



UNIVERSIDAD
POLITECNICA
DE VALENCIA

Prácticas

Boletín Práctica 4

Capa de Persistencia con Entity Framework

Ingeniería del Software

ETS Ingeniería Informática

DSIC – UPV

Curso 2024-2025

1. Objetivo

El objetivo principal de esta sesión es desarrollar la capa de acceso de datos o persistencia de la aplicación. La capa de persistencia ofrecerá a la capa lógica los servicios necesarios para recuperar, modificar, borrar o añadir objetos, sin que la capa de lógica sepa qué mecanismo de persistencia particular se está usando.

Para facilitar el trabajo se va a utilizar **Entity Framework (EF)**. EF es un sistema de mapeo objeto-relacional que va a permitir a nuestra aplicación trabajar directamente con Entidades que son objetos de la lógica de negocio, y no objetos específicos para transferencia de datos. EF se encargará, automáticamente, de gestionar de forma transparente la persistencia de dichos objetos en una base de datos relacional como SQL. El acceso a datos usando Entity Framework se basa en el patrón **Repositorio + Unidad de Trabajo**, tal y como se explica en el tema 6 dedicado a la Persistencia.

Previamente a la realización de la práctica, el estudiante deberá haber repasado el “Tema 6 Diseño de Persistencia” y los seminarios asociados dedicados a “Entity Framework” y “DAL”.

Por otro lado, el caso de estudio de referencia “VehicleRental”, que puede descargar de Poliformat, es una implementación completa, utilizando la arquitectura explicada en clase, que debe servir como ejemplo al estudiante en el desarrollo del caso de estudio que nos ocupa.

Al finalizar la sesión el o la estudiante deberá ser capaz de:

- Describir la arquitectura de la capa de persistencia.
- Configurar el proyecto para poder implementar la capa de persistencia utilizando EF.
- Realizar operaciones con objetos en la base de datos (crear, leer, borrar, actualizar).
- Ejecutar los test unitarios que comprueban el correcto funcionamiento de la capa de persistencia.

A continuación, se describe en primer lugar la arquitectura de la capa de persistencia, y la finalidad de cada una de las clases que la componen, y después se enumerarán las tareas que debe realizar el estudiante para implementar la capa de persistencia del proyecto.

2. Diseño de la capa de acceso a datos

La arquitectura de la capa de acceso a datos que se va a diseñar es la que se muestra en la Figura 1. A continuación se describen los elementos de la arquitectura.

