



« Entity Framework 6 開發實戰 »



多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

部落格：<http://blog.miniasp.com/>



Prerequisites Knowledge of Entity Framework

學習 EF 之前的基礎知識

基礎知識

- .NET Framework
- C#
- Visual Studio
- SQL Server

What's New in the .NET Framework

- [.NET Framework 3.5](#)
- [.NET Framework 4.0](#)
- [.NET Framework 4.5](#)

泛型與 LINQ 命名空間

- [System.Collections.Generic](#)
 - 包含會定義泛型集合的介面和類別，可讓使用者建立強型別集合，提供比起非泛型強型別集合更佳型別安全和效能。
- [System.Linq](#)
 - 提供能夠支援查詢使用 LINQ 的類別和介面。
- [System.Linq.Expressions](#)
 - 包含類別、介面和列舉，可使用運算式樹狀結構格式將這些語言層級程式碼運算式表示為物件。

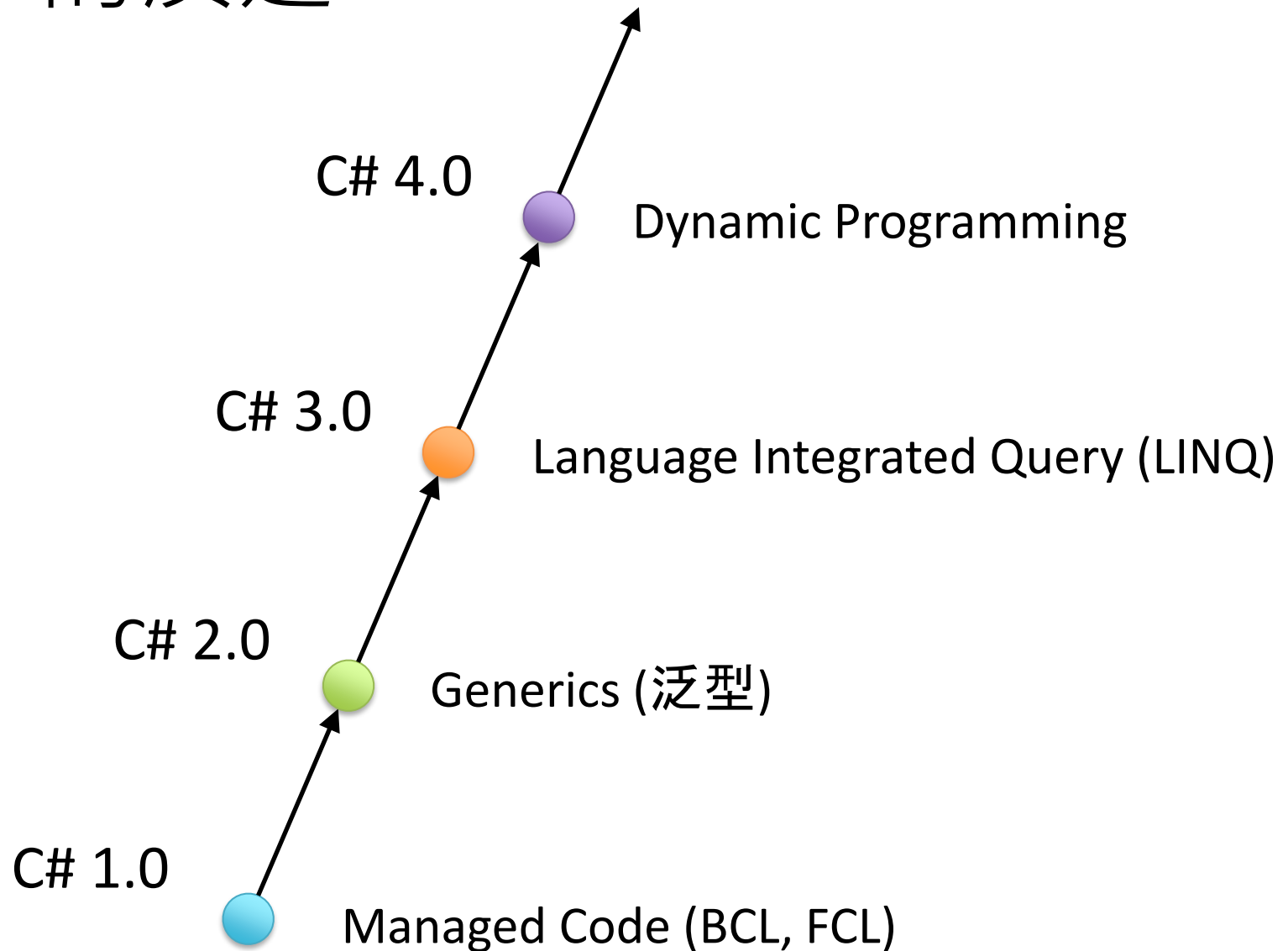
ADO.NET 與 EF 相關命名空間

- [System.Data](#)
 - 可用來存取表示 ADO.NET 架構的類別。
- [System.Data.Entity](#)
 - 包含的類別可提供 Entity Framework 核心功能的存取。(EF6)
- [System.Data.Entity.Infrastructure](#)
 - 包含可支援核心 Entity Framework 功能的類別。(EF6)

中繼資料屬相關命名空間

- [System.ComponentModel](#)
 - 提供類別，用於實作元件和控制項的執行階段和設計階段行為。這個命名空間會包含基底類別 (Base Class) 和介面，用於實作屬性 (Attribute) 和型別轉換子 (Type Converter)、繫結至資料來源，以及授權元件。
- [System.ComponentModel.DataAnnotations](#)
 - 提供用來定義 ASP.NET MVC 和 ASP.NET 資料控制項的中繼資料屬性類別。

C# 的演進



C# 2.0

- C# 2.0

- 部分類別 (Partial class) → partial
- 泛型 (Generics) → List<T>
- 靜態類別 (Static classes) → static
- 產生器功能 (Generator functionality) → yield
 - IEnumerable<T>
- 匿名委派 (Anonymous delegates) → delegate
- 委派的共變數和反變數 (Delegate covariance and contravariance)
- 屬性 (Property) 的 get / set 存取子可不同存取性
(The accessibility of property accessors can be set independently)
- 允許空值的實值型別 (Nullable types) → int?
- null 聯合運算子 (Null-coalescing operator) → ??

