



UNIVERSIDAD
POLITECNICA
DE VALENCIA

Pràctiques

Butlletí Pràctica 4

Capa de Persistència amb Entity Framework

Enginyeria del Programari

ETS Enginyeria Informàtica

DSIC – UPV

Curs 2024-2025

1. Objectiu

L'objectiu principal d'aquesta sessió és desenvolupar la capa d'accés de dades o persistència de l'aplicació. La capa de persistència oferirà a la capa lògica els serveis necessaris per a recuperar, modificar, esborrar o afegir objectes, sense que la capa de lògica conega quin mecanisme de persistència particular s'està utilitzant.

Per a facilitar el treball s'utilitzarà **Entity Framework (EF)**. **EF** és un sistema de mapeo objecte-relacional que permetrà a la nostra aplicació treballar directament amb Entitats que són objectes de la lògica de negoci, i no objectes específics per a transferència de dades (DTO). **EF** s'encarregarà, automàticament, de gestionar de manera transparent la persistència d'aquests objectes en una base de dades relacional com SQL. L'accés a dades amb EF es basa en el patró **Repositori + Unitat de Treball**, tal com s'explica en el tema 6 dedicat a la Persistència.

Prèviament a la realització de la pràctica, l'estudiant haurà d'haver repassat el “Tema 6 Disseny de Persistència” i els seminaris associats dedicats a “Entity Framework” i “DAL”.

D'altra banda, el cas d'estudi de referència “VehicleRental”, que pot descarregar de Poliformat, és una implementació completa, utilitzant l'arquitectura explicada en classe, que ha de servir com a exemple a l'estudiant en el desenvolupament del cas d'estudi que ens ocupa.

En finalitzar la sessió, l'estudiant haurà de ser capaç de:

- Descriure l'arquitectura de la capa de persistència.
- Configurar el projecte per a poder implementar la capa de persistència utilitzant **EF**.
- Realitzar operacions amb objectes en la base de dades (crear, llegir, esborrar, actualitzar).
- Executar els test unitaris que comproven el correcte funcionament de la capa de persistència.

A continuació, es descriu en primer lloc l'arquitectura de la capa de persistència, i la finalitat de cadascuna de les classes que la componen, i després s'enumeraran les tasques que ha de realitzar l'estudiant per a implementar la capa de persistència del projecte.

2. Disseny de la capa d'accés a dades

L'arquitectura de la capa d'accés a dades que es dissenyarà és la que es mostra en la Figura 1. A continuació de descriuen els elements de l'arquitectura.

