UNIVERSIDAD POLITECNICA DE VALENCIA

**Pràctiques**

*Butlletí Pràctica 4*

# Capa de Persistència amb Entity Framework

**Enginyeria del Programari**

ETS Enginyeria Informàtica
DSIC – UPV

**Curs 2021.2022**

## 1. Objectiu

IMPORTANT: Llegir primer tot el document abans de començar la implementació

L'objectiu d'aquesta sessió és el desenvolupament d'una capa d'accés de dades que oferisca a la lògica de l'aplicació els serveis necessaris per a recuperar, modificar, esborrar o afegir objectes a la capa de persistència, sense que la capa de lògica sàpia quin mecanisme de persistència particular s'està usant. Per a facilitar el treball s'utilitzarà **Entity Framework (EF)**.

**EF** és un sistema de mapatge objecte-relacional que va permetre a la nostra aplicació treballar directament amb "*Entitats"* que són objectes de la lògica del negoci, i no objectes específics per a transferència de dades. EF s'encarregarà, automàticament, de gestionar de forma transparent la persistència dels objectes en una base de dades relacional com SQL. L'accés a les dades utilitzant **EF** es basa en el patró **Repositori + Unitat de Treball**, tal i 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.**

L'objectiu d'aquesta sessió és desenvolupar les classes que formen la capa de persistència del cas d'estudi TarongISW de forma anàloga a l'exemple de referència, i provar que la capa de persistència funciona correctament.

A continuació es detallen els passos i aspectes a tenir en conter per a implementar la capa de persistència del projecte.

## 2. Configuració inicial de la solució

Per a treballar amb **EF** es necessari afegir al vostre projecte el paquet d'EF. Per a això, un membre de l'equip (Team Leader) farà el següent: en Visual Studio anar a `Herramientas > Administrador de paquetes NuGet > Administrar paquetes NuGet para la solución` i en el quadre de text `Examinar` escriure Entity Framework. Seleccionar eixe paquet (veure Figura 1) i haurà d'afegir-lo al projecte de biblioteca de la seua solució (projecte que va crear en la pràctica 2 que conté les subcarpetes *BusinessLogic* i *Persistence* que es diu **TarongISWLib**).