C# 3.0

- C# 3.0
 - LINQ 查詢運算式 (language-integrated query)
 - 物件和集合初始設定式 (Object and Collection initializers)
 - 匿名型別 (Anonymous types)
 - 隱含類型區域變數 (Local variable type inference)
 - Lambda 運算式 (Lambda expressions)
 - 運算式樹狀架構 (Expression trees)
 - 自動實作的屬性 (Automatic properties)
 - 擴充方法 (Extension methods)
 - 部分類別和方法 (Partial class and methods)

C# 4.0

- C# 4.0
 - 動態型別 (Dynamic member lookup)
 - 泛型中的共變數和反變數
(Covariant and contravariant generic type parameters)
 - 選擇性參數與具名參數
(Optional parameters and named arguments)
 - 使用 COM Interop 物件時不用再加上 ref 關鍵字
(Optional ref keyword when using COM)
 - 在 COM Interop 程式設計中使用索引的屬性
(Indexed properties)

Visual Studio 2015

- 基本操作
 - 程式碼片段
 - 基本鍵盤快速鍵 (F12 , Ctrl+, , Ctrl+. , ...)
- 工具視窗
 - 方案總管
 - 伺服器總管
 - 錯誤窗格
- 常用套件
 - NuGet Package Manager (已內建)
 - [Entity Framework Power Tools Beta 4](#) ([說明文件](#))

Visual Studio 與 Entity Framework

- Visual Studio 2008 SP1
 - Entity Framework 3.5
- Visual Studio 2010
 - Entity Framework 4.0
 - Entity Framework 4.1
 - Entity Framework 4.2
 - Entity Framework 4.3
 - Entity Framework 5.0
 - Entity Framework 6.0 (runtime only)
- Visual Studio 2012
 - Entity Framework 4.x
 - Entity Framework 5
 - Entity Framework 6
- Visual Studio 2013
 - Entity Framework 5
 - Entity Framework 6
- Visual Studio 2015
 - Entity Framework 5
 - Entity Framework 6

SQL Server

- EF 支援 SQL Server 2005 以上版本
- Management Studio
 - 各式資料庫操作
 - 查詢執行計畫與查看效能報表
- SQL Profiler
 - 分析應用程式對 SQL Server 下達的指令
 - SQL Server 2012 SP1 以後免費下載！
- Database Engine Tuning Advisor
 - 索引與效能分析

Entity Framework Introduction

Entity Framework 簡介

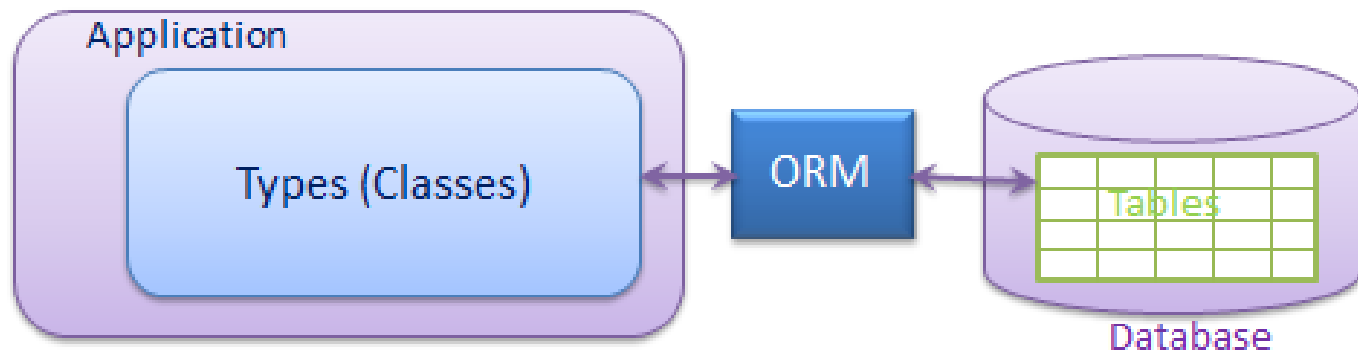


什麼是 Entity Framework ？

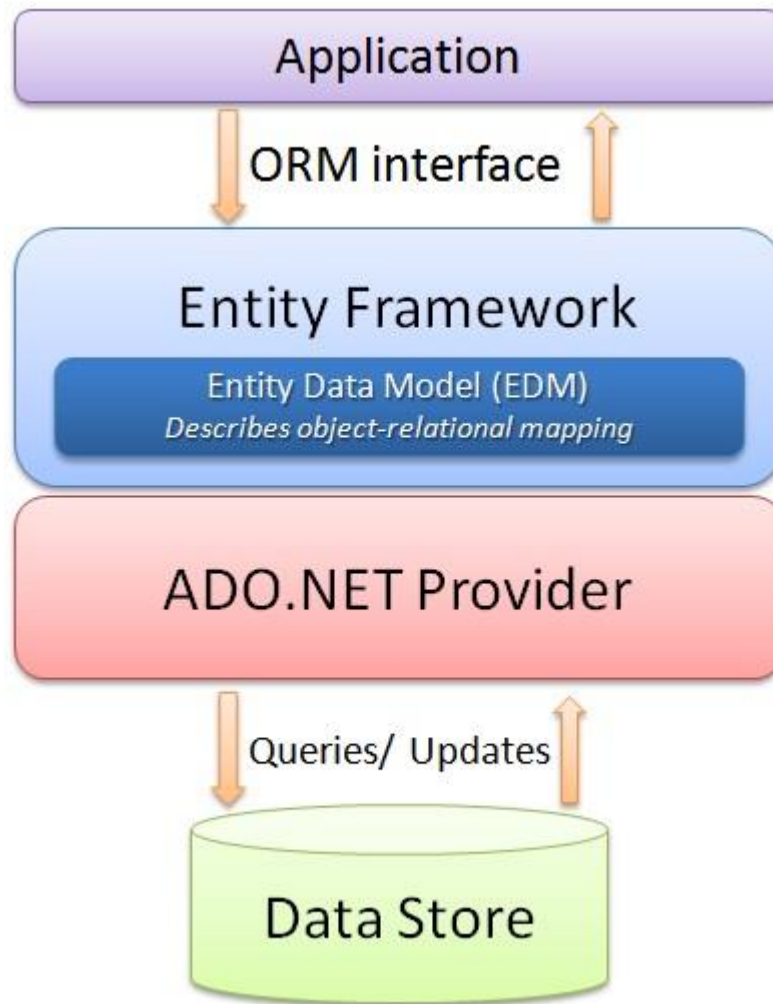
- 一個 O/RM 框架
- 減少資料存取的程式碼
- 透過 LINQ 自動產生 SQL 指令碼
- 透過「強型別」取得與操作物件資料
- 支援變更追蹤、資料識別、延遲載入
- 讓你更專注在商業邏輯，而非繁瑣的程式碼
- 支援三種開發模型 (開發工作流程)
 - DB First, Model First, Code First

