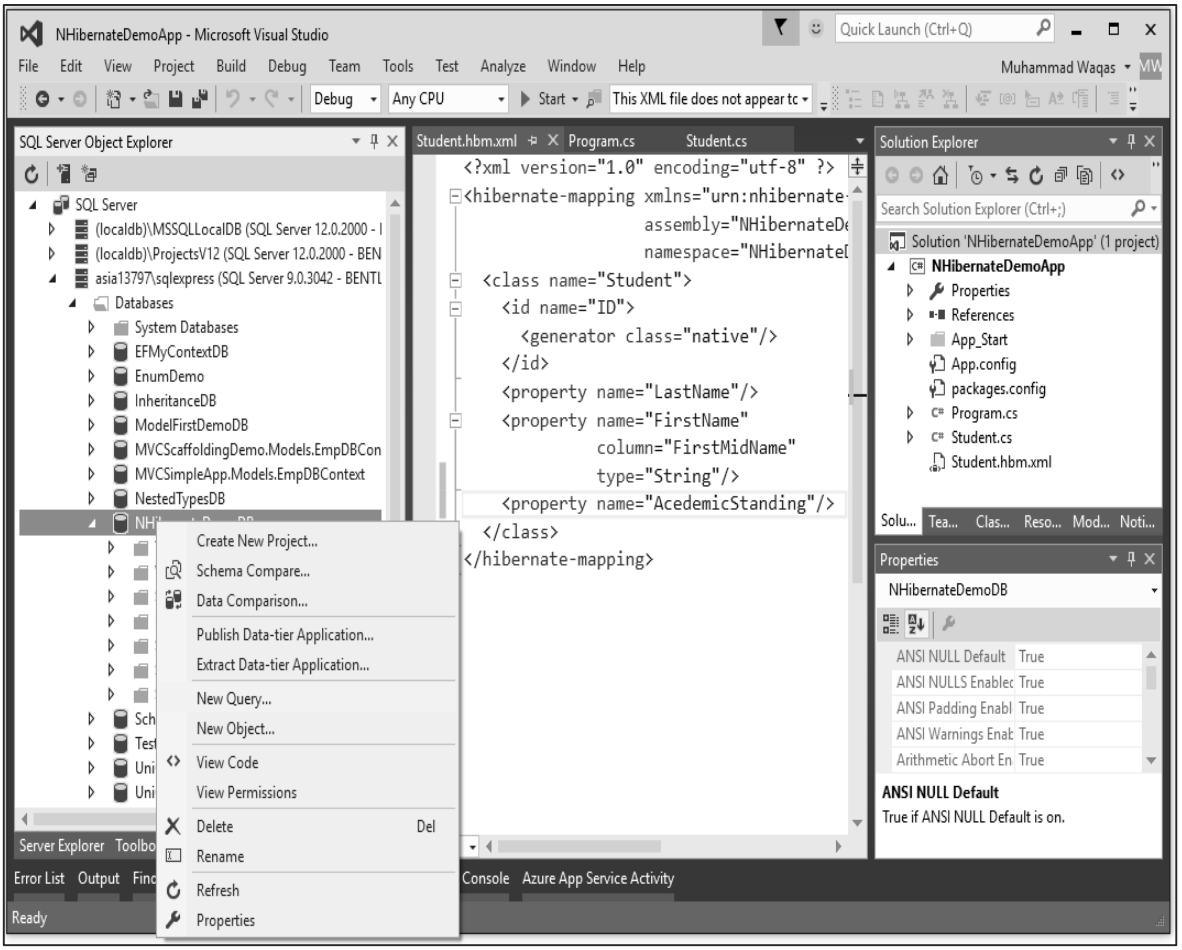
```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NHibernateDemoApp

{

    class Student

    {

        public virtual int ID { get; set; }

        public virtual string LastName { get; set; }

        public virtual string FirstName { get; set; }

        public virtual StudentAcademicStanding AcademicStanding { get; set; }

    }

    public enum StudentAcademicStanding

    {

        Excellent,

        Good,

        Fair,

        Poor,

        Terrible

    }

}
```

As you can see that enumeration has a variety of different values that it can possibly have such as, Excellent, Good, Fair, Poor and Terrible.

Jumping over to the mapping file, you can see that each of these properties are listed out in the mapping file including the newly added **AcademicStanding** property.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemoApp"
                   namespace="NHibernateDemoApp">
  <class name="Student">
    <id name="ID">
      <generator class="native"/>
    </id>
    <property name="LastName"/>
    <property name="FirstName"
              column="FirstMidName"
              type="String"/>
    <property name="AcademicStanding"/>
  </class>
</hibernate-mapping>
```

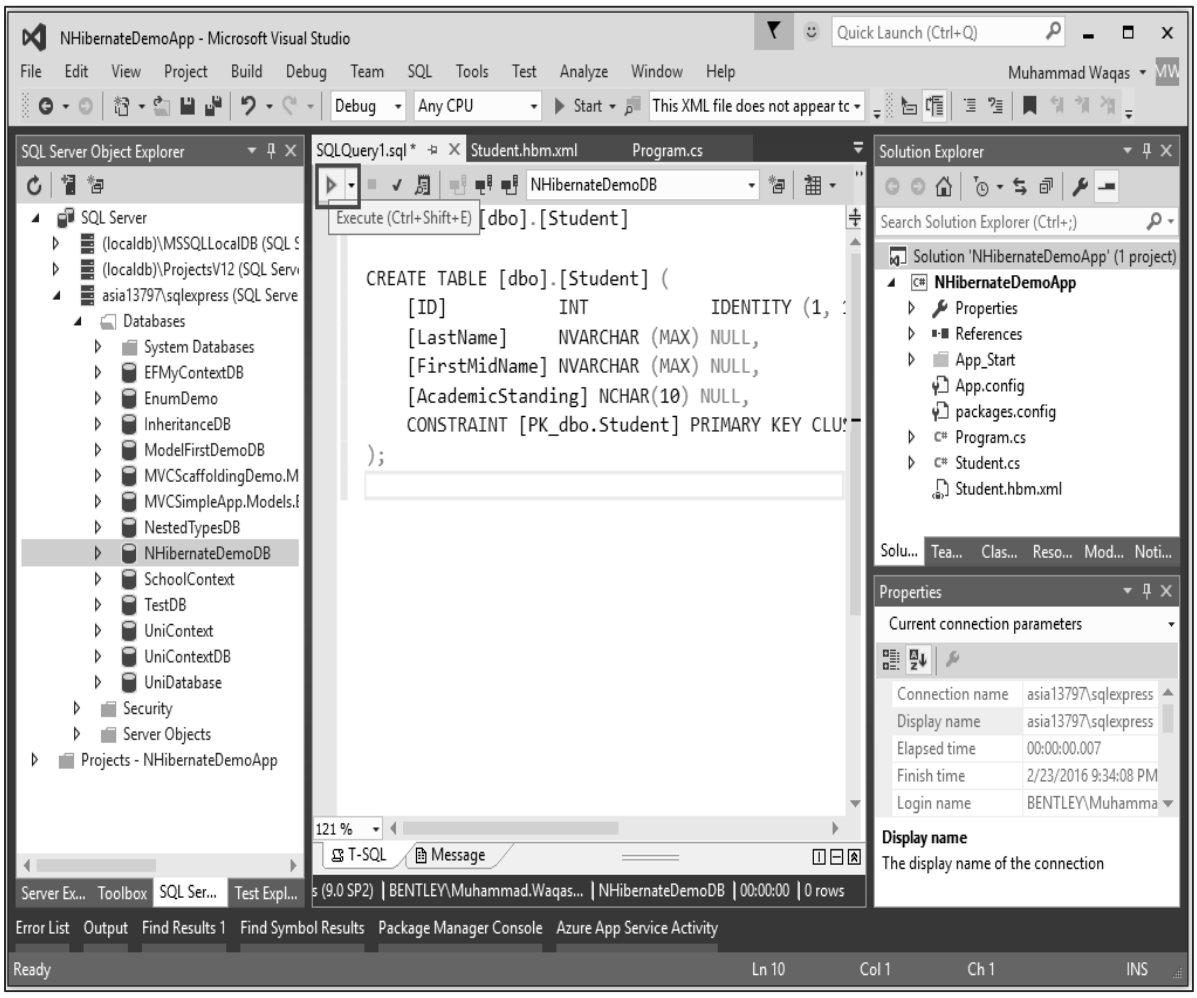
Now we also need to change the database as well, so go to the SQL Server Object Explorer and right-click on the database and select the New Query… option.



It will open the query editor and then specify the below query.

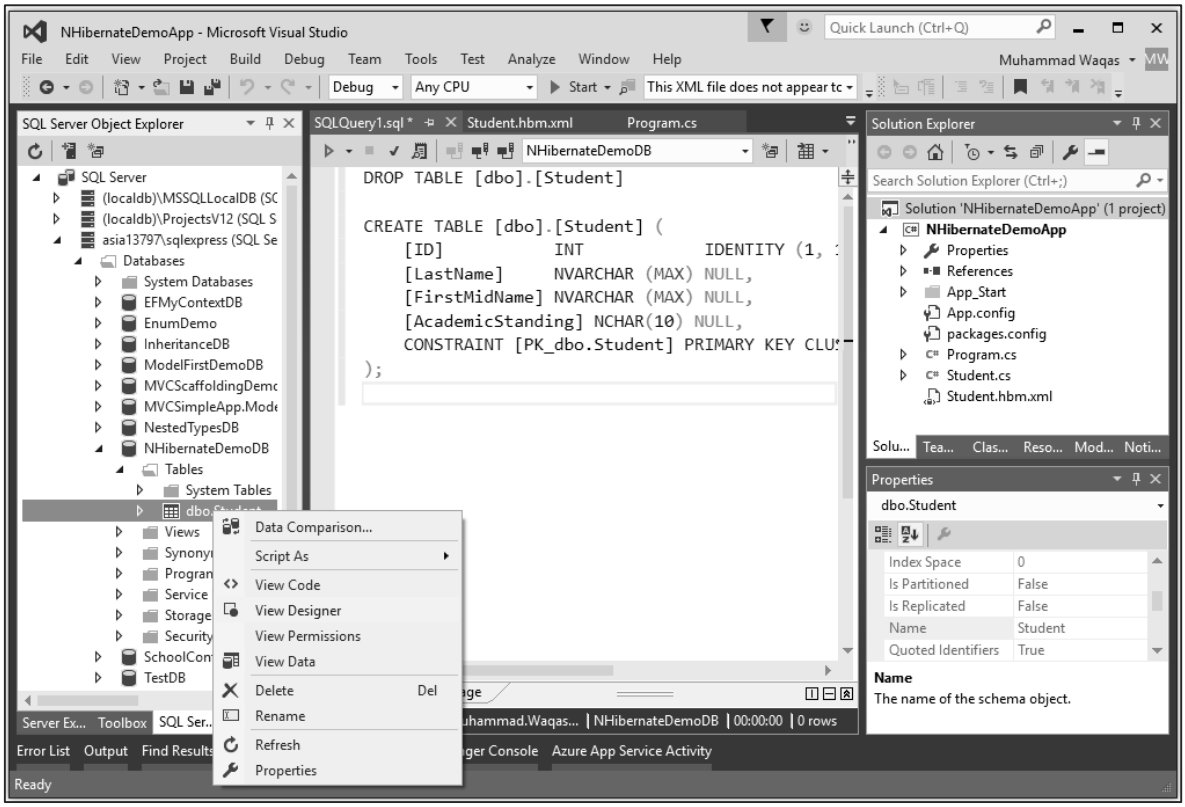
```
DROP TABLE [dbo].[Student]

CREATE TABLE [dbo].[Student] (
    [ID]           INT            IDENTITY (1, 1) NOT NULL,
    [LastName]     NVARCHAR (MAX) NULL,
    [FirstMidName] NVARCHAR (MAX) NULL,
    [AcademicStanding] NCHAR(10) NULL,
    CONSTRAINT [PK_dbo.Student] PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

This query will first drop the existing student table and then create a new table.

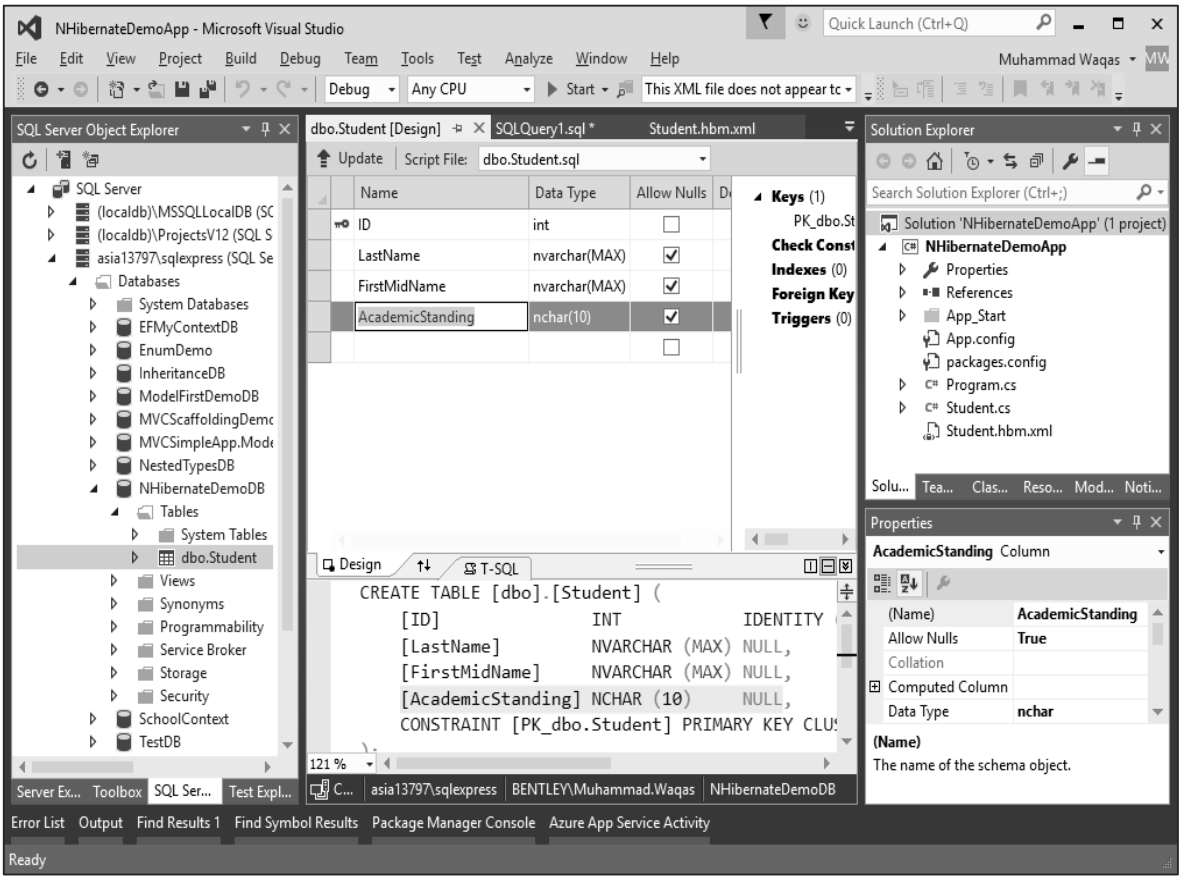


Clcik on the Execute icon as shown above. Once the query is executed succesfully then you see a message.

Expand database and Table dropdown, and then right-click on the Student table and select View Designer.



Now, you will see the newly created table, which also has the new property AcademicStanding.



Let's add two records as shown in the following **Program.cs** file.

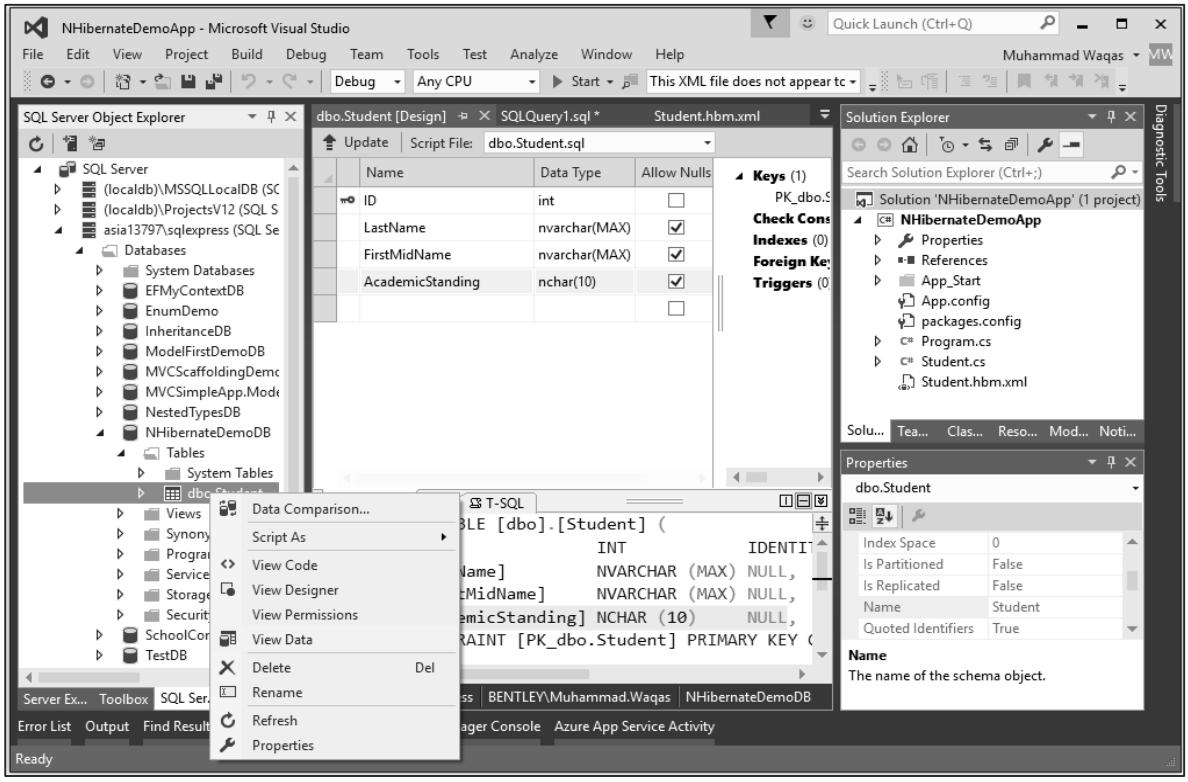
```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
            });
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    var student1 = new Student
                    {
                        ID = 1,
                        FirstName = "Allan",
                        LastName = "Bommer",
```

```
                    AcademicStanding = StudentAcademicStanding.Excellent
                    };

                    var student2 = new Student
                    {
                        ID = 2,
                        FirstName = "Jerry",
                        LastName = "Lewis",
                        AcademicStanding = StudentAcademicStanding.Good
                    };

                    session.Save(student1);
                    session.Save(student2);

                    var students =
session.CreateCriteria<Student>().List<Student>();
                    Console.WriteLine("\nFetch the complete list again\n");
                    foreach (var student in students)
                    {
                        Console.WriteLine("{0} \t{1} \t{2} \t{3}", student.ID,
student.FirstName, student.LastName, student.AcademicStanding);
                    }
                    tx.Commit();
                }
                Console.ReadLine();
            }
        }
    }
}
```

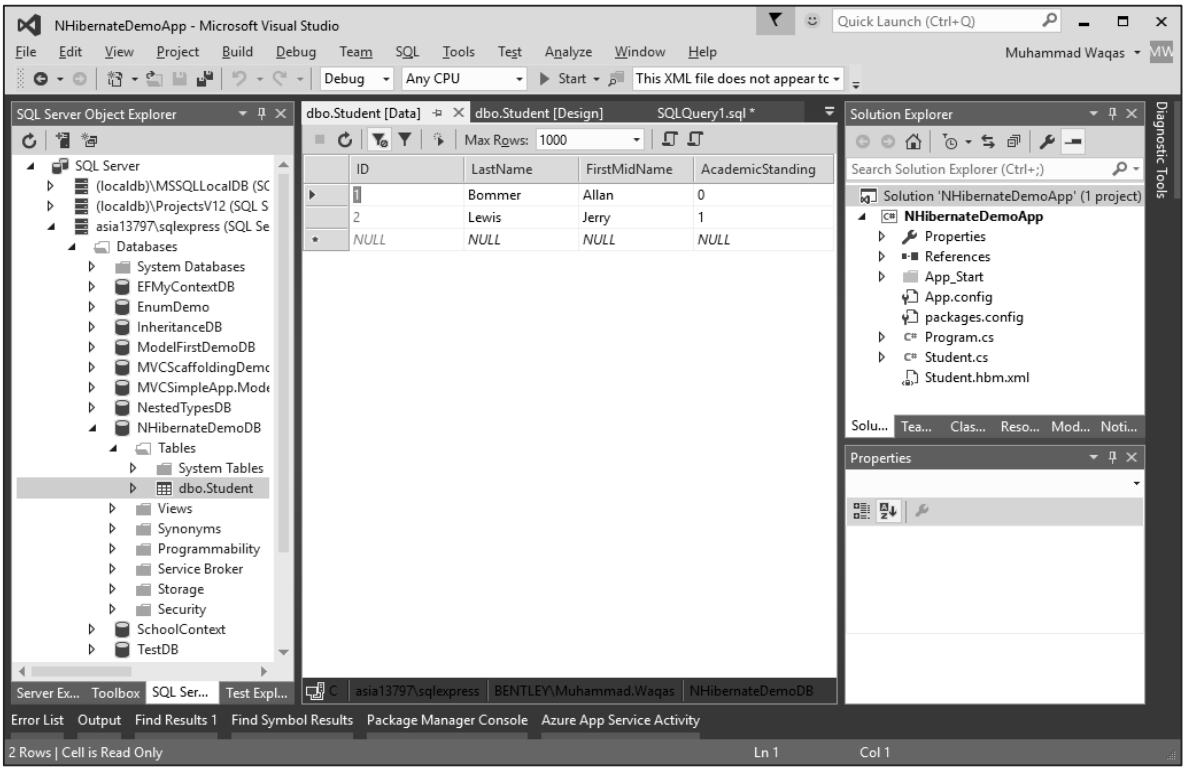
Now let's run your application and you will see the following output on your console window.

```
Fetch the complete list again


1       Allan   Bommer  Excellent
2       Jerry   Lewis   Good
```

Now let's have look into the database by right clicking on the Student Table.



Select View Data and you will see the two records in the student table as shown in the following screenshot.