**Tasca 1:** Afegir el paquet **EF** al vostre projecte d'acord a les indicacions anteriors.

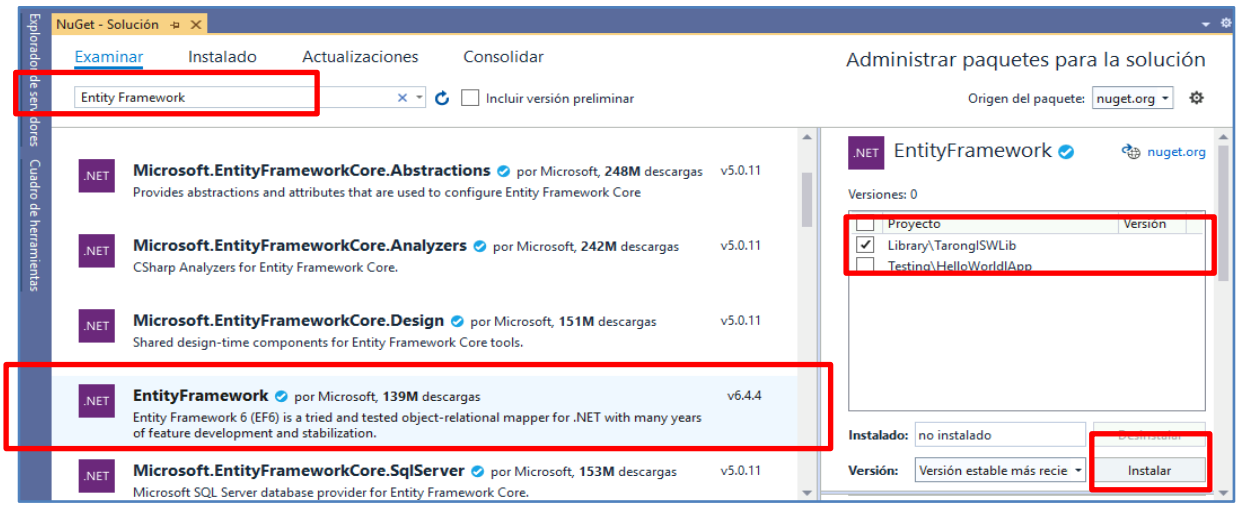


*Figura 1. S'inclou el paquet **EF** al projecte de biblioteca de classes*

Comproveu en l'Explorador de Solucions que en les Referències del vostre projecte s'inclou **EF** (vore Figura 2). Una vegada afegit aquest paquet es protegirà la solució per a pujar els canvis al servidor.

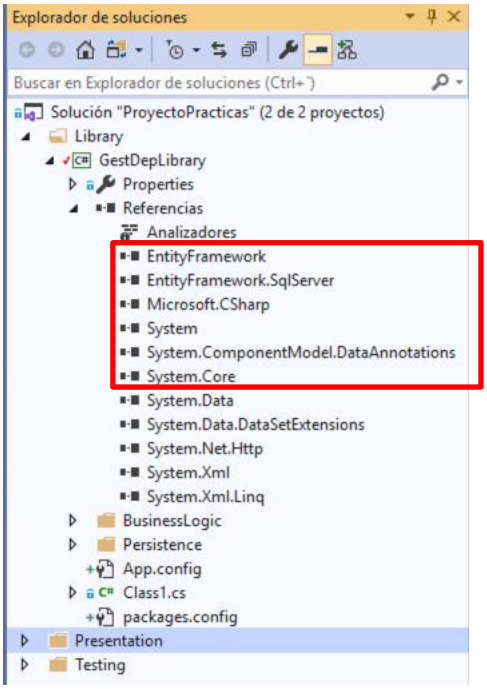


*Figura 2. Comprovant si el paquet EF està instal·lat.*

## 3. Disseny de la capa d'accés a dades (capa de persistència)

La arquitectura de la capa de persistència o d'accés a dades anem a dissenyar-la de forma similar a la que es mostra a la Figura 3. Esta arquitectura es la que es gasta en el projecte d'exemple (VehicleRental). A continuació es descriuen els elements de l'arquitectura.

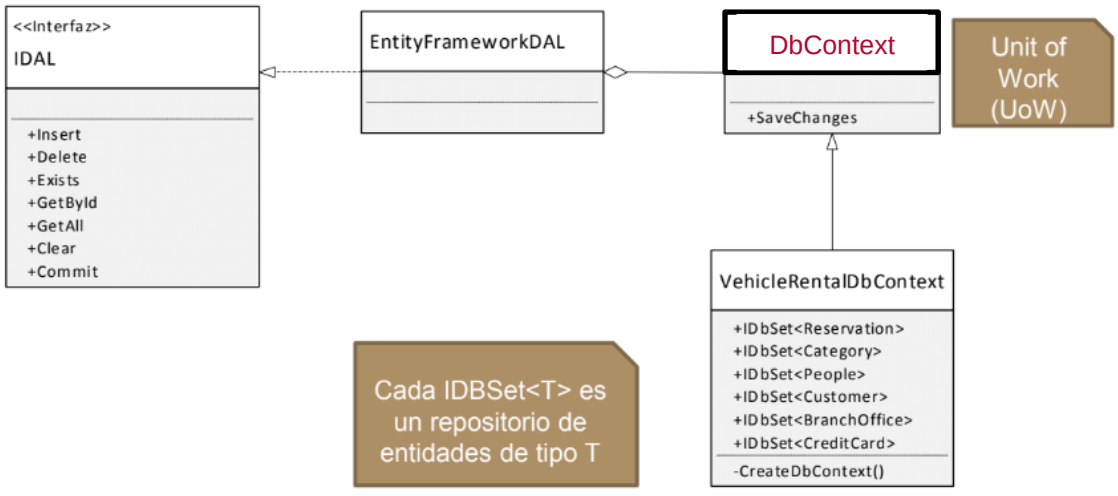


*Figura 3. Arquitectura de la capa d'accés a dades*

**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'utilitze (SQL, XML, etc.).

La Interfície IDAL és un adaptador que combina la funcionalitat dels repositoris (Insert, Delete, GetXXX, Exists), juntament amb la funcionalitat de la unitat de treball (Commit). En efecte, si consultem el projecte d'exemple VehicleRental, veurem que IDAL (arxiu Persistence/Idal.cs) té els següents mètodes.

```
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;
void Commit();
```

Es tracta dels mètodes habituals ja comentats, més un mètode addicional (Clear) per a esborrar la base de dades.