什麼是 O/RM (Object-relational mapping)

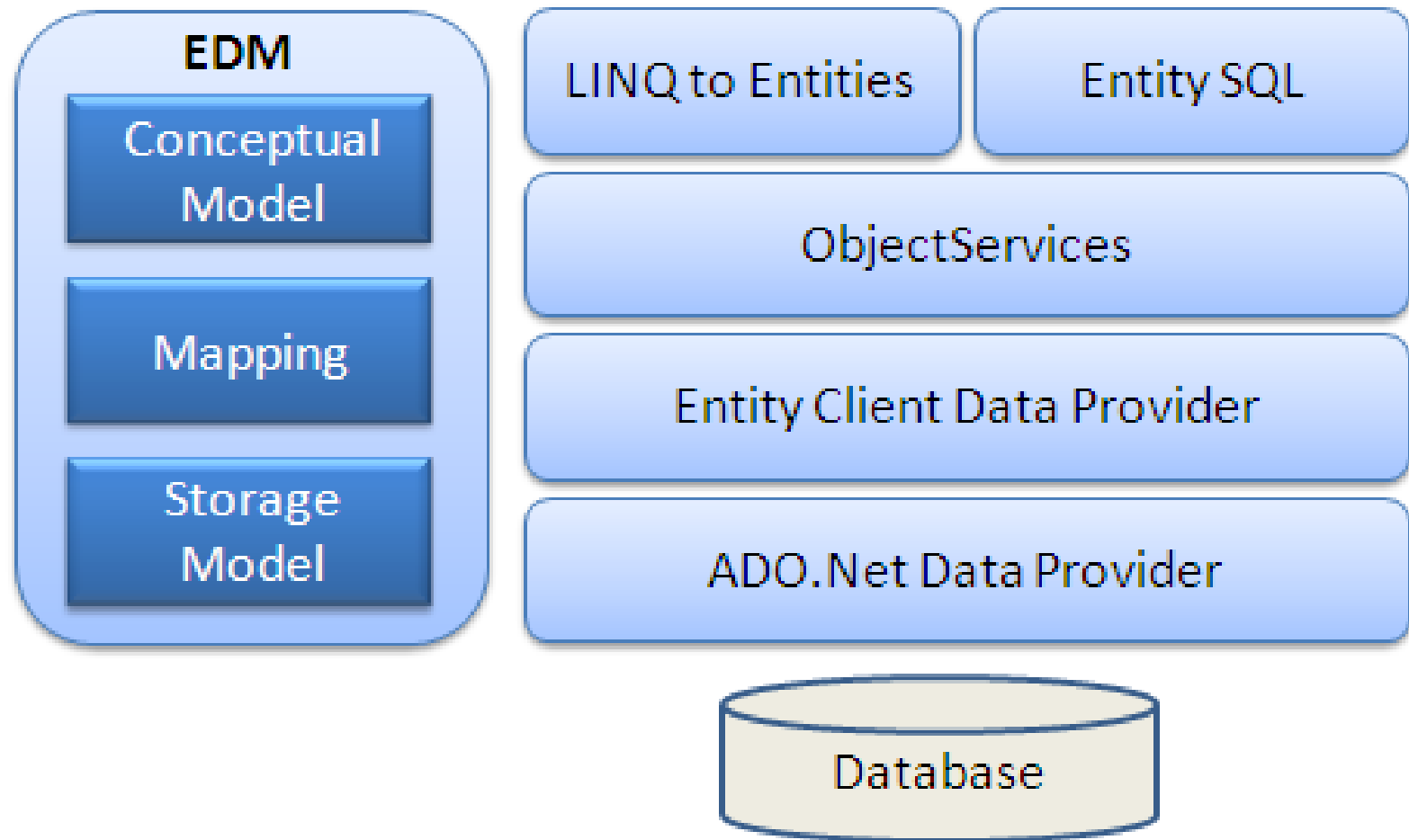
- ORM 做什麼事？
 - 將結構化的關連資料對映成物件導向模型
 - 將物件資料對應成關連資料



