




ADO.NET Entity Framework

Integración de Sistemas

Parte II. Diseño e implementación de
aplicaciones Web con .NET



Contenido

- Introducción
 - ¿Por qué el Entity Framework?
 - ¿Qué es el ADO.NET Entity Framework?
- Arquitectura y componentes
 - Proveedores específicos de EF
 - Entity Data Model (EDM)
 - Entity Client (eSQL)
 - Object Services (eSQL, Linq to Entities)
- Implementación de DAOs con Entity Framework

Introducción

¿Por qué el Entity Framework?

- Desajuste de impedancias (*impedance mismatch*)
 - Diferencias entre los modelos relacionales y los modelos de objetos
- Diseños guiados por modelos de dominio (*Domain Driven Design, DDD*)
 - Proponen centrarse en el modelo conceptual o dominio de trabajo para resolver el problema
 - Dan prioridad al problema a resolver!
- Patrones en DDD
 - VO, Lazy Loading, Data Mapper, Unit of Work

Introducción

¿Por qué el Entity Framework?

- Ignorancia de la persistencia (*Persistence Ignorance*)
- Propugna el trabajo con objetos VO que para nada tengan que saber sobre el almacenamiento subyacente
 - Construcción de objetos POCO (*Plain Old CLR Objects*)
 - Relajación: IPOCO
- EF soportará objetos POCO en la segunda versión. En la primera versión podemos trabajar con objetos IPOCO

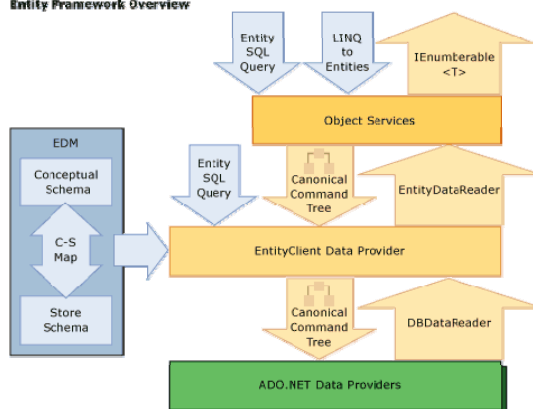
Introducción

¿Qué es el ADO.NET Entity Framework?

- Es un OR/M (Object Relational / Mapping) para .NET
- Incluido en .NET Framework 3.5 SP1 y en VS2008 SP1 (11 de agosto de 2008)
 - Incluye un nuevo proveedor de ADO.NET, llamado **Entity Client**, que habilita el acceso a los modelos conceptuales
- Incluye dos componentes fundamentales:
 - Recursos para el entorno de trabajo:
 - Asistente para diseño en VS y generación de código
 - Librería:
 - Físicamente, en el ensamblado System.Data.Entity.dll
 - Sus espacios de nombres se anidan en **System.Data** (**System.Data.Common**, **System.Data.EntityClient**, **System.Data.Mapping**, **System.Data.Metadata.Edm**, etc.)