You can see that two records are added and Allan has AcademicStanding 0 and Jerry has AcademicStanding 1. This is because in .Net the first enumeration value by default has 0, which is Excellent if you look at **StudentAcademicStanding.** Whereas, in Student.cs file Good is the second one, so it has a value of 1.

11. NHibernate – Configuration

In this chapter, we will look at NHibernate configuration. We have different ways that we can configure NHibernate. It divides into two main groups -

- XML-based configuration
- Code-based configuration

Code-Based Configuration

The code-based configuration is built into NHibernate. It was introduced around the NHibernate 3 and we have used the code bases configuration up till now.

```
cfg.DataBaseIntegration(x =>
{
    x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
    x.Driver<SqlClientDriver>();
    x.Dialect<MsSql2008Dialect>();
    x.LogSqlInConsole = true;
});
cfg.AddAssembly(Assembly.GetExecutingAssembly());
```

All the configurations are specified in the C# code. You can see here that we have got our new configuration object, and then we use **loquacious configuration** that was introduced with NHibernate 3.1 to configure the database. What connection string we are using, what database we are connecting to and the dialect to use. We also add our mapping assembly directly to here.

XML-Based Configuration

If you are using XML-based configuration, you can use a **hibernate.cfg.xml** file, which is just a standalone xml file using the NHibernate schema, or you can embed that NHibernate specific configuration inside of your app or **web.cfg**. The hibernate.cfg.xml name is by default, but we can use an arbitrary name for that xml file as well.

Let's have a look into the XML-based configuration by adding a new xml file to the NHibernateDemoApp project and call it hibernate.cfg.xml.

Enter the following information into the hibernate.cfg.xml file.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
  <session-factory>
    <property name="connection.connection_string">Data
Source=asia13797\\sqlexpress;Initial Catalog=NHibernateDemoDB;Integrated
Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False</property>
    <property
name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
    <property name="dialect">NHibernate.Dialect.MsSql2008Dialect</property>
    <mapping assembly="NHibernateDemoApp"/>
  </session-factory>
</hibernate-configuration>
```

As you can see in the above xml file, we have specified the same configuration as mentioned in the C#.

Now let's comment on this configuration from the Program.cs file and just call the **Configure()** method, which will load the **hibernate.cfg.xml** file as shown below.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;
namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            //cfg.DataBaseIntegration(x =>
```

```
      //{
            //    x.ConnectionString = "Data
Source=asia13797\\sqlexpress;Initial Catalog=NHibernateDemoDB;Integrated
Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
            //    x.Driver<SqlClientDriver>();
            //    x.Dialect<MsSql2008Dialect>();
            //    x.LogSqlInConsole = true;
            //});
            //cfg.AddAssembly(Assembly.GetExecutingAssembly());

            cfg.Configure();

            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    var students =
session.CreateCriteria<Student>().List<Student>();
                    Console.WriteLine("\nFetch the complete list again\n");
                    foreach (var student in students)
                    {
                        Console.WriteLine("{0} \t{1} \t{2} \t{3}", student.ID,
student.FirstName, student.LastName, student.AcademicStanding);
                    }
                    tx.Commit();
                }
                Console.ReadLine();
            }
        }
    }
}
```

Let’s run your application again and you will see the same output.

```
Fetch the complete list again
1       Allan   Bommer  Excellent
2       Jerry   Lewis   Good
```

12. NHibernate – Override Configuration

In this chapter, we will be covering how to override NHibernate configuration. There are just a few things you need to keep in mind.

- First of all, configuration in NHibernate is additive.
- So you don't just have to use a single xml file or you don't just have to use the code-based configuration or fluent NHibernate.
- You can mix and match all of these methods depending on how you want to configure your application.
- The important point to remember is that, lastly configuration wins.

In the following example, you can see that we create our configuration object, configure it using the code-based configuration and finally call the **cfg.configure()** method, which loads the hibernate.cfg.xml file.

```
cfg.DataBaseIntegration(x =>
{
    x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
    x.Driver<SqlClientDriver>();
    x.Dialect<MsSql2008Dialect>();
    x.LogSqlInConsole = true;
});
cfg.Configure();
```

- So anything inside of a hibernate.cfg.xml overrides the settings set by code-based configuration.
- By reversing these two processes we can have the defaults inside of hibernate.cfg.xml and then do our overrides inside of a code-based configuration.
- There is nothing that excludes if you are using code-based configuration and also there is nothing that prevents you from using the hibernate.cfg.xml file.

Let's have a look into a simple example in which we will override the configuration by using a mixture of xml-based and code-based configuration.

Let's also move the connection string to the **app.config** file using the following code.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
  <connectionStrings>
    <add name="default" connectionString="Data
Source=asia13797\\sqlexpress;Initial Catalog=NHibernateDemoDB;Integrated
Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False"/>
  </connectionStrings>
</configuration>
```

The connection string is sitting in some **app.config** file with a default name. Now, we need to mention the default name in hibernate.cfg.xml file instead of the connection string.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
  <session-factory>
    <property name="connection.connection_string">default</property>
    <property
name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
    <property name="dialect">NHibernate.Dialect.MsSql2008Dialect</property>
    <mapping assembly="NHibernateDemoApp"/>
  </session-factory>
</hibernate-configuration>
```

Let's comment on the connection string part, driver, and dialect part from the code-based configuration, because the program will read it from the hibernate.cfg.xml file and the **LogSqlInConsole** part will remain in the code-based configuration.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;
```

```
namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                //x.ConnectionString = "Data
Source=asia13797\\sqlexpress;Initial Catalog=NHibernateDemoDB;Integrated
Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                //x.Driver<SqlClientDriver>();
                //x.Dialect<MsSql2008Dialect>();
                x.LogSqlInConsole = true;
            });
            cfg.Configure();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                   var students =
session.CreateCriteria<Student>().List<Student>();

                    Console.WriteLine("\nFetch the complete list again\n");
                    foreach (var student in students)
                    {
                        Console.WriteLine("{0} \t{1} \t{2} \t{3}", student.ID,
student.FirstName, student.LastName, student.AcademicStanding);
                    }
                    tx.Commit();
                }
```

```
            Console.ReadLine();
           }
        }
    }
}
```

Now when you run the application, you will see that the program has read the log from code-based configuration and other configuration from the hibernate.cfg.xml file.

```
NHibernate: SELECT this_.ID as ID0_0_, this_.LastName as LastName0_0_,
this_.FirstMidName as FirstMid3_0_0_, this_.AcademicStanding as Academic4_0_0_
FROM Student this_


Fetch the complete list again
1       Allan   Bommer  Excellent
2       Jerry   Lewis   Good
```

So now we have got some of our configuration inside of our **hibernate.cfg.xml** file, some of it is inside of the code-based configuration and depending on the order of calling code-based versus **configure()**, we can change which of them overrides the other.

13. NHibernate – Batch Size

In this chapter, we will be covering the batch size update. The batch size allows you to **control the number of updates** that go out in a single round trip to your database for the supported databases.

- The update batch size has been defaulted as of NHibernate 3.2.
- But if you are using an earlier version or need to tune your NHibernate application, you should look at the update batch size, which is a very useful parameter that can be used to tune NHibernate's performance.
- Actually batch size controls how many inserts to push out in a group to a database.
- At the moment, only SQL Server and Oracle support this option because the underlying database provider needs to support query batching.

Let's have a look into a simple example in which we have set the batch size to 10 that will insert 10 records in a set.

```
cfg.DataBaseIntegration(x =>
{
    x.ConnectionString = "default";
    x.Driver<SqlClientDriver>();
    x.Dialect<MsSql2008Dialect>();
    x.LogSqlInConsole = true;
    x.BatchSize = 10;
});
```

Here is the complete implementation in which 25 records will be added to the database.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
```

```
class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
                x.LogSqlInConsole = true;
                x.BatchSize = 10;
            });
            //cfg.Configure();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    for (int i = 0; i < 25; i++)
                    {
                        var student = new Student
                        {
                            ID = 100+i,
                            FirstName = "FirstName"+i.ToString(),
                            LastName = "LastName" + i.ToString(),
                            AcademicStanding = StudentAcademicStanding.Good
                        };
                        session.Save(student);
                    }
                    tx.Commit();
```

```
                    var students =
session.CreateCriteria<Student>().List<Student>();
                    Console.WriteLine("\nFetch the complete list again\n");
                    foreach (var student in students)
                    {
                        Console.WriteLine("{0} \t{1} \t{2} \t{3}", student.ID,
student.FirstName, student.LastName, student.AcademicStanding);
                    }
                }
                Console.ReadLine();
            }
        }
    }
}
```

Now let's run your application and you see all those updates are jumping over to NHibernate profiler. We have 26 individual round trips to the database 25 for insertion and one retrieving the list of students.

Now, why is that? The reason is because NHibernate needs to do a **select scope identity** as we are using the native identifier generation strategy in the mapping file for ID as shown in the following code.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemoApp"
                   namespace="NHibernateDemoApp">
  <class name="Student">
    <id name="ID">
      <generator class="native"/>
    </id>
    <property name="LastName"/>
    <property name="FirstName"
              column="FirstMidName"
              type="String"/>
    <property name="AcademicStanding"/>
  </class>
</hibernate-mapping>
```

So we need to use a different method such as the **guid.comb** method. If we're going to go to guid.comb, we need to go over to our customer and change this to a **guid**. So that will work fine. Now let's change from the native to guid.comb using the following code.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemoApp"
                   namespace="NHibernateDemoApp">
  <class name="Student">
    <id name="ID">
      <generator class="guid.comb"/>
    </id>
    <property name="LastName"/>
    <property name="FirstName"
              column="FirstMidName"
              type="String"/>
    <property name="AcademicStanding"/>
  </class>
</hibernate-mapping>
```

So it's the database that's responsible for generating those IDs. The only way NHibernate can find out what ID was generated was to select it immediately afterwards. Or else, if we have created a batch of students, it will not be able match up the ID of the student that was created.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NHibernateDemoApp
{
    class Student
    {
        public virtual Guid ID { get; set; }
        public virtual string LastName { get; set; }
        public virtual string FirstName { get; set; }
        public virtual StudentAcademicStanding AcademicStanding { get; set; }
```

```
}
    public enum StudentAcademicStanding
    {
        Excellent,
        Good,
        Fair,
        Poor,
        Terrible
    }
}
```

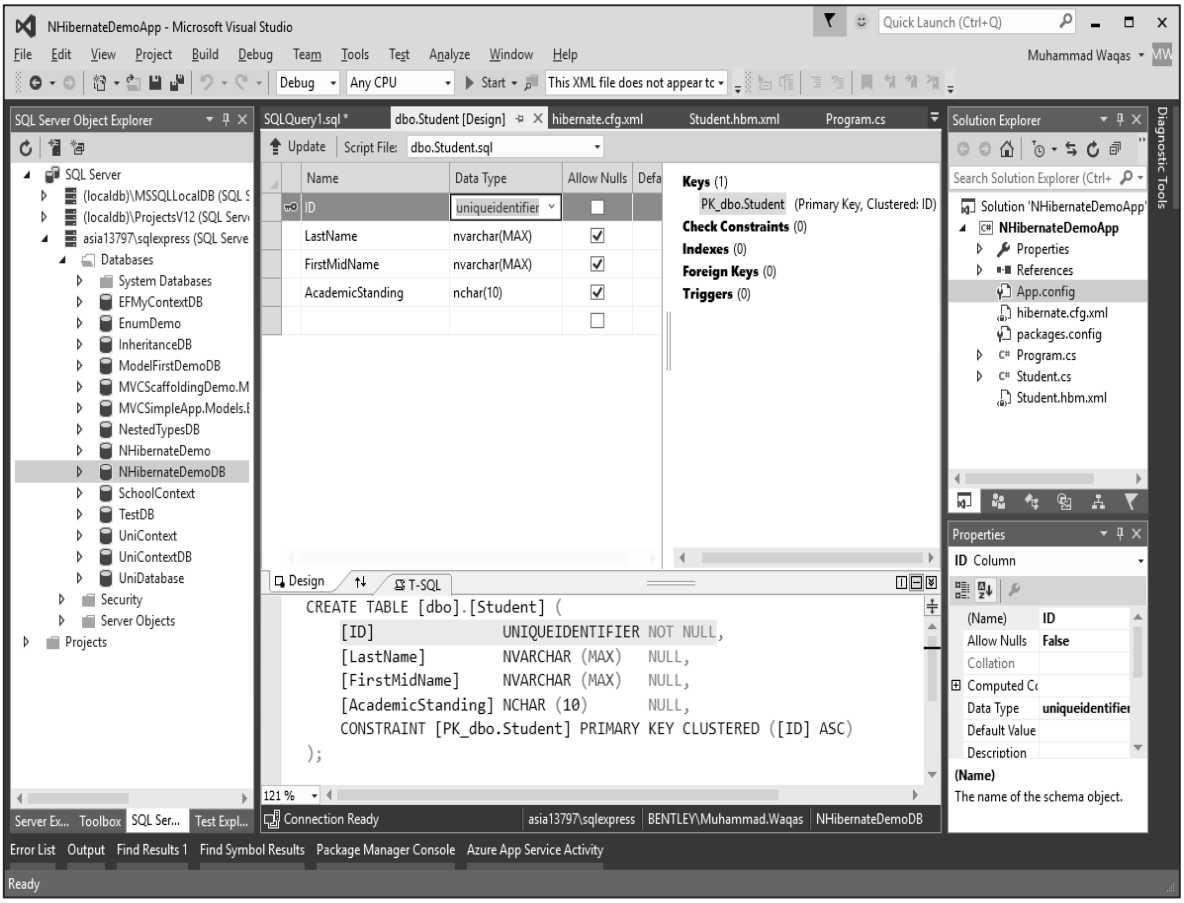
We just need to update our database. Let's drop the student table and create a new table by specifying the following query, so go the SQL Server Object Explorer and right-click on the database and select the **New Query...** option.

It will open the query editor and and then specify the following query.

```
DROP TABLE [dbo].[Student]


CREATE TABLE [dbo].[Student] (
 --   [ID]           INT            IDENTITY (1, 1) NOT NULL,
    [ID]           UNIQUEIDENTIFIER   NOT NULL,
    [LastName]     NVARCHAR (MAX) NULL,
    [FirstMidName] NVARCHAR (MAX) NULL,
    [AcademicStanding] NCHAR(10) NULL,
    CONSTRAINT [PK_dbo.Student] PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

This query will first drop the existing student table and then create a new table. As you can see that we have used **UNIQUEIDENTIFIER** rather than using an integer primary key as an ID.

Execute this query and then go to the **Designer view** and you will see that now the ID is created with a unique identifier as shown in the following image.



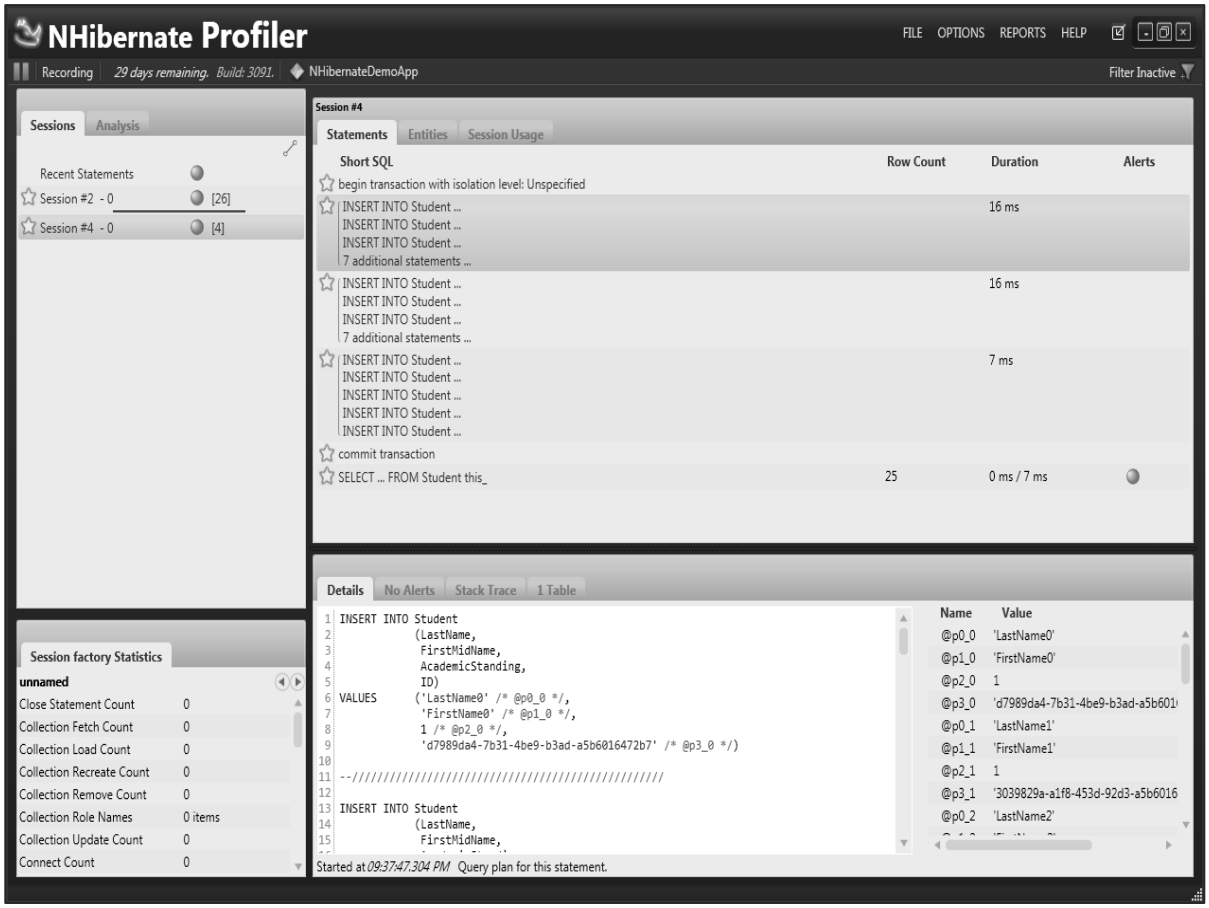
Now we need to remove the ID from the program.cs file, while inserting data, because now it will generate the **guids** for it automatically.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
```

```
     var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
                x.LogSqlInConsole = true;
                x.BatchSize = 10;
            });
            //cfg.Configure();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    for (int i = 0; i < 25; i++)
                    {
                        var student = new Student
                        {
                            FirstName = "FirstName"+i.ToString(),
                            LastName = "LastName" + i.ToString(),
                            AcademicStanding = StudentAcademicStanding.Good
                        };
                        session.Save(student);
                    }
                    tx.Commit();
                    var students =
session.CreateCriteria<Student>().List<Student>();
                    Console.WriteLine("\nFetch the complete list again\n");
                    foreach (var student in students)
                    {
                        Console.WriteLine("{0} \t{1} \t{2} \t{3}", student.ID,
student.FirstName, student.LastName, student.AcademicStanding);
```