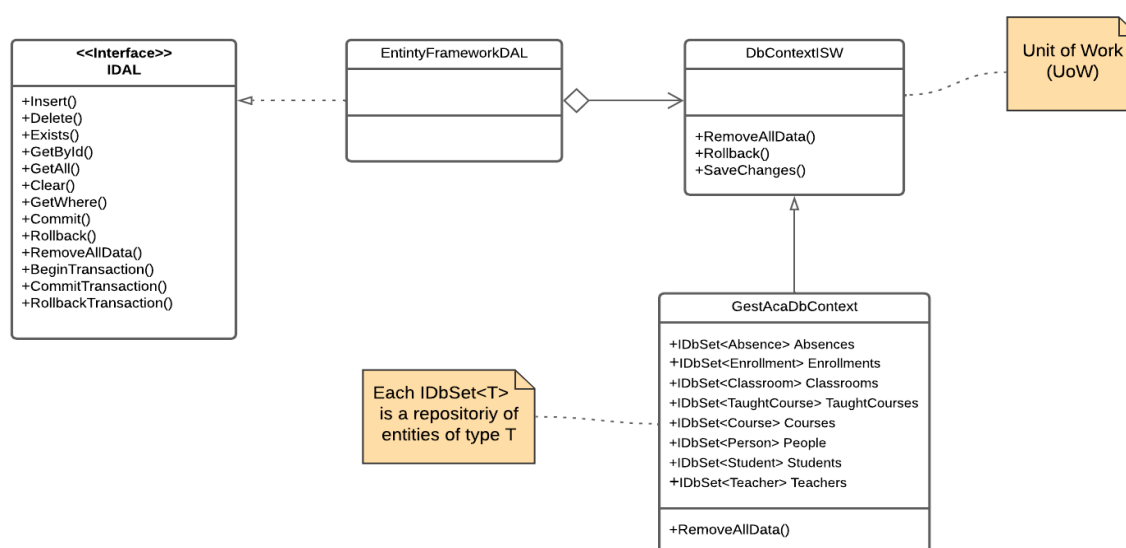


Figura 1. Arquitectura de la capa de acceso a datos

IDAL: La gestión transparente del acceso a datos (agnóstico respecto al mecanismo de persistencia) la vamos a lograr gracias al uso de una **interfaz** entre la capa de lógica y la capa de persistencia real. Esta interfaz, denominada **IDAL** (DAL = Data Access Layer), es la encargada de abstraer los servicios de la capa de acceso a datos del mecanismo de persistencia particular que se utilice (SQL, XML, etc.).

La Interfaz IDAL es un adaptador que combina la funcionalidad de los repositorios, que incluye las operaciones habituales de inserción, borrado, consulta, etc. (Insert, Delete, GetXXX, Exists), junto con la funcionalidad de la unidad de trabajo (Commit, Rollback, BeginTransaction, etc.). Se incluye un método adicional (Clear) para borrar la base de datos. El listado completo de los métodos de la clase IDAL.cs es el siguiente.

```
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();
}
```

El uso de la **genericidad**:

Observe que se ha utilizado *genericidad*, reduciendo notablemente la cantidad de código a implementar. Es decir, en lugar de implementar las siguientes operaciones individuales

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

el mecanismo de genericidad nos permite definir un único método

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

que se instanciará en ejecución en función del tipo **T** (que será una clase) correspondiente.

EntityFrameworkDAL: es una implementación específica de la interfaz IDAL para EF. A continuación, se muestra la parte de código de la clase EntityFrameworkDAL.cs correspondiente a las operaciones más habituales (Insert, Delete, Exists, GetXXX, Clear).

```
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();
    }
    ...
    ...
}
}

```

Se observa que la clase DAL contiene una referencia a un objeto DbContext, tal y como se ilustra en la Figura 1¹ y la implementación particular de cada uno de los servicios que ofrece la interfaz IDAL.

¹ Más exactamente, en la Figura 1 EntityFrameworkDAL contiene DbContextISW, pero téngase en cuenta que esta clase extiende DbContext aunque no se haya representado en el diagrama.

DbContextISW: contiene la implementación de las operaciones sobre la base de datos `RemoveAllData()`, `Rollback()`, así como el tratamiento de excepciones cuando se propagan cambios a la base de datos en `SaveChanges()`. El código de esta clase puede consultarse en el archivo `DbContextISW.cs`.

La implementación de estas tres clases podría reutilizarse en otros proyectos, especialmente las clases `IDAL` y `EntityFrameworkDAL` gracias al uso de la genericidad.

GestAcaDbContext: define el mapeo de objetos del dominio a tablas de la base de datos, siguiendo el patrón Repositorio + UoW. Por tanto, el código de esta clase depende del dominio y de este proyecto en particular. En el código de la clase `GestAcaDbContext.cs` observará las siguientes características:

- Hereda de `DbContext`
- Define los repositorios mediante propiedades que implementan la interfaz `IDbSet<Tipo>` donde “Tipo” es cada una de las clases del modelo que han de ser persistentes en la base de datos: cada `DbSet` constituye un repositorio con los métodos típicos para acceder, añadir o eliminar objetos. En concreto, para este caso de estudio son los siguientes:

```
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; }
```

- Tiene un constructor que proporciona al constructor base el nombre de la cadena de conexión a la base de datos (“`GestAcaDbConnection`”) que deberá definirse en el archivo `App.config` del proyecto de inicio (NO de la biblioteca de enlace dinámico) como se verá más adelante. En el constructor se incluyen también algunos parámetros de configuración.

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

A continuación, se indican las tareas a realizar :

Tarea 1: Configurar la solución.

