



UNIVERSIDAD
POLITECNICA
DE VALENCIA

Laboratory

Lab Session 4

Persistence layer with Entity Framework

Software Engineering

Computer Science School

DSIC – UPV

Year 2024-2025

1. Goal

IMPORTANT: Read the whole document before starting to code.

The goal in this session is the development of a data access layer that will offer the business logic layer all services required to retrieve, modify, remove or add objects to the persistence layer, but ensuring that the business logic layer does not know which persistence mechanism is being used. To do this, we are going to use Entity Framework.

Entity Framework is a persistence framework by Microsoft for .NET, which offers object-relational mapping and is designed around the Repository + Unit of Work patterns (introduced in Chapter 6 Persistence). EF will allow our application to work directly with Entities that are objects of the business logic, and not specific objects for data transfer (DTOs). EF will automatically take care of transparently managing the persistence of these objects in a relational database using SQL sentences.

Before this session, the student should revise chapter 6 “Persistence Design” and the related seminars “Entity Framework” and “DAL”.

Additionally, you should also check the “VehicleRental” study case, which can help you implement the study case you are working on. You can download its complete implementation from Poliformat.

At the end of the session, the student should be able to:

- Describe the architecture of the persistence layer.
- Configure the project in order to be able to implement the persistence layer with EF.
- Perform operations with objects on the database (CRUD operations).
- Run the unit tests to check the correct working of the persistence layer.

Next, the architecture of the persistence layer will be described, including the goal of each class, and then, the tasks that the team should perform to implement the persistence layer of the project will be listed.

2. Design of the Data Access Layer (DAL)

The architecture of the data access layer will be the one shown in Figure 1. The elements of the architecture are described below.

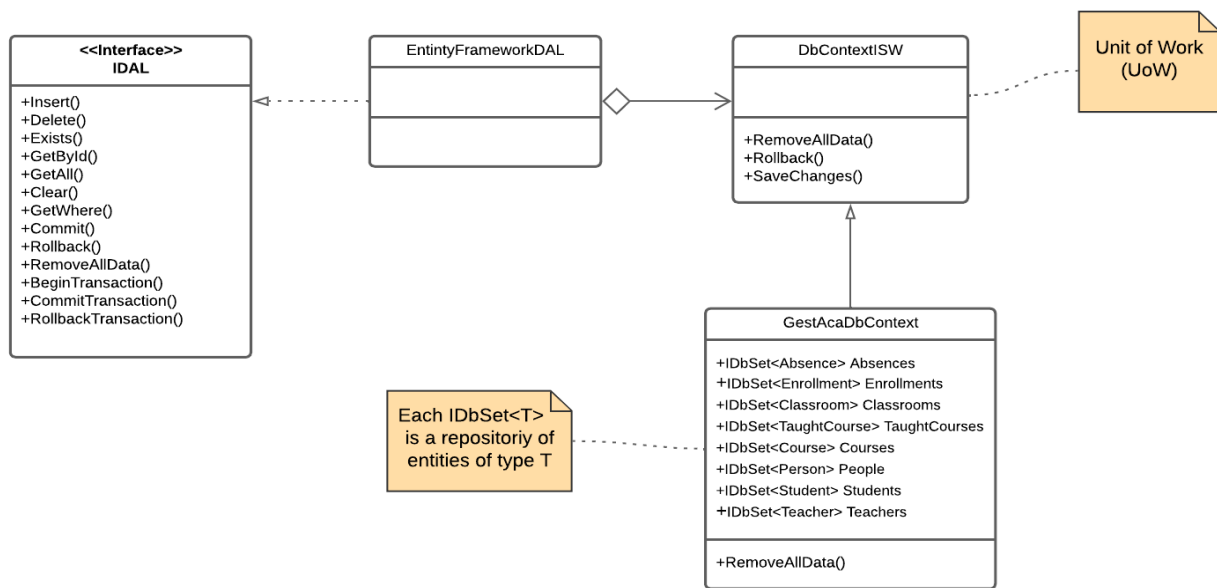


Figure 1. DAL Design class diagram

IDAL: It provides transparent access to the persistence mechanism (persistence agnostic) by bridging the gap between the business logic layer and the persistence layer. This interface, named **IDAL** (DAL = Data Access Layer), is responsible of abstracting the data access services from the underlying persistence mechanism (i.e., SQL, XML, etc.)