```
                    }
                }
                Console.ReadLine();
            }
        }
    }
}
```

Now run the application again and have a look at the NHibernate profiler. Now the NHibernate profiler rather than making 26 round trips will make only four.



It's inserted ten rows into the table, then another ten rows, and later the remaining five. And after commit, it has inserted one more for retrieving all the records.

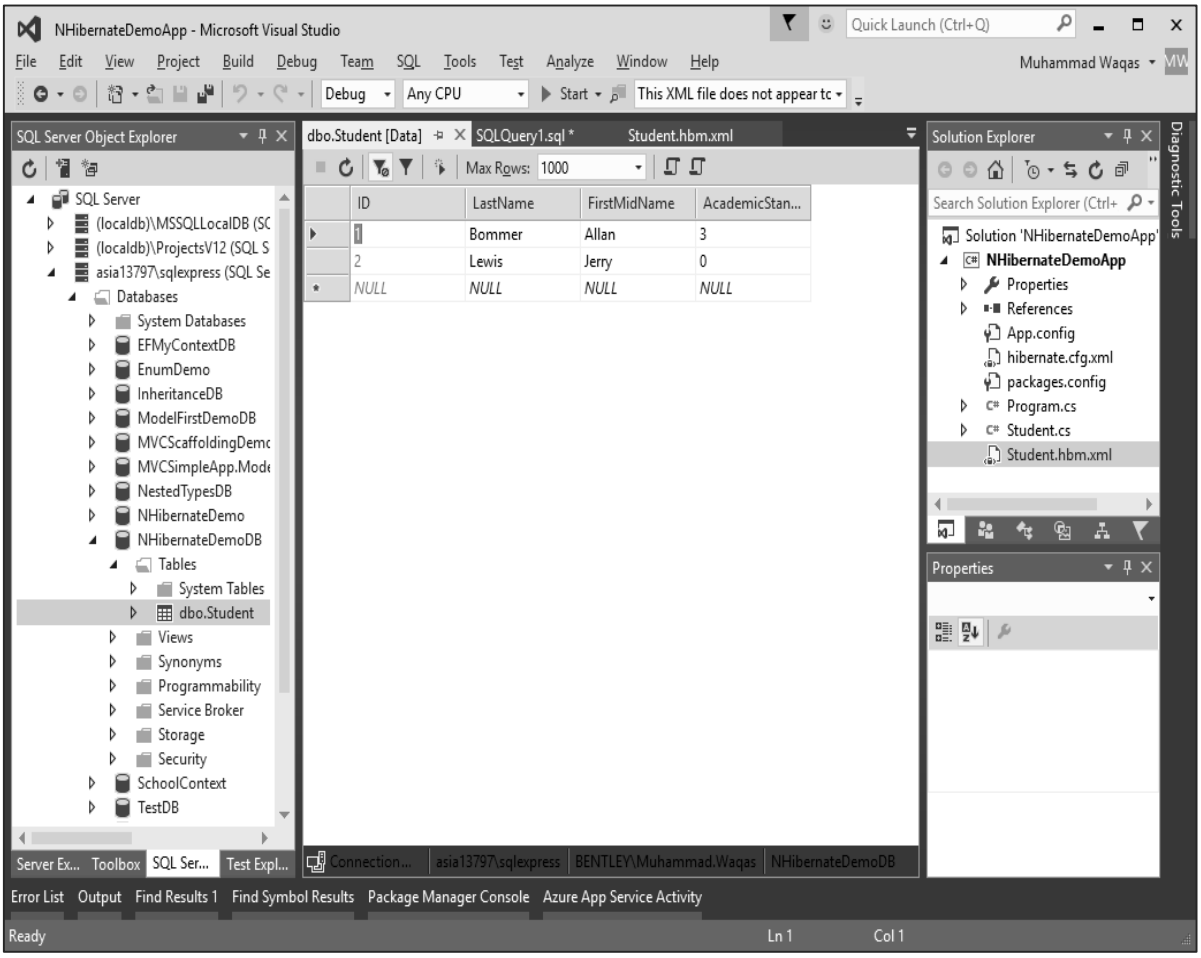
- So it's divided it up into groups of ten, as best it can.
- So if you're doing a lot of inserts, this can dramatically improve the insert performance in your application, because you can batch it up.
- This is because NHibernate assigns those guids itself using the **guid.comb** algorithm, and it doesn't have to rely on the database to do this.
- So using the batch size is a great way to tune it.

14. NHibernate – Caching

In this chapter, we will be covering how the **caching** works in NHibernate applications. It has built-in support for caching. It looks as a simple feature, but in reality, it is one of the most complex features. We will begin with the First Level Cache.

First Level Cache

This cache mechanism is enabled by default in NHibernate and we don't need to do anything for working with cache. To understand this, let's have a look into a simple example, as you can see that we have two records in our database.



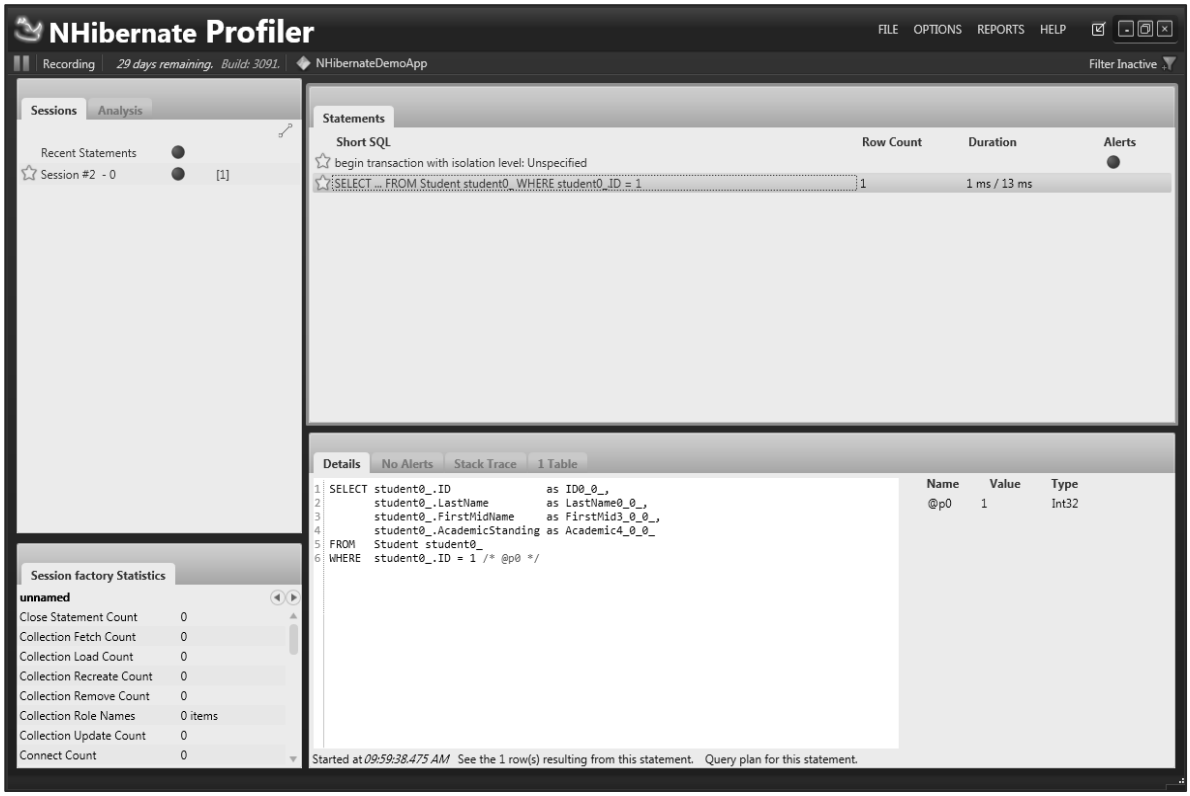
Now in this example, we will retrieve the student whose ID is 1 and we will use the same session query twice as shown in the following code.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cache;
using NHibernate.Cfg;
using NHibernate.Dialect;
```

```
using NHibernate.Driver;
using NHibernate.Linq;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
                x.LogSqlInConsole = true;
                x.BatchSize = 10;
            });
            //cfg.Configure();
            cfg.Cache(c =>
            {
                c.UseMinimalPuts = true;
                c.UseQueryCache = true;
            });
            cfg.SessionFactory().Caching
                .Through<HashtableCacheProvider>()
                .WithDefaultExpiration(1440);
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
```

```
        {
                using (var tx = session.BeginTransaction())
                {
                    var studentUsingTheFirstQuery = session.Get<Student>(1);

                    var studentUsingTheSecondQuery = session.Get<Student>(1);
                }
                Console.ReadLine();
            }
        }
    }
}
```

Now let's run this application and see the result in the NHibernate Profiler.



You will be surprised to see that NHibernate fires only one query. This is how NHibernate uses the first level cache. When the first query is executed, then NHibernate cached the Student with ID = 1 in its first level cache.

So, when the second query is executed then NHibernate first looks up the first level cache Student entity with ID = 1, if it finds that entity, then NHibernate knows that, there is no need to fire another query to retrieve the same employee object again.

15. NHibernate – Mapping Component

In this chapter, we will be talking about mapping components. In NHibernate, **component is a value object**. It does not have an identity of its own.

- An example of this would be a money object, a purse or a wallet might have money in it, but the exact identity of that money is irrelevant.

- It doesn't have its own primary key, but components themselves are persistent in the same table as the owning object.

Let's have a look at a simple example in which a student has an Address, which is an object of **Location class** associated with it.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NHibernateDemoApp
{
    class Student
    {
        public virtual int ID { get; set; }
        public virtual string LastName { get; set; }
        public virtual string FirstName { get; set; }
        public virtual StudentAcademicStanding AcademicStanding { get; set; }
        public virtual Location Address { get; set; }
    }

    public class Location
    {
        public virtual string Street { get; set; }
        public virtual string City { get; set; }
        public virtual string Province { get; set; }
        public virtual string Country { get; set; }
    }
```

```
    public enum StudentAcademicStanding
    {
        Excellent,
        Good,
        Fair,
        Poor,
        Terrible
    }
}
```

Now, we also need to update the database by executing the following query, which will first drop the Student table and then create a new table that will also contain a column for Location class.

```
DROP TABLE [dbo].[Student]


CREATE TABLE [dbo].[Student] (
    [ID]           INT            IDENTITY (1, 1) NOT NULL,
    [LastName]     NVARCHAR (MAX) NULL,
    [FirstMidName] NVARCHAR (MAX) NULL,
    [AcademicStanding] NCHAR(10) NULL,
    [Street]           NVARCHAR (100) NULL,
    [City]         NVARCHAR (100) NULL,
    [Province]     NVARCHAR (100) NULL,
    [Country]      NVARCHAR (100) NULL,
    CONSTRAINT [PK_dbo.Student] PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Now to map those columns that are not directly a part of Student class, but they are properties of Location class and Location class object is defined in the student class. We need a component to map it correctly. Let's create a component in **student.hbm.xml** file as shown in the following code.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemoApp"
                   namespace="NHibernateDemoApp">
  <class name="Student">
    <id name="ID">
      <generator class="native"/>
```

```
   </id>
    <property name="LastName"/>
    <property name="FirstName"
              column="FirstMidName"
              type="String"/>
    <property name="AcademicStanding"/>

    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>
  </class>
</hibernate-mapping>
```

This component is the Address and it has these different properties on it. With this information, NHibernate now has enough that it can actually map this.

Now here is the Program.cs file in which a new student object is created and initialized and then saved to the database. It will then retrieve the list from the database.

```
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cache;
using NHibernate.Caches.SysCache;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;
using System;
using System.Linq;
using System.Reflection;

namespace NHibernateDemoApp
{
    class Program
    {
```

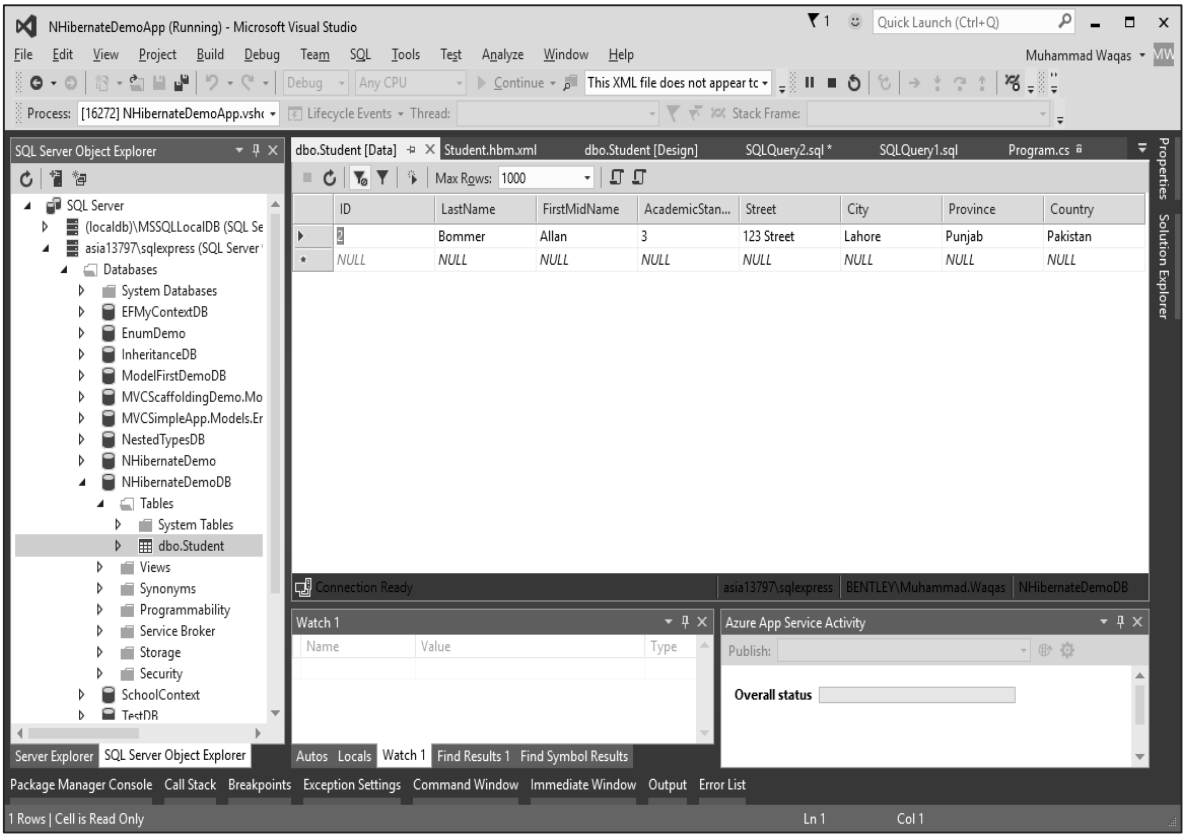
```
  static void Main(string[] args)
        {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x =>
            {
                x.ConnectionString = "Data Source=asia13797\\sqlexpress;Initial
Catalog=NHibernateDemoDB;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False";
                x.Driver<SqlClientDriver>();
                x.Dialect<MsSql2008Dialect>();
            });
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            var sefact = cfg.BuildSessionFactory();
            using (var session = sefact.OpenSession())
            {
                using (var tx = session.BeginTransaction())
                {
                    var student1 = new Student
                    {
                        ID = 1,
                        FirstName = "Allan",
                        LastName = "Bommer",
                        AcademicStanding = StudentAcademicStanding.Poor,
                        Address = new Location
                        {
                            Street = "123 Street",
                            City = "Lahore",
                            Province = "Punjab",
                            Country = "Pakistan"
                        }
                    };
                    session.Save(student1);
                                        tx.Commit();
                    var students = session.Query<Student>().ToList<Student>();
```

```
            Console.WriteLine("\nFetch the complete list again\n");
                  foreach (var student in students)
                  {
                      Console.WriteLine("{0} \t{1} \t{2} \t{3} \t{4} \t{5}
\t{6} \t{7}",
                          student.ID,
                          student.FirstName,
                          student.LastName,
                          student.AcademicStanding,
                          student.Address.Street,
                          student.Address.City,
                          student.Address.Province,
                          student.Address.Country);
                  }
               }
               Console.ReadLine();
            }
         }
    }
}
```

Now we can run this application and NHibernate can save those values to the database. When you run the application, you will see the following output.

```
Fetch the complete list again

2       Allan   Bommer  Poor    123 Street      Lahore  Punjab  Pakistan
```

Here are the values in the database.



The components allow us to separate out columns that are in a database table into their own separate class.

- The other thing to notice here is because the Location is a class, it is not an entity.
- It is a value type object and it doesn't have its own primary key.
- It is saved in the same table as the Student that contains it.
- That's why we're using the component here.
- This allows a lot of flexibility to change our class layer, how our classes are defined versus how our database is laid out.

16. NHibernate – Relationships

In this chapter, we will look at relationships in NHibernate. Let's turn our attention to how we can understand relationships in NHibernate. The easiest way is to think about the relationships from the database perspective.

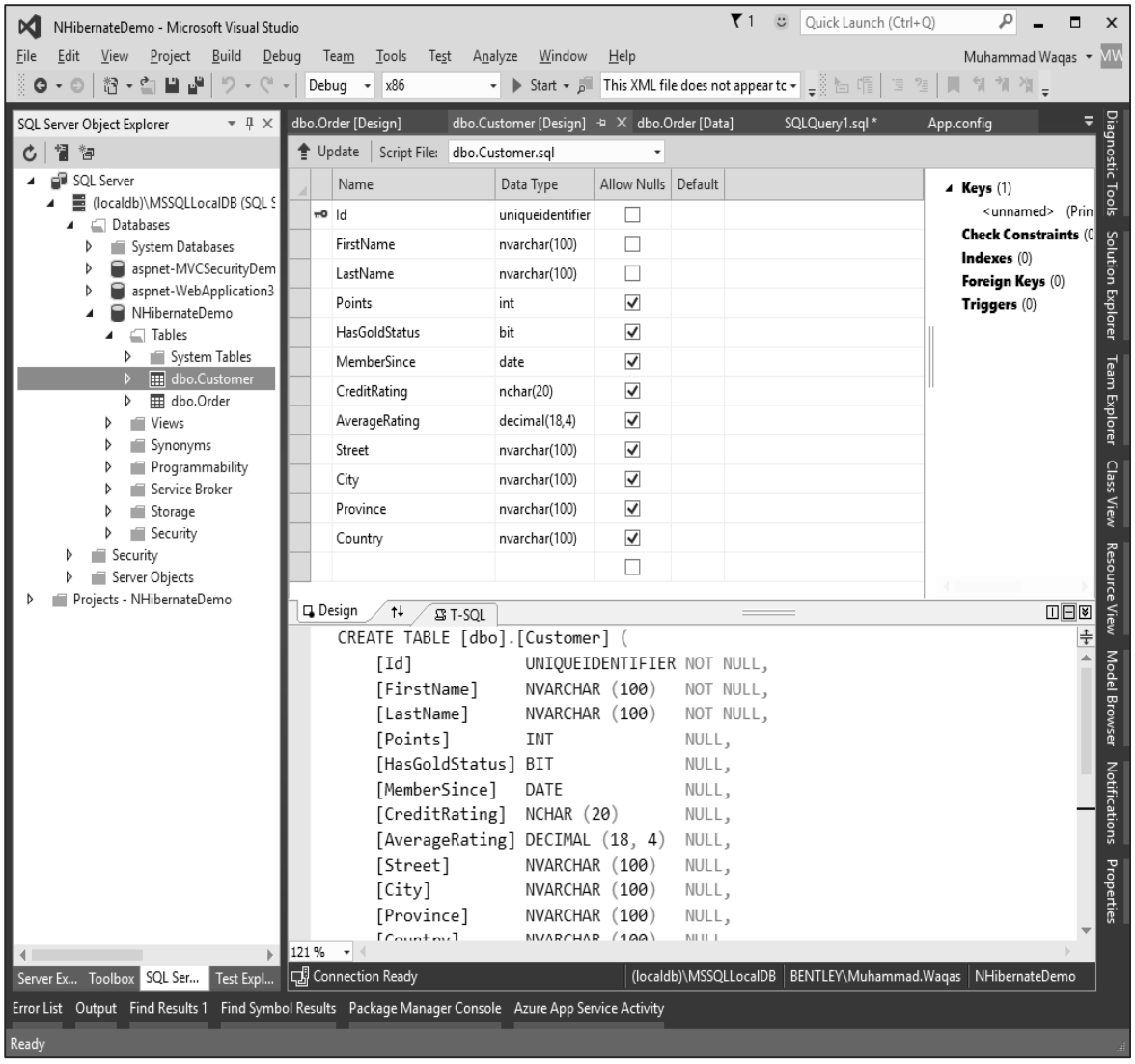
- We will first create a new application in which we will create some relationships among the customer and order entities.
- The first relationship we're going to look at is a classic collection relationship.
- We have a customer with a collection of orders.
- This is a one-to-many relationship and it's represented in the database by 2 tables and there is a customer ID on the orders table and we have a foreign key relationship back to the customer.

First we need to create a database and two tables Customer and Order. You can create this by specifying the following query in SQL Server Explorer.

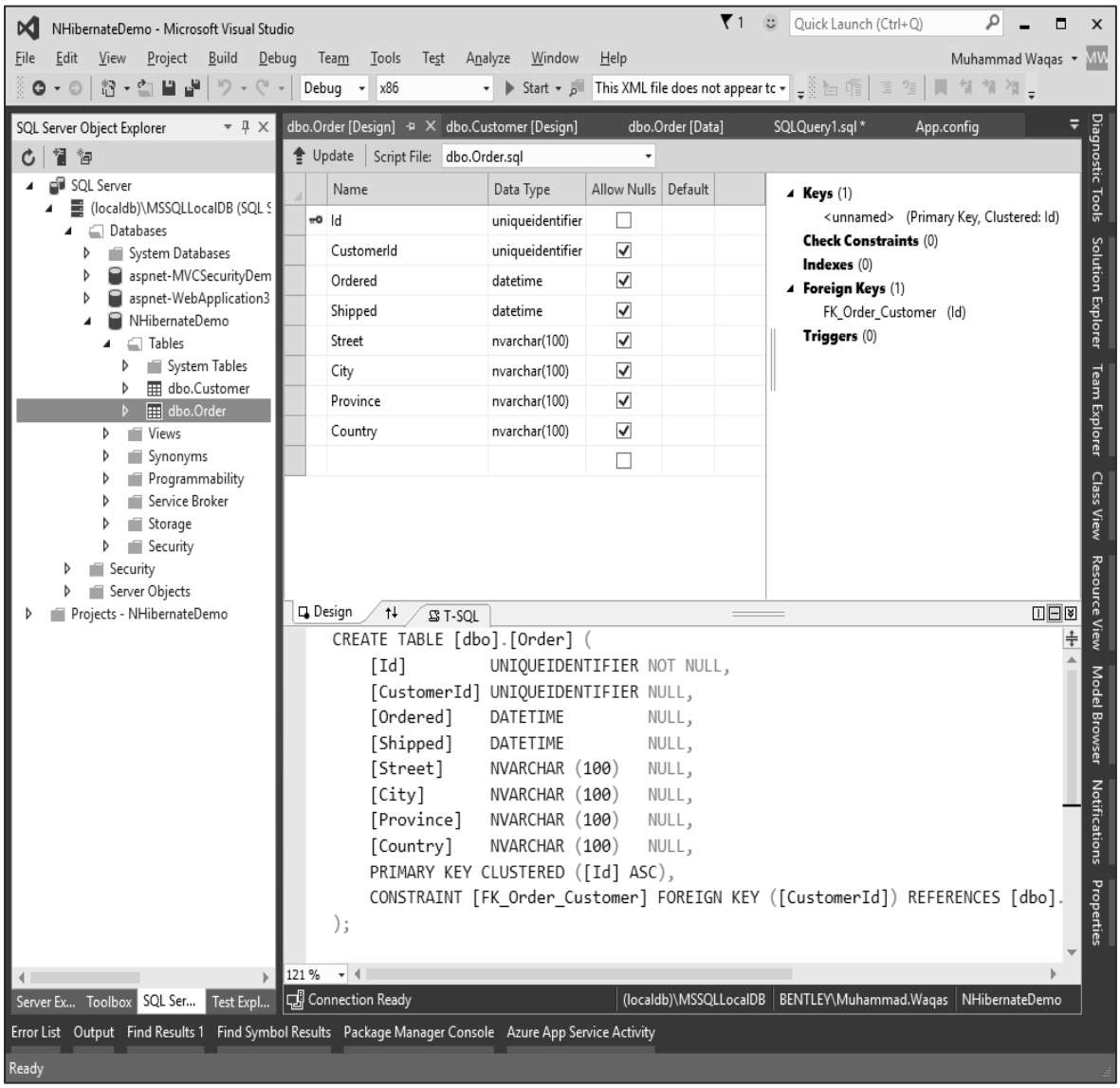
```
USE [master]
GO
CREATE DATABASE [NHibernateDemo]
GO
USE [NHibernateDemo]
GO

CREATE TABLE [dbo].[Customer](
      [Id] [uniqueidentifier] NOT NULL,
      [FirstName] [nvarchar](100) NOT NULL,
      [LastName] [nvarchar](100) NOT NULL,
      [Points] [int] NULL,
      [HasGoldStatus] [bit] NULL,
      [MemberSince] [date] NULL,
      [CreditRating] [nchar](20) NULL,
      [AverageRating] [decimal](18, 4) NULL,
      [Street] [nvarchar](100) NULL,
      [City] [nvarchar](100) NULL,
      [Province] [nvarchar](100) NULL,
      [Country] [nvarchar](100) NULL,
```

```
   PRIMARY KEY CLUSTERED ([Id] ASC)
)
GO
CREATE TABLE [dbo].[Order](
     [Id] [uniqueidentifier] NOT NULL,
     [CustomerId] [uniqueidentifier] NULL,
     [Ordered] [datetime] NULL,
     [Shipped] [datetime] NULL,
     [Street] [nvarchar](100) NULL,
     [City] [nvarchar](100) NULL,
     [Province] [nvarchar](100) NULL,
     [Country] [nvarchar](100) NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
)
GO
```

It will create two tables in the database. The following image shows the Customer Table.



The following image shows the Order Table in which you can see the foreign key relationship back to the customer.



We need to define the connection string in the **app.config** file, here is the implementation of the app.config file.

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

  <connectionStrings>

    <add name="default" connectionString="Data
Source=(localdb)\MSSQLLocalDB;Initial Catalog=NHibernateDemo;Integrated
Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False"/>

  </connectionStrings>

</configuration>
```

To install the NHibernate in your application, run the following command in NuGet Manager Console window.

```
install-package NHibernate
```

To configure the NHibernate configuration, we need to define the configuration in **hibernate.cfg.xml** file as shown in the following code.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
  <session-factory>
    <property name="connection.connection_string_name">default</property>
    <property
name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
    <property name="dialect">NHibernate.Dialect.MsSql2008Dialect</property>
    <property name="show_sql">true</property>
  </session-factory>
</hibernate-configuration>
```

In this example, we will be working two domain classes, Customer and Order.

Here is the Customer.cs file implementation in which we have two classes, one is the Customer class and another is the Location class in which object is used as an address in the Customer class.