Para trabajar con EF es necesario que añada a su proyecto el paquete necesario. Para ello, un miembro del equipo (Team Master) hará lo siguiente: en Visual Studio vaya a Herramientas > Administrador de paquetes NuGet > Administrar paquetes NuGet para la solución y en el cuadro de texto de Examinar escriba Entity Framework. Seleccione ese paquete (ver Figura 2) y añádalo al proyecto de biblioteca de su solución (proyecto que creó en la práctica 2 que contiene las subcarpetas `BusinessLogic` y `Persistence` que se llama **GestAcaLib**).

Compruebe en el Explorador de Soluciones que en las Referencias de su proyecto se incluye EF (ver Figura 3). Una vez añadido este paquete se sincronizará la solución para subir los cambios al servidor.

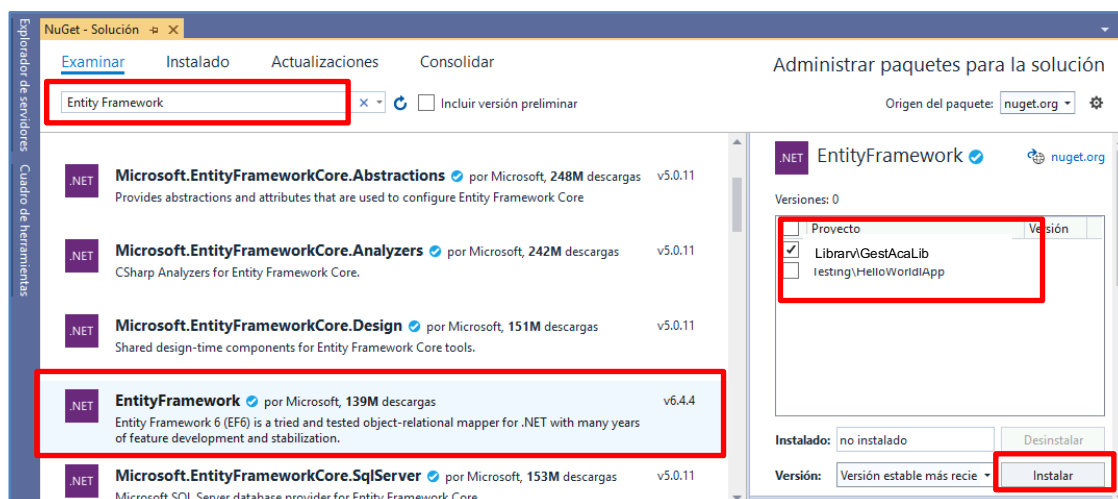


Figura 2. Añadiendo el paquete Entity Framework al proyecto de biblioteca de clases

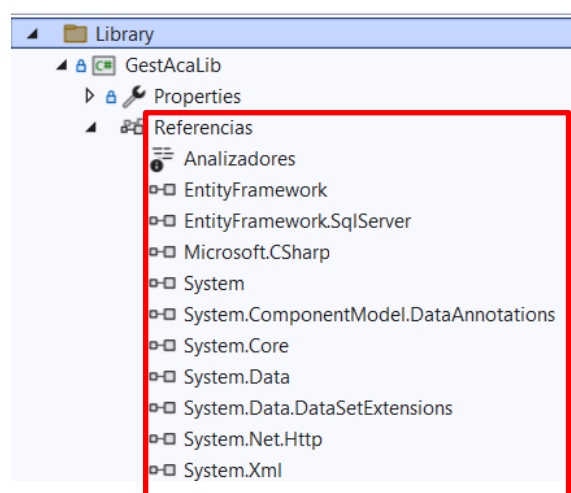


Figura 3. Comprobando que el paquete EF está instalado

Tarea 2: Incluir los archivos necesarios para implementar la capa de persistencia del proyecto.