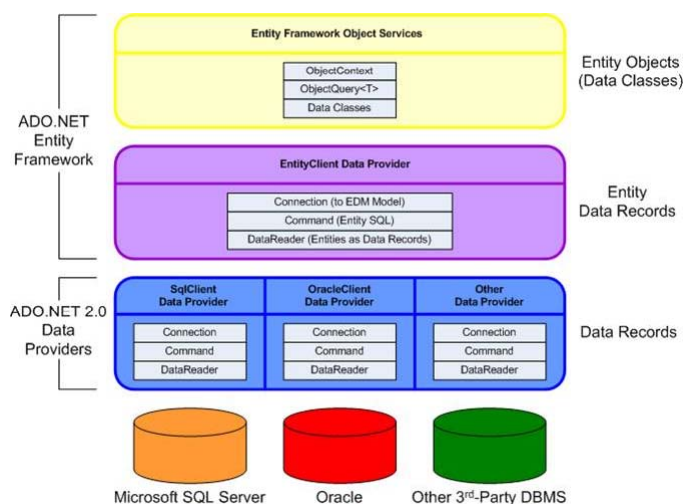
Arquitectura y componentes

Entity Framework Overview



Componentes de la arquitectura de EF

Arquitectura y componentes



Arquitectura y componentes

Proveedores específicos de EF

Fabricante	Provider para
Devart (CoreLab)	Oracle, MySQL, PostgreSQL, SQLite
IBM	DB2, Informix Dynamic Server, U2
MySQL AB	MySQL
Npgsql	PostgreSQL 7.3+ y 8.x
OpenLink	Oracle, Ingres, Informix, Sybase, MySQL, PostgreSQL, DB2, Progress, SQL Server, (cq. Datasource OpenLink ODBC o bridge JDBC)
Phoenix	SQLite Database
Sybase	SQL Anywhere
Vista DB	VistaDB databases
Datadirect Tech.	Oracle, Sybase, SQL Server, DB2...
Firebird	Firebird databases

Arquitectura y componentes

Entity Data Model (EDM)

- El modelo de datos basado en entidades permite
 - Definir los conjuntos de entidades y relaciones entre las entidades de nuestros modelos conceptuales
 - Especificar cómo estos tipos se mapearán a la estructura de la fuente de almacenamiento relacional subyacente
- Para apoyar al EDM, se dispone de una serie de herramientas integradas dentro del entorno
 1. Diseñador de modelos EDM (Entity Data Model Designer)
 2. Asistente de modelos de entidades (Entity Data Model Wizard)
 3. Asistente de actualización de modelos

Arquitectura y componentes > Entity Data Model (EDM)

1. Diseñador de modelos EDM (Entity Data Model Designer)

- Herramienta visual integrada dentro de VS 2008 que permite crear y editar modelos conceptuales
- Componentes:
 - Superficie de diseño: crear y editar modelos
 - Detalles de mapeo: ver y editar mapeos
 - Navegación por el modelo: ver árboles de información sobre el modelo conceptual y el modelo físico
 - Nuevos elementos dentro de la ventana de herramientas

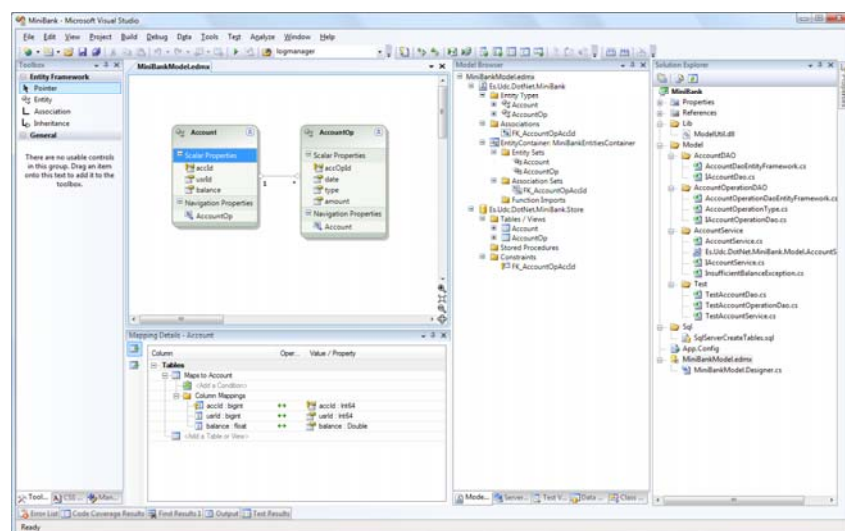
Arquitectura y componentes > Entity Data Model (EDM)

1. Diseñador de modelos EDM (Entity Data Model Designer)

- El EDM designer opera sobre ficheros edmx. Estos ficheros (XML) están formados por tres secciones:
 - SSDL (Storage Schema Definition Language): estructura física de la BD
 - CSDL (Conceptual Schema Definition Language): entidades del modelo conceptual
 - MSL (Mapping Schema Language): también conocida como sección C-S, especifica cómo se relacionan las entidades del modelo conceptual con las tablas, columnas, etc. del modelo físico

Arquitectura y componentes > Entity Data Model (EDM)