Entity Framework 框架



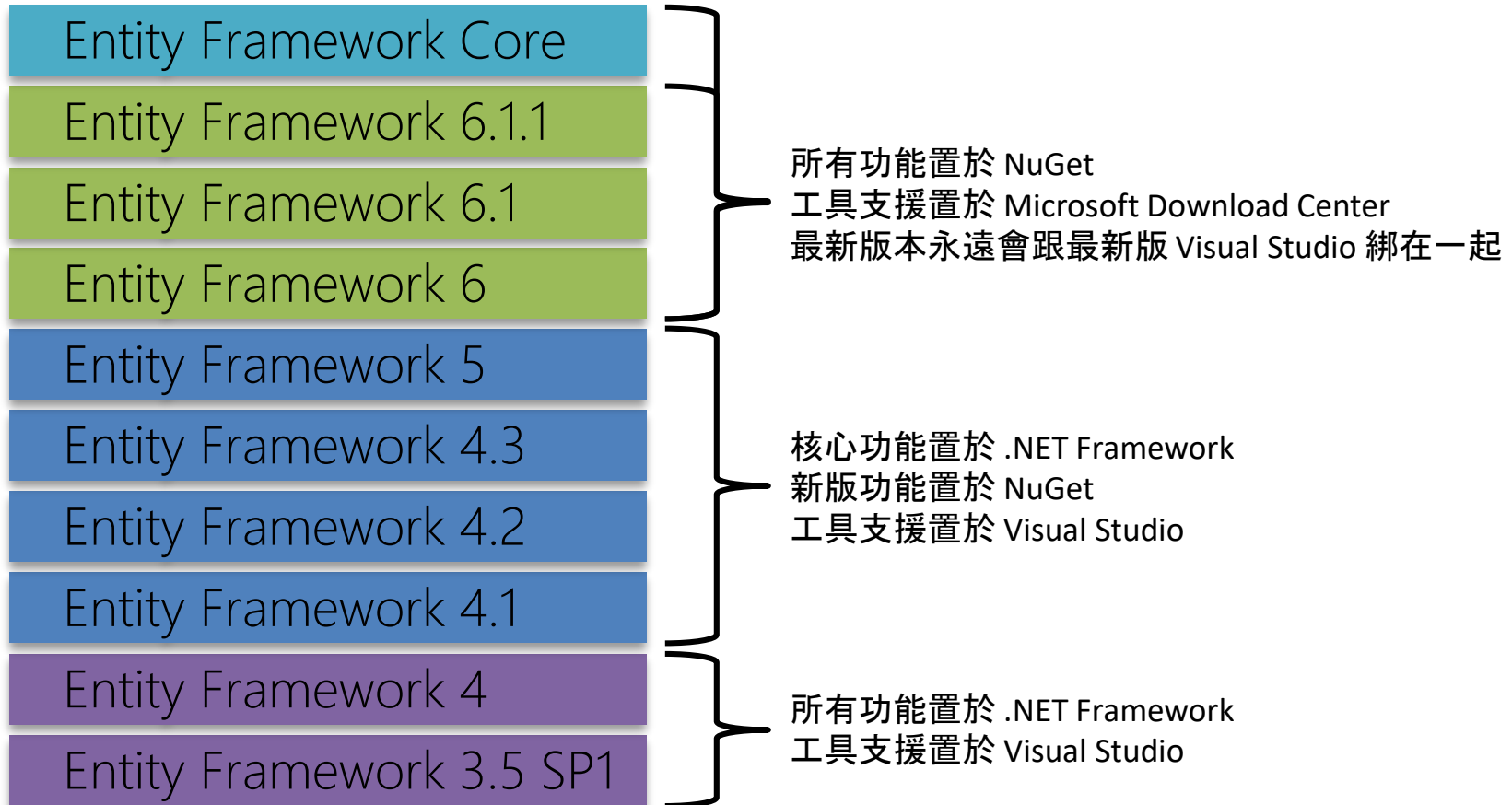
Entity Framework 框架



Entity Framework 的版本歷程

版本	特性
EF 3.5	基本 ORM 框架，僅支援 DB First 開發模式
EF 4.0	開始支援 POCO 、 延遲載入 、 可測試性提升 、 自訂程式碼產生器範本 、支援 Model First 開發模式
EF 4.1	開始支援 NuGet 套件 、增加 DbContext API (取代ObjectContext API)、支援 Code First 開發模式
EF 4.3	支援 Code First Migrations 功能
EF 5.0	EF 開放原始碼、支援 Enum (.NET 4.5)、 Spatial data types 、 Table-Valued Functions (TVFs) 、 支援單一模型多重圖表 、支援 標示實體顏色 、 批次匯入預存程序 、 查詢效能提升 、 EF Power Tools
EF 6.0	支援非同步查詢與儲存 、 斷線自動重連機制 、 基本 DI 支援 、 效能提升 (NGen)

EF 版本差異



選擇 Entity Framework 版本

- Entity Framework 5

- .NET 4.0/4.5 包含部分 EF 核心功能
- 其餘功能可直接從 NuGet 進行安裝 EntityFramework.dll
 - EntityFramework 5.0.0 ([NuGet](#))

```
PM> Install-Package EntityFramework -Version 5.0.0
```

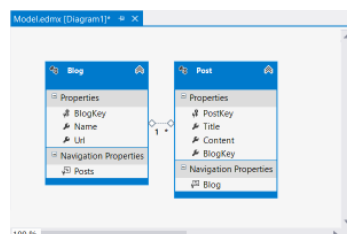
- Entity Framework 6

- EF 6 開始不再與 .NET Framework 有相依性
(須避免引用到相同型別名稱的命名空間)
 - EntityFramework 6.1.3 ([NuGet](#))

```
PM> Install-Package EntityFramework -Version 6.1.3
```

開發工作流程

透過設計工具定義



透過程式碼定義

```
public class Blog
{
    public int BlogKey { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public virtual List<Post> Posts {
    }
}

public class Post
```

全新
資料庫

Model First

在 EF Designer 中建立模型
在 EF Designer 中透過模型產生資料庫
從模型自動建立相關模型類別

Code First

透過程式碼定義類別與關連對應
資料庫依據程式碼定義自動被建立
自動套用模型變更到資料庫 (資料庫移轉)

現有
資料庫

Database First

從 EF Designer 匯入資料庫中現有定義
從模型自動建立相關模型類別

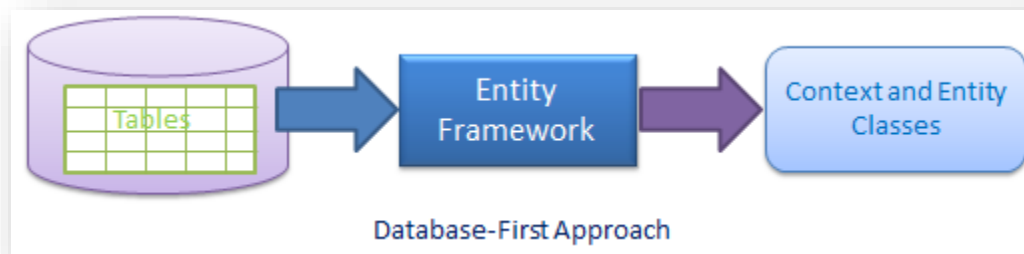
Code First

透過程式碼定義類別與關連對應
透過 EF Power Tools 提供的反向工程工具
(自動建立 Code First 所需的模型類別)

資料庫先行開發模式 (DB First)

- 開發流程

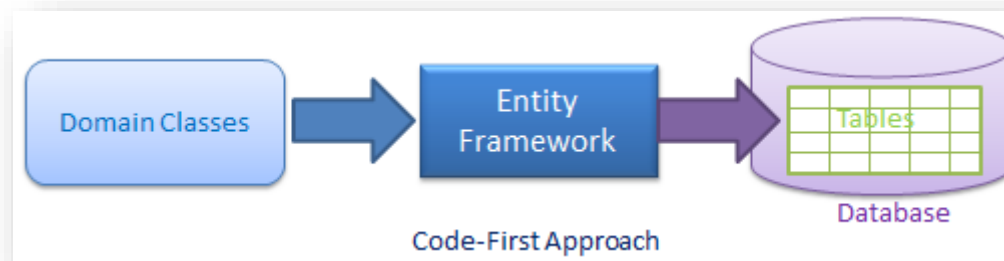
- 先建立資料庫表格結構 (一定要有主索引鍵)
- 再建立 實體資料模型 (EDM)
- VS 會自動從現有的 DB 產生對應的實體類別
- 支援預存程序 (SP)、檢視表 (View)



