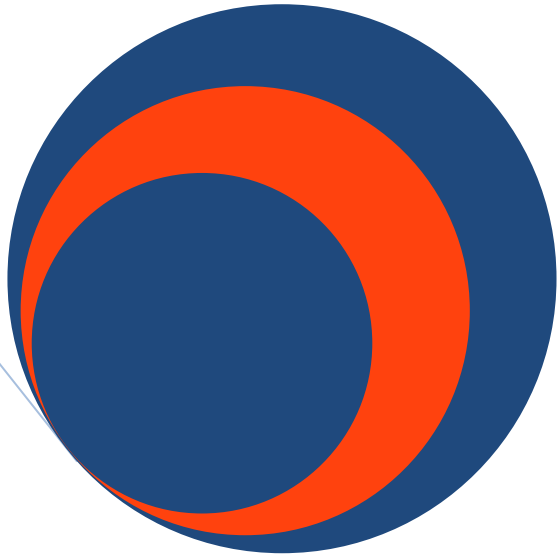

UF3 – Persistència en BDR-BDOR-BDOO

NF1 – Eines de mapatge objecte Relacional

Francisco Fernandez

Índex Mòdul 07

Persistència en BDR-BDOR-BDOO
Eines de mapatge objecte relacional (ORM)

Que és un ORM?

Les sigles ORM (ObjectRelationalMapping) és una tècnica de programació per convertir dades entre el llenguatge de programació a objectes utilitzat i el sitema de base de dades relacional utilitzat en el desenvolupament de la nostra aplicació.

El mapeig objecte-relacional ens ajudarà a oblidar-nos completament de com convertir els objectes en dades primitives per emmagatzemar-les en la base de dades i a l'inrevés.

Ulitilitzar un ORM, ens proporciona certes avantatges, però també diversos incovenients.

Avantatges.

- Velocitat de desenvolupament. La majoria d'eines actuals permeten la creació del model a partir de l'esquema de la base de dades.
- Abstracció de la base de dades. Utilitzar un ORM ens separa del sistema de base de dades que utilitzem i si en un futur hem de canviar de motor de base de dades, això no afectarà al nostre sistema.
- Seguretat. Els ORM implementen sistemes per evitar atacs com SQL injections.
- Manteniment de codi. Ens facilita el manteniment de codi degut a la correcta ordenació de la capa de dades, fent que manteniment del codi sigui més senzill.
- Llenguatge propi per realitzar les consultes. Aquest llenguatges de mapeig porten un llenguatge propi per realitzar les consultes, els usuaris ha de deixar d'utilitzar SQL per utilitzar el llenguatge propi de cada eina.

Inconvenients.

- Temps utilitzat en l'aprenentatge. Aquestes eines són complexes i es triga en tenir coneixement de l'eina.
- Aplicacions una mica més lentes. Això es degut a que totes les consultes que es facin a la base de dades, el sistema primer haurà de transformar al llenguatge propi de l'eina, després llegir el registres i per últim crear els objectes.

Els ORM més utilitzat actualment són Hibernate per Java (NHibernate per .NET), EntityFramework per .NET, Doctrine per PHP, Propel per PHP, ...

EntitityFramework, primera visió

EntityFramework és un conjunt de tecnologies dʼADO.NET que permeten el desenvolupament dʼaplicacions de software orientades a dades. Permet treballar als programadors amb dades en forma dʼobjectes i propietats, sense tenir que pensar en les taules de les bases de dades que hi ha per sota. Donat que EntityFramework és un component de .NET Framework, les aplicacions dʼEntityFramework es poden executar en qualsevol equip que tingui instal·lat el .NET Framework a partir de la versió 3.5 SP1.

EntityFramework admet Entity Data Model (EDM) per definir dades en el nivell conceptual. Quan sʼutilitza ADO.NET Entity Data Model Designer la informació sobre lʼassignació, del model conceptual i lʼemmagatzemament es guarda en un arxiu .edmx.

Consultant el model dʼentitats de dades

Els models de dades es poden consultar amb LINQ to Entities i amb Entity SQL. També es poden utilitzar mètodes especials (alguns basats en LINQ i dʼaltres en la classe ObjecteQueryd'EntityFramework). A de quedar clar que es consulta el model no la base de dades.