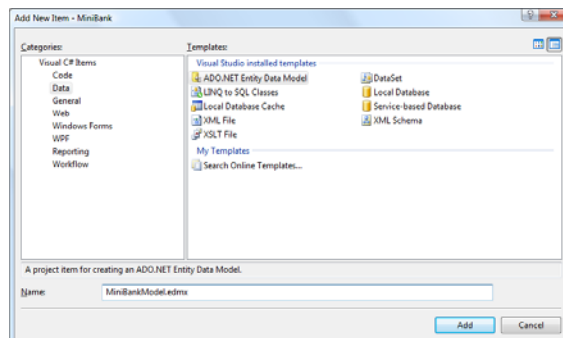
1. Diseñador de modelos EDM (Entity Data Model Designer)



Arquitectura y componentes > Entity Data Model (EDM)

2. Asistente de modelos de entidades (Entity Data Model Wizard)

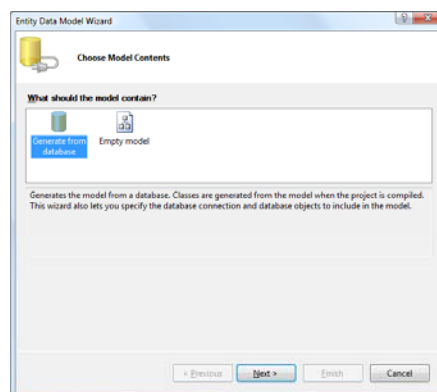
- Encargado de generar el archivo edmx
- Permite crear el modelo a partir de una BD ya existente o generar un modelo vacío



Agregando un modelo de EDM

Arquitectura y componentes > Entity Data Model (EDM)

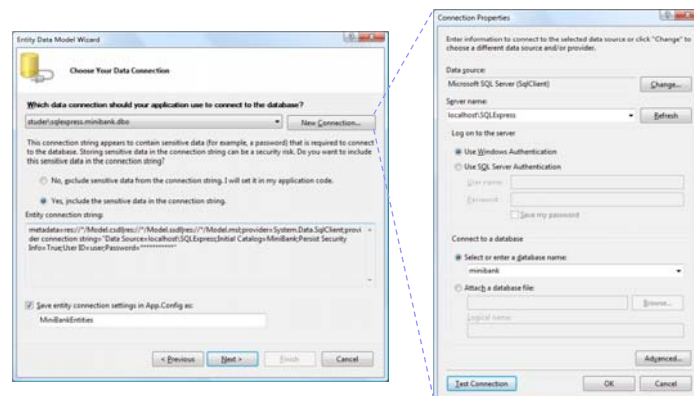
2. Asistente de modelos de entidades (Entity Data Model Wizard)



Asistente de creación de EDM (1)

Arquitectura y componentes > Entity Data Model (EDM)

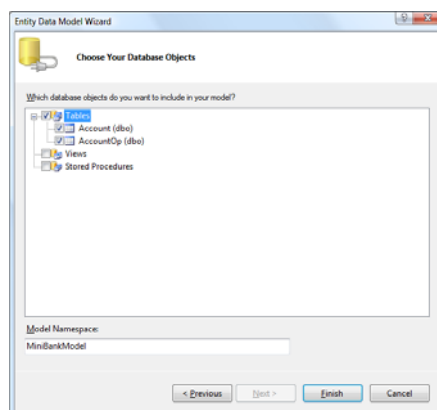
2. Asistente de modelos de entidades (Entity Data Model Wizard)



Asistente de creación de EDM (2)

Arquitectura y componentes > Entity Data Model (EDM)

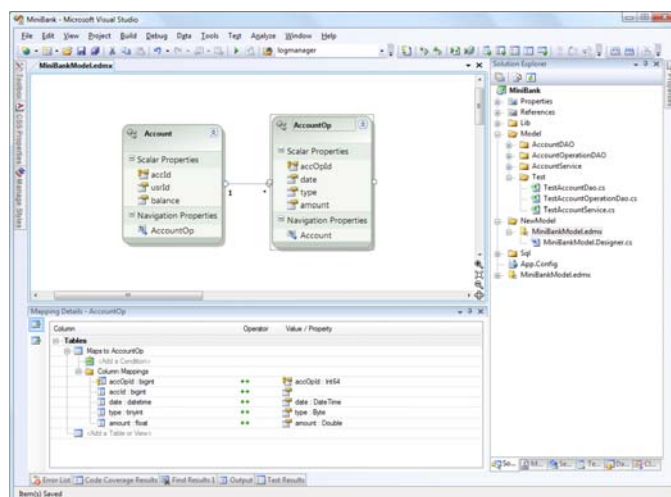
2. Asistente de modelos de entidades (Entity Data Model Wizard)