The IDAL interface is an adapter combining the functionality of the repositories (Insert, Delete, GetXXX, Exists) and the functionality of the UoW (Commit, Rollback, BeginTransaction, etc). An additional method (Clear) to remove the database's data is included. The full list of IDAL.cs methods is this one:

```

public interface IDAL
{
    void Insert<T>(T entity) where T : class;
    void Delete<T>(T entity) where T : class;
    IEnumerable<T> GetAll<T>() where T : class;
    T GetById<T>(IComparable id) where T : class;
    bool Exists<T>(IComparable id) where T : class;
    void Clear<T>() where T : class;
    IEnumerable<T> GetWhere<T>(Expression<Func<T, bool>> predicate)
        where T : class;
    void Commit();
    void Rollback();
    void RemoveAllData();
    void BeginTransaction();
    void CommitTransaction();
    void RollbackTransaction();
}
  
```

Using Generics

Note that generic type parameters `T` have been used, significantly reducing the amount of code to be implemented. That is, instead of implementing the following individual operations:

```
void InsertCourse(Course c);
void InsertStudent(Student s);
void InsertAbsence(Absence a);
...
```

We can define a unique method by using generic types:

```
void Insert<T>(T entity) where T : class;
```

This method will be instantiated in execution time according to the corresponding `T` type (which will be a class).

EntityFrameworkDAL: It is a specific implementation of IDAL for EF. You can find the code for this class below:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Linq.Expressions;

namespace GestAca.Persistence
{
    public class EntityFrameworkDAL : IDAL
    {
        private readonly DbContext dbContext;

        public EntityFrameworkDAL(DbContext dbContext)
        {
            this.dbContext = dbContext;
        }

        public void Insert<T>(T entity) where T : class
        {
            dbContext.Set<T>().Add(entity);
        }

        public void Delete<T>(T entity) where T : class
        {
            dbContext.Set<T>().Remove(entity);
        }

        public IEnumerable<T> GetAll<T>() where T : class
        {
            return dbContext.Set<T>();
        }

        public T GetById<T>(IComparable id) where T : class
        {
            return dbContext.Set<T>().Find(id);
        }

        public bool Exists<T>(IComparable id) where T : class
```

```

    {
        return dbContext.Set<T>().Find(id) != null;
    }

    public void Clear<T>() where T : class
    {
        dbContext.Set<T>().RemoveRange(dbContext.Set<T>());
    }

    public IEnumerable<T> GetWhere<T>(Expression<Func<T, bool>> predicate)
        where T : class
    {
        return dbContext.Set<T>().Where(predicate).AsEnumerable();
    }

    public void Commit()
    {
        dbContext.SaveChanges();
    }
    ...
    ...
    ...
}
}

```

As you can see in this code, the DAL class contains a reference to a DbContext object, as illustrated in Figure 1¹ and the implementation of each of the services offered by the IDAL interface.

It is important to remember that after completing a transaction (one or more operations that imply a change in the data), the method Commit should be called to persist the changes. Although there is not a specific method to update an object, **if objects have been internally modified, it is also needed to execute the method Commit.**

DbContextISW: contains the implementation of the operations on the database such as RemoveAllData(), Rollback(), but also the exception handling when changes are passed to the DB in SaveChanges(). The code of this class can be seen in the file DbContextISW.cs

GestAcaDbContext: defines the mapping of domain objects to database tables, following the Repository + UoW pattern. Therefore, this class's code depends on the domain, and on this project in particular. In this class, you can observe the following features:

- It inherits from DbContext
- It has several properties of type IDbSet<Type> where Type is one of the classes in the domain model that have to be persisted (these are called entities). Each IDbSet is a repository with the typical methods to access, add or remove objects. Specifically, for our project:

```

public IDbSet<Absence> Absences { get; set; }
public IDbSet<Enrollment> Enrollments { get; set; }
public IDbSet<Classroom> Classrooms { get; set; }
public IDbSet<TaughtCourse> TaughtCourses { get; set; }
public IDbSet<Course> Courses { get; set; }
public IDbSet<Person> People { get; set; }
public IDbSet<Student> Students { get; set; }
public IDbSet<Teacher> Teachers { get; set; }

```

¹ More specifically, what is shown is that EntityFrameworkDAL contains DbContextISW, but it must be remembered that DbContextISW extends DbContext, even though it has not been shown in the diagram.

- It has a constructor which forwards to the base class constructor the name of the database connection string ("GestAcaDbConnection"). This connection string is defined in the configuration file App.config. We can also include some configuration directives in the constructor, like the caching policy (see seminar on Entity Framework)