```
using System;
using System.Text;
using Iesi.Collections.Generic;

namespace NHibernateDemo {
    public class Customer {
        public Customer() {
            MemberSince = DateTime.UtcNow;
            Orders = new HashedSet<Order>();
        }
        public virtual Guid Id { get; set; }
        public virtual string FirstName { get; set; }
        public virtual string LastName { get; set; }
        public virtual double AverageRating { get; set; }
        public virtual int Points { get; set; }
        public virtual bool HasGoldStatus { get; set; }
```

```
   public virtual DateTime MemberSince { get; set; }
        public virtual CustomerCreditRating CreditRating { get; set; }
        public virtual Location Address { get; set; }

        public virtual ISet<Order> Orders { get; set; }

        public virtual void AddOrder(Order order) {
            Orders.Add(order);
            order.Customer = this;
        }

        public override string ToString() {
            var result = new StringBuilder();
            result.AppendFormat("{1} {2} ({0})\r\n\tPoints:
{3}\r\n\tHasGoldStatus: {4}\r\n\tMemberSince: {5} ({7})\r\n\tCreditRating:
{6}\r\n\tAverageRating: {8}\r\n", Id, FirstName, LastName, Points,
HasGoldStatus, MemberSince, CreditRating, MemberSince.Kind, AverageRating);
            result.AppendLine("\tOrders:");
            foreach(var order in Orders) {
                result.AppendLine("\t\t" + order);
            }
            return result.ToString();
        }
    }

    public class Location {
        public virtual string Street { get; set; }
        public virtual string City { get; set; }
        public virtual string Province { get; set; }
        public virtual string Country { get; set; }
    }

    public enum CustomerCreditRating {
        Excellent, VeryVeryGood, VeryGood, Good, Neutral, Poor, Terrible
    }
}
```

Here is the mapping file **Customer.hbm.xml** in which Customer class is mapped to the Customer table.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Customer">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="FirstName"/>
    <property name="LastName"/>
    <property name="AverageRating"/>
    <property name="Points"/>
    <property name="HasGoldStatus"/>
    <property name="MemberSince" type="UtcDateTime"/>
    <property name="CreditRating" type="CustomerCreditRatingType"/>

    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>
  </class>
</hibernate-mapping>
```

We also have an Order Class and here is the implementation of **Order.cs** file.

```
using System;
using Iesi.Collections.Generic;

namespace NHibernateDemo {
    public class Order {
        public virtual Guid Id { get; set; }
        public virtual DateTime Ordered { get; set; }
        public virtual DateTime? Shipped { get; set; }
```

```
   public virtual Location ShipTo { get; set; }
        public virtual Customer Customer { get; set; }

        public override string ToString() {
            return string.Format("Order Id: {0}", Id);
        }
    }
}
```

Many-to-One Relationship

We also need to map the Order class to the Order table in the database, so here is the implementation of the **Order.hbm.xml** file.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Order" table="`Order`">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="Ordered"/>
    <property name="Shipped"/>
    <component name="ShipTo">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>
    <!--<many-to-one name="Customer" column="CustomerId" cascade="save-
update"/>-->
  </class>
</hibernate-mapping>
```

One-to-Many Relationship

Here, we are going to take a look at a one-to-many relationship, in this case, between customer and orders. We've got our customer here, we're creating a new one, and you can see that the collection is initialized with the following pair of orders.

```
private static Customer CreateCustomer() {
    var customer = new Customer {
                                    FirstName = "John",
                                    LastName = "Doe",
                                    Points = 100,
                                    HasGoldStatus = true,
                                    MemberSince = new DateTime(2012, 1, 1),
                                    CreditRating = CustomerCreditRating.Good,
                                    AverageRating = 42.42424242,
                                    Address = CreateLocation()
                                };
    var order1 = new Order {
        Ordered = DateTime.Now
    };
    customer.AddOrder(order1);
    var order2 = new Order {
                                Ordered = DateTime.Now.AddDays(-1),
                                Shipped = DateTime.Now,
                                ShipTo = CreateLocation()
                            };
    customer.AddOrder(order2);
    return customer;
}
```

So we will create a new customer and then save it, after saving it, we will find the ID and then reload it in another session in the Main method as shown in the following program.

```
private static void Main() {
    var cfg = ConfigureNHibernate();
    var sessionFactory = cfg.BuildSessionFactory();

    Guid id;
    using(var session = sessionFactory.OpenSession())
    using(var tx = session.BeginTransaction()) {
        var newCustomer = CreateCustomer();
        Console.WriteLine("New Customer:");
        Console.WriteLine(newCustomer);
```

```
   session.Save(newCustomer);
        id = newCustomer.Id;
        tx.Commit();
    }

    using(var session = sessionFactory.OpenSession())
    using(var tx = session.BeginTransaction()) {
        var reloaded = session.Load<Customer>(id);
        Console.WriteLine("Reloaded:");
        Console.WriteLine(reloaded);
        tx.Commit();
    }

    Console.WriteLine("Press <ENTER> to exit...");
    Console.ReadLine();
}
```

Here is the complete **Program.cs** file implementation.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            Guid id;
            using(var session = sessionFactory.OpenSession())
```

```
          using(var tx = session.BeginTransaction()) {
                var newCustomer = CreateCustomer();
                Console.WriteLine("New Customer:");
                Console.WriteLine(newCustomer);
                session.Save(newCustomer);
                id = newCustomer.Id;
                tx.Commit();
            }
            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var reloaded = session.Load<Customer>(id);
                Console.WriteLine("Reloaded:");
                Console.WriteLine(reloaded);
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Customer CreateCustomer() {
            var customer = new Customer {
                                            FirstName = "John",
                                            LastName = "Doe",
                                            Points = 100,
                                            HasGoldStatus = true,
                                            MemberSince = new DateTime(2012, 1,
1),
                                            CreditRating =
CustomerCreditRating.Good,
                                            AverageRating = 42.42424242,
                                            Address = CreateLocation()
                                        };
            var order1 = new Order {
                Ordered = DateTime.Now
```

```
      };
            customer.AddOrder(order1);
            var order2 = new Order {
                                        Ordered = DateTime.Now.AddDays(-1),
                                        Shipped = DateTime.Now,
                                        ShipTo = CreateLocation()
                                    };
            customer.AddOrder(order2);
            return customer;
        }
        private static Location CreateLocation() {
            return new Location
                   {                        Street = "123 Somewhere Avenue",
                       City = "Nowhere",
                       Province = "Alberta",
                       Country = "Canada"
                   };
        }
        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
 }
    }
}
```

When you run this application, you will see the following output.

```
New Customer:
John Doe (00000000-0000-0000-0000-000000000000)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Unspecified)
        CreditRating: Good
        AverageRating: 42.42424242
        Orders:
                Order Id: 00000000-0000-0000-0000-000000000000
                Order Id: 00000000-0000-0000-0000-000000000000

Reloaded:
John Doe (9b0fcf10-83f6-4f39-bda5-a5b800ede2ba)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:

Press <ENTER> to exit...
```

As you can see that initially the customer has 2 orders, but when we reload it, there are no orders to be seen. If you look at **customer.hbm.xml** file, you can see here that we do not map actual orders collection. So NHibernate knows nothing about it. Let's go ahead and add it.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Customer">
    <id name="Id">
```

```
  <generator class="guid.comb"/>
    </id>
    <property name="FirstName"/>
```

```
    <property name="LastName"/>
    <property name="AverageRating"/>
    <property name="Points"/>
    <property name="HasGoldStatus"/>
    <property name="MemberSince" type="UtcDateTime"/>
    <property name="CreditRating" type="CustomerCreditRatingType"/>

    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>

    <set name="Orders" table="`Order`">
      <key column="CustomerId"/>
      <one-to-many class="Order"/>
    </set>
  </class>
</hibernate-mapping>
```

This is a set and the name of this collection is 'Orders', which is stored in a table called order. We need to specify a key which is the name of the foreign key or to find orders. These orders are identified or belong to a customer through the customer ID. And then I have to note that this is a one-to-many relationship and it is with the order class.

We also need to slightly change the Main method by saving the new customer orders to the database as well as shown in the following program.

```
private static void Main() {
    var cfg = ConfigureNHibernate();
    var sessionFactory = cfg.BuildSessionFactory();

    Guid id;
    using(var session = sessionFactory.OpenSession())
    using(var tx = session.BeginTransaction()) {

   var newCustomer = CreateCustomer();
        Console.WriteLine("New Customer:");
        Console.WriteLine(newCustomer);
```

```
        session.Save(newCustomer);
        foreach (var order in newCustomer.Orders)
        {
            session.Save(order);
        }
        id = newCustomer.Id;
        tx.Commit();
    }

    using(var session = sessionFactory.OpenSession())
    using(var tx = session.BeginTransaction()) {
        var reloaded = session.Load<Customer>(id);
        Console.WriteLine("The orders were ordered by: ");
        foreach (var order in reloaded.Orders)
        {
            Console.WriteLine(order.Customer);
        }
        tx.Commit();
    }

    Console.WriteLine("Press <ENTER> to exit...");
    Console.ReadLine();
}
```

We have also specified which customer ordered that particular product. So we need to create a many-to-one relationship to relate that order back to that customer.

So let's go into the **Order.hbm.xml** file and add a many-to-one, and then name the customer field and the column with the customer ID.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Order" table="`Order`">

    <id name="Id">
      <generator class="guid.comb"/>
    </id>
```

```
    <property name="Ordered"/>
    <property name="Shipped"/>
    <component name="ShipTo">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>
    <many-to-one name="Customer"
                 column="CustomerId"/>
  </class>
</hibernate-mapping>
```

Let's run this application again and now you will see the following output.

```
New Customer:
John Doe (00000000-0000-0000-0000-000000000000)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Unspecified)
        CreditRating: Good
        AverageRating: 42.42424242
        Orders:
                Order Id: 00000000-0000-0000-0000-000000000000
                Order Id: 00000000-0000-0000-0000-000000000000

Reloaded:
John Doe (660a6f29-650e-4380-99e0-a5b800febbde)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:

         Order Id: 57314deb-e023-4e55-ac1e-a5b800febbe3
                Order Id: fc065683-d5f5-484b-ae42-a5b800febbe3
```

```
The orders were ordered by:
John Doe (660a6f29-650e-4380-99e0-a5b800febbde)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: 57314deb-e023-4e55-ac1e-a5b800febbe3
                Order Id: fc065683-d5f5-484b-ae42-a5b800febbe3

John Doe (660a6f29-650e-4380-99e0-a5b800febbde)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: 57314deb-e023-4e55-ac1e-a5b800febbe3
                Order Id: fc065683-d5f5-484b-ae42-a5b800febbe3

Press <ENTER> to exit...
```

17. NHibernate – Collection Mapping

In this chapter, we will be covering how to represent collections. There are different types of collections that we can use within the NHibernate such as –

- Lists
- Sets
- Bags

Now, from the .NET perspective, we generally deal with lists or like very simple data structures, lists, dictionaries. .NET does not have a wide variety of different collection types. So why does NHibernate need all these different types? It really comes back to the database.

List

- A list is an ordered collection of elements that are not necessarily unique.
- We can map this using the **IList <T>**.
- So although we might conventionally have a list of addresses, and from application point of view we know that the elements are unique, nothing in the list prevents us from inserting duplicate elements in that list.

Set

- A set is an unordered collection of unique elements. If you try to insert 2 duplicate elements into a set, it will throw an exception.
- There's nothing specific in NHibernate about it.
- It's just a convenient way a have a generic set implementation. If you're on .NET 4, you can use the new **HashSet <T>** to represent these, but in most NHibernate applications, we represent this is an ISet.
- It is an unordered, if you pull back a list of addresses from a database or a list of orders, you don't know what order they're coming in unless you put in a specific Order by clause.
- So in general, the data that you're pulling back from a database are sets.
- They are unique collections of elements that are unordered.

Bag

- Another common collection that we will see in the database world is a bag, which is just like a set except it can have duplicate elements.
- In the .NET world, we represent this by an IList.

Sets are probably the most common, but you will see lists and bags as well depending on your application. Let's have a look into a below **customer.hbm.xml** file from the last chapter in which Set orders are defined.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Customer">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="FirstName"/>
    <property name="LastName"/>
    <property name="AverageRating"/>
    <property name="Points"/>
    <property name="HasGoldStatus"/>
    <property name="MemberSince" type="UtcDateTime"/>
    <property name="CreditRating" type="CustomerCreditRatingType"/>

    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>

    <set name="Orders" table="`Order`">
      <key column="CustomerId"/>
      <one-to-many class="Order"/>
    </set>
  </class>
</hibernate-mapping>
```

As you can see, we have mapped the orders collection as a set. Remember that a set is an unordered collection of unique elements.

Now, if you look at the Customer class, you will see that Orders property is defined with an ISet as shown in the following program.

```
public virtual ISet<Order> Orders { get; set; }
```

Now when this application is run, you will see the following output.

```
New Customer:
John Doe (00000000-0000-0000-0000-000000000000)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Unspecified)
        CreditRating: Good
        AverageRating: 42.42424242
        Orders:
                Order Id: 00000000-0000-0000-0000-000000000000
                Order Id: 00000000-0000-0000-0000-000000000000

Reloaded:
John Doe (1f248133-b50a-4ad7-9915-a5b8017d0ff1)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: c41af8f2-7124-42a7-91c5-a5b8017d0ff6
                Order Id: 657f6bb0-1f42-45fc-8fc7-a5b8017d0ff7

The orders were ordered by:
John Doe (1f248133-b50a-4ad7-9915-a5b8017d0ff1)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: c41af8f2-7124-42a7-91c5-a5b8017d0ff6
                Order Id: 657f6bb0-1f42-45fc-8fc7-a5b8017d0ff7

John Doe (1f248133-b50a-4ad7-9915-a5b8017d0ff1)
        Points: 100
```

```
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: c41af8f2-7124-42a7-91c5-a5b8017d0ff6
                Order Id: 657f6bb0-1f42-45fc-8fc7-a5b8017d0ff7

Press <ENTER> to exit...
```

If the items in the collection didn't need to be unique, if you could have multiple orders with the same primary key occurring multiple times in this collection, then this would be better mapped as a bag as shown in the following program.

```
<bag name="Orders" table="`Order`">
   <key column="CustomerId"/>
   <one-to-many class="Order"/>
</bag>
```

Now, if you run this application you will get an exception because if we take a look at the customer class, you'll notice that the orders are marked as an ISet in the C# code.

So we will also need to change this to an IList and then here, we would need to change from the HashSet to a List in the constructor.

```
public class Customer {
    public Customer() {
        MemberSince = DateTime.UtcNow;
        Orders = new List<Order>();
    }
    public virtual Guid Id { get; set; }
    public virtual string FirstName { get; set; }
    public virtual string LastName { get; set; }
    public virtual double AverageRating { get; set; }
    public virtual int Points { get; set; }
    public virtual bool HasGoldStatus { get; set; }
    public virtual DateTime MemberSince { get; set; }
    public virtual CustomerCreditRating CreditRating { get; set; }
    public virtual Location Address { get; set; }
```

```
    public virtual IList<Order> Orders { get; set; }

    public virtual void AddOrder(Order order) {
        Orders.Add(order);
        order.Customer = this;
    }

    public override string ToString() {
        var result = new StringBuilder();
        result.AppendFormat("{1} {2} ({0})\r\n\tPoints: {3}\r\n\tHasGoldStatus:
{4}\r\n\tMemberSince: {5} ({7})\r\n\tCreditRating: {6}\r\n\tAverageRating:
{8}\r\n", Id, FirstName, LastName, Points, HasGoldStatus, MemberSince,
CreditRating, MemberSince.Kind, AverageRating);
        result.AppendLine("\tOrders:");
        foreach(var order in Orders) {
            result.AppendLine("\t\t" + order);
        }
        return result.ToString();
    }
}
```

When you run the application, you will see the same behavior. But, now we can have an order occurring multiple times in the same collection.

```
John Doe (00000000-0000-0000-0000-000000000000)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Unspecified)
        CreditRating: Good
        AverageRating: 42.42424242
        Orders:
                Order Id: 00000000-0000-0000-0000-000000000000
                Order Id: 00000000-0000-0000-0000-000000000000

Reloaded:
John Doe (fbde48f5-d620-4d1c-9a7f-a5b8017c3280)
        Points: 100
        HasGoldStatus: True
```

```
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: 6dd7dbdb-354f-4c82-9c39-a5b8017c3286
                Order Id: 9b3e2441-a81b-404d-9aed-a5b8017c3287

The orders were ordered by:
John Doe (fbde48f5-d620-4d1c-9a7f-a5b8017c3280)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: 6dd7dbdb-354f-4c82-9c39-a5b8017c3286
                Order Id: 9b3e2441-a81b-404d-9aed-a5b8017c3287

John Doe (fbde48f5-d620-4d1c-9a7f-a5b8017c3280)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: 6dd7dbdb-354f-4c82-9c39-a5b8017c3286
                Order Id: 9b3e2441-a81b-404d-9aed-a5b8017c3287

Press <ENTER> to exit...
```

18. NHibernate – Cascades

In this chapter, we will be covering how to use the Cascade feature. If you have a set or a collection of items or a relationship between two classes such as our customer and order and have a foreign key relationship. If we delete the customer by default, NHibernate doesn't do anything to the child objects, so the ones that belong to that customer and we could be orphaning orders.

- We could also be violating foreign key constraints, so we can use the notion of cascades.
- By default, NHibernate does not cascade operations to child objects.
- The reason for this is that you can have relationships such as a customer having a default shipping address and that shipping address is shared with many different customers.
- So you wouldn't want to cascade that relationship necessarily because other customers are still referring to it.
- So the whole notion of cascades is to tell NHibernate how to handle its child entities.

There are different options for cascading, which are as follows;

- **none:** which is the default and it means no cascading.
- **all:** which is going to cascade saves, updates, and deletes.
- **save-update:** it will cascade, saves and updates.
- **delete:** it will cascade deletes.
- **all-delete-orphan:** it is a special one which is quite frequently used and is the same as All Except, if it finds Delete-orphan rows, it will delete those as well.

You can specify the default in your **hbm.xml** file, so you can provide a default cascade on that Hibernate mapping element or you can also specify it for specific collections and relationships such as the many-to-one.

Let's have a look into simple example cascades, let's fix the problem in the program, where we have to manually cascade the save to the orders as shown in the following code.

```
using(var session = sessionFactory.OpenSession())
using(var tx = session.BeginTransaction()) {
    var newCustomer = CreateCustomer();
    Console.WriteLine("New Customer:");
    Console.WriteLine(newCustomer);
    session.Save(newCustomer);
    foreach (var order in newCustomer.Orders)
```

```
    {
        session.Save(order);
    }
    id = newCustomer.Id;
    tx.Commit();
}
```

In the above code snippet, you can see that we are manually saving all the orders for the customer. Now let's remove manual cascade code in which all the orders are saved.

```
using(var session = sessionFactory.OpenSession())
using(var tx = session.BeginTransaction()) {
    var newCustomer = CreateCustomer();
    Console.WriteLine("New Customer:");
    Console.WriteLine(newCustomer);
    session.Save(newCustomer);
    id = newCustomer.Id;
    tx.Commit();
}
```

We need to specify the cascade option in **customer.hbm.xml**.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Customer">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="FirstName"/>
    <property name="LastName"/>
    <property name="AverageRating"/>
    <property name="Points"/>
    <property name="HasGoldStatus"/>
    <property name="MemberSince" type="UtcDateTime"/>
    <property name="CreditRating" type="CustomerCreditRatingType"/>
```

```
    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>

    <set name="Orders" table="`Order`" cascade="all-delete-orphan">
      <key column="CustomerId"/>
      <one-to-many class="Order"/>
    </set>
  </class>
</hibernate-mapping>
```

- Now, orders fully belong to the customer. So if the customers were deleted from the database, our application here would want to delete all of those orders, including any that might have been orphaned.
- It will end up doing a delete. By that, it will say delete from order table, where the customer ID equals the customer that you're deleting.
- So you can actually cascade these deletes. So with the **All**, it will do saves, updates, and deletes.

Now when you run this application, you will see the following output.

```
New Customer:
John Doe (00000000-0000-0000-0000-000000000000)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Unspecified)
        CreditRating: Good
        AverageRating: 42.42424242
        Orders:
                Order Id: 00000000-0000-0000-0000-000000000000
                Order Id: 00000000-0000-0000-0000-000000000000

Reloaded:
John Doe (10b2a3d7-7fcf-483c-b1da-a5bb00b8512e)
        Points: 100
```

```
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: e6680e30-5b3b-4efa-b017-a5bb00b85133
                Order Id: b03858e7-8c36-4555-8878-a5bb00b85134

The orders were ordered by:
John Doe (10b2a3d7-7fcf-483c-b1da-a5bb00b8512e)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: e6680e30-5b3b-4efa-b017-a5bb00b85133
                Order Id: b03858e7-8c36-4555-8878-a5bb00b85134

John Doe (10b2a3d7-7fcf-483c-b1da-a5bb00b8512e)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: e6680e30-5b3b-4efa-b017-a5bb00b85133
                Order Id: b03858e7-8c36-4555-8878-a5bb00b85134

Press <ENTER> to exit...
```

As you can see that we have deleted the code from the program that manually cascaded and our application is still working.