Asistente de creación de EDM (3)

Arquitectura y componentes > Entity Data Model (EDM)

2. Asistente de modelos de entidades (Entity Data Model Wizard)



Vista gráfica del archivo edmx

Arquitectura y componentes > Entity Data Model (EDM)

2. Asistente de modelos de entidades (Entity Data Model Wizard)

```

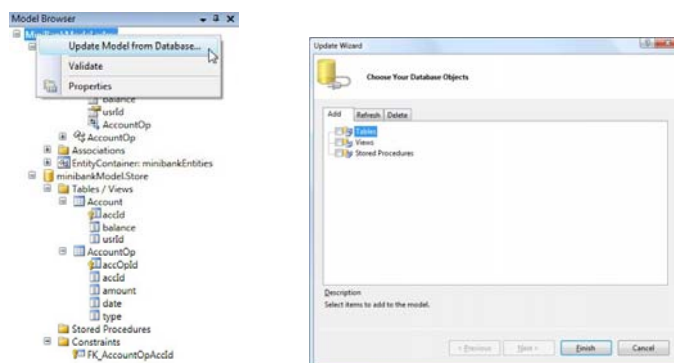
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="4.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- EDM content -->
    <edmx:StorageModel>
      <!-- EDM content -->
    </edmx:StorageModel>
    <!-- EDM content -->
    <edmx:ConceptualModel>
      <!-- EDM content -->
    </edmx:ConceptualModel>
    <!-- EDM content -->
    <edmx:MappingModel>
      <!-- EDM content -->
    </edmx:MappingModel>
  </edmx:Runtime>
  <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <edmx:Designer>
    <!-- EDM content -->
  </edmx:Designer>
</edmx:Edmx>
  
```

Vista XML del archivo edmx

Arquitectura y componentes > Entity Data Model (EDM)

3. Asistente de actualización de modelos

- Permite actualizar el modelo EDM después de que se hayan realizado cambios en la BD



Arquitectura y componentes > Entity Data Model (EDM)

Herencia

- El Modelo Relacional no soporta directamente el concepto de herencia
 - La herencia se representa comúnmente en una base de datos de una de las tres siguientes formas:
 - Mapeando una jerarquía entera de herencia a una sola tabla
 - Tabla por jerarquía, *Table per Hierarchy (TPH)*
 - Mapeando cada tipo en una jerarquía de herencia a una tabla diferente
 - Tabla por tipo concreto, *Table per Type (TPT)*
 - Mediante una aproximación híbrida donde la información común está en una tabla única y existen tablas adicionales que contienen las columnas añadidas para cada tipo derivado
 - Tabla por subclase, *Table per Subclass (TPS)*
- El EF soporta mapeado a cualquiera de estos tres modelos de herencia

Arquitectura y componentes

Entity Client

- Es un nuevo proveedor de ADO.NET
- En lugar de trabajar con modelos físicos trabaja con modelos EDM
 - Es agnóstico con respecto a la BD subyacente
 - Implementa una arquitectura abierta y es capaz de trabajar con diferentes proveedores de EF específicos
 - Estos proveedores se encargan de traducir las consultas sobre el modelo en consultas en el dialecto específico de la BD subyacente, así como de la ejecución de dichas consultas y la recuperación de los resultados
 - El lenguaje utilizado para consultar los modelos de EDM se llama **Entity SQL (eSQL)**
 - Es una variante de los dialectos de SQL
 - Mejora ciertos aspectos de las consultas, como la navegación entre tablas

Arquitectura y componentes > Entity Client

Entity SQL

- Ej: consultar el número de cuentas

```
String connectionString = ConfigurationManager.  
    ConnectionStrings["MiniBankEntities"].ToString();  
Int64 userId = 1234;  
  
using (EntityConnection connection =  
    new EntityConnection(connectionString))  
{  
    connection.Open();  
    EntityCommand command = connection.CreateCommand();  
  
    //Entity SQL does not support the count(*) aggregate. Use count(0) instead.  
    command.CommandText =  
        "SELECT count(0) " +  
        "FROM MiniBankEntities.Account as t " +  
        "WHERE t.userId = @userId";  
    command.CommandType = CommandType.Text;
```