程式碼先行開發模式 (Code First)

- 開發流程

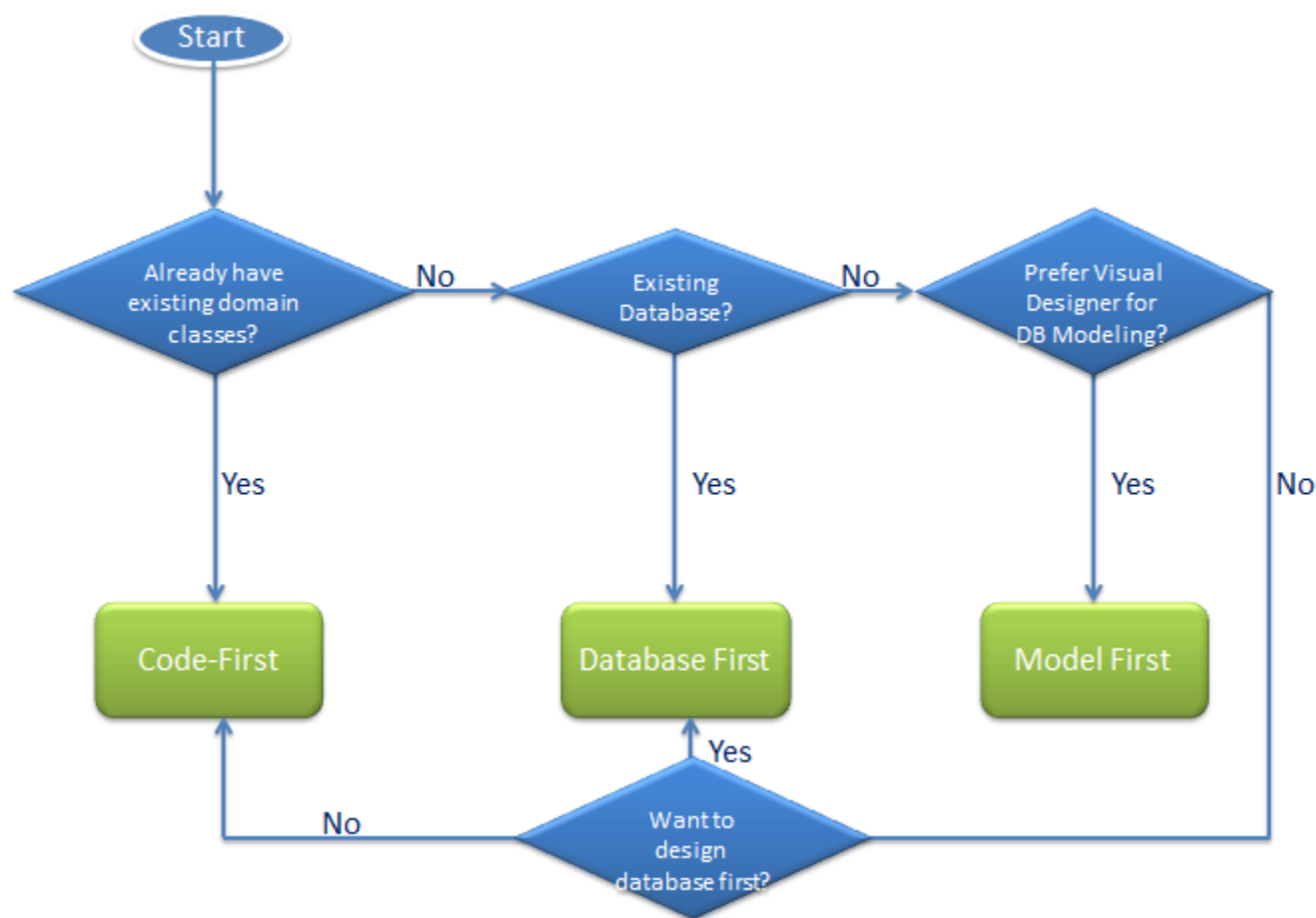
- 先手動建立 實體類別 (*.cs) 模型
 - 雖然沒有 EDMX 檔，但 EF 必須有 EDM 才能運作
- 透過 實體類別 (Entity Class) 自動 建立 資料庫
- EF6 支援預存程序 (SP)、不支援檢視表 (View)



模型先行開發模式 (Model First)

- 開發流程
 - 先建立 實體資料模型 (EDM)
 - 透過 實體資料模型 (EDM) 自動產生資料庫 DDL
 - 自動建立資料庫 Schema
 - VS 會自動從現有的 DB 產生對應的實體類別
 - 不支援預存程序 (SP)、檢視表 (View)

如何選擇適當的開發工作流程





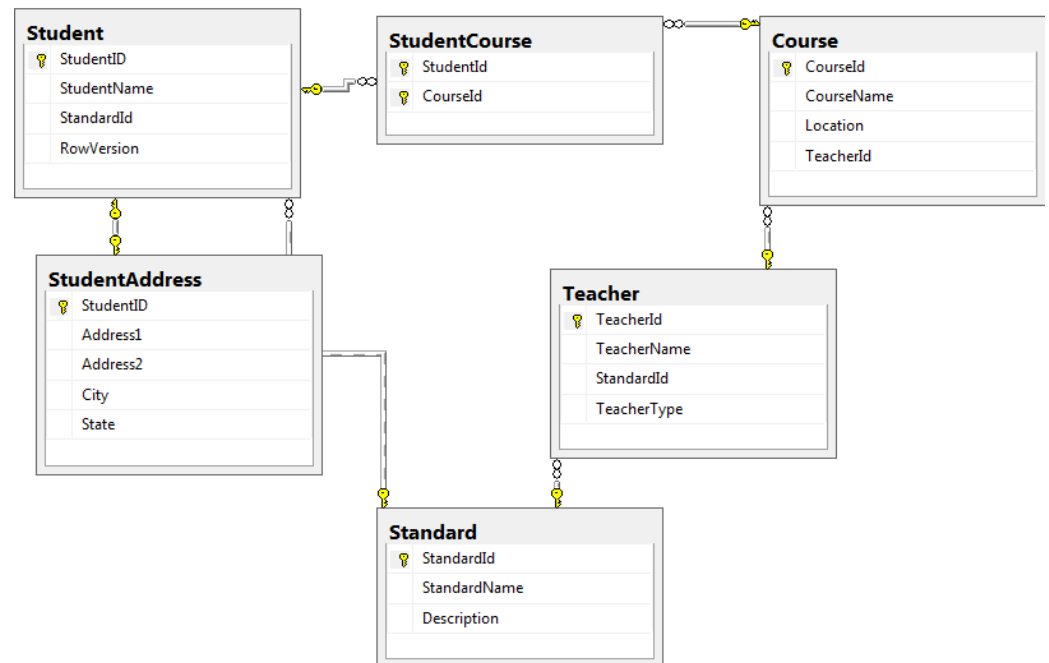
Entity Framework Quick Start (Database First)