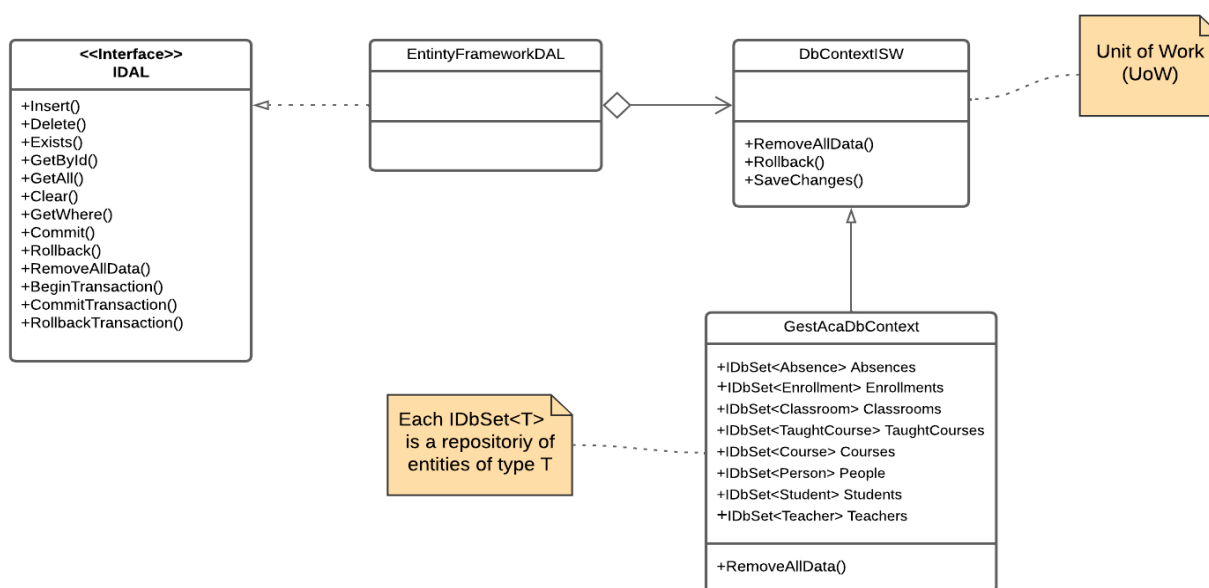


Figura 1. Arquitectura de la capa d'accés a dades

IDAL: La gestió transparent de l'accés a les dades (agnòstic respecte al mecanisme de persistència) l'aconseguirem gràcies a l'ús d'una interfície entre la capa de lògica i la capa de persistència real. Aquesta interfície, denominada IDAL (**DAL = Data Access Layer**), és l'encarregada d'abstraure els serveis de la capa d'accés a dades del mecanisme de persistència particular que s'utilitzi (SQL, XML, etc.).

La Interfície IDAL és un adaptador que combina la funcionalitat dels repositoris, amb les operacions habituals d'inserció, borrat, consulta, et. (Insert, Delete, GetXXX, Exists), juntament amb la funcionalitat de la unitat de treball (Commit, Rollback, BeginTransaction, etc). S'inclou un mètode addicional (Clear) per a esborrar la base de dades. El llistat complet dels mètodes de la classe IDAL.cs és el següent:

```
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();
}
```

L'ús de la genericitat:

Podeu observar que s'ha utilitzat *genericitat*, reduint notablement la quantitat de codi a implementar. Es a dir, en lloc de implementar las següents operacions individuals

```
void InsertCourse(Course p);
void InsertStudent(Student u);
void InsertAbsence(Absence a);
...
```

el mecanisme de genericitat ens permet definir un únic mètode

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

que s'instanciarà en execució en funció del tipo **T** (que serà una classe) corresponent.

EntityFrameworkDAL: és una implementació específica de la interfície IDAL para EF. A continuació, es mostra la part de codi de la classe EntityFrameworkDAL.cs corresponen a les operacions més habituals (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();
}
...
...
...
}
}

```

S'observa que la classe DAL conté una referència a un objecte DbContext, tal i com al com s'il·lustra en la Figura 1¹, i la implementació particular de cadascun dels serveis que ofereix la interfície IDAL..

¹ Concretament, en la Figura 1 EntityFrameworkDAL conté DbContextISW, però s'ha de tenir en conter que aquesta classe es una especialització de DbContext encara que no es està representat al diagrama.

DbContextISW: conté la implementació de les operacions sobre la base de dades `RemoveAllData()`, `Rollback()`, així com el tractament d'excepcions quan es propaguen canvis a la base de dades en `SaveChanges()`. El codi d'aquesta classe pot consultar-se en l'arxiu `DbContextISW.cs`.

La implementació d'aquestes tres classes podria reutilitzar-se en altres projectes, especialment les classes `IDAL` y `EntityFrameworkDAL` gràcies a l'ús de la genericitat.

GestAcadbContext: defineix el mapatge d'objectes del domini a taules de la base de dades, seguint el patró `Repositori + UoW`. Per tant, el codi d'aquesta classe depèn del domini i d'aquest projecte en particular. En el codi de la classe `GestAcadbContext.cs` observarà les següents característiques:

- Hereta de `DbContext`
- Defineix els repositoris mitjançant propietats que implementen la interfície `IDbSet` on "Tipus" és cadascuna de les classes del model que han de ser persistents en la base de dada: cada `DbSet` constitueix un repositori amb els mètodes típics per a accedir, afegir o eliminar objectes. En concret, per a aquest cas d'estudi són els següents:

```
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; }
```

- Té un constructor que proporciona al constructor base el nombre de la cadena de connexió a la base de dades ("`GestAcadbConnection`") que deurà definir-se en l'arxiu `App.config` del projecte d'inici (NO de la biblioteca de enllaç dinàmic) como es veurà més endavant. En el constructor s'inclouen també alguns paràmetres de configuració.