**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 InsertOffice(BranchOffice o);
void InsertReservation(Reservation r);
void InsertCategory(Category c);
...
```

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 per a EF. El codi d'aquesta classe, que en el projecte d'exemple es troba a `Persistence/EntityFrameworkImp/EntityFrameworkDAL.cs`, és el següent:

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

namespace VehicleRental.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 void Commit()
        {
            dbContext.SaveChanges();
        }

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

    }
}
```

S'observa que la classe DAL conté una referència a un objecte DbContext, tal i com al com s'il·lustra en la Figura 3, i la implementació particular de cadascun dels serveis que ofereix la interfície IDAL. DbContext defineix el mapatge d'objectes del domini a taules de la base de dades, seguint el patró Repositori + UoW.

**VehicleRentalDbContext**: aquesta classe estén `DbContext` i en ella es defineixen les propietats que implementen els repositoris per a persistir les classes del negoci, i algunes opcions de configuració i creació de la base de dades.

En el projecte d'exemple si observeu la classe `VehicleRentalDbContext`, que es troba en `Persistence/EntityFrameworkImp/Vehiclerentaldbcontext.cs`, observarà les següents característiques:

- Hereta de `DbContext`
- Té propietats de tipus `DbSet<Tipo>` on "`Tipo`" és cadascuna de les classes del model que han de ser persistents en la base de dades (cada IDbSet constitueix un repositori amb els mètodes típics per a accedir, afegir o eliminar objectes). En concret, en el projecte exemple tenim els següents:

```
public IDbSet<BranchOffice> BranchOffices { get; set; }
public IbSet<Reservation> Reservations { get; set; }
public IDbSet<Category> Categories { get; set; }
public IDbSet<Person> People { get; set; }
```

```
public IDbSet<Customer> Customers { get; set; }
public IDbSet<CreditCard> CreditCards { get; set; }
```

- Té un constructor que proporciona al constructor base el nom de la cadena de connexió a la base de dades ("VehicleRentalDBConnection") que està definida en l'arxiu `App.config` del projecte d'inici (NO de la biblioteca d'enllaç dinàmic). En el constructor també s'inclouen alguns paràmetres de configuració.

```
public VehicleRentalDbContext() : base("Name=VehicleRentalDbConnection") { … }
```

Per a implementar la capa de persistència del projecte del cas d'estudi, hi ha que tindre en compte lo següent:

- En PoliformaT estarà disponible un arxiu EntityFrameworkImp.zip que conté la implementació de les classes `DbContextISW`, `EntityFrameworkDAL` i `IDAL`.
- La implementació de les classes `IDAL` i `EntityFrameworkDAL` és genèrica i es pot reutilitzar en el projecte. La classe `DbContextISW` es necessària per a implementar la classe Unitat de Treball del projecte: `TarongISWDbContext`.
- Haurà de crear una estructura com la de la Figura 4, agregant la carpeta `EntityFrameworkImp` en `Persistence` i els arxius `Dbcontext.cs`, `Entityframeworkdal.cs` i `Idal.cs`.
- Per a afegir un arxiu al projecte, premar amb el botó dret del ratolí sobre la carpeta on vulga incloure'l i seleccionar Agregar > Element Existent …. A continuació, seleccione l'arxiu o arxius .cs que vulga afegir.
- La unitat de treball, classe `TarongISWDbContext`, s'implementarà estenent la classe `DbContextISW`, que al seu torn, estén la classe `DbContext` d'EF, que proporciona un mètode anomenat `RemoveAllData()` que implementa l'esborrat de la base de dades.
- Per a implementar la classe `TarongISWDbContext`, fixe's en la implementació de `VehicleRentalDbContext` i faça-la de forma similar, definint una propietat `IDbSet` per a cadascuna de les classes persistents del model. Per exemple:

```
public IDbSet<Person> People { get; set; }
```

Com les classes del model estan definides en un `namespace` diferent (`TarongISW.Entities`) haurà d'incloure la corresponent directiva using TarongISW.Entities.