Para implementar la capa de persistencia del proyecto del caso de estudio siga los siguientes pasos:

1. Descargue el archivo EntityFrameworkImp.zip de PoliformaT que contiene la implementación de las clases DbContextISW, EntityFrameworkDAL, IDAL y GestAcaDbContext.
2. Cree la carpeta EntityFrameworkImp en la carpeta Persistence con botón derecho del ratón > Agregar > Nueva Carpeta. Pulse el botón derecho del ratón sobre la carpeta recién creada y elija la opción "Abrir carpeta en el explorador de archivos".
3. Extraiga las clases descargadas en el paso 1, cópielas y péguelas en la carpeta Persistence > EntityFrameworkImp que ha abierto en el paso 2.
4. Añada a la carpeta EntityFrameworkImp los archivos DbContextISW.cs, EntityFrameworkDAL.cs, IDAL.cs y GestAcaDbContext.cs con botón derecho del ratón sobre la carpeta > Agregar > Elemento Existente A continuación, seleccione los archivos .cs indicados y acepte para añadirlos al proyecto.

Compruebe que el resultado es como el que se muestra en la Figura 4. Fíjese que todas las clases creadas forman parte del mismo namespace: `GestAca.Persistence` por lo que es necesario incluir la directiva `using GestAca.Entities`; en los archivos de clase donde se utilicen las clases del modelo.

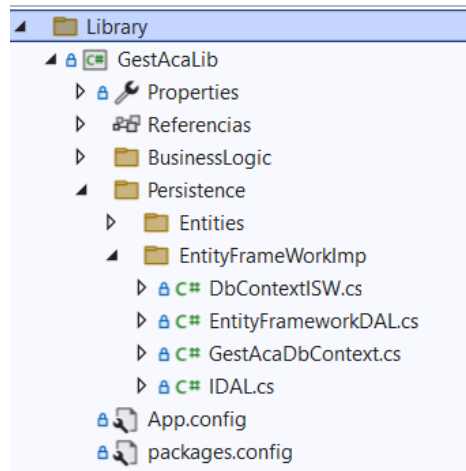


Figura 4. Estructura de la capa de persistencia en el proyecto de VS

Tarea 3: Anotar el código de la capa lógica con Data Annotations.

A veces es necesario modificar el código de la capa lógica mediante las **Data Annotations** para proporcionar cierta información a EF para que construya la BD correctamente. En el seminario 6.1 de teoría sobre **Entity Framework** se dan algunas pautas en relación al uso de anotaciones. Concretamente, preste atención a las anotaciones:

```
[Key]
[InverseProperty("XXX")]
[Required]
```

Deberá escribir la anotación delante de la definición de la propiedad correspondiente.

NOTA: si quiere que un atributo clave de tipo `int` no sea considerado autoincremental, debe utilizar la anotación siguiente:

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

Tarea 4: Revisar la implementación de los constructores.

En la práctica anterior definió los constructores necesarios para crear los objetos de la lógica de negocio. Siguiendo las instrucciones, definió un constructor por defecto y un constructor con argumentos.

El constructor por defecto lo utiliza EF cuando tiene que materializar un objeto en memoria después de recuperar la información de las tablas de la BD. Por eso, el constructor no tiene que inicializar los atributos del objeto (ya lo hace EF) pero sí tiene que inicializar los atributos de tipo colección.

El otro constructor con argumentos es el que se debe utilizar en el código del programa cuando sea necesario crear un objeto. Como ya sabrá, EF se encarga de dar valor a los **atributos ID de tipo numérico** de manera automática en el momento en el que se persiste el objeto por primera vez. Por lo tanto, no es necesario dar valor a dichos atributos en el constructor del objeto ya que EF le cambiará el valor en cuando lo persista por primera vez.

En la sesión anterior, los constructores ya se diseñaron teniendo en cuenta este aspecto: los ID de tipo `string` o los ID numéricos no autoincrementales deberán ser inicializados en el constructor. Los ID de tipo numérico autoincrementales son gestionados por el gestor de base de datos, por lo que no tienen que inicializarse explícitamente en el constructor y por tanto su valor no se pasa como argumento al constructor.



Una vez haya finalizado el desarrollo de la capa de persistencia sincronice su solución con un comentario como “Capa Persistencia Finalizada (versión beta)”.

Tarea 5: Ejecutar los test unitarios de la capa de persistencia.