So depending on your relationship, you might want to cascade those. Now, let's take a look at a different cascade relationship. Let’s go to the **Order.hbm.xml** file and we can cascade that many-to-one relationship.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Order" table="`Order`">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="Ordered"/>
    <property name="Shipped"/>
    <component name="ShipTo">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>
    <many-to-one name="Customer"
                 column="CustomerId"
                 cascade="save-update"/>
  </class>
</hibernate-mapping>
```

So if we create a new order and there's a new customer attached to it and we say, save that order, we might want to cascade that. But one thing that we'd probably don't want to do is if an order is deleted to delete the corresponding customer.

So here, we would want to do a save update, so using a save-update, it will cascade any saves or updates to that customer. So, if we get a new customer or if we are changing the customer, it will cascade that. If it is a delete, it won't delete that from the database.

So running our application again, everything still works as expected.

```
New Customer:
John Doe (00000000-0000-0000-0000-000000000000)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Unspecified)
        CreditRating: Good
        AverageRating: 42.42424242
        Orders:
```

```
                Order Id: 00000000-0000-0000-0000-000000000000
                Order Id: 00000000-0000-0000-0000-000000000000

Reloaded:
John Doe (10b2a3d7-7fcf-483c-b1da-a5bb00b8512e)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: e6680e30-5b3b-4efa-b017-a5bb00b85133
                Order Id: b03858e7-8c36-4555-8878-a5bb00b85134

The orders were ordered by:
John Doe (10b2a3d7-7fcf-483c-b1da-a5bb00b8512e)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: e6680e30-5b3b-4efa-b017-a5bb00b85133
                Order Id: b03858e7-8c36-4555-8878-a5bb00b85134

John Doe (10b2a3d7-7fcf-483c-b1da-a5bb00b8512e)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: e6680e30-5b3b-4efa-b017-a5bb00b85133
                Order Id: b03858e7-8c36-4555-8878-a5bb00b85134

Press <ENTER> to exit...
```

Now you should have a look at your application, remember that the default is None and you have to think about your entities and your relationships between them to determine the appropriate cascades for each of your entities as well as each of your relationships in that database.

19. NHibernate – Lazy Loading

In this chapter, we will be covering the lazy loading feature. It is an entirely different concept by default and NHibernate doesn't have lazy loading, for example if you load a customer, it's not going to load all of the orders.

- The order collection will be loaded on demand.
- Any association, whether it be a many-to-one or a collection is lazy loaded by default, it requires an **Open ISession**.
- If you have closed your session, or if you have committed your transaction, you can get a lazy load exception that it cannot pull in those additional objects.
- You have to be careful about lazy loading and how much data you actually need.
- You can turn off lazy loading for an entire association or you could put lazy equals false or you can also specify a fetching strategy.

Here is the **Program.cs** file implementation.

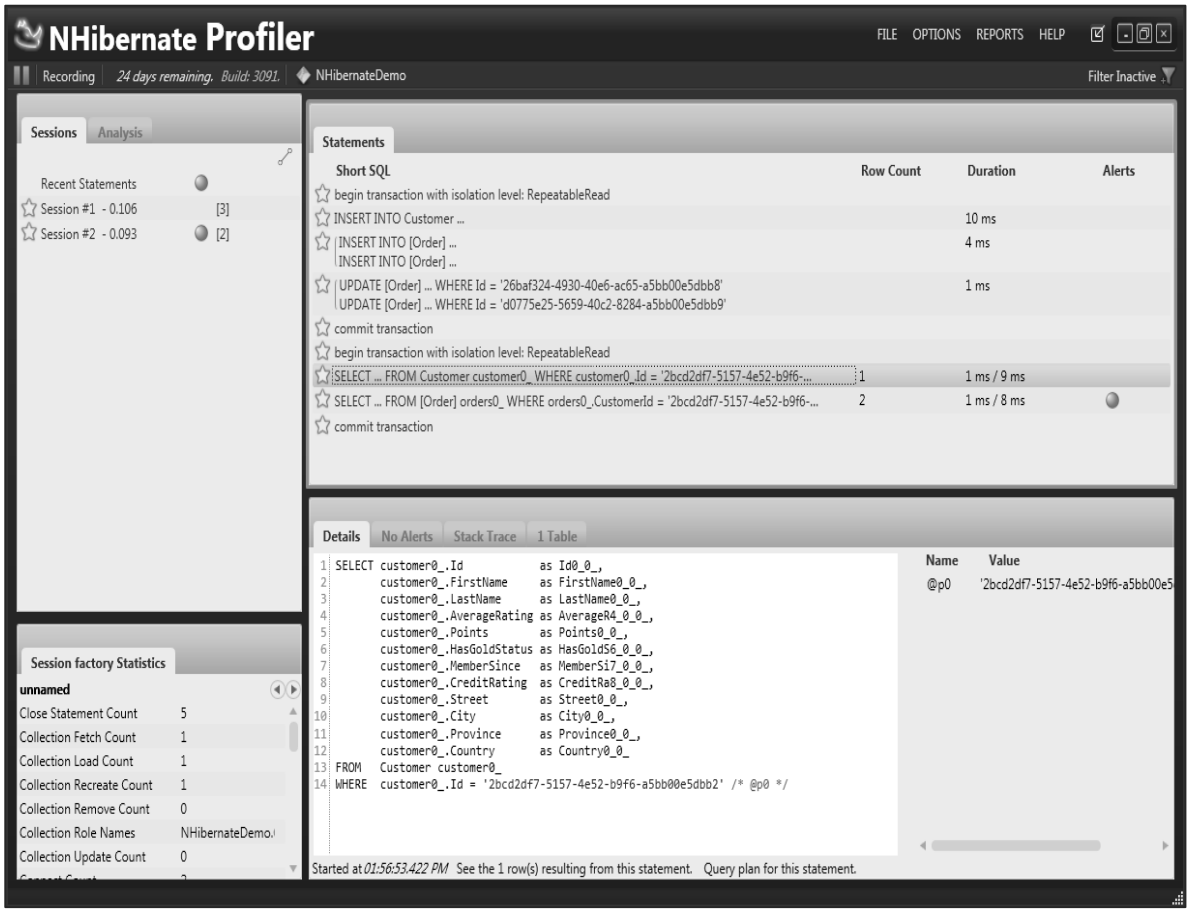
```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            Guid id;
            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
```

```
          var newCustomer = CreateCustomer();
                Console.WriteLine("New Customer:");
                Console.WriteLine(newCustomer);
                session.Save(newCustomer);
                id = newCustomer.Id;
                tx.Commit();
            }

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var reloaded = session.Load<Customer>(id);
                Console.WriteLine("Reloaded:");
                Console.WriteLine(reloaded);
                Console.WriteLine("The orders were ordered by: ");
                foreach (var order in reloaded.Orders)
                {
                    Console.WriteLine(order.Customer);
                }
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Customer CreateCustomer() {
            var customer = new Customer {
                                            FirstName = "John",
                                            LastName = "Doe",
                                            Points = 100,
                                            HasGoldStatus = true,
                                            MemberSince = new DateTime(2012, 1,
1),
                                            CreditRating =
CustomerCreditRating.Good,

                                        AverageRating = 42.42424242,
```

```
                                        Address = CreateLocation()
                                    };
            var order1 = new Order {
                Ordered = DateTime.Now
            };
            customer.AddOrder(order1);
            var order2 = new Order {
                                    Ordered = DateTime.Now.AddDays(-1),
                                    Shipped = DateTime.Now,
                                    ShipTo = CreateLocation()
                                };
            customer.AddOrder(order2);
            return customer;
        }

        private static Location CreateLocation() {
            return new Location
                    {
                        Street = "123 Somewhere Avenue",
                        City = "Nowhere",
                        Province = "Alberta",
                        Country = "Canada"
                    };
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                    x.ConnectionStringName = "default";
                                    x.Driver<SqlClientDriver>();
                                    x.Dialect<MsSql2008Dialect>();
                                    x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                    x.Timeout = 10;

                                x.BatchSize = 10;
```

```
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

To understand this, let's run the application and take a look at the NHibernate Profiler.

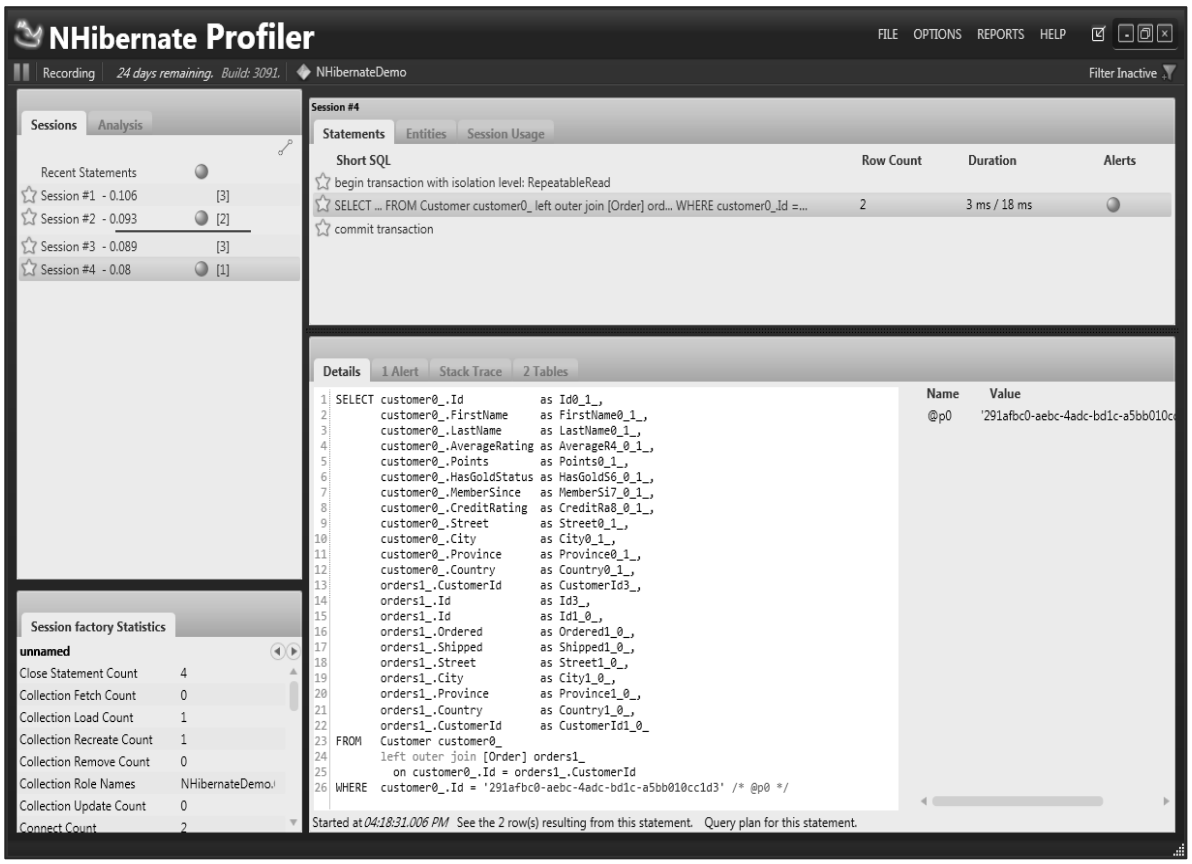


As you can see that we have the Select From Customer, given a particular customer ID and then we also have another Select From Orders table, when it actually accesses that customer's collection.

So we have 2 roundtrips to the database. Now, sometimes, we would want to optimize this. To do this, let's go to the **customer.hbm.xml** file and add a fetching strategy and ask it to do a join fetch.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Customer">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="FirstName"/>
    <property name="LastName"/>
    <property name="AverageRating"/>
    <property name="Points"/>
    <property name="HasGoldStatus"/>
    <property name="MemberSince" type="UtcDateTime"/>
    <property name="CreditRating" type="CustomerCreditRatingType"/>

    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>

    <set name="Orders" table="`Order`" cascade="all-delete-orphan"
fetch="join">
      <key column="CustomerId"/>
      <one-to-many class="Order"/>
    </set>
  </class>
</hibernate-mapping>
```

As you can see that we haven't changed any code in our application, we have just added a fetching strategy in the **customer.hbm.xml**. Let's run this application again, it still behaves exactly the same way. Let's look at NHibernate Profiler.



- Before, program had two round trips to the database, now, it only has one and that's because it's doing a left outer join here.
- We can see that it's doing a left outer join between the customer table and the order table based on the customer ID, and therefore, it's able to load in all of that information at once.
- We have saved 1 roundtrip to the database.
- The down side is that the customer information will be duplicated on both lines and that's the way that a SQL left outer join works.
- So with the fetching strategy, we are pulling back a bit more data and we are saving a roundtrip.

You can also do this at the query level, so let's go to the **Program.cs** file and look at the simpler reloaded example.

```
using(var session = sessionFactory.OpenSession())
using(var tx = session.BeginTransaction()) {
    //var query = from customer in session.Query<Customer>()
    //                select customer;
    //var reloaded = query.Fetch(x => x.Orders).ToList();
```

```
    var reloaded = session.Load<Customer>(id);
    Console.WriteLine("Reloaded:");
    Console.WriteLine(reloaded);
    Console.WriteLine("The orders were ordered by: ");
    foreach (var order in reloaded.Orders)
    {
        Console.WriteLine(order.Customer);
    }
    tx.Commit();
}
```

Here, we are doing a load by the customer. Now let's change it to a query and we will use a link query as shown in the following code.

```
using(var session = sessionFactory.OpenSession())
using(var tx = session.BeginTransaction()) {

    var query = from customer in session.Query<Customer>()
                where customer.Id == id
                select customer;
    var reloaded = query.Fetch(x => x.Orders).ToList().First();
    Console.WriteLine("Reloaded:");
    Console.WriteLine(reloaded);
    tx.Commit();
}
```

Let's also remove the fetching strategy from the **customer.hbm.xml** file.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Customer">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="FirstName"/>
    <property name="LastName"/>
```

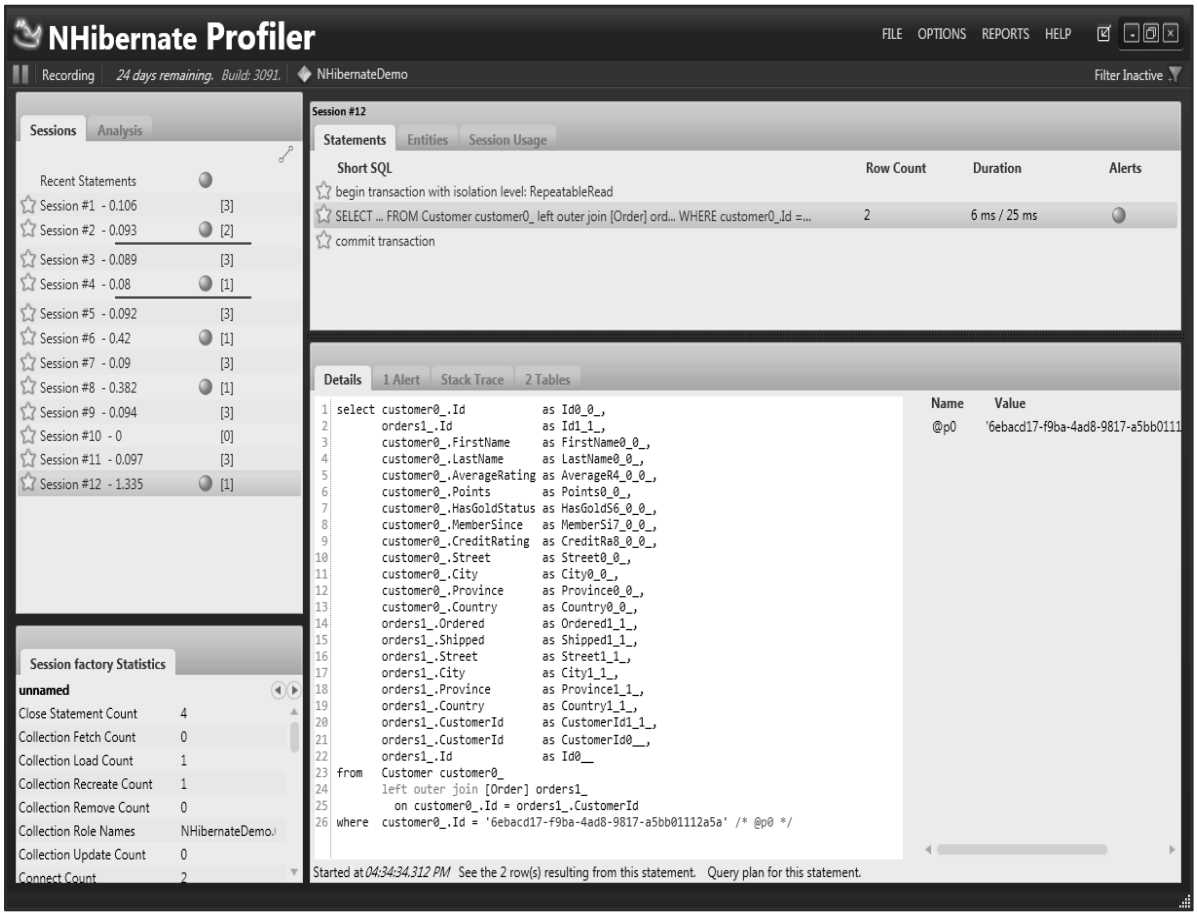
```
    <property name="AverageRating"/>
    <property name="Points"/>
    <property name="HasGoldStatus"/>
    <property name="MemberSince" type="UtcDateTime"/>
    <property name="CreditRating" type="CustomerCreditRatingType"/>

    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>

    <set name="Orders" table="`Order`" cascade="all-delete-orphan">
      <key column="CustomerId"/>
      <one-to-many class="Order"/>
    </set>
  </class>
</hibernate-mapping>
```

Let's run this application again and you will see the following output.

```
New Customer:
John Doe (00000000-0000-0000-0000-000000000000)
        Points: 100
        HasGoldStatus: True
        MemberSince: 1/1/2012 12:00:00 AM (Unspecified)
        CreditRating: Good
        AverageRating: 42.42424242
        Orders:
                Order Id: 00000000-0000-0000-0000-000000000000
                Order Id: 00000000-0000-0000-0000-000000000000

Reloaded:
John Doe (6ebacd17-f9ba-4ad8-9817-a5bb01112a5a)
        Points: 100
        HasGoldStatus: True
```

```
        MemberSince: 1/1/2012 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 42.4242
        Orders:
                Order Id: 16a6596b-d56e-41c7-9681-a5bb01112a60
                Order Id: d41d615b-0f21-4032-81db-a5bb01112a61

Press <ENTER> to exit...
```

Now let's look at the NHibernate Profiler, you can see that we've got this eager join fetch happening once again, but this time, it's based on the query.



20. NHibernate – Inverse Relationships

In this chapter, we will be covering another feature which is Inverse Relationships. It is an amusing option that you will see on collection that are inversely equal to true and it also confuses a lot of developers. So let's talk about this option. To understand this, you really have to think about the relational model. Let's say you have a bidirectional associations using a single foreign key.

- From a relational standpoint, you have got one foreign key, and it represents both customer to order and orders to customer.
- From the OO model, you have unidirectional associations using these references.
- There is nothing that says that two unidirectional associations represent the same bidirectional association in the database.
- The problem here is that NHibernate doesn't have enough information to know that **customer.orders** and **order.customer** represent the same relationship in the database.
- We need to provide **inverse equals true** as a hint, it is because the unidirectional associations are using the same data.
- If we try to save these relationships that have 2 references to them, NHibernate will try to update that reference twice.
- It will actually do an extra roundtrip to the database, and it will also have 2 updates to that foreign key.
- The inverse equals true tells NHibernate which side of the relationship to ignore.
- When you apply it to the collection side and NHibernate will always update the foreign key from the other side, from the child object side.
- Then we only have one update to that foreign key and we don't have additional updates to that data.
- This allows us to prevent these duplicate updates to the foreign key and it also helps us to prevent foreign key violations.