- Haurà d'assignar totes les classes creades al mateix `namespace`, es a dir, TarongISW.Persistence.

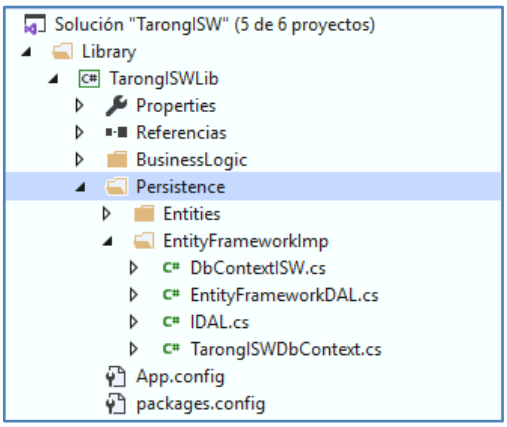


*Figura 4. Estructura de la capa de persistència en el projecte*

**Tasca 2:** Implementar la capa de persistència per al projecte, d'acord amb les consideracions anteriors i tenint com a referencia el projecte exemple.

## 4. Utilització de *Data Annotations*

En ocasions, es necessari modificar el codi de la capa lògica fent ús de les **Data Annotations** per a proporcionar certa informació a EF. En el seminari del tema 6 de teoria sobre EF es donen algunes pautes en relació a l'ús de les anotacions. Concretament, l'equip ha de prestar atenció a les anotacions:

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

**Tasca 3:** Revisa el codi de la capa lògica i inclou les anotacions EF que considereu necessàries.

## 5. Modificació dels constructores

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 ho fa EF) però sí que cal 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 persistixca per primera vegada. Els constructors ja es varen dissenyar en la sessió anterior tenint en compte este aspecte. Únicament els ID de tipo `string` deuran ser inicialitzats en el constructor. El ID de tipus numèric no s'inicialitzaran explícitament en el constructor, i per lo tant, el seu valor no es passa com argument.

**Tasca 4:** Revisar les classes de la lògica i comprovar que els constructores estan correctament implementats segons s'indica en el paràgraf anterior.

## 6. Prova de la capa de persistència (projecte de consola)

 Una vegada finalitzat el desenvolupament de la capa de persistència, l'equip deu protegix la seua solució amb un comentari com a "Capa Persistència Finalitzada (versió beta)".

És el moment de provar la capa de persistència. Per a això es pot crear un projecte de consola i escriure un programa per a crear alguns objectes de la lògica de negoci i persitir-los gastant el serveis proporcionats per el DAL. El passos a seguir serien:

- Crear un projecte de consola dins de ***Testing*** anomenat DBTest.

- Incloure una referència a la seua biblioteca de classes (TarongISWLib). Per a això, en el seu projecte d'aplicació de consola (DBTest): Botó dret del ratolí sobre Referencias > Agregar referencia … i seleccionar dins de Proyectos el seu projecte de biblioteca de classes.

- Incloure el paquet Entity Framework gastant el gestor de paquets NuGet, de forma similar a com ja s'explicar en l'apartat 2

• Modificar l'arxiu de configuració App.config que s'haurat creat en DBTest, afegint la cadena de connexió a base de dades. Si gastem SQLServer com a motor de base de dades i denominem "TarongISWDB" a la base de dades a crear, la cadena de connexió quedaria com segueix:

```
 <connectionStrings>
    <clear />
    <add name="TarongDBConnection"
connectionString="Server=(localdb)\mssqllocaldb;Database=TarongISWDB;Trusted_Connection=Tru
e;MultipleActiveResultSets=true" providerName="System.Data.SqlClient" />
  </connectionStrings>
```