Como ya sabe, las pruebas de código pueden sistematizarse mejor mediante un motor de pruebas unitarias como MSTest². Al igual que hizo para probar la implementación de la capa lógica en la sesión anterior, tendrá que ejecutar las pruebas unitarias diseñadas por el profesorado. Descargue el proyecto de pruebas GestAcaPersistenceTests.zip de PoliformaT, añádalo a su proyecto y ejecútelo tal y como ya se le ha explicado.

Antes de pasar a la siguiente tarea deberá asegurarse que todos los tests se han ejecutado correctamente.



Una vez haya pasado todos los test sincronice su solución con un comentario como “Capa Persistencia Finalizada”

Tarea 6: Persistir algunos objetos en la base de datos.

A continuación, se crearán algunos objetos de la lógica de negocio y se harán persistentes en la base de datos dejándola en un estado consistente (deben cumplirse todas las restricciones expresadas por las relaciones y cardinalidades del modelo de clases). En concreto se pide crear los objetos necesarios para que se cumpla el siguiente enunciado:

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.

Para ello puede crear un proyecto de consola, tal y como se vio en la primera sesión de prácticas, y escribir un programa que cree los objetos de la lógica de negocio y los haga persistentes usando los servicios ofrecidos por el DAL. Los pasos a seguir serían:

1. Cree un proyecto de consola de .NET Framework dentro de **Testing** llamado **DBTest**.
2. Incluya una referencia a su biblioteca de clases (GestAcaLib). Para ello, en su proyecto de aplicación de consola (DBTest): Botón derecho del ratón sobre Referencias > Agregar referencia ... y seleccionar dentro de Proyectos su proyecto de biblioteca de clases.
3. Agregue el paquete Entity Framework usando el administrador de paquetes NuGet, de forma similar a como se ha explicado en la Tarea 1.
4. Modifique el archivo de configuración App.config que se habrá creado en DBTest, añadiendo la sección connectionStrings que define la cadena de conexión a la base de datos. Si usamos SQLServer como motor de base de datos y denominamos “GestAcaDB” a la base de datos a crear, la sección que debe añadir es ésta:

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

² Puede informarse sobre pruebas unitarias en VS en:

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

En el código de prueba (implementado en Program.cs) deberá incluir los espacios de nombres GestAca.Persistence y GestAca.Entities para poder utilizar las clases de EntityFrameworkImp y las clases de la lógica. Para poder utilizar los servicios del DAL deberá crear en primer lugar una instancia de EntityFrameworkDAL pasándole como argumento una instancia de GestAcaDbContext. **Al crear la instancia de GestAcaDbContext por primera vez se creará la base de datos.**

Para facilitar el trabajo y asegurar que la base de datos se crea correctamente, le proporcionamos el archivo Program.cs con parte del código necesario y algunas utilidades para mostrar por consola los mensajes de error en caso de que se produzca alguna excepción. Copie el contenido de este archivo en el archivo Program.cs creado en el proyecto DBTest. La estructura resultante del proyecto será como la de la Figura 5.

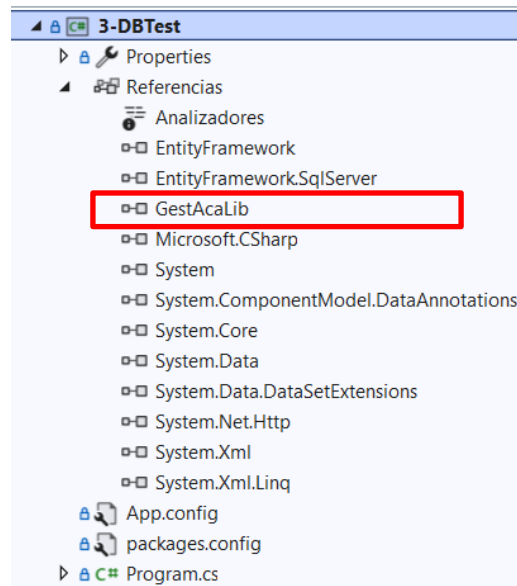


Figura 5. Estructura del proyecto de consola de Test

Observará un método prácticamente vacío denominado createSampleDB que habrá que rellenar con el código necesario para crear los objetos de negocio y persistirlos en la base de datos. Como punto de partida, le proporcionamos el código para crear dos cursos (course).

```
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
}
```

Como se puede observar en el código después de borrar el contenido de la base de datos (`dal.RemoveAllData()`), se crearán dos instancia de `Course`. Para que los datos persistan en la base de datos debemos añadir el objeto al repositorio de `Course`, con la instrucción `dal.Insert<Course>(aCourse1)`, y a continuación realizar el commit en la base de datos. De forma similar, creamos y persistimos el objeto `aCourse2`.