Entity Framework 快速上手

準備範例資料庫

- 建立資料表
 - 需有主索引鍵
- 建立表格關聯
 - 一對一
 - 一對多
 - 多對多
- 建立檢視表
- 建立預存程序

ContosoUniversity.sql



請用 Management Studio 連接 LocalDB 資料庫: (localdb)\v11.0

建立第一個範例專案

- 建立新專案 (Console 或 Web Application)
- 使用 伺服器總管
 - 資料連接
 - (localdb)\v11.0 VS2013
 - (localdb)\MSSQLLocalDB VS2015
- 使用 SQL Server 物件總管
- 建立 ADO.NET 實體資料模型

建立 ADO.NET 實體資料模型

- 來自資料庫的 EF Designer
 - 資料庫物件和設定
 - [] 產生的物件名稱複數化或單數化
 - [■] 在模型中包含外部索引鍵資料行
 - [■] 將選取的預存程序和函數匯入實體模型
- 介紹 Entity Framework 設計工具
- 介紹 EDMX 檔案內容結構
- 介紹 T4 與自動產生的程式碼

介紹 Entity Framework 設計工具

- 屬性視窗 (Property Window)

屬性視窗 (Property Window) 顯示 ContosoUniversityModel.Department 的屬性。

文件	屬性
名稱	Department
存取	Public
抽象	False
基底類型	(無)
填滿色彩	0, 122, 204
實體集名稱	Department

名稱
實體的名稱。

屬性視窗 (Property Window) 顯示 ContosoUniversityModel.Department.Name 的屬性。

文件	屬性
完整描述	
摘要	
可為 Null	(無)
名稱	Name
並行模式	None
固定長度	False
型別	String
最大長度	50
預設值	(無)
實體索引鍵	False

Getter
決定 Getter 屬性的存取。

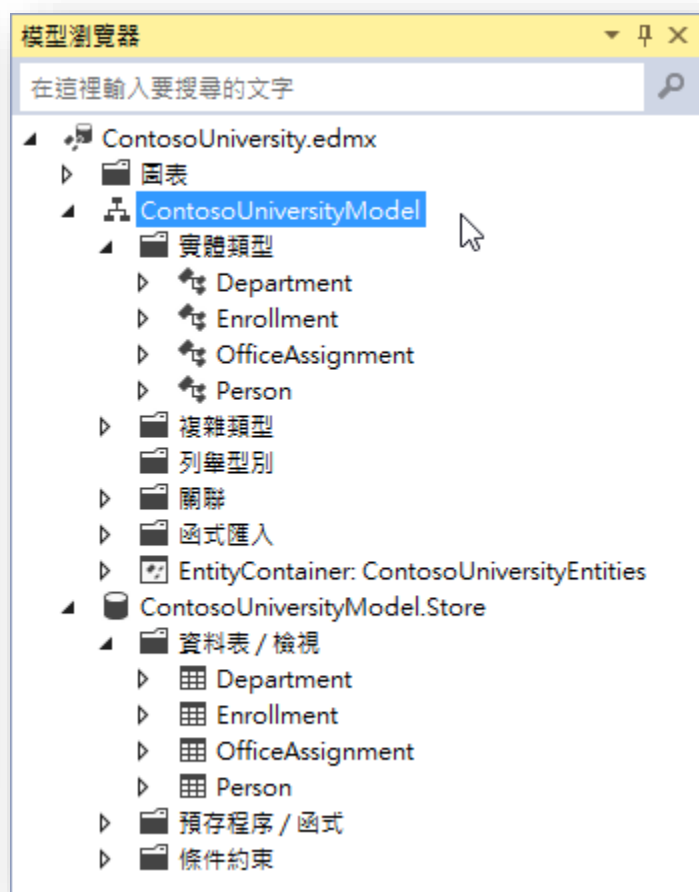
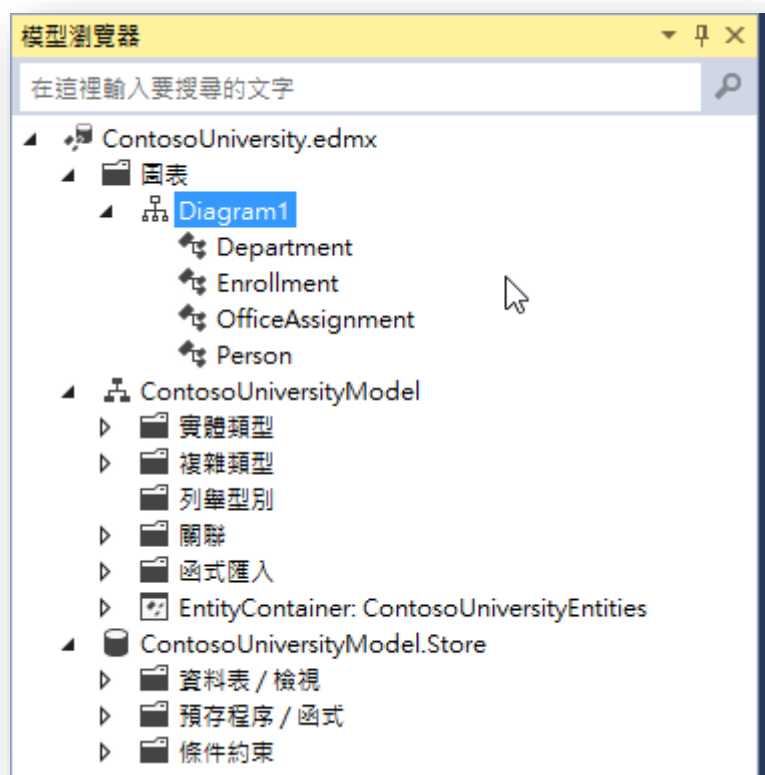
屬性視窗 (Property Window) 顯示 ContosoUniversityModel.Department.DepartmentID 的屬性。