Let's have a look at the **customer.cs** file in which you will see the **AddOrder** method and the idea here is that we now have this back pointer from order back to customer and it needs to be set. So when an order is added to a customer, that customer's back pointer is set, otherwise, it would be null, so we need this to keep this connected properly together in the object graph.

```
using System;
using System.Text;
using Iesi.Collections.Generic;
```

```
namespace NHibernateDemo {
    public class Customer {
        public Customer() {
            MemberSince = DateTime.UtcNow;
            Orders = new HashedSet<Order>();

   }
        public virtual Guid Id { get; set; }
        public virtual string FirstName { get; set; }
        public virtual string LastName { get; set; }
        public virtual double AverageRating { get; set; }
        public virtual int Points { get; set; }
        public virtual bool HasGoldStatus { get; set; }
        public virtual DateTime MemberSince { get; set; }
        public virtual CustomerCreditRating CreditRating { get; set; }
        public virtual Location Address { get; set; }

        public virtual ISet<Order> Orders { get; set; }

        public virtual void AddOrder(Order order) {
            Orders.Add(order);
            order.Customer = this;
        }

        public override string ToString() {
            var result = new StringBuilder();
            result.AppendFormat("{1} {2} ({0})\r\n\tPoints:
{3}\r\n\tHasGoldStatus: {4}\r\n\tMemberSince: {5} ({7})\r\n\tCreditRating:
{6}\r\n\tAverageRating: {8}\r\n", Id, FirstName, LastName, Points,
HasGoldStatus, MemberSince, CreditRating, MemberSince.Kind, AverageRating);
            result.AppendLine("\tOrders:");
            foreach(var order in Orders) {
                result.AppendLine("\t\t" + order);
            }
            return result.ToString();
        }
    }
```

```
    public class Location {
        public virtual string Street { get; set; }
        public virtual string City { get; set; }
        public virtual string Province { get; set; }
        public virtual string Country { get; set; }

   }
    public enum CustomerCreditRating {
        Excellent, VeryVeryGood, VeryGood, Good, Neutral, Poor, Terrible
    }
}
```

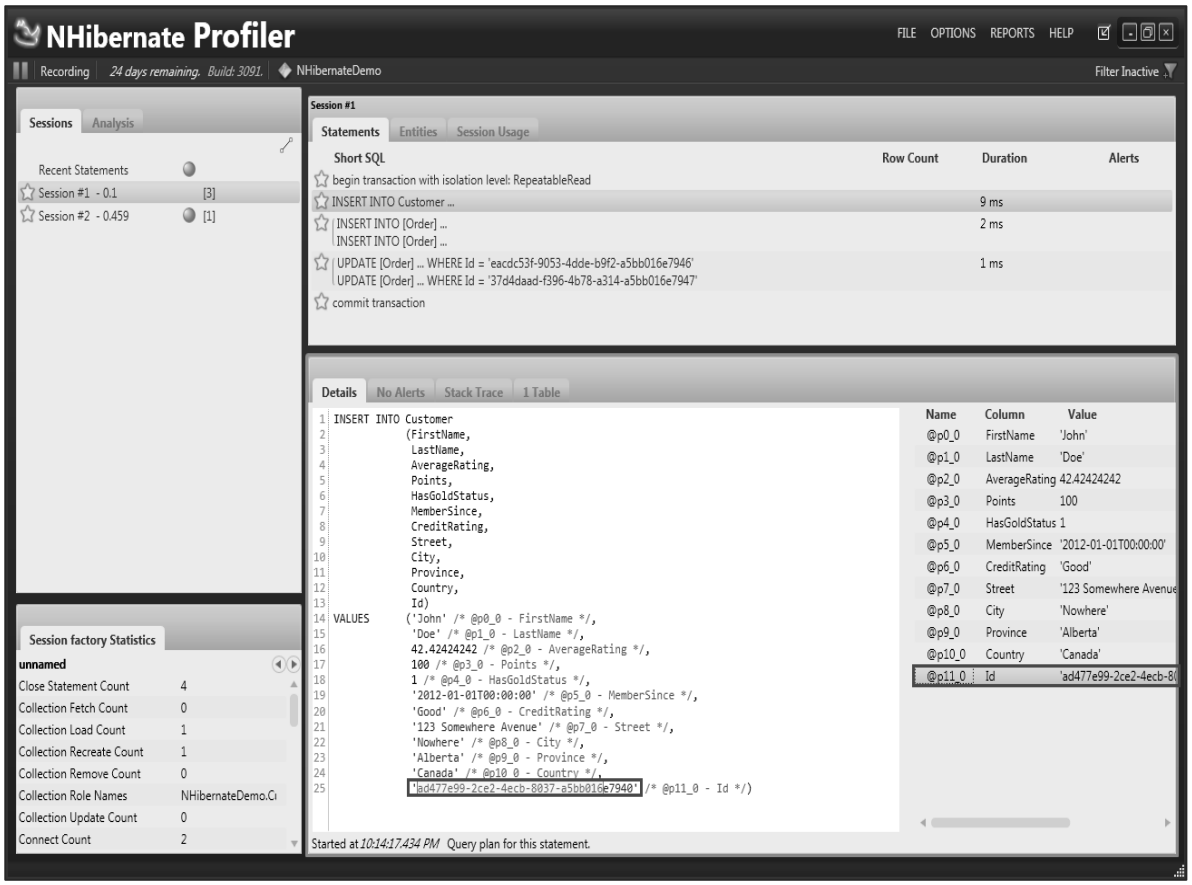
Here is the **Program.cs** file implementation.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            Guid id;
            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var newCustomer = CreateCustomer();
                Console.WriteLine("New Customer:");
                Console.WriteLine(newCustomer);
                session.Save(newCustomer);
                id = newCustomer.Id;
```

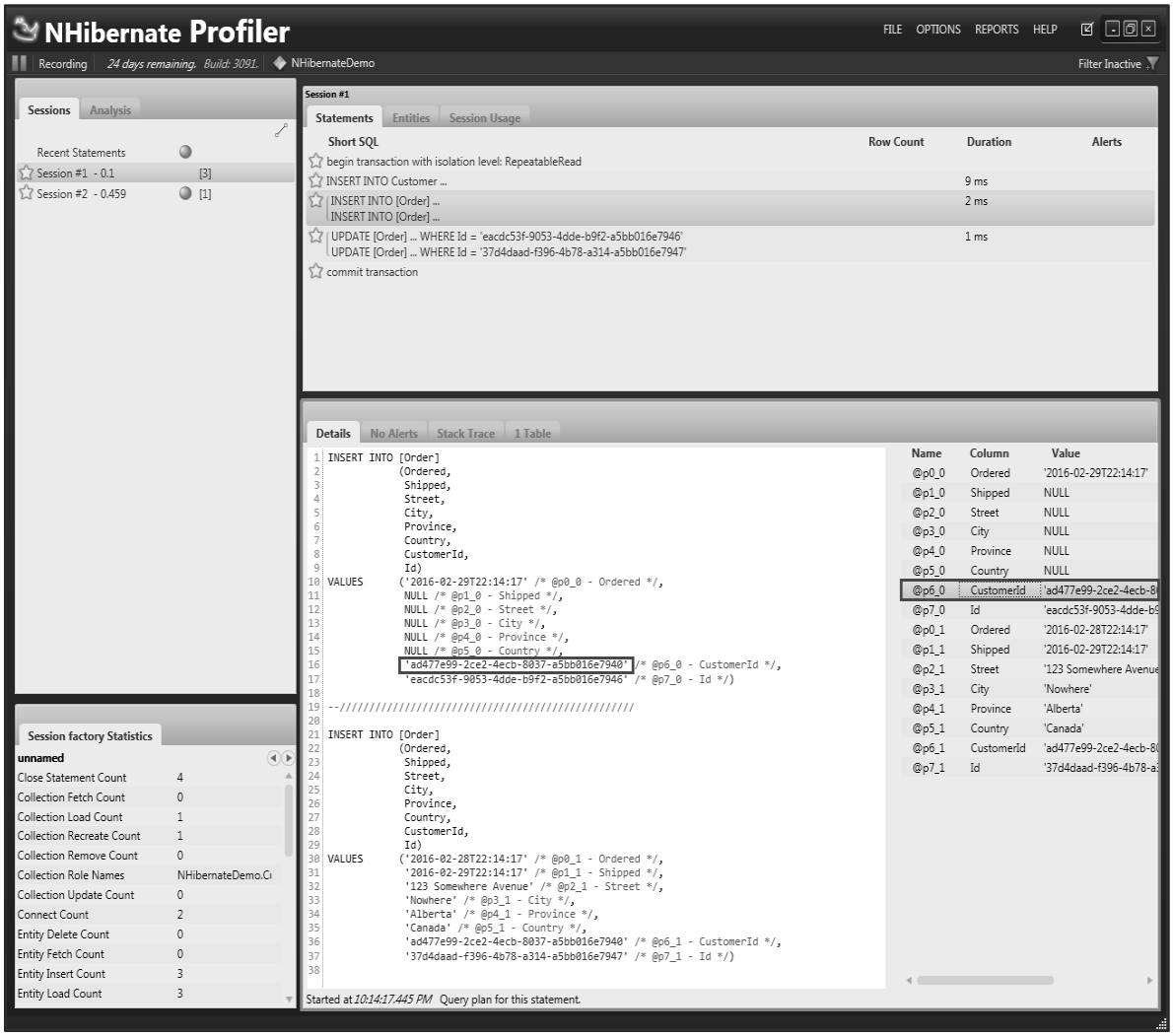
```
                tx.Commit();
            }

            using(var session = sessionFactory.OpenSession())

          using(var tx = session.BeginTransaction()) {
                var query = from customer in session.Query<Customer>()
                            where customer.Id == id
                            select customer;
                var reloaded = query.Fetch(x => x.Orders).ToList().First();
                Console.WriteLine("Reloaded:");
                Console.WriteLine(reloaded);
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Customer CreateCustomer() {
            var customer = new Customer {
                                        FirstName = "John",
                                        LastName = "Doe",
                                        Points = 100,
                                        HasGoldStatus = true,
                                        MemberSince = new DateTime(2012, 1,
1),
                                        CreditRating =
CustomerCreditRating.Good,
                                        AverageRating = 42.42424242,
                                        Address = CreateLocation()
                                    };
            var order1 = new Order {
                Ordered = DateTime.Now
            };
            customer.AddOrder(order1);
            var order2 = new Order {
```

```
                                    Ordered = DateTime.Now.AddDays(-1),
                                    Shipped = DateTime.Now,
                                    ShipTo = CreateLocation()

                            };
            customer.AddOrder(order2);
            return customer;
        }

        private static Location CreateLocation() {
            return new Location
                   {                        Street = "123 Somewhere Avenue",
                       City = "Nowhere",
                       Province = "Alberta",
                       Country = "Canada"                   };
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;        }
    }
}
```

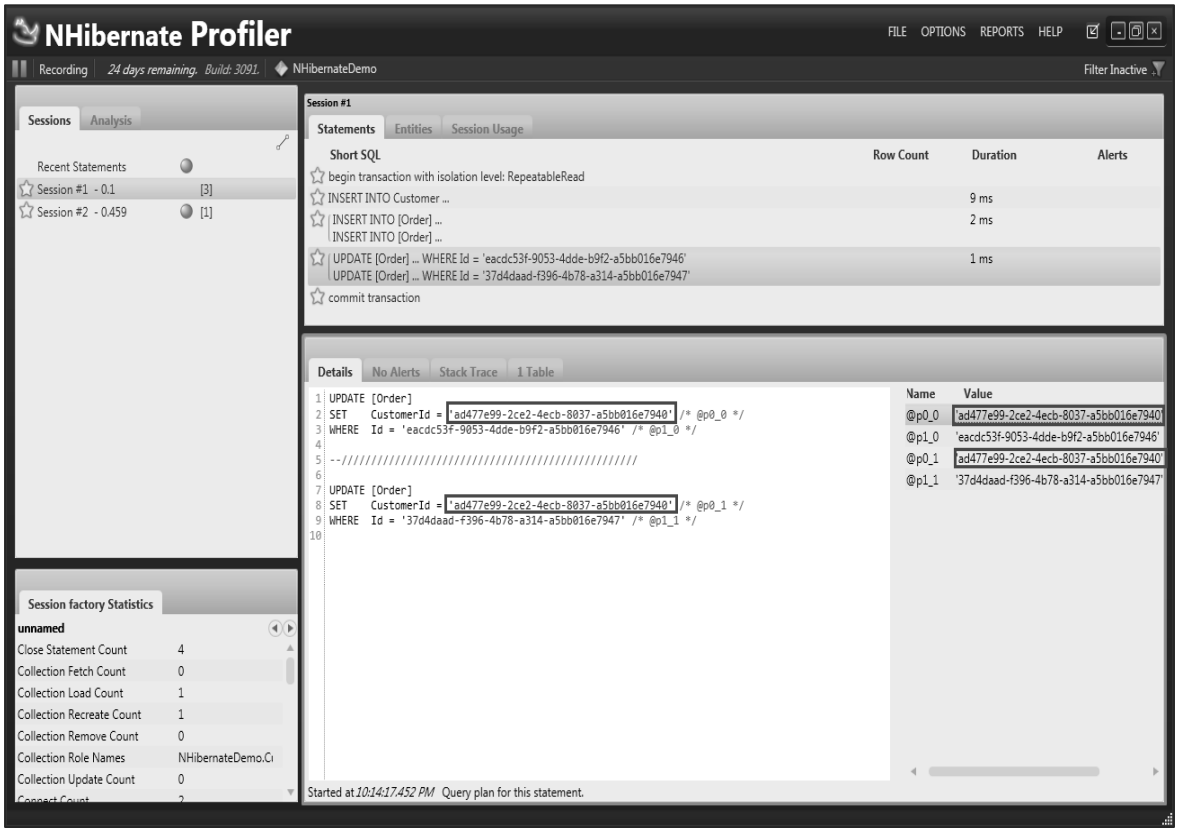
It is going to save that to the database and then reload it. Now let's run your application and open the NHibernate Profiler and see how it actually saved it.



You will notice that we have 3 groups of statements. The first one will insert the customer, and that customer's ID is the Guid, which is highlighted. The second statement is insert into the orders table.



You will notice the same Customer Id Guid is set in there, so have that foreign key set. The last statement is the update, which will update the foreign key to the same customer id once again.



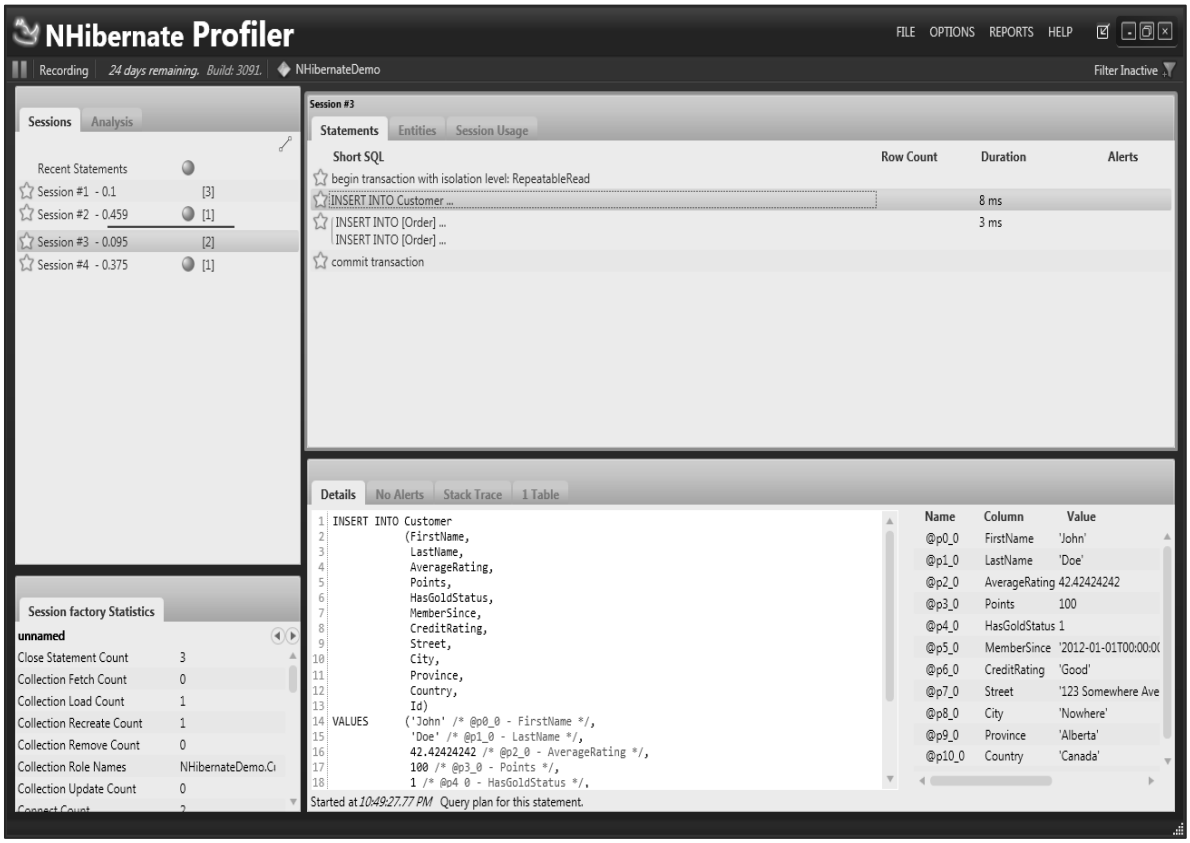
Now the problem is that the customer has the orders, and the orders has the customer, there's no way that we haven't told NHibernate that it's actually the same relationship. The way we do this is with inverse equals true.

So let's go to our **customer.hbm.xml** mapping file and set the inverse equal to true as shown in the following code.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                   assembly="NHibernateDemo"
                   namespace="NHibernateDemo">
  <class name="Customer">
    <id name="Id">
      <generator class="guid.comb"/>
    </id>
    <property name="FirstName"/>
    <property name="LastName"/>
    <property name="AverageRating"/>
    <property name="Points"/>
    <property name="HasGoldStatus"/>
    <property name="MemberSince" type="UtcDateTime"/>
```

```
    <property name="CreditRating" type="CustomerCreditRatingType"/>

    <component name="Address">
      <property name="Street"/>
      <property name="City"/>
      <property name="Province"/>
      <property name="Country"/>
    </component>

    <set name="Orders" table="`Order`" cascade="all-delete-orphan"
inverse="true">
      <key column="CustomerId"/>
      <one-to-many class="Order"/>
    </set>
  </class>
</hibernate-mapping>
```

When saving the orders, it will set that foreign key from the order side. Now let's run this application again and open the NHibernate profiler.



If we look at how those are inserted, we get the insert in the customer, and the insert into orders, but we don't have that duplicate update of the foreign key because it's being updated when the orders are being saved.

- Now, you should note that if you only have a unidirectional association and it's the set that is maintaining this relationship, then if you turn inverse equals true, that foreign key is never going to be set, and those items are never going to have their foreign keys set in the database.

- If you look at the many-to-one relationship in the **Order.hbm.xml** file and you look for inverse, it doesn't actually have an inverse attribute.

- It always is set from the child item, but if you have a many-to-many collection, you can set it from either side.

21. NHibernate – Load/Get

In this chapter, we will be covering how the Load and Get features are working and how we can use them. These are two very similar APIs provided by **ISession** for loading an object by primary key.

- **Get:** it will return the object or a null.
- **Load:** it will return the object or it will throw an **ObjectNotFoundException**.

Now, why do we have these two different APIs?

Load

- It's because Load can optimize database round trips much more efficiently.
- Load actually returns a proxy object and doesn't need to access the database right when you issue that Load call.
- When you access that proxy, the object doesn't happen to be in the database, it can throw an ObjectNotFoundException at that point.

Get

- Conversely, with Get because of limitations of the CLR or **Common Language Runtime** and NHibernate must go to the database immediately, check if the objects are there and return null, if it's not present.
- It doesn't have the object option of delaying that fetch, that roundtrip to the database to a later time because it cannot return a proxy object and that swapped that proxy object out for a null, when the user actually accesses it.

Let's have a look into a simple example in which you will see how these are actually used and the difference between Get and Load. We will continue with same domain classes **Customers** and **Orders** and similarly the same mapping files from the last chapter.

In this example, we will first use the Get as shown in the following program.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
```

```
using NHibernate.Linq;
namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var id1 = Guid.Parse("4e97c816-6bce-11e1-b095-6cf049ee52be");
                var id2 = Guid.Parse("AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE");

                var customer1 = session.Get<Customer>(id1);
                Console.WriteLine("Customer1 data");
                Console.WriteLine(customer1);

                var customer2 = session.Get<Customer>(id2);
                Console.WriteLine("Customer2 data");
                Console.WriteLine(customer2);
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
```

```
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

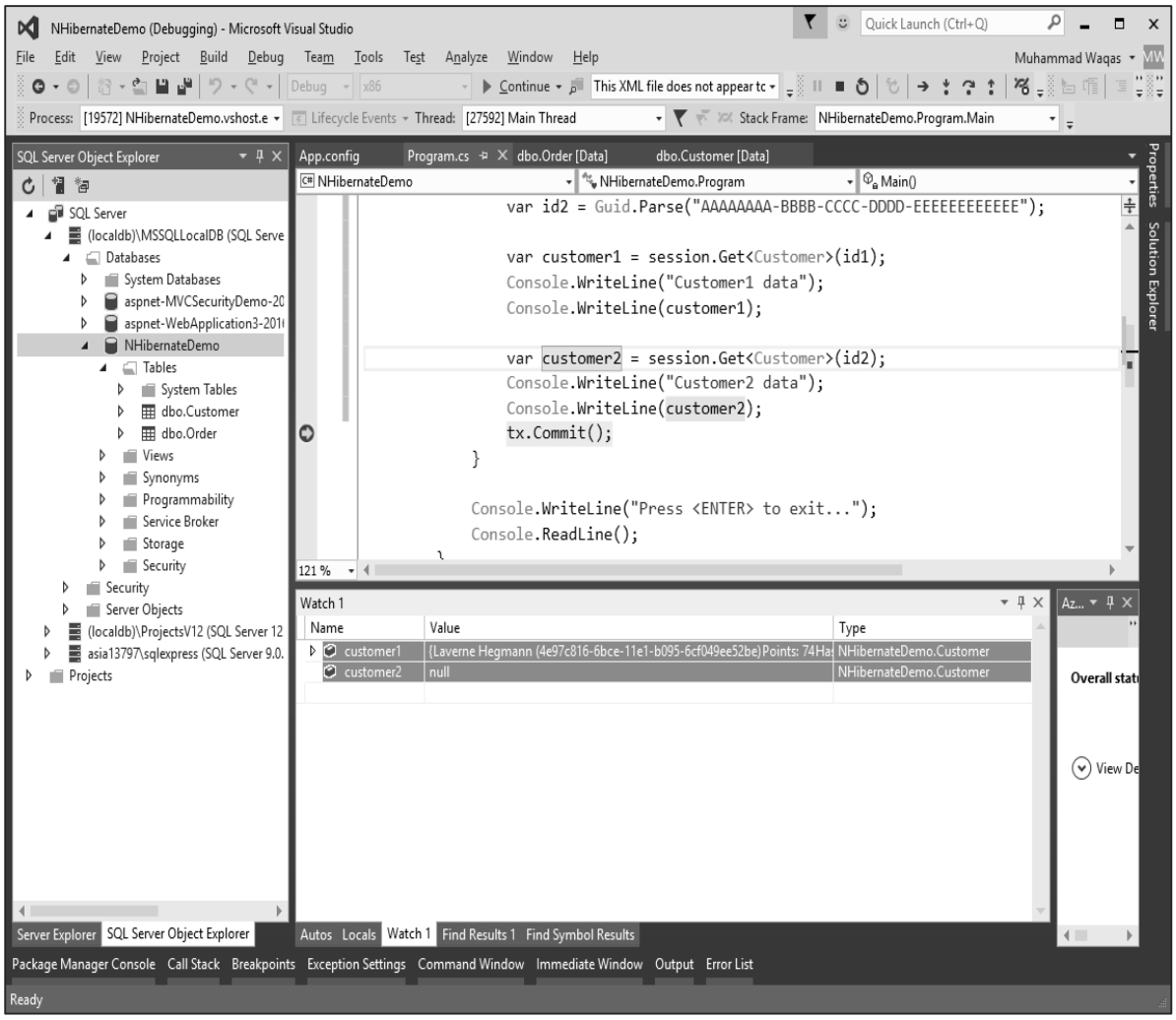
As you can see that we have two **Guid** ID's, the first one is a good ID, it's the ID of a customer that we know is in the database. While the second ID is not present in the database. Both these ID's are passed as a parameter to **Get()** method and then the result is printed on the console.

When the above code is compiled and executed you will see the following output.

```
Customer1 data
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Customer2 data

Press <ENTER> to exit...
```

As you can see that Customer1 data is printed but the Customer2 data is empty, that is because the Customer2 record is not available in the database.

When you run your application again, we can insert a break point before the commit statement and then let's look at both customers in the Watch window.



As you can see that Customer1 data is available, while Customer2 is null and the type is **NHibernateDemo.Customer** for both.