Per realitzar una consulta el primer que tenim que fer es generar un context a partir del EntityContainer.

```
aulacepModel.aulacepEntities contexto = new aulacepModel.aulacepEntities();
```

Consultant amb LINQ to Entities

La sintaxi dʼuna consulta bàsica LINQ to Entities és la següent:

```
var ciclos =
       from c in contexto.cicles
       where c.id == 2
       select c;
```

També podem consultar amb mètodes LINQ:

```
var ciclos =
       contexto.cicles
      .Where(c => c.id == 2);
```

Projeccions en les consultes

Si no volem retornar tota una entitat realitzem una projecció. Per exemple

```
var ciclos = from c in contexto.cicles
             where c.id == 2
             select new {c.codi, c.descripcio};
```

Si ho fem amb mètodes LINQ.

```
var ciclos = contexto.cicles
             .Where(c => c.id == 2)
             .Select(c =>new {c.codi, c.descripcio});
```

Utilitzant navegació en les consultes

Un dels beneficis d'utilitzar EDM per realitzar consultes és que les relacions es creen dintre del model i normalment no fartemjoins per accedir a les dades relacionades.

Navegant a una EntityReference

Una manera d'accedir a les dades relacionades es fer una projecció de les dades relacionades. Per exemple:

```
var ciclos = from c in contexto.cicles
             where c.id == 2
             select new {c, c.cursos};
```

També podem fer projecció de camps de la entitat relacionada, sempre i quan l'entitat relacionada no sigui una col·lecció. Per exemple:

```
var cursos = from c in contexto.cursos
             where c.id == 40
             select new {c, c.cicles.codi, c.cicles.descripcio};
```

Si utilitzem mètodes LINQ per consultar:

```
var cursos = contexto.cursos
             .Where(c => c.id == 40)
             .Select(c =>new {c, c.cicles.codi, c.cicles.descripcio});
```

Filtrant i ordenant amb una EntityReference

Es poden fer filtres i ordenar basant-se en propietats d'un EntityReference, s'hagin seleccionat o no. Per exemple:

```
Var cursos = from c in contexto.cursos
             where c.cicles.codi == "ASI"
             select c;
```

Navegant per col·leccions d'entitats

A més de navegar per una entitat podem navegar per una col·lecció d'entitats. Per exemple:

```
var ciclos = from c in contexto.cicles
             where c.codi == "ASI"
             select new {c, c.cursos};
```

Projectant propietats des de entitats de EntityCollection

Podem seleccionar propietats particulars de la EntityCollection.

```
var ciclos = from c in contexto.cicles
             wherec.codi == "ASI"
             select new {
             c,
             cursos_light = from cu inc.cursos
             select new {cu.codi, cu.descripcio}
             };
```

Filtrant amb EntityCollections

Per filtrar amb una col·lecció d'entitats podem utilitzar el mètode ANY de LINQ to Entities. Per exemple:

```
var ciclos = from c in contexto.cicles
             where c.cursos.Any(cu =>cu.codi.Contains("ASI"))
             select c;
```

Funcions d'agregació amb EntityCollections

Els mètodes que hi ha són Count, Sum, Average, Min i Max.

```
Var ciclos = from c in contexto.cicles
             select new { c, c.cursos.Count };


var cursos = from c in contexto.cursos
             select new { c.codi, MaxHores = c.assignatures.Max(a =>a.hores) };
```

Funcions d'agregació en mètodes LINQ

Els agregats de LINQ són mètodes, no operadors de consulta. Per exemple:

```
var cursos = contexto.cursos
             .Select((c) =>new { c.codi, MaxHores = c.assignatures.Max(a =>a.hores) });
```

Agrupació

LINQ to Entities té un operador Group. A més té la clàusula opcional INTO amb la que poden especificar un nom a un grup. No es una propietat, però amb aquest nom podem aplicar funcions al grup. Per exemple:

```
Var cursos = from c in contexto.cursos
             group c by c.id_cicle into cicle
             select new { cicle.Key, cicle, Max = cicle.Count()};
```

Filtrant amb condicions de grup

Si volem filtrar per una propietat del grup amb LINQ utilitzarem l'operador WHERE. Per exemple:

```
Var cursos = from c in contexto.cursos
             group c by c.id_cicle into cicle
             where cicle.Count() > 2
             select new { cicle.Key, cicle, max = cicle.Count()};
```