**En el codi de prova (implementat en `Program.cs`) hi haurà que crear en primer lloc una instància de `EntityFrameworkDAL`, pasant-li una instància de `TarongISWDbContext`. En crear la instància de `TarongISWDbContext` per primera vegada es crearà la base de dades. A continuació, haurà de crear alguns objectes de la lògica de negoci i persistir-los usant els serveis oferits pel `DAL`.**

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. Aquest arxiu haurà d'agregar-lo al projecte `DBTest`. L'estructura resultant del projecte serà com la de la Figura 5:

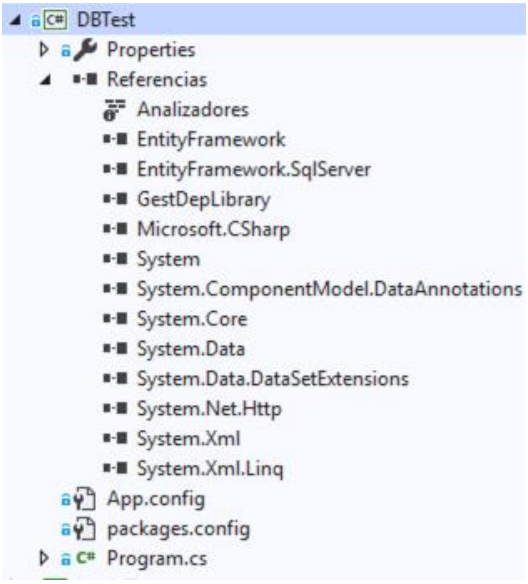


*Figura 3. Estructura del projecte de consola de Test*

Afegir aquest projecte (DBTest) a la solució i òbriga l'arxiu Program.cs. Observarà un mètode pràcticament buit denominat `createSampleDB` que caldrà emplenar amb el codi necessari per a crear els objectes de negoci i persistir-los en la base de dades.

```
private void createSampleDB(IDAL dal)
 {
     // Remove all data from DB
     dal.RemoveAllData();

     Console.WriteLine("Creando los datos y almacenándolos en la BD.");

     Person p1 = new Person("12345678Z", "Juan Abelló");
     dal.Insert<Person>(p1);
     dal.Commit();
```

```
            Parcel parcel = new Parcel("1234567AB9999C0001DE", "El Lobillo, Alhambra (Ciudad
Real)", Product.Kiwi, 10000, p1);
            dal.Insert<Parcel>(parcel);
            p1.Parcels.Add(parcel);
            dal.Commit();

           // Populate here the rest of the database with data

        }
```

Com es pot observar en el codi, després de borrar el contigut de la base de dates (dal.RemoveAllData()), es crea una instància de Person. Per a que les dades es persitixquen en la bases de dates hem d'afegix l'objecte al repositori de People, amb la instrucció `dal.Insert<Person>(p1)` i a continuació fer el `Commit` en la base de dades.

De forma similar, creem un objecte Parcel. Es aquest cas, a més hi haurà que mantindré la relació entre Person i Parcel, agregant l'objecte Parcel a la col·lecció Parcels de Person (`p1.Parcels.Add(parcel); )`

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`.

**Tasca 5:** Crear un programa de prova seguint les indicacions anteriors que persistixca en la base de dades una instància de cadascuna de les classes del model. Per a comprovar si la informació s'ha emmagatzemat en la base de dades consultar el següent apartat.

## 7. Ferramentes d'Inspecció de la BD

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 (vore 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 (SqlClient)" i com a arxiu de dades seleccionar 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\TarongISWDB.mdf`.