Recuerde que para poder ejecutar este proyecto deberá indicar que es el proyecto de inicio. Para ello, deberá pulsar con el botón derecho del ratón en el proyecto y en el menú contextual escoger la opción “Establecer como proyecto de inicio”.

Es importante recordar que tras completar una transacción (una o más operaciones que implican un cambio en los datos), debe invocarse al método `Commit` para persistir todos los cambios. Aunque no se ofrece un método específico para actualizar un objeto, **si se han modificado objetos internamente, igualmente tendremos que ejecutar el método `Commit`.**

Para comprobar que la BD se ha actualizado correctamente podría consultar su contenido desde el código o bien utilizar la herramienta de inspección incluida en VS, tal y como se explica en el siguiente apartado correspondiente a la última tarea que debe realizar.

Tarea 7: Utilice la herramienta de inspección de la BD para verificar que se ha almacenado la información en cada una de las tablas.

Desde el propio entorno de desarrollo en Visual Studio es posible conectarse a cualquier base de datos para ver sus tablas y su contenido. Para conectarse a una base de datos existente local (creada como un fichero local) tendrá que realizar los siguientes pasos (Figura 6):

- Herramientas > Conectar con la Base de Datos, y
- Seleccionar como origen de datos “Archivo de base de datos de Microsoft SQL Server”, y
- Seleccionar como archivo de datos el fichero con extensión “.mdf” creado. Dada la configuración realizada en el fichero `App.config`, el fichero de base de datos “.mdf” se crea en `C:\Users\nombreUsuario\GestAcadB.mdf`.