Arquitectura y componentes > Entity Client

Entity SQL

- Ej: consultar el número de cuentas (cont.)

```
EntityParameter usrIdParameter = command.CreateParameter();
usrIdParameter.ParameterName = "usrId";
usrIdParameter.DbType = DbType.Int64;
usrIdParameter.Value = userId;
command.Parameters.Add(usrIdParameter);

/* It caches the query plan of the SQL dialect generated */
command.EnablePlanCaching = true;

int numberOfAccounts = (int)command.ExecuteScalar();

connection.Close();

Console.WriteLine("Number of Accounts: " + numberOfAccounts);
}
```

Arquitectura y componentes > Entity Client

Entity SQL

- Es posible consultar el SQL generado

```
String generatedSQL = command.ToTraceString();
```

- Para el ejemplo anterior

```
SELECT
1 AS [C1],
[GroupBy1].[A1] AS [C2]
FROM ( SELECT
        COUNT(0) AS [A1]
        FROM [dbo].[Account] AS [Extent1]
        WHERE [Extent1].[usrId] = @usrId
    ) AS [GroupBy1]
```

Arquitectura y componentes


Entity Client

- Con **Entity Client** y **eSQL** podemos cubrir la mayoría de las necesidades de una capa de acceso a datos
 - Podemos consultar modelos conceptuales, de forma similar a cómo se consultaban BD en ADO.NET 2.0
 - Disponemos de clases equivalentes a las ya conocidas
 - EntityConnection, EntityParameter, EntityCommand, EntityDataReader
- Sin embargo, todavía tenemos que realizar una transformación de los datos recuperados a objetos del dominio (**materialización**)
- Para evitar este paso, EF ofrece una nueva capa: **Object Services**

Arquitectura y componentes

Object Services

- Conjunto de clases que permiten consultar y obtener resultados en términos de objetos
 - Se reduce la cantidad y la complejidad del código
- Las consultas pueden realizarse
 - Entity SQL (eSQL)
 - Indep. del SGBD
 - Strings (Interpretados en tiempo de ejecución)
 - Linq (Language Integrated Queries) to Entities
 - Lenguaje común y semántico
 - Interpretado en tiempo de compilación
- Las consultas realizadas en Entity SQL y Linq-to-Entities son convertidas internamente a **Canonical Query Tree**, que se convierte a su vez en una pregunta entendible por el almacén de datos subyacente (e.g., en SQL en el caso de una BD relacional)
- Permite seguir los cambios en los **entity objects** y gestionar las relaciones entre ellos



Arquitectura y componentes

Object Services

- Object Services
 - EntityObject
 - ObjectContext
 - ObjectStateManager



Arquitectura y componentes

Object Services

- ObjectContext
 - Permite trabajar con el modelo conceptual
 - Consultas: ObjectQuery;
 - Inserciones: .AddToXXX(XXX entity); .AddObject(...),
 - Borrado: .DeleteObject
 - Persistencia: .SaveChanges();
 - Gestión de la conexión
 - Almacén en memoria de objetos
 - Tracking de estado objetos:
 - .Attach(..), .Detach(..)
 - ObjectStateManager
 - MergeOption

Arquitectura y componentes

Object Services

■ ObjectStateManager

- Seguimiento del estado de entidades
- Gestiona entradas EntityStateEntry para cada Entidad en almacén en memoria
 - Cuando se cargan (Query, Attach): **Unchanged**
 - Cuando se crean (AddObject): **Added**
 - Cuando se modifican: **Changed**
 - Cuando se borran: **Deleted**
 - Cuando se destruye elObjectContext: **Detached**
 - Al aplicar ObjectContext.SaveChanges() en Added, Changed, cambia a Unchanged

Arquitectura y componentes

Object Services. Consultas. Entity SQL

■ Ej: consultar el número de cuentas

```
using (MiniBankEntities context = new MiniBankEntities())
{
    String query = "SELECT VALUE account " +
                  "FROM Account " +
                  "WHERE account.usrId = @userId";

    ObjectParameter param = new ObjectParameter("userId", userId);

    int result =
        context.CreateQuery<Account>(query, param).Count();

    Console.WriteLine(result);
}
```