```
public GestAcadbContext() : base("GestAcadbConnection") { ... }
```

A continuació, s'enumeraran les tasques que ha de realitzar l'estudiant per a implementar la capa de persistència del projecte.

Tasca 1: Configurar la solució.

Per a treballar amb EF és necessari afegir al projecte el paquet necessari. Per a això, un membre de l'equip (Team Leader) farà el següent:

en Visual Studio en l'apartat **Eines > Administrador de paquets NuGet > Administrar paquets NuGet per a la solució** i en el quadre de text d'Examinar escriure Entity Framework. Seleccionar aqueix paquet (veure Figura 2) i afegir-lo al projecte de biblioteca de la vostra solució (projecte que es va crear en la pràctica 2 i que conté les subcarpetes BusinessLogic i Persistence que es diu **GestAcaLib**).

Comprovar en l'Explorador de Solucions que en les Referències del vostre projecte s'inclou **EF** (veure Figura 3). Una vegada afegit aquest paquet se sincronitzarà la solució per a pujar els canvis al servidor.

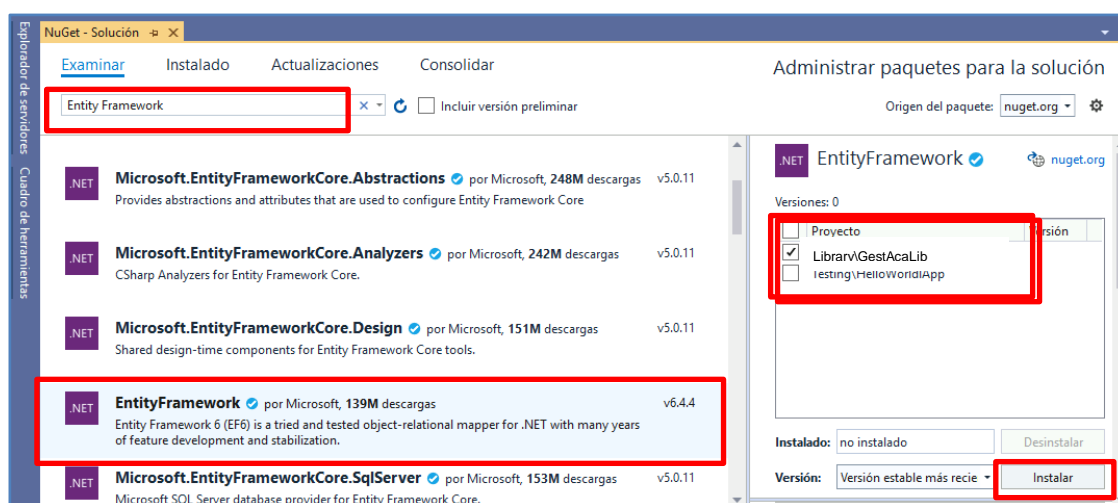


Figura 2. Afegint el paquet Entity Framework al projecte de biblioteca de classes

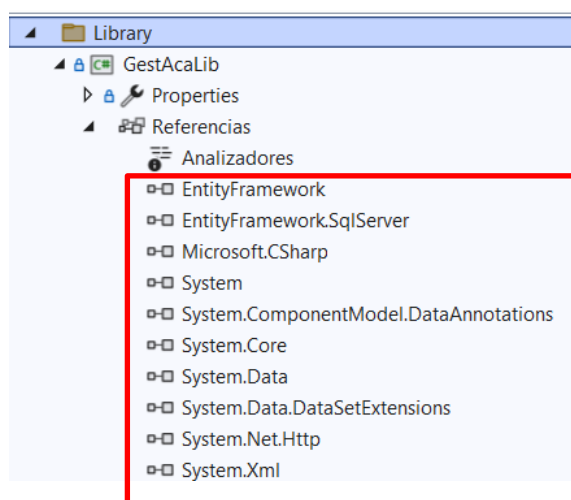


Figura 3. Comprovant que el paquet EF està instal·lat

Tasca 2: Incloure els arxius necessaris per a implementar la capa de persistència del projecte.

Per a implementar la capa de persistència del projecte (cas d'estudi) es deuen seguir els següents passos:

1. Descarregar l'arxiu EntityFrameworkImp.zip de PoliformaT que conté la implementació de las classes DbContextISW, EntityFrameworkDAL, IDAL i GestAcaDbContext.
2. Crear la carpeta EntityFrameworkImp en la carpeta Persistence amb el botó dret del ratolí > Agregar > Nueva Carpeta. Prémer botó dret del ratolí sobre la carpeta acabada de crear i elegir l'opció "Abrir carpeta en el explorador de archivos".
3. Extraure las classes descarregades en el pas 1, copiar i pegar en la carpeta Persistence > EntityFrameworkImp que s'ha obert en el pas 2.
4. Afegir a la carpeta EntityFrameworkImp els arxius DbContextISW.cs, EntityFrameworkDAL.cs, IDAL.cs i GestAcaDbContext.cs amb el botó dret del ratolí sobre la carpeta > Agregar > Elemento Existente A continuació, seleccionar els arxius .cs indicats i acceptar per afegir-los al projecte.

Comprovar que el resultat és como el que es mostra en la Figura 4. Hi ha que comprovar que totes les classes creades formen part del mateix namespace: GestAca.Persistence, per lo que és necessari incloure la directiva `using GestAca.Entities;` en els arxius de classe on s'invoquen les classes del model.

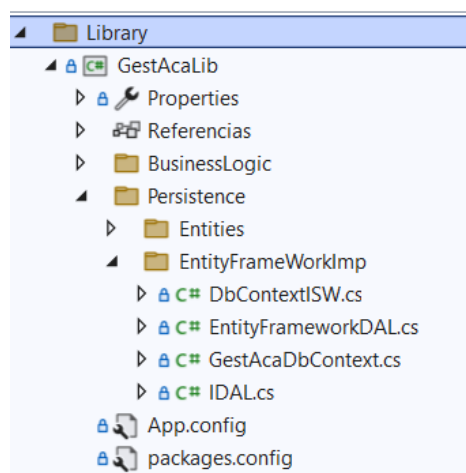


Figura 4. Estructura de la capa de persistència en el projecte de VS

Tasca 3: Anotar el codi de la capa lògica amb *Data Annotations*.

A vegades es necessari modificar el codi de la capa lògica utilitzant les **Data Annotations** per a proporcionar certa informació a **EF** per construir la BD correctament. En el seminari 6.1 de teoria sobre **EF** es donen algunes pautes en relació a l'ús de anotacions. En concret preste atenció a les anotacions:

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

Cada equip deurà escriure l'anotació adequada davant la definició de la propietat corresponen, sempre que es necessita. **NOTA:** en cas de que un atribut clau (identificador) de tipus int no desitgem que es considere autoincremental, es deu utilitzar la següent anotació:

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

Tasca 4: Revisar la implementació dels constructors.

En la pràctica anterior va definir els constructors necessaris per a crear els objectes de la lògica de negoci. Seguint les instruccions, va definir un constructor per defecte i un constructor amb arguments.

El constructor per defecte l'utilitza EF quan ha de materialitzar un objecte en memòria després de recuperar la informació de les taules de la BD. Per això, el constructor no ha d'inicialitzar els atributs de l'objecte (ja el fa EF) però sí que ha d'inicialitzar els atributs de tipus col·lecció.

L'altre constructor amb arguments és el que s'ha d'utilitzar en el codi del programa quan siga necessari crear un objecte. Com ja sabrà, EF s'encarrega de donar valor als **atributs ID de tipus numèric** de manera automàtica en el moment en el qual es persisteix l'objecte per primera vegada. Per tant, no és necessari donar valor a aquests atributs en el constructor de l'objecte ja que EF li canviarà el valor en quan el persistisca per primera vegada.

En la sessió anterior, els constructors ja es van dissenyar tenint en compte aquest aspecte: els ID de tipus string o el ID numèrics no autoincrementals hauran de ser inicialitzats en el constructor. Els ID de tipus numèric autoincremental són gestionats per el gestor de la base de dades, per tant, no s'inicialitzen explícitament en el constructor i el seu valor no es passa com a argument al constructor.



Una vegada haja finalitzat el desenvolupament de la capa de persistència sincronitze la seua solució amb un comentari tipus "Capa Persistència Finalitzada (versió beta)".

Tasca 5: Executar els test unitaris de la capa de persistència.

Les proves de codi poden sistematitzar-se millor mitjançant un motor de proves unitàries com MSTest. Igual que es va fer per a provar la implementació de la capa lògica en la sessió anterior, hauran d'executar-se les proves unitàries dissenyades pel professorat per a la capa de persistència.

Descarregar el projecte de proves GestAcaPersistenceTests.zip de PoliformaT, afegir-lo al vostre projecte i executar-lo tal com ja s'ha explicat per a les proves de la capa lògica.

Abans de passar a la següent tasca, l'equip es deurà assegurar que tots els test s'han executat correctament.



Una vegada passats tots els tests, sincronitzar la solució amb un comentari com "Capa Persistència Finalitzada"