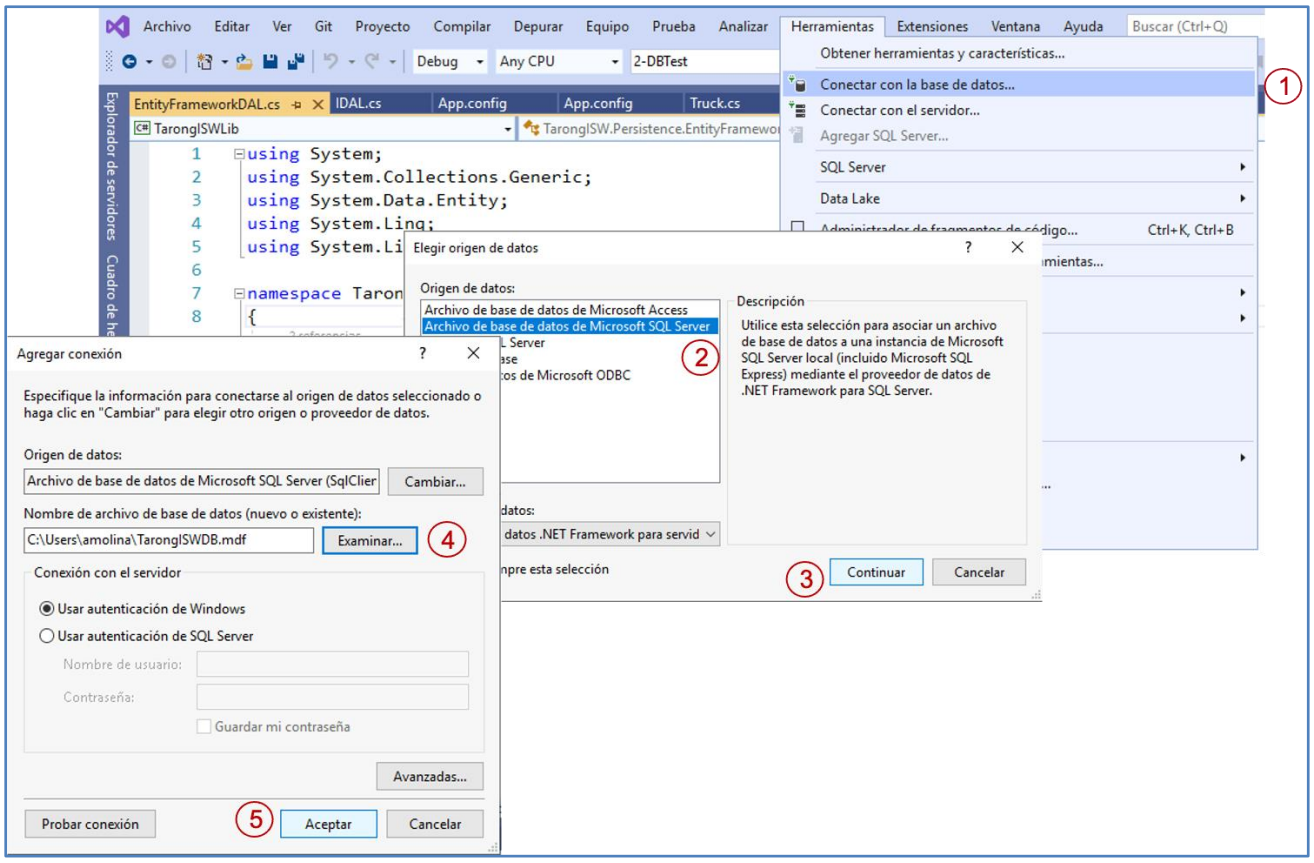


*Figura 6. Connectant amb la base de dades des de Visual Studio*

Una vegada creada la connexió és possible explorar les taules de la base de dades en l'explorador de servidors que apareixerà a l'entorn de desenvolupament, tal com es mostra a la Figura 7.

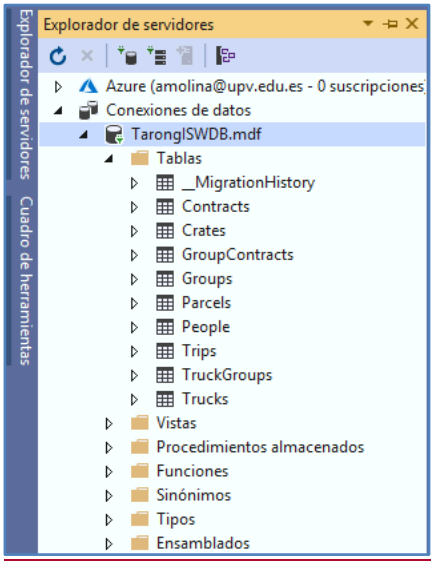


*Figura 7. Taules de la BD des de l'explorador de servidors*

Fent doble-click sobre una taula es pot veure la seua estructura. Per a veure les dades d'una taula es farà click sobre el botó dret de ratolí sobre la `tabla > Mostrar datos tabla`. Per exemple, en la taula **Parcels**, després d'executar el programa de prova i haver afegit varies parcel·les, podem veure el contingut (Figura 8).