Arquitectura y componentes

Object Services. Consultas. Entity SQL

- Ej: recuperar las cuentas de un usuario (implementando Page-by-Page)

```
using (MiniBankEntities context = new MiniBankEntities())
{
    String query = "SELECT value account " +
                   "FROM Account " +
                   "WHERE account.usrId = @usrId " +
                   "ORDER BY account.accId";

    ObjectParameter param = new ObjectParameter("usrId", usrId);

    List<Account> accounts =
        context.CreateQuery<Account>(query, param).
            Execute(MergeOption.NoTracking).Skip(startIndex).
            Take(count).ToList();

    foreach (Account a in accounts)
    {
        Console.WriteLine(a.accId + ", " + a.balance);
    }
}
```

Arquitectura y componentes

Object Services. Consultas. LINQ-to-Entities

- Ej: consultar el número de cuentas

```
using (MiniBankEntities context = new MiniBankEntities())
{
    int result =
        (from acc in context.Account
         where acc.usrId == usrId
         select acc).Count();

    Console.WriteLine(result);
}
```


Arquitectura y componentes

Object Services. Consultas. LINQ-to-Entities

- Ej: recuperar las cuentas de un usuario (implementando Page-by-Page)

```
using (MiniBankEntities context = new MiniBankEntities())
{
    List<Account> accounts =
        (from a in context.Account
         where a.userId == userId
         orderby a.accId
         select a).Skip(startIndex).Take(count).ToList();

    foreach (Account a in accounts)
    {
        Console.WriteLine(a.accId + ", " + a.balance);
    }
}
```

Implementación de DAOs con Entity Framework

- Siguiendo el enfoque visto en la primera parte de la asignatura, se ha diseñado un **DAO genérico** con las operaciones comunes a todas las clases persistentes:
 - Create, Find, Exists, Update, Remove
- Cada entidad persistente tendrá su propio DAO, que extenderá el genérico para añadir operaciones propias
- El DAO genérico se encuentra en el proyecto ModelUtil, utilizado por MiniBank y MiniPortal como .dll
- La interfaz parametrizada del DAO genérico recibe 2 argumentos:
 - E, es la clase persistente para la que se implementará el DAO
 - PK, define el tipo del identificador de la clase persistente
- Los métodos están definidos en base a esos parámetros y no están acoplados a ninguna tecnología de persistencia

Implementación de DAOs con Entity Framework

- Interfaz del DAO genérico

```
public interface IGenericDao<E, PK>
{
    void Create(E entity);

    /// <exception cref="InstanceNotFoundException"></exception>
    E Find(PK id);

    Boolean Exists(PK id);

    E Update(E entity);

    /// <exception cref="InstanceNotFoundException"></exception>
    void Remove(PK id);
}
```

Implementación de DAOs con Entity Framework

- Implementación del DAO genérico con Entity Framework

```
public class GenericDaoEntityFramework<E, PK> :
    IGenericDao<E, PK>
    where E : IEntityWithKey
{
    // entityClass is set in the constructor of this class
    private Type entityClass;

    // context must be set by means of Context property
    private ObjectContext context;

    private String entityContainerName;

    public GenericDaoEntityFramework()
    {
        this.entityClass = typeof(E);
    }
}
```

Implementación de DAOs con Entity Framework

■ Implementación del DAO genérico con Entity Framework

```
[Dependency]
public ObjectContext Context
{
    set
    {
        context = value;

        entityContainerName = (context.MetadataWorkspace.
            GetItems<EntityContainer>(DataSpace.CSpace))[0].Name;

        context.DefaultContainerName = entityContainerName;

        // Forces the load of the metadata
        context.MetadataWorkspace.LoadFromAssembly(
            entityClass.Assembly);
    }
    get
    {
        return context;
    }
}
```

Implementación de DAOs con Entity Framework

■ Implementación del DAO genérico con Entity Framework

```
public EntityKey CreateEntityKey(PK id)
{
    EntityType entityType =
        (EntityType)context.MetadataWorkspace.GetType(entityClass.Name,
            entityClass.Namespace, DataSpace.CSpace);

    /* We assume that the DAO works only with single field primary
     * key classes
     */
    String primaryKeyFieldName =
        ((EntityType)entityType).KeyMembers.First().ToString();

    // Create the entityKey
    EntityKey entityKey =
        new EntityKey(entityContainerName + "." + entityClass.Name,
            new EntityKeyMember[] {
                new EntityKeyMember(primaryKeyFieldName, id) });

    return entityKey;
}
```