Tasca 6: Crear objectes i que s'emmagatzemen en la base de dades

A continuació es crearan alguns objectes de la lògica de negoci i es faran persistents en la base de dades deixant-la en un estat consistent (han de complir-se totes les restriccions expressades per les relacions i cardinalitats del model de classes). En concret es demana crear els objectes necessaris d'acord al següent enunciat:

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.

Per realitzar aquesta tasca es pot crear un projecte de consola, tal i com es va veure en el butlletí 1 de pràctiques (sessió 1) i escriure un programa per a crear els objectes de la lògica de negoci i els faça persistents mitjançant els serveis que proporciona el DAL. El passos a seguir serien:

1. Crear un projecte de consola de .NET Framework dins de la carpeta **Testing** anomenat **DBTest**.
2. Incloure una referència a la seua biblioteca de classes (**GestAcaLib**). Per a això, en el seu projecte d'aplicació de consola (DBTest): Botó dret del ratolí sobre Referencias > Agregar referencia ... i seleccionar dins de Projectes el seu projecte de biblioteca de classes.
3. Agregar el paquet Entity Framework (EF) amb l'administrador de paquets NuGet, de manera similar a com s'ha explicat en l'apartat 2.
4. Modificar l'arxiu de configuració **App.config** que s'haurà creat en **DBTest**, afegint la secció **connectionStrings** que defineix la cadena de connexió a la base de dades. Si gastem SQLServer com a motor de base de dades i denominem "**GestAcaDB**" a la base de dades a crear, la secció que ha d'afegir és aquesta:

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

En el codi de prova (implementat en Program.cs) haurà d'incloure els espais de noms **GestAca.Persistence** i **GestAca.Entities** per a poder utilitzar les classes de **EntityFrameworkImp** i les classes de la lògica. Per a poder utilitzar els serveis del DAL haurà de crear en primer lloc una instància de **EntityFrameworkDAL** passant-li com a argument una instància de **GestAcaDbContext**. **En crear la instància de **GestAcaDbContext** per primera vegada es crearà la base de dades.**

Per a facilitar-li el treball i assegurar que la base de dades es crea correctament, li proporcionem l'arxiu **Program.cs** amb part del codi necessari i algunes utilitats per a mostrar per consola els missatges d'error en cas que es produïska alguna excepció. Copiar el contingut d'aquest arxiu en l'arxiu **Program.cs** creat en el projecte **DBTest**. L'estructura resultant del projecte serà com la de la Figura 5.

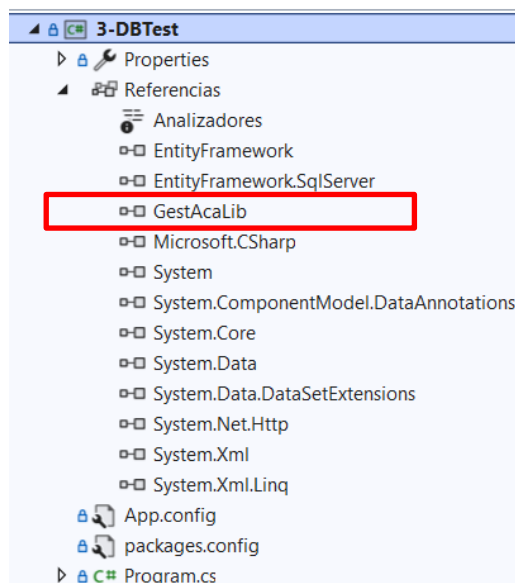


Figura 5. Estructura del projecte de consola de Test

Observar que en Program.cs hi ha un mètode pràcticament buit denominat createSampleDB que caldrà ampliar amb el codi necessari per a crear els objectes de negoci i persistir-los en la base de dades. Com a punt de partida, li proporcionem el codi per a crear un parell de cursos.