Carregant informació relacionada

Fins ara, totes les consultes que volien retornar dades relacionades han de preguntar explícitament per aquestes dades en la consulta. Però també les tenim sense preguntar explícitament per elles. Les dades relacionades no són retornades per la consulta original, si no que cada vegada que utilitzem les dades relacionades al codi és realitza una consulta al servidor de base de dades per obtenir les dades relacionades que es necessitin. Per exemple:

```
int[] num_cursos;
int i;

var ciclos = from c in contexto.cicles
             select c;
i = 0;
foreach (var ciclo in ciclos)
{
       num_cursos[i] = ciclo.cursos.Count();
       i++;
}
```

En l'exemple anterior, fins que no accedim a cursos no s'obtenen de la base de dades. Això és conegut com càrrega de dades peresosa.

Utilitzant el mètode Include per la carrega de dades ansiosa.

En els cassos en els que necessitem tota la informació relacionada, és més eficient que l'obtenció d'aquestes dades formi part de la consulta original. Això es pot obtenir fent una projecció que inclogui les dades relacionades a la clàusula SELECT, el mètode Include és un altre manera de fer el mateix però que es preferible per diverses raons. La raó més important és que el objectes resultants poden ser les nostres pròpies entitats i no tipus anònims. Un exemple d'utilització:

```
var ciclos = from c in contexto.cicles.Include("cursos")
             where c.codi.StartsWith("A")
             select c;
```

Crear i afegir objectes

Si volem inserir dades a l'origen de dades, hem de crear una instància d'un tipus d'entitat i afegir l'objecte al context de l'objecte. Per poder guardar un objecte nou a l'origen de dades, primer hem d'establir totes les propietats que no admeten valors null. Quan treballem amb classes generades per Entity Framework, també es pot treballar amb el mètode estàtic **Create**NomObjecte del tipus d'entitat per crear una nova instància d'un tipus d'entitat.

Per agregar els nous objectes a un context d'objectes utilitzant el mètode AddObject sobre ObjectSet.

```
var ciclo = new aulacepModel.cicles();

ciclo.id = int.Parse(TextBoxIdentificador.Text);
ciclo.codi = TextBoxCodi.Text;
ciclo.descripcio = TextBoxDescripcio.Text;

contexto.cicles.AddObject(ciclo);

contexto.SaveChanges();
```

Eliminar objectes

Si cridem el mètode DeleteObject sobre ObjectSet, es marca l'objecte especificat per a la seva eliminació. La fila no s'elimina de l'origen de dades fins que no cridem a SaveChanges. El comportament d'eliminar objectes difereix en Entity Framework depenent del tipus de relació al que pertany l'objecte.

En una relació on la clau principal de l'entitat principal forma part de la clau principal de l'entitat depenent, a l'eliminar un objecte es poden eliminar també els objectes relacionats. Els objectes relacionats no poden existir sense una relació definida amb l'objecte primari. Al eliminar l'objecte primari, també s'eliminaran tots els objectes secundaris.

En una relació on la clau externa accepta valors nuls, s'estableixen a null els objectes depenents quan s'elimina un objecte princiapal.

Per exemple

```
var ciclos = from c in contexto.cicles
             where c.codi == TextBoxCodi.Text
             select c;

 var ciclo = ciclos.First();

 contexto.cicles.DeleteObject(ciclo);

 contexto.SaveChanges();
```

Modificar objectes

Al modificar els objectes s'han de tenir en compte:

- Quan es canvia el valor de qualsevol propietat d'un objecte complex, l'estat de l'objecte entitat de per sobre canvia a modificat.
- Al canviar una associació de clau externa, l'estat de l'objecte depenent canvia a modificat.

Per exemple:

```
var ciclos = from c in contexto.cicles
             where c.codi == TextBoxCodi.Text
             select c;

var ciclo = ciclos.First();

ciclo.descripcio = "Administració de Sistemes Informàtics";

contexto.SaveChanges();
```

Data Binding amb aplicacions ASP .NET (EntityDataSource)

Entenent com amb EntityDataSource s'obtenen i es modifiquen les dades

En temps d'execució quan EntityDataSource necessita omplir-se, comença llegint les propietats EntityConnectionString, EntityContainer, i EntitySet que han estat definides per nosaltres. Llavors crea un nou ObjectContext utilitzant el nom d'EntityConnectionString, i un ObjectQuery utilitzant EntitySet. Si volem triar propietats individuals de l'entitat, hem de crear un stringEntitySQL utilitzant el nom de l'EntityContainer i els noms de les propietats seleccionades. La consulta es crearà automàticament basant-se en les propietats del control EntityDataSource.