```
public GestAcaDbContext() : base("GestAcaDbConnection") { ... }
```

Task 1: Configure the solution (team leader only).

It is needed to add a certain package to your class library project to work with Entity Framework. After connecting and downloading the latest version of the solution in Visual Studio, the team leader should select the option Herramientas > Administrador de paquetes NuGet > Administrar paquetes NuGet para la solución, and in the text field called "Examinar" type Entity Framework. This package will show first on the list (see Figure 2) and you will have to install it into the class library project created in sessions 2 and 3 (**GestAcaLib**, which contains the folders BusinessLogic and Persistence).

Check in the Explorador de Soluciones that Entity Framework is included in the references of your project. Once this is done, the team leader **must commit and synchronize** the changes.

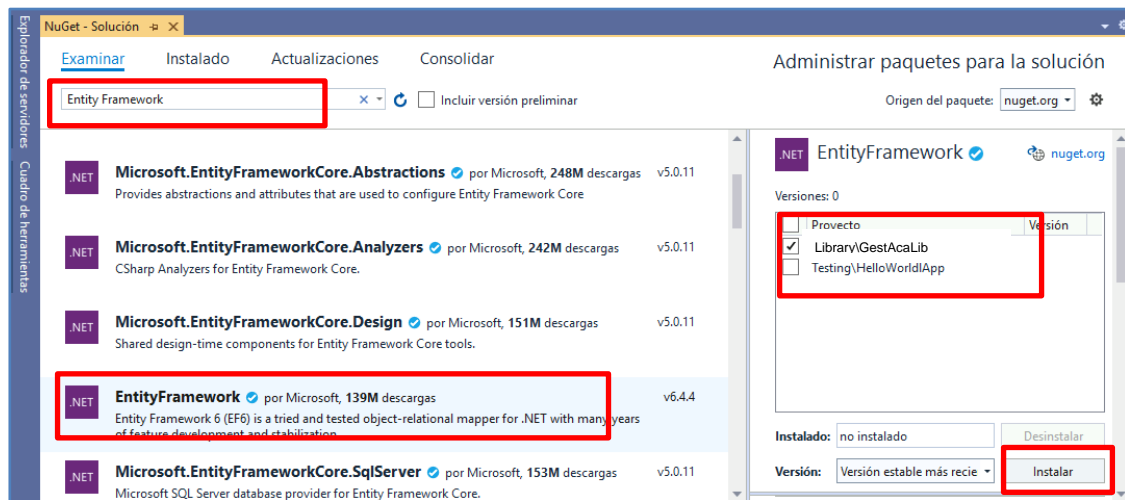


Figure 2. adding the EntityFramework package to the Class Library project

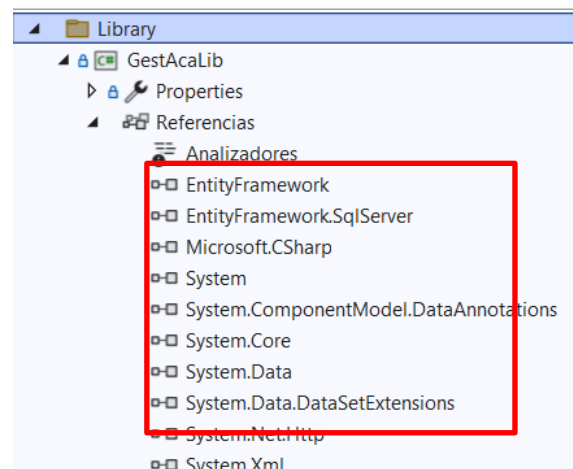


Figure 3. Checking if EF is installed

Task 2: Include all files needed to implement the persistence layer

In order to implement the persistence layer for the case study's project, follow the next steps:

1. Download the file EntityFrameworkImp from PoliformaT, which contains the implementation of the classes DbContextISW, EntityFrameworkDAL, IDAL and GestAcaDbContext.
2. Add a new EntityFrameworkImp folder inside the Persistence folder (right-click on the folder's name > Agregar (Add) > Nueva carpeta (New folder). After that, right click on the new folder and select "Abrir carpeta en el explorador de archivos" (Open folder in File explorer).
3. Unzip the file from the first step and paste the files inside the folder you just opened in step 2.
4. In Visual Studio again, right-click on EntityFrameworkImp and select Agregar > Elemento existente (Existing element) and add the files you pasted in step 3.

Check that the final folder structure you have is the same as in Figure 4. Please, be aware that all added classes are part of the same namespace (GestAca.Persistence), so it is necessary to include the `using GestAca.Entities;` directive in the class files in which the model classes are used.

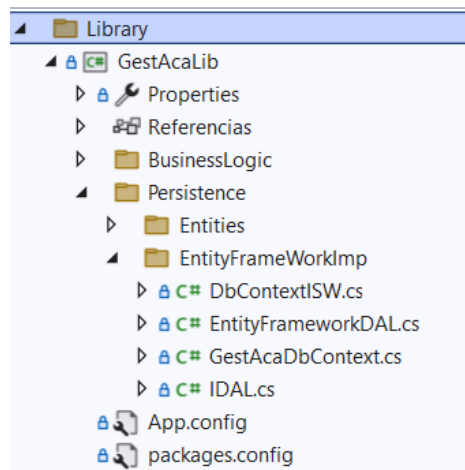


Figure 4: persistence layer's structure in VS project.

Task 3. Review the logic layer code and add any EF annotations you feel are necessary.

Sometimes it is necessary to modify the code of the business logic classes (in the folder Persistence/entities) using the necessary **Data Annotations** to provide some extra information to the Entity Framework. In fact, some guidelines regarding the use of annotations are given in the **Entity Framework** theory seminar 6.1. In particular, pay attention to the annotations:

[Key]
[Required]

You will have to add the annotations before the definition of the corresponding property.

HINT: if a key attribute is of type int and you don't want to be automatically incremented by the DBMS you must add the following data annotation:

```
[DatabaseGeneratedAttribute(DatabaseGeneratedOption.None), Key()]
```

Task 4: check the constructor implementation.

In the previous session, you defined the necessary constructors to create the objects of the business logic. Following the instructions, you defined a default constructor and a constructor with arguments.

The default constructor is used by the Entity Framework when it needs to create an object in memory after retrieving the information from the tables in the database. Therefore, you do not have to initialize the attributes of the object in this constructor (EF already does), but you do have to initialize the attributes of type collection.

The other constructor with arguments is the one that should be used in the code when it is necessary to create an object. As you already know, EF gives values to the ID attributes of numerical type automatically in the moment in which the object is persisted for the first time. Therefore, it is not necessary to give value to these attributes in the object's constructor since EF will change the value when it persists the object for the first time.

The constructors were already designed in the previous session with this aspect in mind. Only the string type IDs and the non auto-incremental int IDs should be initialised in the constructor. All auto-incremental numeric IDs are initialized by EF and do not have to be passed as parameters in the constructors.



Once you have finished the implementation of the persistence layer, commit and synchronize your solution with a comment like "Finished the persistence layer (beta version)".

Task 5. Run the unit tests for the persistence layer.

As you already know, code testing can be better systematized through a unit test engine, like MSTest². In the same way you did to test the object design implementation in the previous bulletin, you will have to run the unit test implemented by the teachers. Download the test project GestAcaPersistenceTests.zip from PoliformaT, add it to your project and run it as it has already been explained.

Before you move on to the next task, please ensure that all tests have run without failures.



Once your project has passed all tests, commit and synchronize your solution with a comment like "Persistence layer completed".

Task 6. Persisting some objects in the database

Next, let us create some business logic objects and persist them in the database, leaving it in a consistent state afterwards (that is, all restrictions expressed by the relationships and the cardinalities must be fulfilled). Specifically, you are asked to create the necessary objects so that the following statement holds:

There will be two courses (aCourse1, aCourse2) and two teachers (aTeacher1, aTeacher2). Two objects TaughtCourse (aTaughtCourse1, aTaughtCourse2) for aCourse1 and aCourse2 respectively. Create 10 students enrolled in aTaughtCourse1. Create two classrooms (c1 and c2) for aTaughtCourse1, aTaughtCourse2 respectively. Create two absences (in different dates) a1 and a2 for one enrolment.

² You can get more information about unit testing in VS at:

<https://docs.microsoft.com/es-es/visualstudio/test/getting-started-with-unit-testing?view=vs-2019>

To accomplish this, we may create a console application inside the Testing folder, in a similar way we did in the first lab session. In that Project, we will create some business objects and persist them using the services offered by the DAL. Follow these steps:

1. Create a folder named DBTest inside the solution folder Testing
2. Inside, add a console application project of the type .NET Framework. Name it DBTest (you can follow the instructions in bulletin 1)
3. Add a reference to your class library (GestAcaLib): expand the project in the solution view and open the context menu on Referencias > Agregar referencia ... Then, inside the projects section, select the class library project.
4. Include the Entity Framework using the NuGet packages manager, in a similar way as it is explained above.
5. Add a data base connection string to the configuration file App.config. As we use SQLServer as database engine and "GestAcaDB" as the name of the database, then the connection string will result as:

```
<connectionStrings>

    <clear />
    <add name="GestAcaDbConnection"
connectionString="Server=(localdb)\mssqllocaldb;Database=GestAcaDB;Trusted_Connection
=True;MultipleActiveResultSets=true" providerName="System.Data.SqlClient" />
</connectionStrings>
```

In the test code (implemented in Program.cs) you will have to include the namespaces GestAca.Persistence and GestAca.Entities in order to be able to use the EntityFrameworkImp classes and the business logic classes. Then, you will have to create an instance of EntityFrameworkDAL by passing it an instance of GestAcaDbContext. When creating the GestAcaDbContext instance for the first time the database will be generated. Then, you will have to create some business logic objects and persist them using the services offered by the DAL.

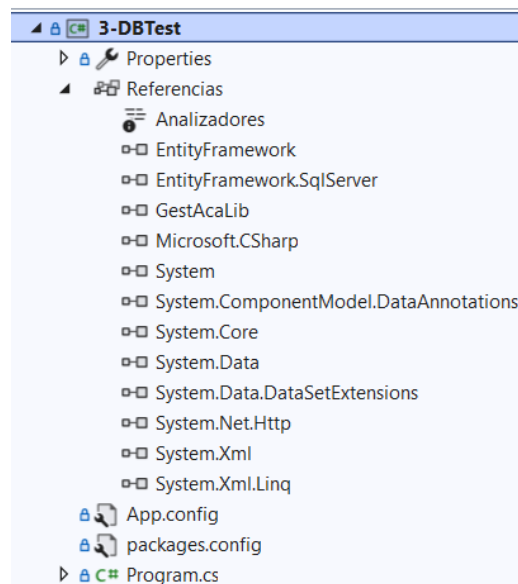


Figure 5. Console Test project structure

To make your work easier and to ensure that the database has been correctly created, we provide you with some of the necessary code. The resulting structure of the project will be as shown in Figure 5:

In the file Program.cs that is available in poliformat, you will find a method named createSampleDB (IDAL dal) that should be completed with the code required to create business objects and persist them in the database. If you set this project as the Initial project (context menu and choose “Establecer como Proyecto de inicio”), you could execute it and check whether the database has been created properly.

```
private void CreateSampleDB(IDAL dal)
{
    dal.RemoveAllData();

    Console.WriteLine("CREANDO LOS DATOS Y ALMACENANDOLOS EN LA BD");
    Console.WriteLine("=====");

    Console.WriteLine("\n// CREACIÓN DE CURSOS");
    //public Course(string descr, string name)
    Course aCourse1 = new Course("Curso Introductorio Ingenieria Software", "Software Engineering");
    dal.Insert<Course>(aCourse1);
    dal.Commit();
    Course aCourse2 = new Course("Curso Introductorio de Estructuras de datos", "Data Structures");
    dal.Insert<Course>(aCourse2);
    dal.Commit();

