# 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.

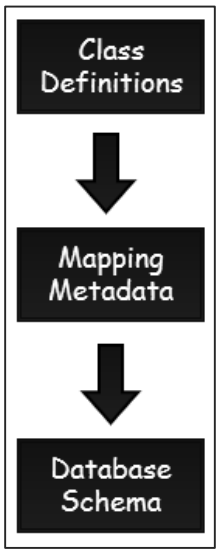
- In the object world, we are working with pointers that are unidirectional, whereas in with the relational world, we have foreign keys which are inherently bidirectional.

- 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**.

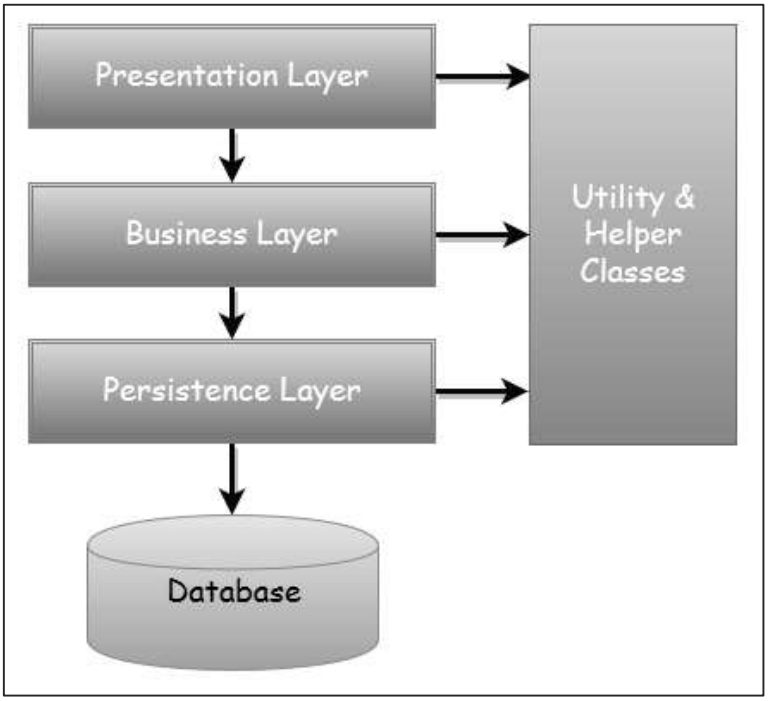
# 2. NHibernate – Architecture

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

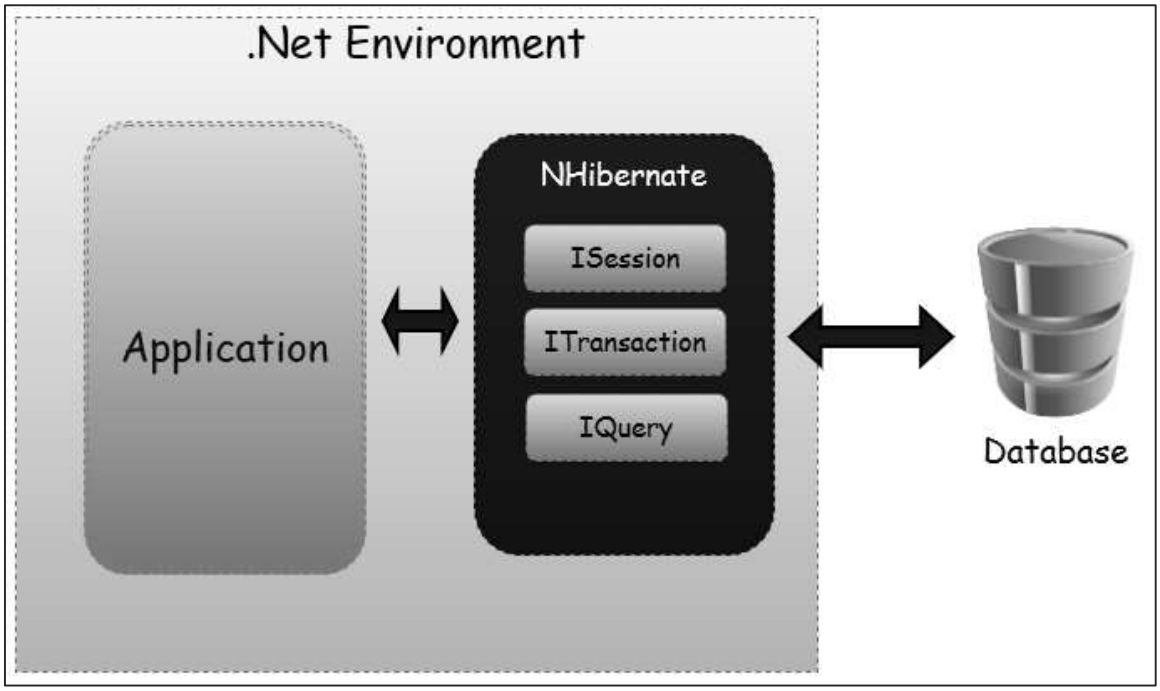
- The database exists outside the .NET application.
- It's the actual, persistent representation of the system state.
- If a SQL database is used, the database includes the relational schema and possibly stored procedures.