```
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
}
```

Com es pot observar en el codi després d'esborrar el contingut de la base de dades (dal.RemoveAllData()), es creen dos instàncies de Course. Per a que les dades persistisquen en la base de dades hem d'afegir els objectes al repositori de Course, amb la instrucció dal.Insert<Course>(aCourse1), i a continuació realitzar el commit en la base de dades.

Recordar que per a poder executar aquest projecte haurà d'indicar que és el projecte d'inici. Per a això, haurà de prémer amb el botó dret del ratolí en el projecte i en el menú contextual triar l'opció “Establecer como proyecto de inicio”.

És important recordar que després de completar una transacció (una o més operacions que impliquen un canvi en les dades), ha d'invocar-se al mètode Commit per a persistir tots els canvis. Encara que no s'ofereix un mètode específic per a actualitzar un objecte, **si s'han modificat objectes internament, igualment haurem d'executar el mètode Commit.**

Per a comprovar que la BD s'ha actualitzat correctament podria consultar el seu contingut des del codi o bé utilitzar l'eina d'inspecció inclosa en VS, tal com s'explica en el següent apartat corresponent a l'última tasca que ha de realitzar.

Tasca 7: Utilitzar l'eina d'inspecció de la BD per a verificar que s'ha emmagatzemat la informació en cadascuna de les taules.

Des del propi entorn de desenvolupament en Visual Studio és possible connectar-se a qualsevol base de dades per a veure les seues taules i el seu contingut. Per a connectar-se a una base de dades existent local (creada com un fitxer local) haurà de realitzar els següents passos (Figura 6):

- Herramientas > Conectar con la Base de Datos, i
- Seleccionar com a origen de dades “Archivo de base de datos de Microsoft SQL Server”, i
- Seleccionar com a arxiu de dades el fitxer amb extensió “.mdf” creat. Donada la configuració realitzada en el fitxer App.config, el fitxer de base de dades “.mdf” es crea en **C:\Users\nombreUsuario\GestAca.mdf.**

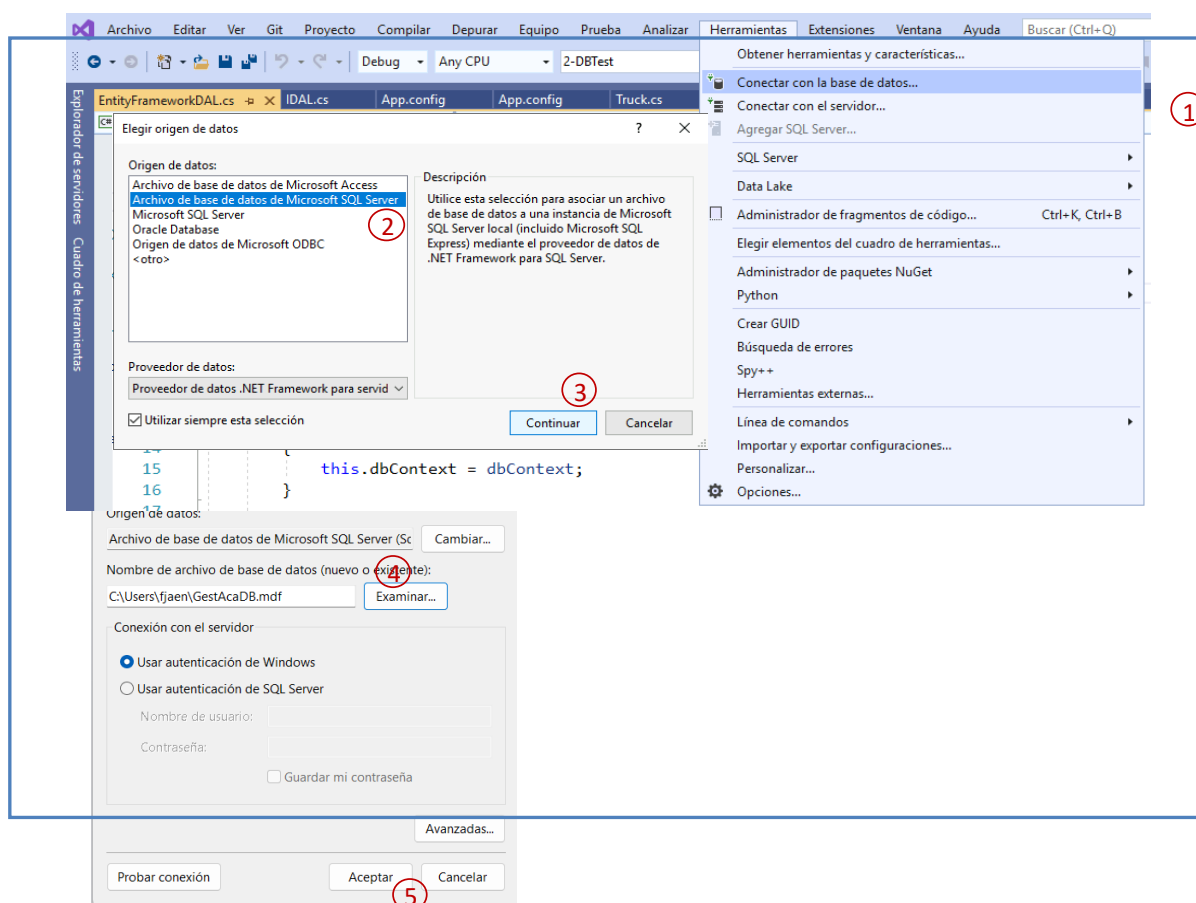


Figura 6: Connectant amb la base de dades des de VS

Una vegada creada la connexió és possible explorar les taules de la base de dades en l'explorador de servidors que apareixerà a l'esquerra a l'entorn de desenvolupament, o des de **Ver > Explorador de servidores**, com a mostra la Figura 7.

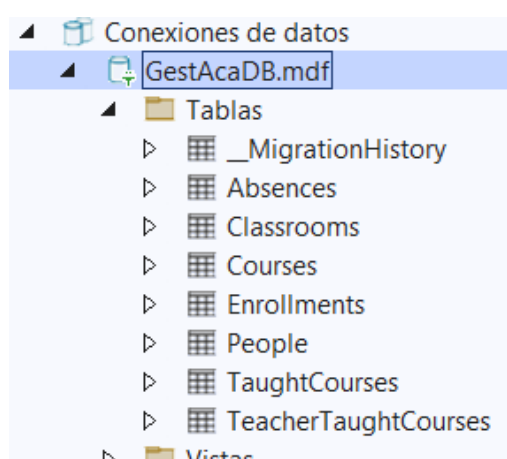
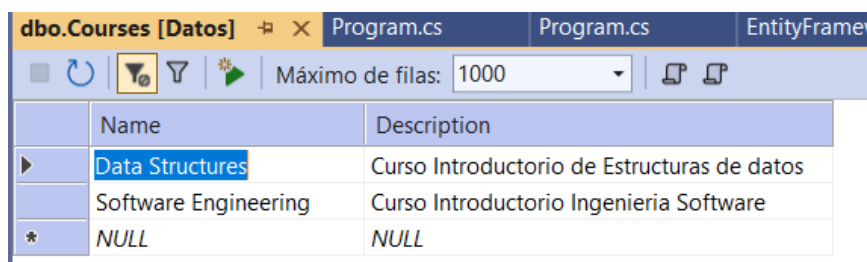



Figura 7. Taules de la BD des de l'explorador de servidors

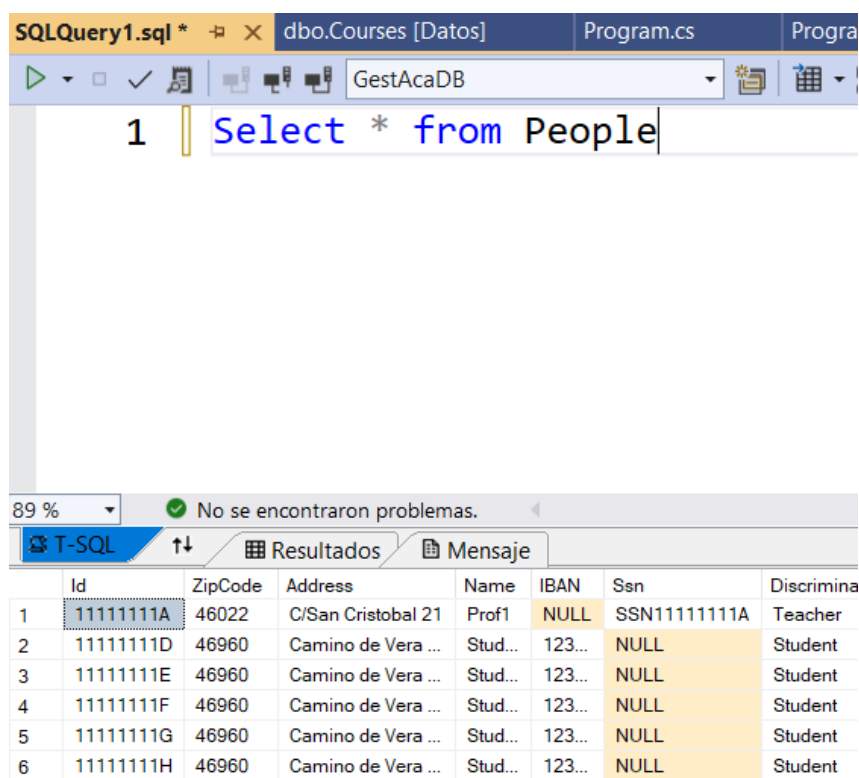
Fent doble-clic sobre una taula es pot veure la seua estructura. Per a veure les dades emmagatzemades en una taula es farà clic amb el botó dret de ratolí sobre la taula > **Mostrar datos de tabla**. La Figura 8 mostra el contingut de la taula Courses després d'afegir el cursos del DBtest.



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

Figura 8. Contingut de la taula Courses

De manera similar, des de l'explorador de servidors podem fer clic amb el botó dret de ratolí sobre la connexió a la BD (GestAca.mdf) i triar l'opció Nueva consulta, la qual cosa ens permetrà escriure sentències SQL que treballen sobre la nostra base de dades. Per exemple, fàcilment podrem visualitzar tots els registres que hi ha en una taula TTT de la base de dades escrivint “Select * From TTT” i executant la consulta amb . La Figura 9 mostra el resultat de consultar la taula 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

Figura 9: Exemple de consulta a la BD



Una vegada haja comprovat que la BD s'ha poblat correctament sincronitze de nou la seua solució amb un comentari tipus “Creació i persistència d'objectes finalitzada”. S'ha de tenir en compte que la base de dades NO se sincronitza en el repositori, per la qual cosa és possible que cada membre de l'equip visualitze un contingut diferent depenent dels objectes que haja creat.