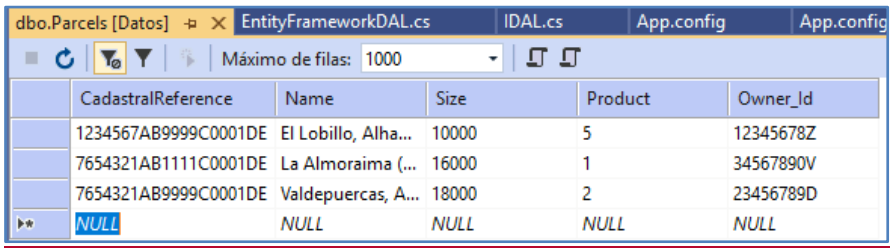


*Figura 8. Contingut de la taula Parcels*

De forma similar, des-de l'explorador de servidors podem prémer el botó dret del ratolí sobre la connexió a la BD (TarongISWDB.mdf) i triar l'opció `Nueva consulta` que 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 la nostra en una taula TTT de la base de dades, més concretament en la taula denominada TTT, escrivint "`Select` * `From` TTT" i executant la consulta (botó triangle verd "play" del SQLQuery creat). La Figura 9 mostra el resultat de consultar la tabla Parcels.

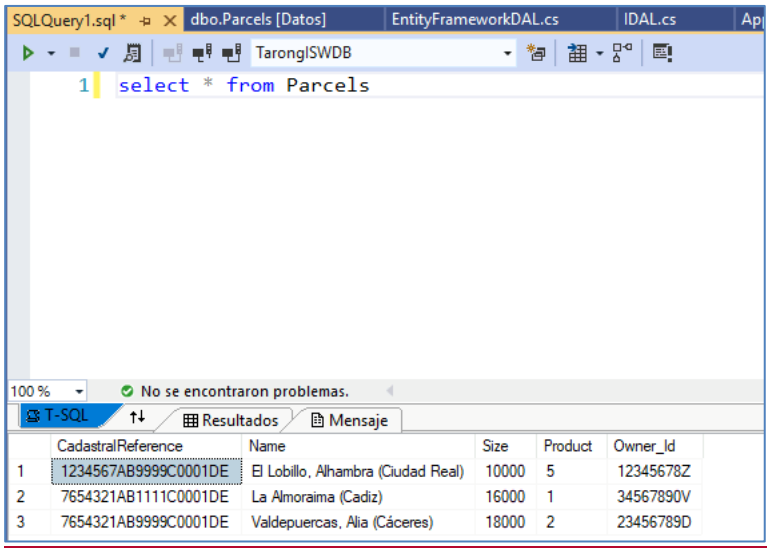


*Figura 9. Exemple de consulta a la BD*

**Tasca 6:** Comprovar en la base de dades que s'ha emmagatzemat la informació en cadascuna de les taules.

**IMPORTANT: Durant la realització d'aquestes activitats, recorde sincronitzar la seua solució quan ho considere convenient i afija comentaris informatius dels canvis introduïts. Una vegada validat que el programa funciona correctament realitze el corresponent `commit`.**

## 8. Proves unitàries (MS Test)

Les proves de codi poden sistematitzar-se millor amb un motor de proves unitàries com MSTest[1]. En Visual Studio es poden definir casos de prova per a garantir la cobertura de codi i detectar errors en el mateix. Visual Studio proporciona ferramentes per a la execució sistemàtica de les proves unitàries i la visualització del resultat de les mateixes.

**Tasca 7.** Com a última tasca abans de passar a la implementació de la lògica dels casos d'ús, el equip deurà executar les proves unitàries que indique el seu professor o professora de practiques.

[1] Informació sobre les proves unitàries en VS en:
https://docs.microsoft.com/es-es/visualstudio/test/getting-started-with-unit-testing?view=vs-2019