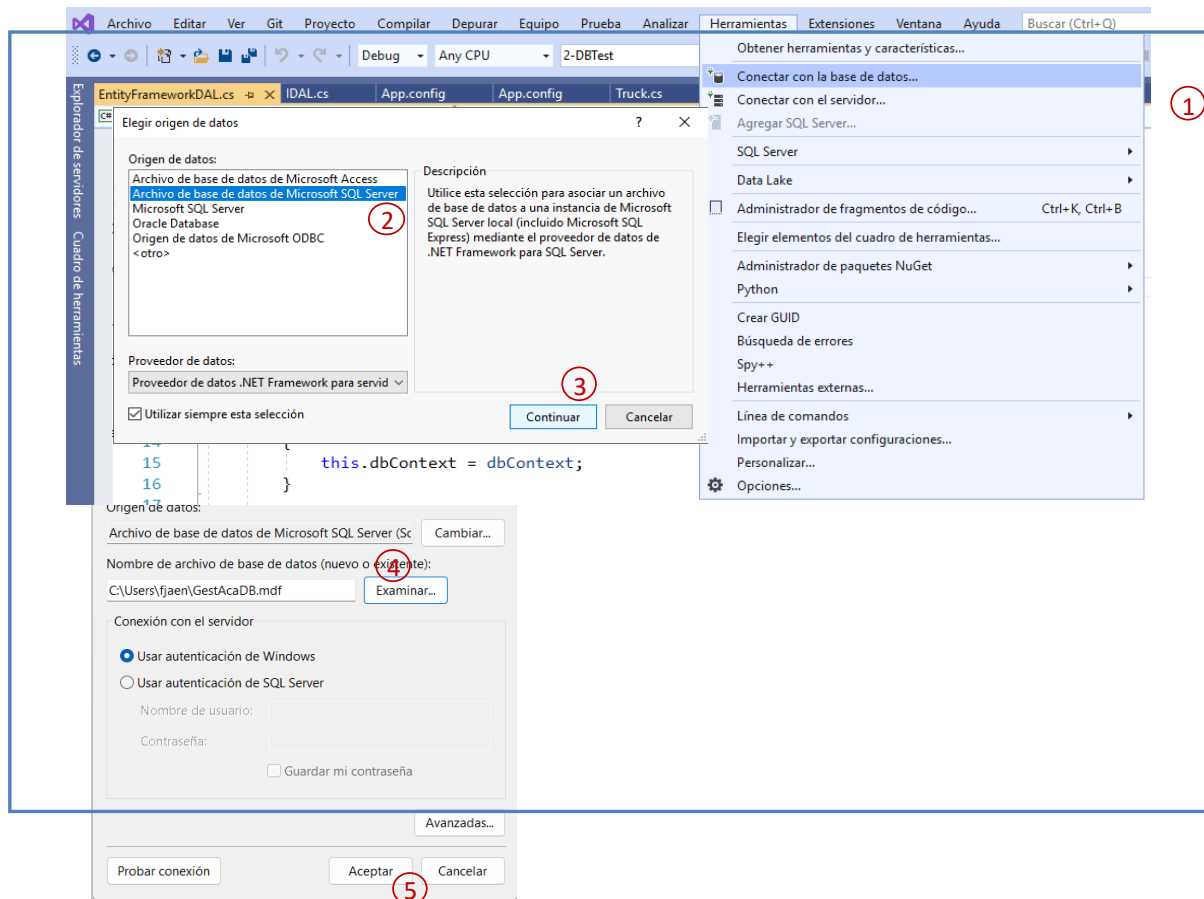


Figura 6: Conectando con la base de datos desde VS

Una vez creada la conexión es posible explorar las tablas de la base de datos en el explorador de servidores que aparecerá a la izquierda en el entorno de desarrollo, o desde Ver > Explorador de servidores, como muestra la Figura 7.

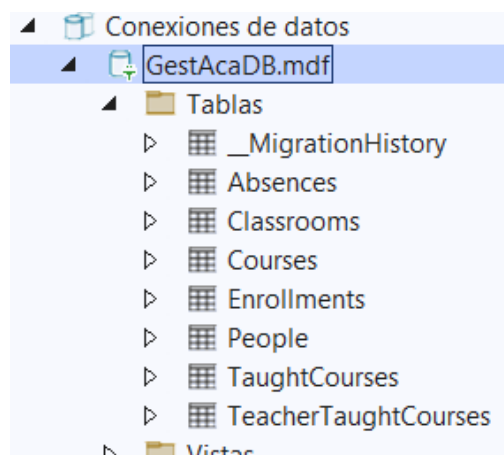



Figura 7. Tablas de la BD desde el explorador de servidores

Haciendo doble-click sobre una tabla se puede ver su estructura. Para ver los datos almacenados en una tabla se hará click con el botón derecho de ratón sobre la tabla > Mostrar datos de tabla. La Figura 8 muestra el contenido de la tabla Courses después de añadir dos cursos la base de datos.

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

Figura 8. Contenido de la tabla Courses

De forma similar, desde el explorador de servidores podemos hacer click con el botón derecho de ratón sobre la conexión a la BD (GestAcaDB.mdf) y elegir la opción Nueva consulta, lo que nos permitirá escribir sentencias SQL que trabajen sobre nuestra base de datos. Por ejemplo, fácilmente podremos visualizar todos los registros que hay en una tabla TTT de la base de datos escribiendo “`Select * From TTT`” y ejecutando la consulta con . La Figura 9 muestra el resultado de consultar la tabla People.

SQLQuery1.sql *

dbo.Courses [Datos]

Program.cs

Programa

GestAcaDB

1

Select * from People

89 %

No se encontraron problemas.

T-SQL

Resultados

Mensaje

	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

Figura 9: Ejemplo de consulta a la base de datos



Una vez haya comprobado que la BD se ha poblado correctamente sincronice de nuevo su solución con un comentario tipo “Creación y persistencia de objetos finalizada”. Se debe tener en cuenta que la base de datos NO se sincroniza en el repositorio, por lo que es posible que cada miembro del equipo tenga un contenido distinto dependiendo de los objetos que haya creado.