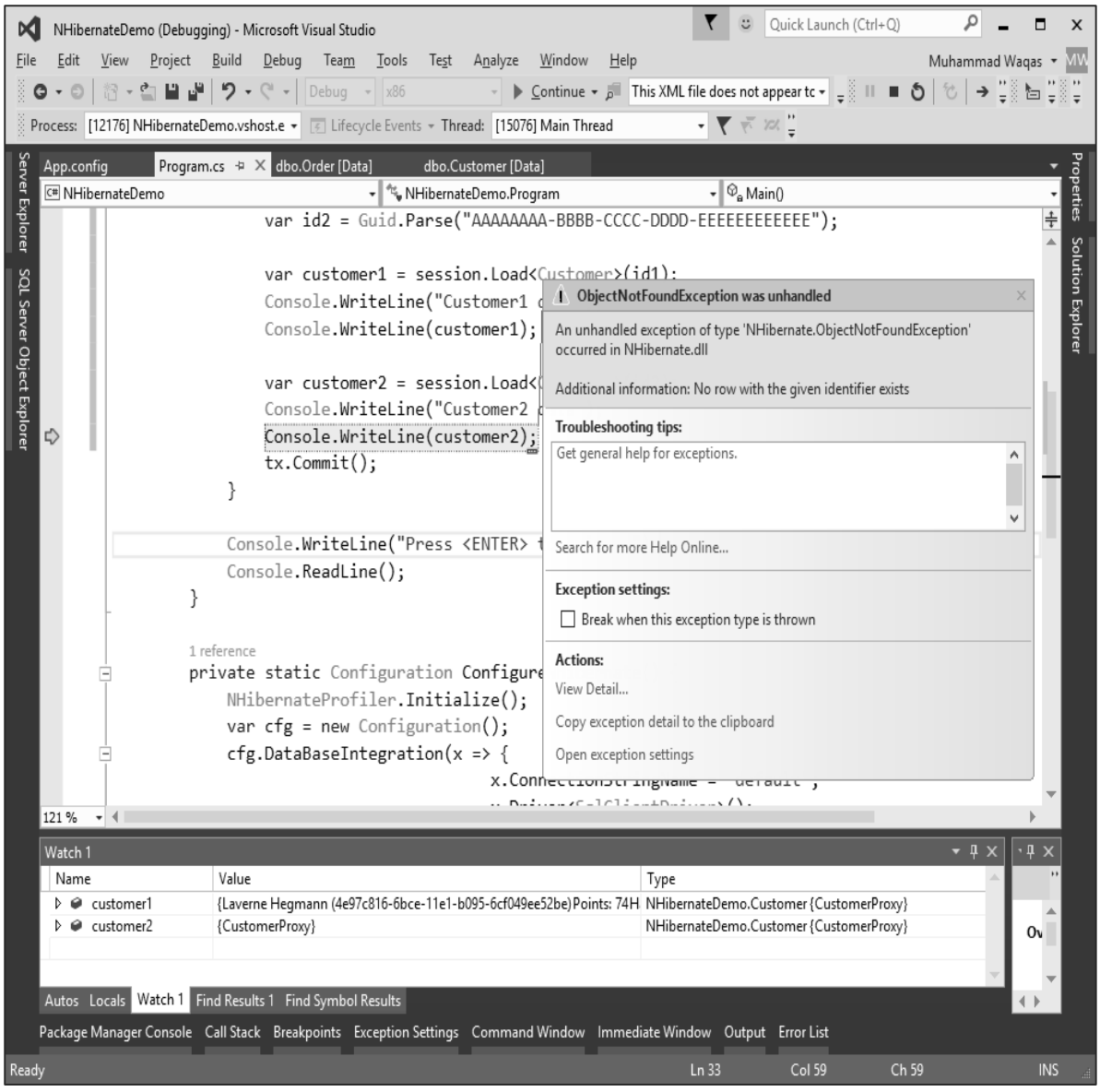
Now let's use the Load method instead of Get in the same example as shown in the following code.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
```

```
using NHibernate.Linq;
namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var id1 = Guid.Parse("4e97c816-6bce-11e1-b095-6cf049ee52be");
                var id2 = Guid.Parse("AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE");

                var customer1 = session.Load<Customer>(id1);
                Console.WriteLine("Customer1 data");
                Console.WriteLine(customer1);

                var customer2 = session.Load<Customer>(id2);
                Console.WriteLine("Customer2 data");
                Console.WriteLine(customer2);
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
```

```
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

Now let's run this example and you will see that the following exception is thrown as seen in the screenshot.



Now if you look at the Watch window, you will see the type is customer proxy for both objects. And you also see the same data for Customer1 on the console window.

```
Customer1 data
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Customer2 data
```

22. NHibernate – LINQ

In this chapter, we will be covering another common API that people will use is the NHibernate LINQ provider. Its access through an extension method on ISession and the signature is a **Query <T>**. There are two types of syntax while using LINQ –

- Query Chaining Syntax
- Query Comprehension Syntax

Query Chaining Syntax

You can access any record from the database using the method chain syntax as shown in the following program.

```
var customer = session.Query<Customer>()
               .Where(c => c.FirstName == "Laverne")
```

- You can see that we have query, and also WHERE clause, you can have additional WHERE clauses and similarly select clause.
- This is a standard method chain syntax that you can use in normal LINQ.
- LINQ to Objects or LINQ to SQL, any of the other LINQ providers you might be familiar with.

Let's have a look into a simple example in which we will retrieve the customer whose first name is Laverne. Now it is a possibility that we might have more than one customer whose first name is Laverne, so we will retrieve the first one only.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
```

```
     private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {

                var customer = session.Query<Customer>()
                                .Where(c => c.FirstName == "Laverne").First();
                Console.WriteLine(customer);
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

Now when the above code is compiled and executed you will see the following output.

```
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Press <ENTER> to exit...
```

Query Comprehension Syntax

There's also the query comprehensions syntax, which looks a lot more like SQL using the from, where and select keywords.

So let's have a look into the same example, but this time we use LINQ comprehensions syntax, which looks a lot more like SQL as shown in the following program.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();
```

```
            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {

                var customer = (from c in session.Query<Customer>()
                                where c.FirstName == "Laverne"
                                select c).First();
                Console.WriteLine(customer);
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

Now let's run this application again and you will see the following output.

```
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Press <ENTER> to exit...
```

Let's have a look into another example in which we will retrieve all those customers, whose FirstName starts with the letter H.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
```

```
                var customers = session.Query<Customer>()
                                .Where(c => c.FirstName.StartsWith("H"));

                foreach (var customer in customers.ToList())
                {
                    Console.WriteLine(customer);
                }

                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

Similarly, the query comprehension syntax will look like the following program.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {

                var customers = from c in session.Query<Customer>()
                                where c.FirstName.StartsWith("H")
                                select c;

                foreach (var customer in customers.ToList())
                {
                    Console.WriteLine(customer);
                }

                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }
```

```
        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

Let's run this application again and you will see all the customers, whose first name starts with the alphabet H.

```
Herman Crooks (4ead3480-6bce-11e1-b15c-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 12/3/2010 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ead3480-6bce-11e1-b15d-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b15e-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b15f-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b160-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b161-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b162-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b163-6cf049ee52be
```

```
Hudson Bins (4ec03f80-6bce-11e1-b2b7-6cf049ee52be)
        Points: 56
        HasGoldStatus: False
        MemberSince: 10/20/2008 12:00:00 AM (Utc)
        CreditRating: Terrible
        AverageRating: 0
        Orders:
                Order Id: 4ec03f80-6bce-11e1-b2b8-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2b9-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2ba-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bb-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bc-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bd-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2be-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bf-6cf049ee52be

Hettie Feest (4ec50240-6bce-11e1-b300-6cf049ee52be)
        Points: 82
        HasGoldStatus: False
        MemberSince: 4/10/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ec50240-6bce-11e1-b301-6cf049ee52be
                Order Id: 4ec50240-6bce-11e1-b302-6cf049ee52be
                Order Id: 4ec50240-6bce-11e1-b303-6cf049ee52be

Press <ENTER> to exit...
```

23. NHibernate – Hibernate Query Language

In this chapter, we will be covering Hibernate Query Language. HQL is shared across both Java's Hibernate and NHibernate.

- It is the oldest query mechanism along with **Criteria**.
- It was implemented very early and it is a string-based query **API**.
- You access it through **ISession CreateQuery**, and it is almost similar to SQL.
- It uses many of the same keywords, but has a simplified syntax.
- It is one of the most common examples, if you're looking for how to perform a query you'll often find HQL examples.

The following is a simple example of HQL –

```
var customers = session.CreateQuery("select c from Customer c
                                    where c.FirstName = 'Laverne'");
```

- So here you can see that they select C from customer, it looks a lot like SQL. This is an opaque string as far as NHibernate is concerned, so you don't know whether this is a valid HQL until runtime, which is one of the disadvantages.

- One of the strengths of the LINQ provider is you can get to compile time support.

- But HQL, is one of the most flexible query mechanisms oftenly used. It is said that, if there's no other way to do it then there's a way to do it in HQL.

Let's have a look into a simpe example in which we will recreate our LINQ queries using HQL instead. You can get access to HQL by calling the **session.CreateQuery** and pass as a parameter using an HQL string.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
```

```
 internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var customers = session.CreateQuery("select c from Customer c
where c.FirstName = 'Laverne'");
                foreach (var customer in customers.List<Customer>())
                {                    Console.WriteLine(customer);
                }

                tx.Commit();            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
      }
    }
}
```

- This HQL string looks a lot like SQL, the main difference is that FirstName is the property name and not the column name.
- So, if there's a discrepancy between the two, you use the property name. Same thing, it looks like a table name, but it's actually the name of the class that we are selecting from.
- If the back end table was named as Customers, we would still use Customer in our HQL query.

Let's run this application and you will see the following output.

```
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Press <ENTER> to exit...
```

Let's have a look into another simple example in which we will retrieve all those customers whose FirstName starts with the letter H using HQL.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;

using NHibernate.Linq;
namespace NHibernateDemo {
    internal class Program {
```

```
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var customers = session.CreateQuery("select c from Customer c
where c.FirstName like 'H%'");
                foreach (var customer in customers.List<Customer>())
                {
                    Console.WriteLine(customer);
                }

                tx.Commit();
            }
            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }
        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
                  cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

Let's run your application again and you will see that all of the customers whose name starts with H are returned from this query.

```
Herman Crooks (4ead3480-6bce-11e1-b15c-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 12/3/2010 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ead3480-6bce-11e1-b15d-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b15e-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b15f-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b160-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b161-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b162-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b163-6cf049ee52be

Hudson Bins (4ec03f80-6bce-11e1-b2b7-6cf049ee52be)
        Points: 56
        HasGoldStatus: False
        MemberSince: 10/20/2008 12:00:00 AM (Utc)
        CreditRating: Terrible
        AverageRating: 0
        Orders:
                Order Id: 4ec03f80-6bce-11e1-b2b8-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2b9-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2ba-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bb-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bc-6cf049ee52be

          Order Id: 4ec03f80-6bce-11e1-b2bd-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2be-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bf-6cf049ee52be

Hettie Feest (4ec50240-6bce-11e1-b300-6cf049ee52be)
        Points: 82
        HasGoldStatus: False
```

```
        MemberSince: 4/10/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ec50240-6bce-11e1-b301-6cf049ee52be
                Order Id: 4ec50240-6bce-11e1-b302-6cf049ee52be
                Order Id: 4ec50240-6bce-11e1-b303-6cf049ee52be

Press <ENTER> to exit...
```

We can do more complicated things like wanting all orders where customers with an order count is greater than 9. Following is the HQL query for the same.

```
var customers = session.CreateQuery("select c from Customer c where
size(c.Orders) > 9");
foreach (var customer in customers.List<Customer>())
{
    Console.WriteLine(customer);
}
```

We also need to indicate that we need a size here or count or length. In HQL, we have the option of using the special size method as shown above.

The other way to write this, if you prefer is **c.Orders.size**, and this has the exact effect.

```
var customers = session.CreateQuery("select c from Customer c where
c.Orders.size > 9");
foreach (var customer in customers.List<Customer>())
{
    Console.WriteLine(customer);
}
```

Let's run this application.

```
Lindsay Towne (4ea3aef6-6bce-11e1-b0cb-6cf049ee52be)
        Points: 50
        HasGoldStatus: False
        MemberSince: 4/13/2007 12:00:00 AM (Utc)
        CreditRating: VeryGood
        AverageRating: 0
        Orders:
```

```
                Order Id: 4ea3aef6-6bce-11e1-b0cc-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0cd-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0ce-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0cf-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0d0-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0d1-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0d2-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0d3-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0d4-6cf049ee52be
                Order Id: 4ea3aef6-6bce-11e1-b0d5-6cf049ee52be

Wyman Hammes (4ea61056-6bce-11e1-b0e2-6cf049ee52be)
        Points: 32
        HasGoldStatus: False
        MemberSince: 2/5/2011 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 0
        Orders:
                Order Id: 4ea61056-6bce-11e1-b0e3-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0e4-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0e5-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0e6-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0e7-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0e8-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0e9-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0ea-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0eb-6cf049ee52be
                Order Id: 4ea61056-6bce-11e1-b0ec-6cf049ee52be
```

```
Press <ENTER> to exit...
```

You can see that all the customers, who have more than 9 orders are retrieved from the database.

24. NHibernate – Criteria Queries

In this chapter, we will be covering criteria queries mechanism. The **NHibernate Query by Criteria API** lets you build a query by manipulating criteria objects at runtime.

- This approach lets you specify constraints dynamically without direct string manipulations, but it doesn't lose much of the flexibility or power of HQL.
- On the other hand, queries expressed as criteria are often less readable than queries expressed in HQL.
- The classic criteria syntax is an object based query API as shown in the following program.

```
var customers = session.CreateCriteria<Customer>()
                .Add(Restrictions.Like("FirstName", "H%"));
```

- As you can see we are doing a session create criteria on the customer, and now we're adding restriction object to that query.
- This is useful for query pages where users can select certain options, but not others.
- It's easier to build up the query as the sort of tree like query structure rather than in HQL or LINQ, where you can use the AND or OR in WHERE clause.
- It is easier just to add on additional restrictions using these criteria objects.

Let's have a look into a simple example in which we will create a query and get access to the criteria API through **createCriteria** and then add a restriction that the first name starts with H.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;
namespace NHibernateDemo {
```

```
 internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var customers = session.CreateCriteria<Customer>()
                                .Add(Restrictions.Like("FirstName", "H%"));
                foreach (var customer in customers.List<Customer>())
                {
                    Console.WriteLine(customer);
                }
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }
        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

When the above code is compiled and executed you will see the following output.

```
Herman Crooks (4ead3480-6bce-11e1-b15c-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 12/3/2010 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ead3480-6bce-11e1-b15d-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b15e-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b15f-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b160-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b161-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b162-6cf049ee52be
                Order Id: 4ead3480-6bce-11e1-b163-6cf049ee52be

Hudson Bins (4ec03f80-6bce-11e1-b2b7-6cf049ee52be)
        Points: 56
        HasGoldStatus: False
        MemberSince: 10/20/2008 12:00:00 AM (Utc)
        CreditRating: Terrible
        AverageRating: 0
        Orders:
                Order Id: 4ec03f80-6bce-11e1-b2b8-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2b9-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2ba-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bb-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bc-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bd-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2be-6cf049ee52be
                Order Id: 4ec03f80-6bce-11e1-b2bf-6cf049ee52be

Hettie Feest (4ec50240-6bce-11e1-b300-6cf049ee52be)
        Points: 82
        HasGoldStatus: False
        MemberSince: 4/10/2009 12:00:00 AM (Utc)
```

```
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ec50240-6bce-11e1-b301-6cf049ee52be
                Order Id: 4ec50240-6bce-11e1-b302-6cf049ee52be
                Order Id: 4ec50240-6bce-11e1-b303-6cf049ee52be

Press <ENTER> to exit…
```

Let's have a look into another simple example in which we will retrieve the customer whose first name is equal to "Laverne"

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var customers = session.CreateCriteria<Customer>()
                                .Add(Restrictions.Eq("FirstName", "Laverne"))
                                .List<Customer>();
                foreach (var customer in customers)
                {
```

```
            Console.WriteLine(customer);
                }
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

Let's run this application again and you will see the following output.

```
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
```

```
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Press <ENTER> to exit...
```

Now, one of the major disadvantages of the criteria API are these opaque strings in the property names. So, if the first name was refactored to be something else, the refactoring tool would not necessarily pick up the opaque string.

25. NHibernate – QueryOver Queries

In this chapter, we will be covering QueryOver Queries. It is a new syntax which is more like LINQ using the method chain syntax as shown in the following query.

```
var customers = session.QueryOver<Customer>()
                .Where(x => x.FirstName == "Laverne");
```

- It is still criteria under the covers, but now our queries are strongly typed.
- As we have seen in the criteria query, the first name is just an opaque string, now we're actually using an **x.FirstName**, so the first name gets refactored and renamed that gets changed in the link style criteria query using the query over.
- We can still do many similar things, but you cannot use the query comprehension syntax with query over, you have to use the method chain syntax and you can't mix and match the link and the criteria.
- For a lot of queries, the query over API is very useful and provides a much easier to comprehend object syntax than using Criteria directly.

Let's have a look into a simple example in which we will retrieve a customer whose first name is Laverne using a query over.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();
```

```
            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {
                var customers = session.QueryOver<Customer>()
                                .Where(x => x.FirstName == "Laverne");
                foreach (var customer in customers.List())
                {
                    Console.WriteLine(customer);
                }
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                    x.ConnectionStringName = "default";
                                    x.Driver<SqlClientDriver>();
                                    x.Dialect<MsSql2008Dialect>();
                                    x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                    x.Timeout = 10;
                                    x.BatchSize = 10;
                                });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

As you can see that it is still Criteria underneath the covers, but is just a nicer syntax.

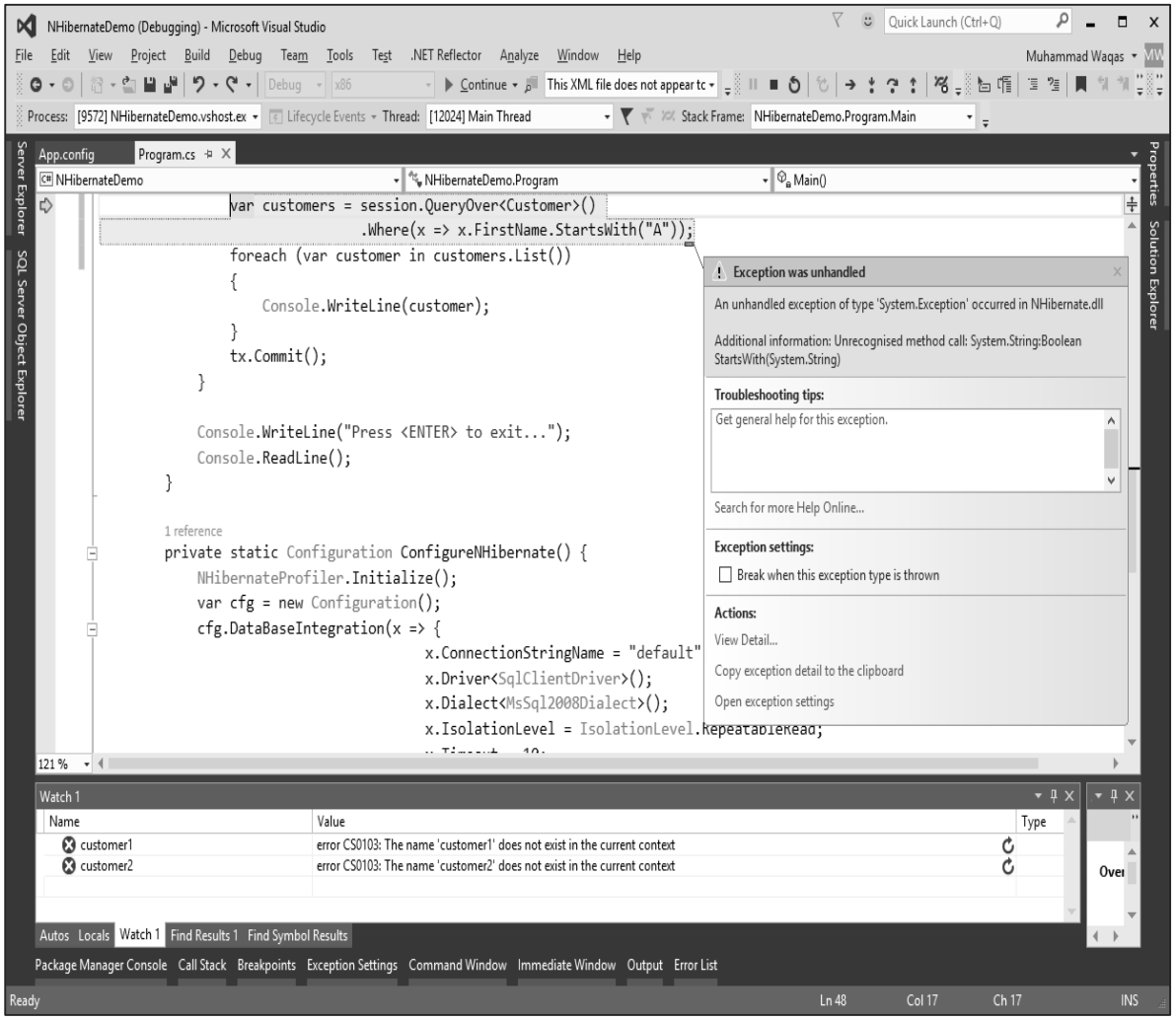
When the above code is compiled and executed, you will see the following output.

```
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Press <ENTER> to exit...
```

One of the disadvantages is that, let's say we want to say that **FirstName.StartsWith("A")** as shown in the following program.