文件	屬性
完整描述	
摘要	
可為 Null	False
名稱	DepartmentID
並行模式	None
型別	Int32
預設值	(無)
實體索引鍵	True

StoreGeneratedPattern
決定在進行插入和更新作業時，是否會在資料庫中自動產生對應的資料行

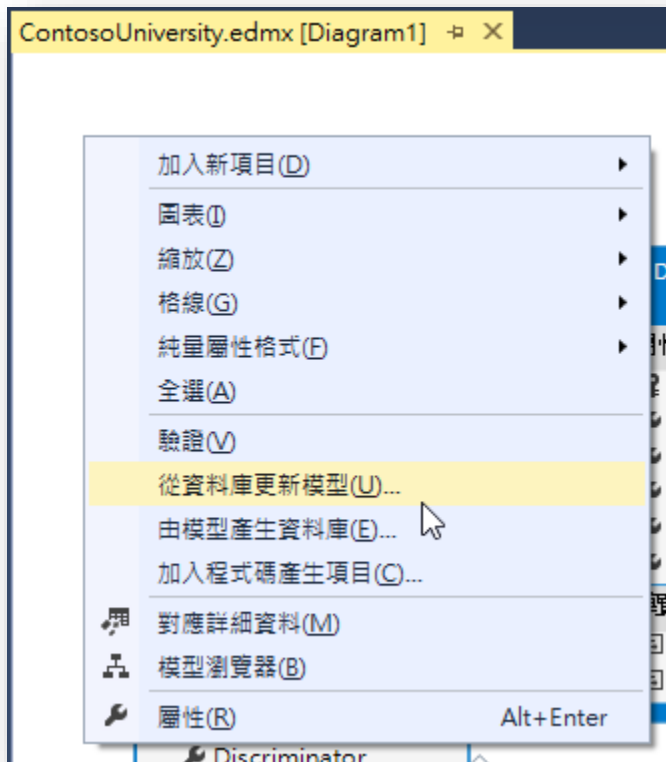
介紹 Entity Framework 設計工具

- 模型瀏覽器 (Model Browser)



介紹 Entity Framework 設計工具

- 從資料庫更新模型

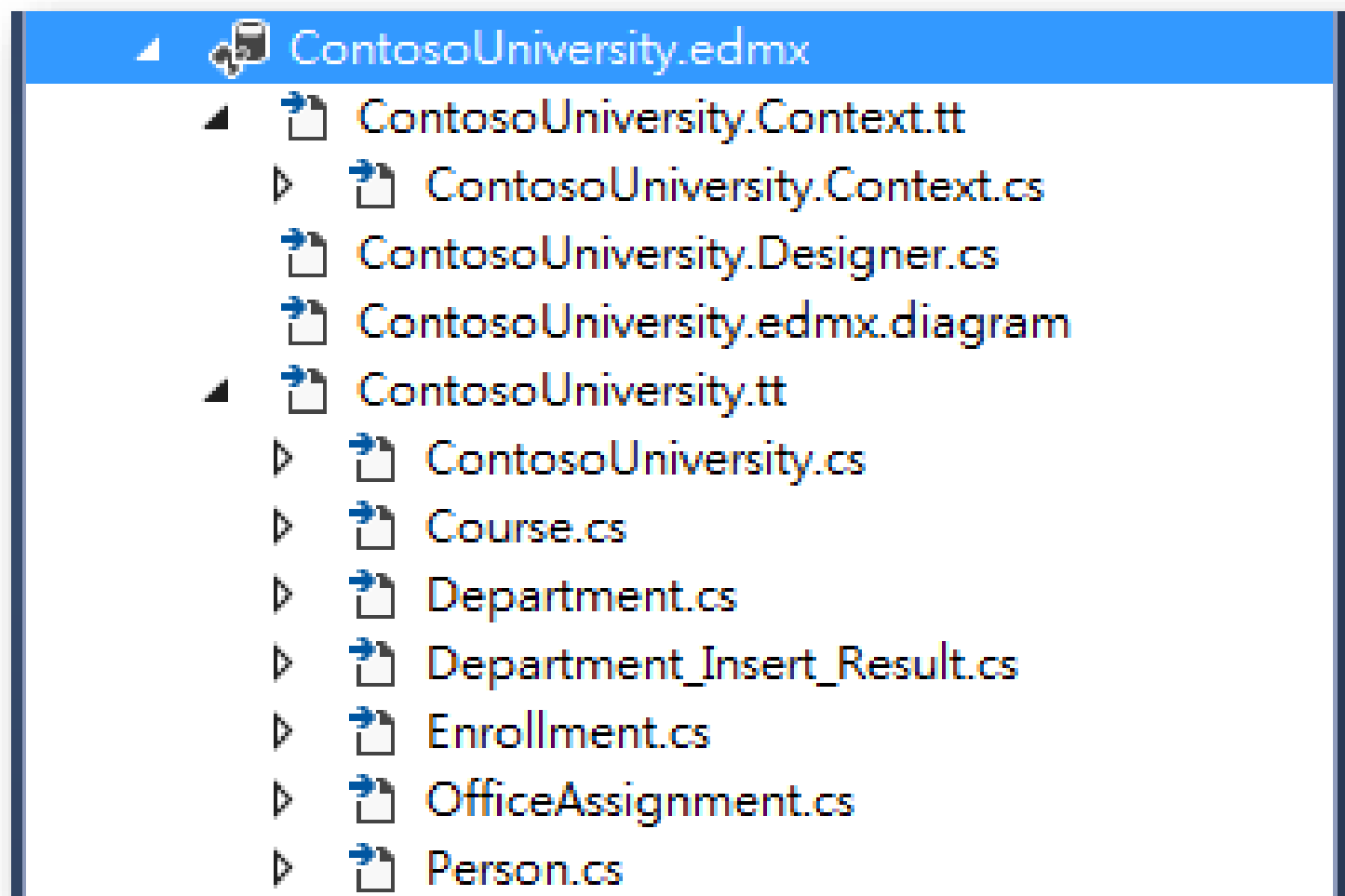


執行 Entity Framework 的
「從資料庫更新模型」為何不更新

介紹 EDMX 檔案內容結構

- edmx:Runtime
 - edmx:StorageModels
 - **SSDL** = store schema definition language
 - edmx:ConceptualModels
 - **CSDL** = Conceptual schema definition language
 - edmx:Mappings
 - **MSL** = mapping specification language
 - C-S Mapping
- Designer
- CSDL、SSDL 和 MSL 規格