    // Populate here the rest of the database
    // Add missing code here
}
```

As you can see in the code after deleting the content of the database (dal.RemoveAllData()), two instances of Course are created. For the data to persist in the database we must add the object to the Courses repository, with the instructions dal.Insert<Course>(aCourse1) dal.Insert<Course>(aCourse2), and then perform the commit in the database.

It is important to remember that after completing a transaction (one or more operations that imply a change in the data), the method Commit should be called to persist the changes. Although there is not a specific method to update an object, **if objects have been internally modified, it is also needed to execute the method Commit.**

In order to check that the DB has been correctly updated you could run some code to check that, or you can use the DB inspection tools included in VS, as it is explained and shown next.

Task 7. Use the DB inspector tool to check that the information has been correctly saved on each table.

From Visual Studio IDE itself, it is possible to connect to any database to see its tables and content. In order to connect to an existing local database (which is saved as a local file), you will have to follow the next steps (

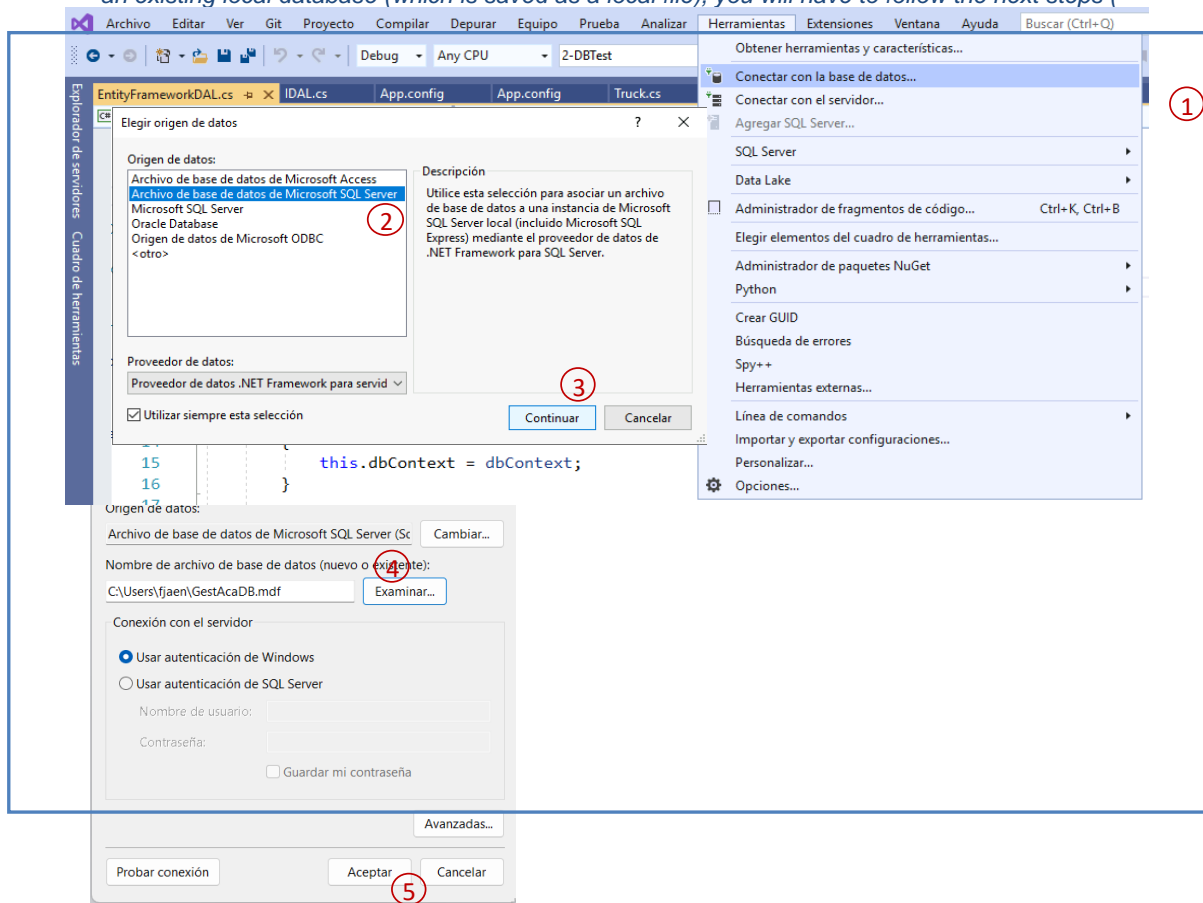


Figure 6):

- Select Herramientas > Conectar con la Base de Datos
- Select as data source: “Archivo de base de datos de Microsoft SQL Server (SqlClient)”
- Select the file with extension “.mdf” created by your application. Notice that the file should be in the path: C:\Users\userName\ GestAcaDB.mdf (it depends on the app.config file).

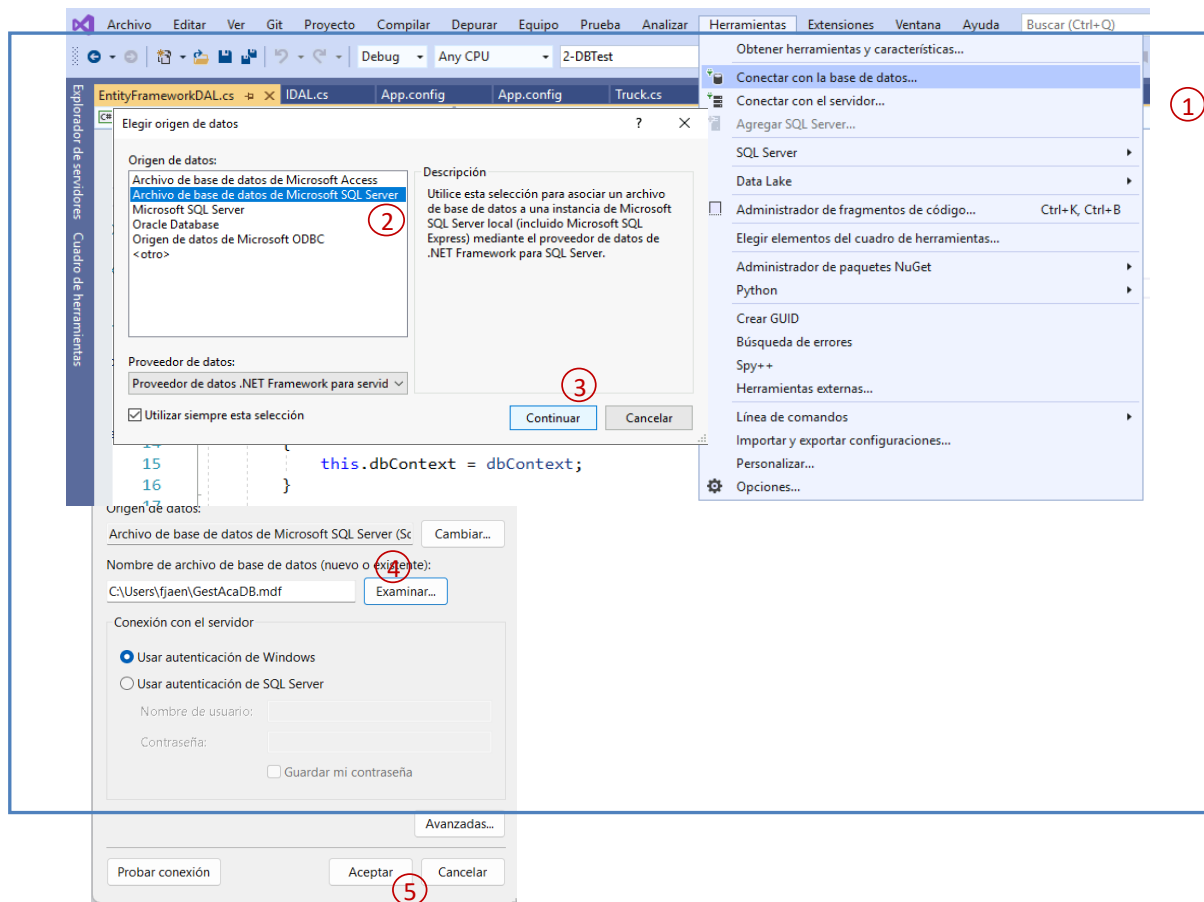


Figure 6: connecting to the DB from VS.

Once the connection has been established, the tables may be explored using the server explorer that appears to the left in Visual Studio, or from “Ver > Explorador de Servidores” menu to the top.

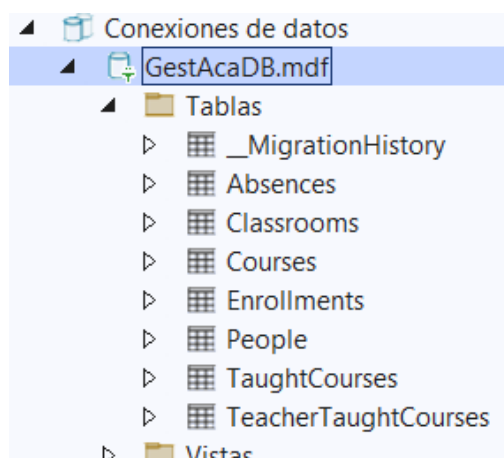
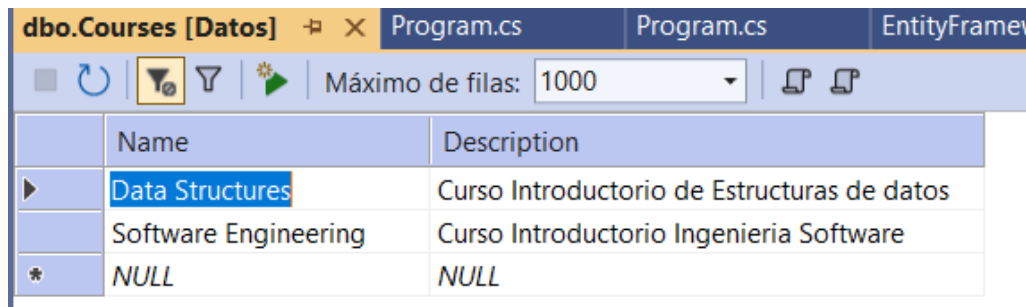


Figure 7. BD tables from the server explorer

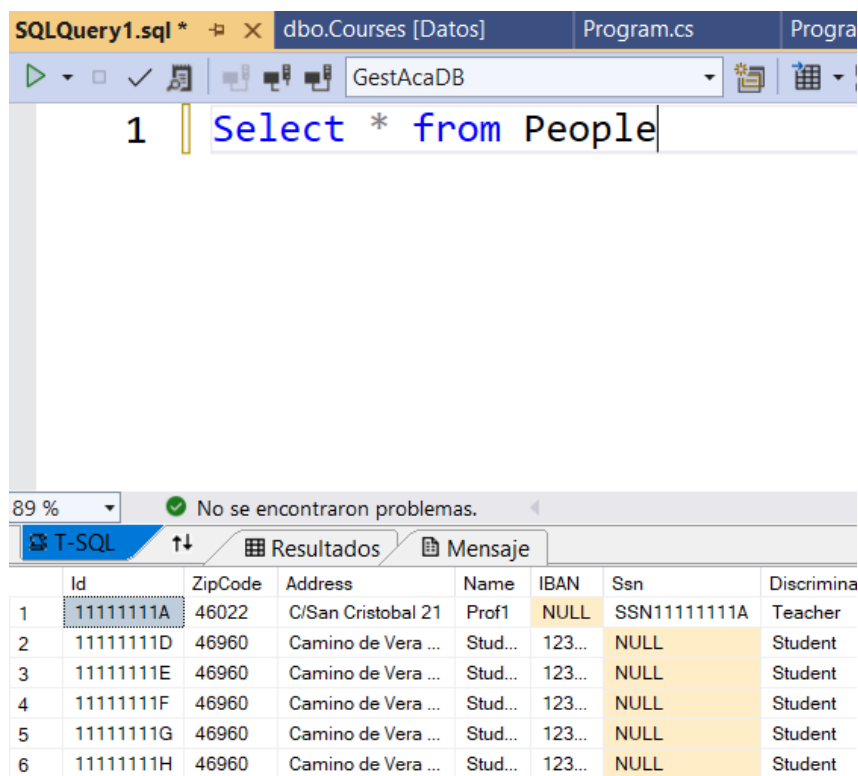
If you double-click on any table, you will see its structure. To see the data stored on a table, you should click on the table using the mouse right-button and selecting the option *Mostrar datos tabla*. In Figure 8, we can see the content of the *Courses* table after executing the console test project.



	Name	Description
▶	Data Structures	Curso Introdutorio de Estructuras de datos
	Software Engineering	Curso Introdutorio Ingenieria Software
*	NULL	NULL

Figure 8. *Courses* table

In a similar way, you can select the option *Nueva consulta* by right-clicking on the DB connection (*GestAca.mdf*), which allows you to write SQL sentences for your database. For example, you can easily visualize all the records of a table by writing and executing (▶ button) the following sentence: *Select * From TableName*. Figure 9 shows the result of running a query to the table *People*.



	Id	ZipCode	Address	Name	IBAN	Ssn	Discrimina
1	11111111A	46022	C/San Cristobal 21	Prof1	NULL	SSN11111111A	Teacher
2	11111111D	46960	Camino de Vera ...	Stud...	123...	NULL	Student
3	11111111E	46960	Camino de Vera ...	Stud...	123...	NULL	Student
4	11111111F	46960	Camino de Vera ...	Stud...	123...	NULL	Student
5	11111111G	46960	Camino de Vera ...	Stud...	123...	NULL	Student
6	11111111H	46960	Camino de Vera ...	Stud...	123...	NULL	Student

Figure 9: *example of a query to the database*



Once you have checked that the DB has been populated correctly, commit and synchronize your solution with a comment like "Creation and persistence of objects finished". Please, take into account that the DB file is NOT saved in the repo, so each team member might have different data depending on what they have created.