Implementación de DAOs con Entity Framework

■ Implementación del DAO genérico con Entity Framework

```
public void Create(E entity)
{
    String entitySetName =
        entityContainerName + "." + entityClass.Name;
    context.AddObject(entitySetName, entity);
    context.SaveChanges();
    context.AcceptAllChanges();
}

/// <exception cref="InstanceNotFoundException"/>
public E Find(PK id)
{
    EntityKey entityKey = this.CreateEntityKey(id);
    try
    {
        E result = (E)context.GetObjectByKey(entityKey);
        return result;
    }
    catch (ObjectNotFoundException)
    {
        throw new InstanceNotFoundException(id, entityClass.FullName);
    }
}
```

Implementación de DAOs con Entity Framework

■ Implementación del DAO genérico con Entity Framework

```
public Boolean Exists(PK id)
{
    Boolean objectFound = true;
    EntityKey entityKey = this.CreateEntityKey(id);
    try
    {
        object result = context.GetObjectByKey(entityKey);
    }
    catch (ObjectNotFoundException)
    {
        objectFound = false;
    }
    return objectFound;
}

public E Update(E entity)
{
    // Last Updates are sent to database
    context.Refresh(RefreshMode.ClientWins, entity);
    context.SaveChanges();
    context.AcceptAllChanges();
    return (E)context.GetObjectByKey(entity.EntityKey);
}
```

Implementación de DAOs con Entity Framework

- Implementación del DAO genérico con Entity Framework

```
/// <exception cref="InstanceNotFoundException"/>
public void Remove(PK id)
{
    E objectToRemove = default(E);
    try
    {
        // First we need to find the object
        objectToRemove = Find(id);
        context.DeleteObject(objectToRemove);
        context.SaveChanges();
        context.AcceptAllChanges();
    }
    catch (InstanceNotFoundException)
    {
        throw;
    }
}
```

Implementación de DAOs con Entity Framework

- Implementación del DAO genérico con Entity Framework

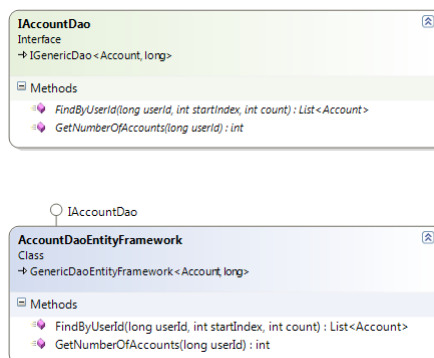
```
catch (OptimisticConcurrencyException)
{
    context.Refresh(RefreshMode.ClientWins, objectToRemove);
    context.DeleteObject(objectToRemove);
    context.SaveChanges();
    context.AcceptAllChanges();
}
catch (InvalidOperationException)
{
    throw new InstanceNotFoundException(id, entityClass.FullName);
}
}
```

Implementación de DAOs con Entity Framework

- Para implementar la persistencia utilizando Entity Framework, el DAO necesita un objetoObjectContext
 - Se asigna y recupera a través de la propiedad Context
 - Se utilizará inyección de dependencias para establecer el valor de la propiedad Context
- Para implementar las operaciones podemos elegir EntitySQL o Linq-to-Entities (hemos visto ejemplos de ambos)

Implementación de DAOs con Entity Framework

- Ej.: MiniBank > AccountDao



Implementación de DAOs con Entity Framework

- Ej.: MiniBank > AccountOperationDao

IAccountOperationDao
Interface
↳ IGenericDao<AccountOp, long>

Methods

- FindByDate(long accountId, DateTime startDate, DateTime endDate, int startIndex, int count) : List<AccountOp>
- GetNumberOfAccountOperations(long accountId, DateTime startDate, DateTime endDate) : int

IAccountOperationDao

AccountOperationDaoEntityFramework
Class
↳ GenericDaoEntityFramework<AccountOp, long>

Methods

- FindByDate(long accountId, DateTime startDate, DateTime endDate, int startIndex, int count) : List<AccountOp>
- GetNumberOfAccountOperations(long accountId, DateTime startDate, DateTime endDate) : int