L'assistent configura només les propietats més elementals de EntityDataSource, però el control té més propietats, i alguna d'aquestes serveixen per definir la consulta:

Per exemple:

Include, Where, OrderBy, ...

EntityDataSource i el seu ObjectContext

Per defecte cada EntityDataSource crea el seu propi ObjectContext. Si tenim més d'un EntityDataSource, tenim molts ObjectContext i diverses connexions a la base de dades.Cada vegada que una pàgina es recarrega, el seus contextos es creen i s'esborren.

EntityDataSource té un esdeveniment anomenat ContextCreating, on podem forçar el context del EntityDataSource.

Utilitzar un sol context té les seves avantatges. Per exemple les relacions entre entitats no són reconegudes a no ser que el mateix context gestioni les entitats. Per crear un context:

```
aulacepModel.aulacepEntities contexto = new aulacepModel.aulacepEntities();
```

Per utilitzar el context creat a un EntityDataSource en concret hem de programar l'esdeveniment ContextCreating:

```
   Protected void EntityDataSourceCiclos_ContextCreating(object sender,
   EntityDataSourceContextCreatingEventArgs e)
{
e.Context = contexto;
    }
```

Esdeveniments del context d'EntityDataSource

ContextCreating

e.Context. Marca el context i és on podem midificar-lo si no volem el que es crea per defecte.

ContextCreated

Retorna el context de l'EntityDataSource.

ContextDisposing

e.Context. Retorna el context.

e.Cancel. Podem detenir el context.

EntityDataSource i el ViewState

El control EntityDataSource no només manté els valors actuals de les dades, si no , també els valors originals.

Tot això es guarda en el ControlState. Aquest valors es mantenen quan recarreguem les pàgines.

Accedint a les claus foranes quan no està la propietat de ForeignKey.

Quan les claus foranes s'utilitzen en el model, les propietats són incloses en el control EntityDataSource, però les propietats de navegació no.

Per poder utilitzar les associacions amb EntityFramework, hem de tenir la propietat EneableFlat del control EntityDataSource a cert.

Treballant amb dades relacionades EntityReference

Si volem accedir a les dades relacionades a la nostra entitat, es pot fer amb un sol EntityDataSource .

Utilitzant EntityDataSource.Include per obtenir dades relacionades

Podem carregar les dades relacionades utilitzant la propietat Include del control EntityDataSource. Això afegirà un mètode Include en la consulta de resultats.

Include és molt pràctic per mostrar dades de només lectura. Per fer modificacions és millor un control EntityDataSource addicional.

Mostrant dades que venen des de propietats de navegació.

Els passos que hem de seguir són:

1. Canviar la propietat Include d'EntityDataSource.
2. Seleccionar el control on estem mostrant les dades com per exemple una GridView, obrirem les tasques i seleccionarem Editar columnas ...
3. En els BoudField farem doble-clic sobre el camp de navegació que volem possar com camp seleccionat.
4. Moure el camp a la posició de la grid on volem que aparegui.
5. Editar la propietat DataField del nou camp i posem el camp de la entitat relacionada que volem que surti.
6. Posem el camp de només lectura i el convertim en un camp de plantilla.

Per poder ordenar també hem de canviar la propietat SortExpression i modificar-la amb el mateix valor del DataField.

Utilitzar un nou control EntityDataSource per habilitar l'edició de propietats de navegació

Necessitem 2 elements, un nou EntityDataSource i una DropDownList. Hem de realitzar canvis molt semblants a com es fan amb el desconnectat:

1. Afegim un nou EntityDataSource.
2. Tornem a generar el projecte per a que el nou EntityDataSource reconegui el model.
3. Configurem l'origen de dades del nou EntityDataSource.
4. En el EditTemplate de la propietat de navegació col·locarem un DropDownList que tindrà com DataSource el EntityDataSource nou. Hem de definir DataTextField amb el nom de la propietat que volem mostrar i DataValueField amb el nom de la propietat que volem que sigui el valor seleccionat.