```
var customers = session.QueryOver<Customer>()
                .Where(x => x.FirstName.StartsWith("A"));
foreach (var customer in customers.List())
{
    Console.WriteLine(customer);
}
tx.Commit();
```

Now let's run the application again and you will see that this is not a LINQ provider as it doesn't know what this **StartsWith** method is, so you will get a **RunTime exception**.



The exception says unrecognized method call. Here we are doing the obvious thing, but it doesn't necessarily work.

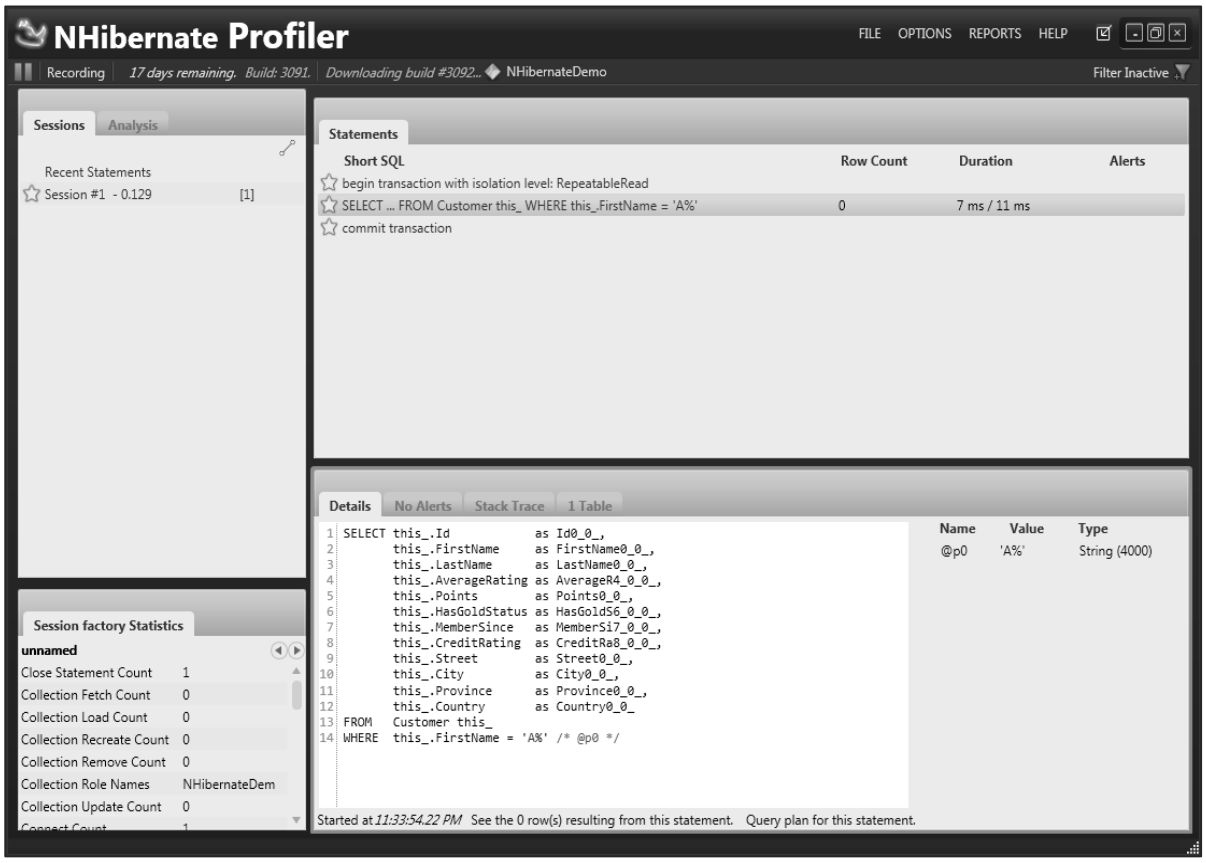
Let's try something else, like FirstName is equal to "A%" as shown in the following code.

```
var customers = session.QueryOver<Customer>()
                .Where(x => x.FirstName == "A%");
foreach (var customer in customers.List())
{
    Console.WriteLine(customer);
}
```

Let's run this once again and you will see that we're not going to get any results back as shown below.

```
Press <ENTER> to exit...
```

To understand this why we are not getting any results, let's have a look at NHibernate profiler.



As you can see that the first name is equal to A% which is not. A% is used in SQL using with the like operator. Now we need to create a restriction into WHERE clause as shown in the following program.

```
var customers = session.QueryOver<Customer>()
                .Where(Restrictions.On<Customer>(c =>
c.FirstName).IsLike("A%"));
foreach (var customer in customers.List())
{
    Console.WriteLine(customer);
}
```

Let's run your application again and you will see that all the customers are retrieved with first name starts with A.

```
Alejandrin Will (4ea3aef6-6bce-11e1-b0b4-6cf049ee52be)
        Points: 24
        HasGoldStatus: False
        MemberSince: 10/1/2011 12:00:00 AM (Utc)
```

```
        CreditRating: VeryVeryGood
        AverageRating: 0
        Orders:
                Order Id: 4ea3aef6-6bce-11e1-b0b5-6cf049ee52be

Austyn Nolan (4ea871b6-6bce-11e1-b110-6cf049ee52be)
        Points: 67
        HasGoldStatus: True
        MemberSince: 12/29/2007 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea871b6-6bce-11e1-b111-6cf049ee52be

Antonia Murphy (4ea871b6-6bce-11e1-b121-6cf049ee52be)
        Points: 72
        HasGoldStatus: True
        MemberSince: 6/15/2009 12:00:00 AM (Utc)
        CreditRating: Terrible
        AverageRating: 0
        Orders:
                Order Id: 4ea871b6-6bce-11e1-b122-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b123-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b124-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b125-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b126-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b127-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b128-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b129-6cf049ee52be
                Order Id: 4ea871b6-6bce-11e1-b12a-6cf049ee52be
```

It works the same way as it did before, except using this new **QueryOver** syntax. Many developers find that LINQ syntax is more approachable and often does the right things.

If LINQ can't handle it, then you will start looking at HQL or Criteria to see if that's going to be more suitable.

It just gives you a different syntax, so Criteria, both the create criteria and the QueryOver provide you just yet another querying mechanism that allows you to pull data out of the database using NHibernate.

26. NHibernate – Native SQL

In this chapter, we will be covering how to use the native SQL queries in NHibernate. If you have been using handwritten SQL for a number of years, you may be concerned that ORM will take away some of the expressiveness and flexibility you are used to.

- NHibernate's powerful query facilities allow you to do almost anything you would in SQL, and in some cases more.
- For the rare cases where you can't make NHibernate's own query facilities do exactly what you want.
- NHibernate allows you to retrieve objects using your database's native SQL dialect.

Let's have a look into a simple example of the Native SQL queries in NHibernate.

```
using System;
using System.Data;
using System.Linq;
using System.Reflection;
using HibernatingRhinos.Profiler.Appender.NHibernate;
using NHibernate.Cfg;
using NHibernate.Criterion;
using NHibernate.Dialect;
using NHibernate.Driver;
using NHibernate.Linq;
using NHibernate;

namespace NHibernateDemo {
    internal class Program {
        private static void Main() {
            var cfg = ConfigureNHibernate();
            var sessionFactory = cfg.BuildSessionFactory();

            using(var session = sessionFactory.OpenSession())
            using(var tx = session.BeginTransaction()) {

                IQuery sqlQuery = session.CreateSQLQuery("SELECT * FROM
CUSTOMER").AddEntity(typeof(Customer));
```

```
          var customers = sqlQuery.List<Customer>();
                foreach (var customer in customers)
                {
                    Console.WriteLine(customer);
                }
                tx.Commit();
            }

            Console.WriteLine("Press <ENTER> to exit...");
            Console.ReadLine();
        }

        private static Configuration ConfigureNHibernate() {
            NHibernateProfiler.Initialize();
            var cfg = new Configuration();
            cfg.DataBaseIntegration(x => {
                                        x.ConnectionStringName = "default";
                                        x.Driver<SqlClientDriver>();
                                        x.Dialect<MsSql2008Dialect>();
                                        x.IsolationLevel =
IsolationLevel.RepeatableRead;
                                        x.Timeout = 10;
                                        x.BatchSize = 10;
                                    });
            cfg.SessionFactory().GenerateStatistics();
            cfg.AddAssembly(Assembly.GetExecutingAssembly());
            return cfg;
        }
    }
}
```

The above example uses **CreateSQLQuery()** to get back a list of objects, and you will also notice that the root entity type you want the query to return is specified as Customer.

Let's run your application and you will see that all the customers are retrieved from the database.

```
Emerson Prosacco (4ec2a0e0-6bce-11e1-b2cf-6cf049ee52be)
        Points: 17
        HasGoldStatus: False
        MemberSince: 6/22/2007 12:00:00 AM (Utc)
        CreditRating: Excellent
        AverageRating: 0
        Orders:
                Order Id: 4ec2a0e0-6bce-11e1-b2d0-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2d1-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2d2-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2d3-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2d4-6cf049ee52be

Kaci Friesen (4ec2a0e0-6bce-11e1-b2d5-6cf049ee52be)
        Points: 30
        HasGoldStatus: True
        MemberSince: 5/25/2007 12:00:00 AM (Utc)
        CreditRating: VeryVeryGood
        AverageRating: 0
        Orders:
                Order Id: 4ec2a0e0-6bce-11e1-b2d6-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2d7-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2d8-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2d9-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2da-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2db-6cf049ee52be

Eveline Waters (4ec2a0e0-6bce-11e1-b2dc-6cf049ee52be)
        Points: 58
        HasGoldStatus: False
        MemberSince: 10/29/2009 12:00:00 AM (Utc)
        CreditRating: Good
        AverageRating: 0
        Orders:
```

```
                Order Id: 4ec2a0e0-6bce-11e1-b2dd-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2de-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2df-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e0-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e1-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e2-6cf049ee52be

Molly Kuhn (4ec2a0e0-6bce-11e1-b2e3-6cf049ee52be)
        Points: 73
        HasGoldStatus: False
        MemberSince: 12/16/2007 12:00:00 AM (Utc)
        CreditRating: VeryGood
        AverageRating: 0
        Orders:
                Order Id: 4ec2a0e0-6bce-11e1-b2e4-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e5-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e6-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e7-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e8-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2e9-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2ea-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2eb-6cf049ee52be
                Order Id: 4ec2a0e0-6bce-11e1-b2ec-6cf049ee52be
```

Here is another way of writing native SQL query as shown below.

```
IList<Customer> customers = session.CreateSQLQuery("SELECT * FROM CUSTOMER")
                                   .AddScalar("Id", NHibernateUtil.Guid)
                                   .AddScalar("FirstName",
NHibernateUtil.String)
                                   .AddScalar("LastName", NHibernateUtil.String)
                                   .List<Customer>();
```

- As you can see that the above query specified the SQL query string and the columns and types to return.
- This will return an IList of Object arrays with scalar values for each column in the Customer table.
- Only these three columns will be returned, even though the query is using * and could return more than the three listed columns.

Let's have a look into another simple example.

```
IList<Customer> customers = session.CreateSQLQuery("SELECT * FROM CUSTOMER
WHERE FirstName = 'Laverne'")
                        .AddEntity(typeof(Customer))
                        .List<Customer>();
 foreach (var customer in customers)
 {
     Console.WriteLine(customer);
 }
```

Let's run your application again and you will see the following output.

```
Laverne Hegmann (4e97c816-6bce-11e1-b095-6cf049ee52be)
        Points: 74
        HasGoldStatus: True
        MemberSince: 4/4/2009 12:00:00 AM (Utc)
        CreditRating: Neutral
        AverageRating: 0
        Orders:
                Order Id: 4ea14d96-6bce-11e1-b095-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b096-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b097-6cf049ee52be
                Order Id: 4ea14d96-6bce-11e1-b098-6cf049ee52be

Press <ENTER> to exit...
```

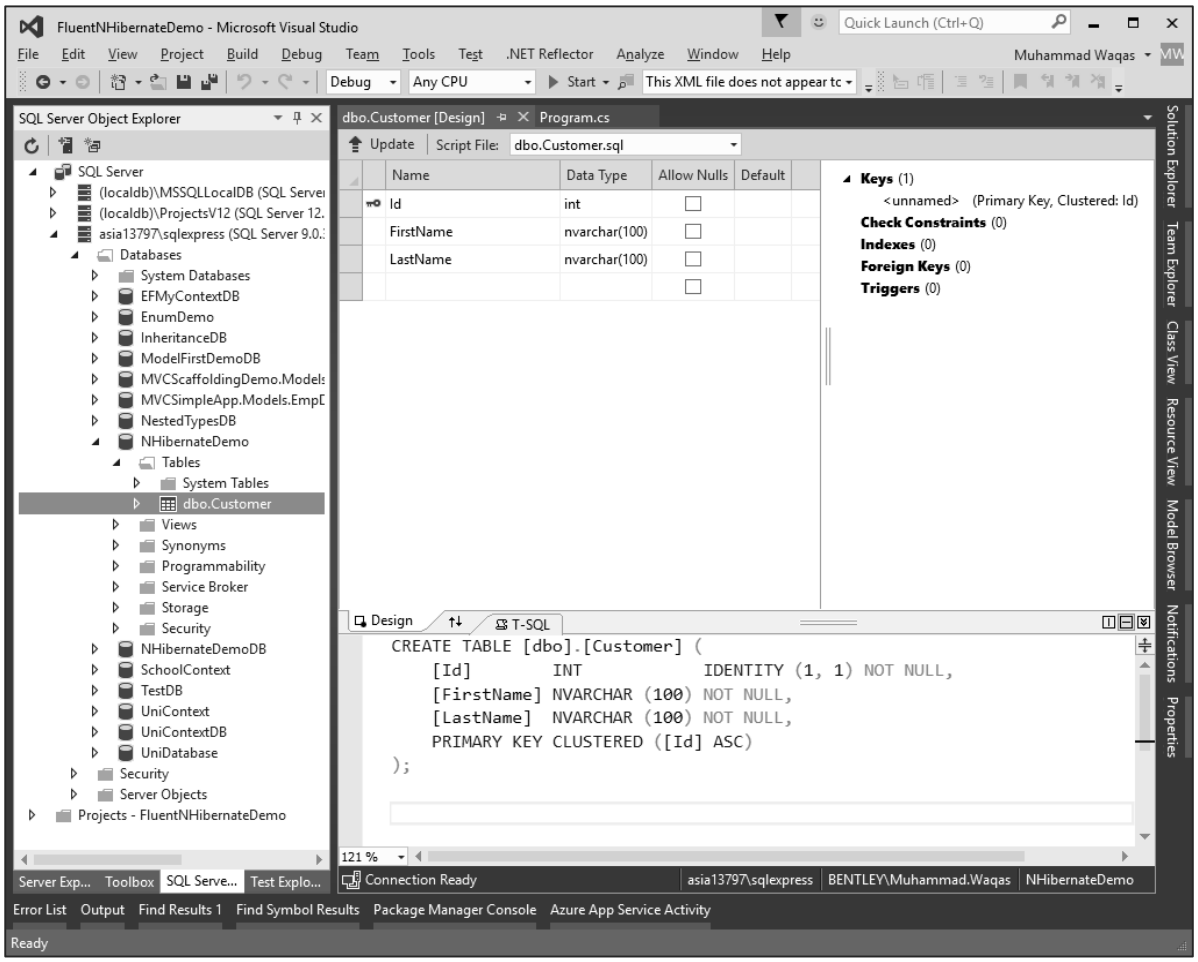
Similarly, you can specify any type of SQL query to retrieve data from the database.

27. NHibernate – Fluent Hibernate

In this chapter, we will be covering fluent NHibernate. Fluent NHibernate is another way of mapping or you can say it is an alternative to NHibernate's standard XML mapping files. Instead of writing XML (**.hbm.xml files**) documents. With the help of Fluent NHibernate, you can write mappings in strongly typed C# code.

- In Fluent NHibernate mappings are compiled along with the rest of your application.
- You can easily change your mappings just like your application code and the compiler will fail on any typos.
- It has a conventional configuration system, where you can specify patterns for overriding naming conventions and many other things.
- You can also set how things should be named once, then Fluent NHibernate does the rest.

Let's have a look into a simple example by creating a new console project. In this chapter, we will use a simple database in which we have a simple Customer table as shown in the following image.

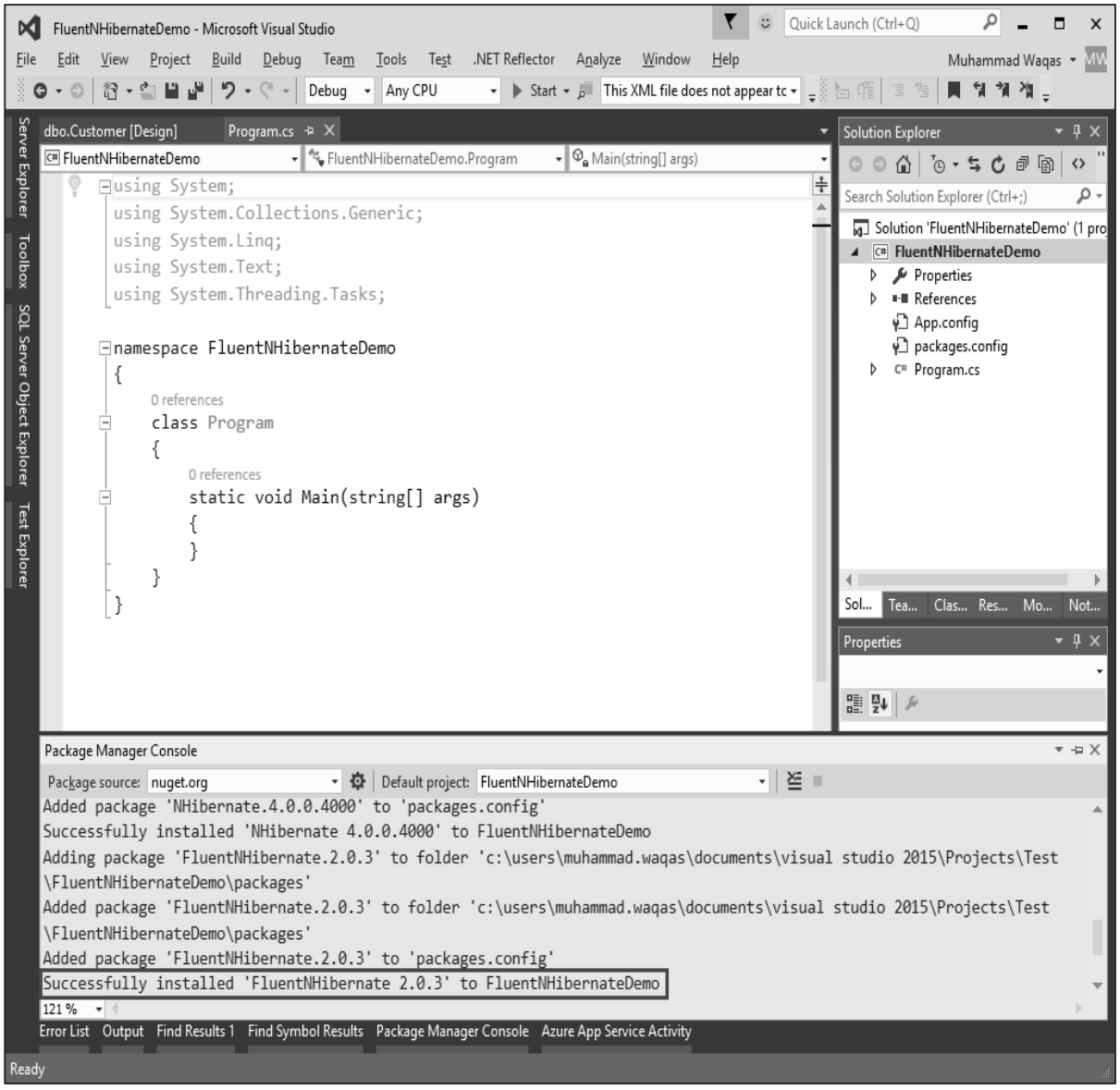


Install Fluent NHibernate

The first step is to start Fluent NHibernate is to install Fluent NHibernate package. So open the **NuGet Package Manager Console** and enter the following command.

```
PM> install-package FluentNHibernate
```

Once it is installed successfully, you will see the following message.



Let's add a simple model class of Customer and the following program shows the Customer class implementation.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace FluentNHibernateDemo
{
    class Customer
    {
        public virtual int Id { get; set; }
        public virtual string FirstName { get; set; }
        public virtual string LastName { get; set; }
    }
}
```

Now we need to create Mappings using fluent NHibernate, so add one more class **CustomerMap** in your project. Here is the implementation of the CustomerMap class.

```
using FluentNHibernate.Mapping;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FluentNHibernateDemo
{
    class CustomerMap : ClassMap<Customer>
    {
        public CustomerMap()
        {
            Id(x => x.Id);
            Map(x => x.FirstName);
            Map(x => x.LastName);
            Table("Customer");
        }
    }
}
```

Let's add another class **NHibernareHelper** in which we will set different configuration settings.

```
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;
using NHibernate;
using NHibernate.Tool.hbm2ddl;

namespace FluentNHibernateDemo
{
    public class NHibernateHelper
    {
        private static ISessionFactory _sessionFactory;

        private static ISessionFactory SessionFactory
        {
            get
            {
                if (_sessionFactory == null)

                    InitializeSessionFactory();
                return _sessionFactory;
            }
        }

        private static void InitializeSessionFactory()
        {
            _sessionFactory = Fluently.Configure()
                .Database(MsSqlConfiguration.MsSql2008
                                .ConnectionString(
                                    @"Data Source=asia13797\sqlexpress;Initial
Catalog=NHibernateDemo;Integrated Security=True;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWri
te;MultiSubnetFailover=False")
                                .ShowSql()
                )
                .Mappings(m =>
                          m.FluentMappings
```

```
                        .AddFromAssemblyOf<Program>())
                .ExposeConfiguration(cfg => new SchemaExport(cfg)
                                                .Create(true, true))
                .BuildSessionFactory();
        }

        public static ISession OpenSession()
        {
            return SessionFactory.OpenSession();
        }
    }
}
```

Now let's move to the **Program.cs** file in which we will start a session and then create a new customer and save that customer to the database as shown below.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FluentNHibernateDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var session = NHibernateHelper.OpenSession())
            {
                using (var transaction = session.BeginTransaction())
                {
                    var customer = new Customer { FirstName = "Allan",
                                                  LastName = "Bomer" };
                    session.Save(customer);
                    transaction.Commit();
```

```
            Console.WriteLine("Customer Created: " + customer.FirstName +
                                                        "\t" +
customer.LastName);
                }
                Console.ReadKey();
            }
        }
    }
}
```

Let's run your application and you will see the following output.

```
if exists (select * from dbo.sysobjects where id = object_id(N'Customer') and
OBJECTPROPERTY(id, N'IsUserTable') = 1) drop table Customer


    create table Customer (
        Id INT IDENTITY NOT NULL,
       FirstName NVARCHAR(255) null,
       LastName NVARCHAR(255) null,
       primary key (Id)
    )
NHibernate: INSERT INTO Customer (FirstName, LastName) VALUES (@p0, @p1);
select SCOPE_IDENTITY();@p0 = 'Allan' [Type: String (4000)], @p1 = 'Bomer'
[Type: String (4000)]
Customer Created: Allan Bomer
```

As you can see the new customer is created. To see the customer record, let's go to the database and see the View Data and you will see that 1 Customer is added.