## 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?

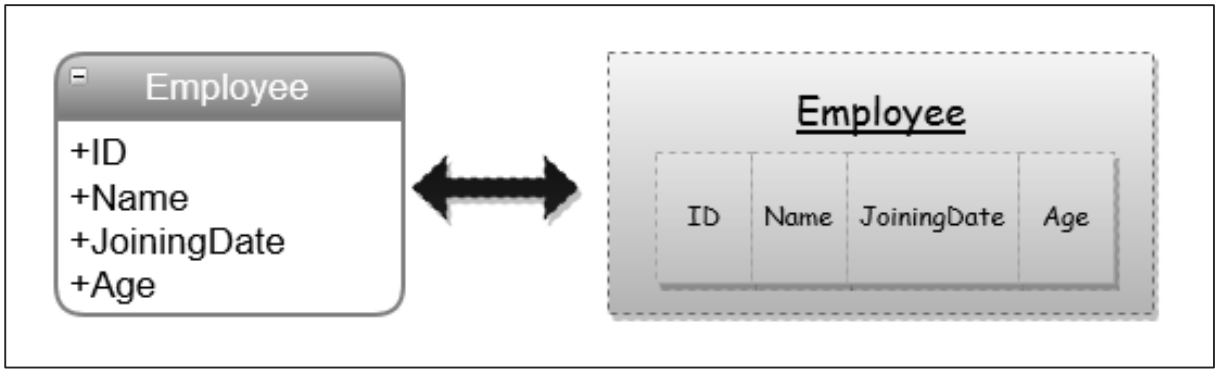
Object-Relational Mapping (ORM) is a **programming technique** for converting data between incompatible type systems in object-oriented programming languages. In other words, it is the concept of mapping an application's business objects to relational database tables, so that the data can be easily accessed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

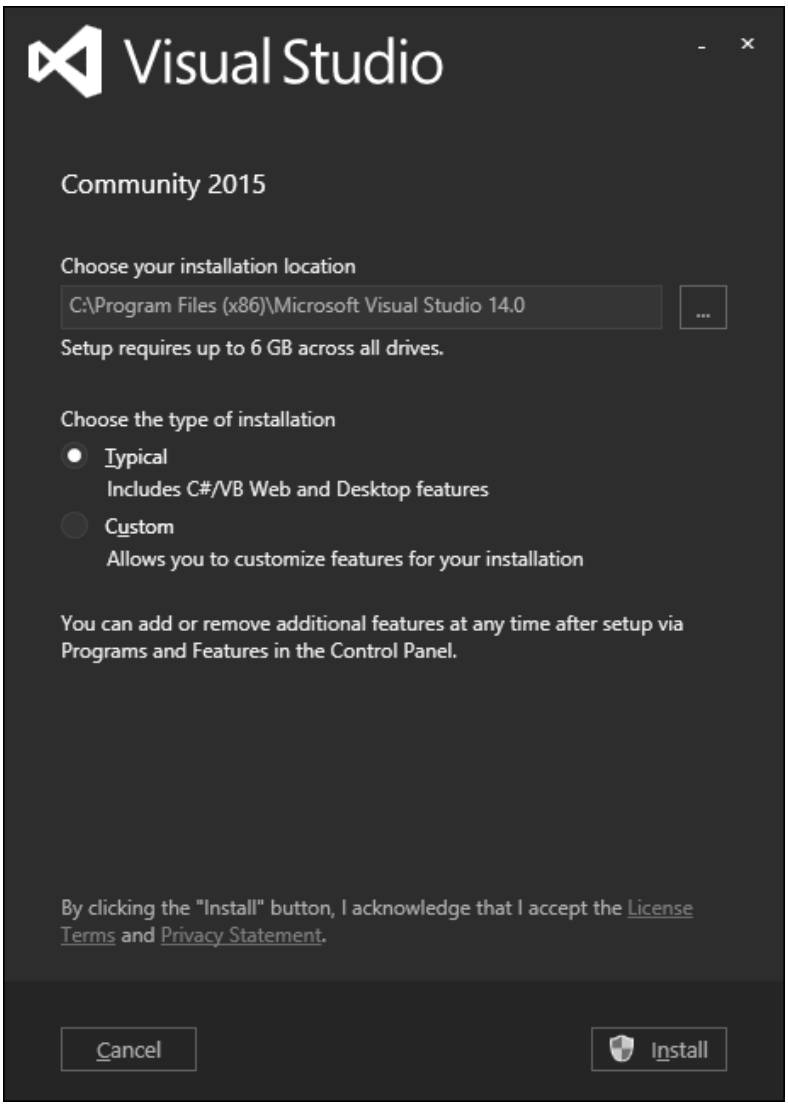
# 4. NHibernate – Environment Setup

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.