介紹 T4 與自動產生的程式碼



DbContext 檔案內容摘要

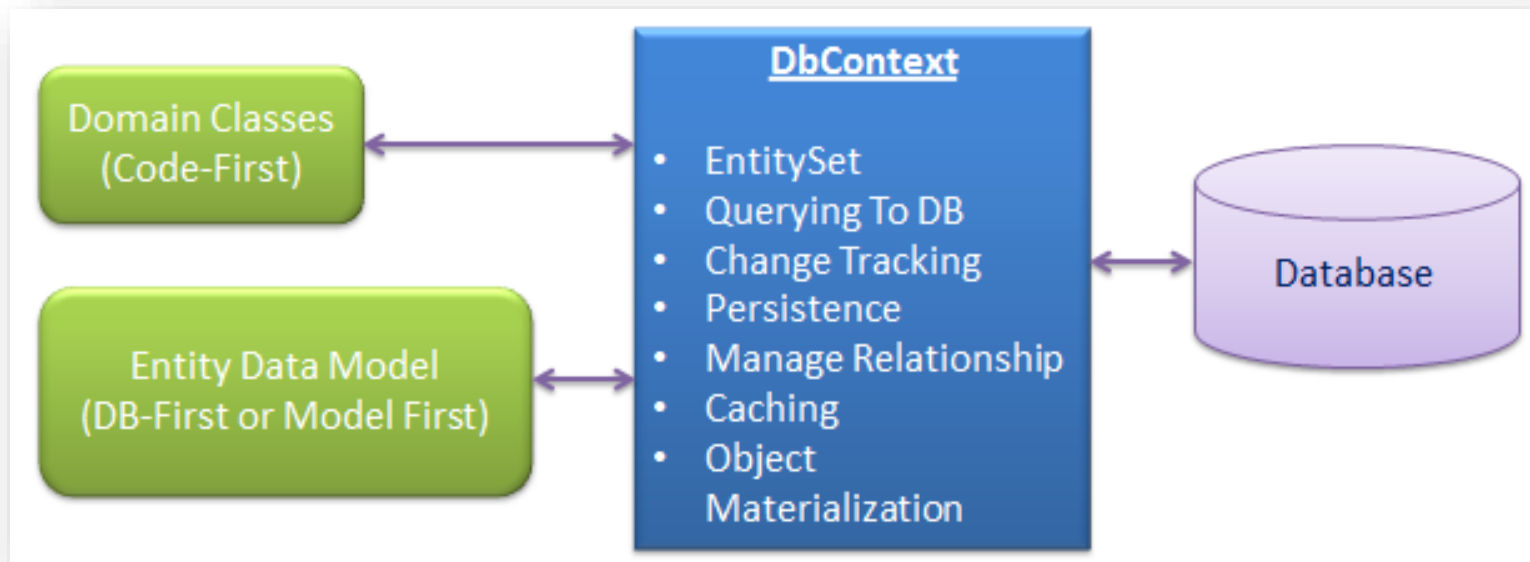
- ContosoUniversity.Context.cs

```
10 namespace ConsoleApplication1
11 {
12     using System;
13     using System.Data.Entity;
14     using System.Data.Entity.Infrastructure;
15     using System.Data.Entity.Core.Objects;
16     using System.Linq;
17
18     2 個參考
19     public partial class ContosoUniversityEntities : DbContext
20     {
21         1 個參考
22         public ContosoUniversityEntities()
23             : base("name=ContosoUniversityEntities")
24         {
25         }
26
27         0 個參考
28         protected override void OnModelCreating(DbModelBuilder modelBuilder)
29         {
30             throw new UnintentionalCodeFirstException();
31         }
32
33         0 個參考
34         public virtual DbSet<Course> Course { get; set; }
35         2 個參考
36         public virtual DbSet<Department> Department { get; set; }
37         0 個參考
38         public virtual DbSet<Enrollment> Enrollment { get; set; }
39         0 個參考
```



DbContext 衍生類別的三個部分

- 類別建構式 → 指定連接字串名稱
- `OnModelCreating` → Fluent API (Code First)
- `DbSet<T>` → 實體資料集 (Entity Set)



類別建構式：指定連接字串名稱

- 從 web.config 或 app.config 讀取連接字串

```
public ContosoUniversityEntities()  
    : base("name=ContosoUniversityEntities")  
{  
}
```

- 從程式動態指定連接字串

```
public ContosoUniversityEntities()  
    : base("name=ContosoUniversityEntities")  
{  
    base.Database.Connection.ConnectionString =  
        @"data source=(localdb)\v11.0; initial catalog=ContosoUniversity;  
        integrated security=True; MultipleActiveResultSets=True;  
        App=EntityFramework";  
}
```

DbContext 的主要職責

- 實體集合 (EntitySet)
 - 也就是 DbSet<TEntity> 的這些實體，主要用來對應到資料庫中的資料表
- 查詢資料 (Querying)
 - DbContext 轉換 LINQ to Entities 語法成 SQL 查詢語法，並將指令送到資料庫
- 變更追蹤 (Change Tracking)
 - 當資料從 DB 查詢出來後，進一步追蹤所有實體物件 (Entity object) 的各種狀態
- 存續資料 (Persisting Data)
 - 依據實體物件的狀態 (EntityState) 執行資料的新增、更新、刪除等操作
- 快取 (Caching)
 - DbContext 預設會將資料庫中的資料快取起來 (僅限於 DbContext 的執行生命週期)
- 管理關聯性 (Manage Relationship)
 - DbContext 管理實體與實體之間的關聯性
 - 可透過 EDM (DB First) 或 Fluent API (Code First) 來定義實體之間的關聯性
- 物件實體化 (Object Materialization)
 - DbContext 將 DB 取得到的原始資料轉換成強型別的實體物件 (entity objects)

使用 DbContext API

- 了解 Entity Framework 的生命週期範圍

```
using (var ctx = new ContosoUniversityEntities())  
{  
    // 透過 ctx 操作 CRUD 等動作  
}
```

- 取得早期ObjectContext物件的方式

```
using System.Data.Entity.Infrastructure;  
using (var ctx = new ContosoUniversityEntities())  
{  
    var objectContext  
        = (ctx as IObjectContextAdapter).ObjectContext;  
}
```

EF 的實體物件類型 (1)

- POCO 實體物件 (Plain Old CLR Object)
 - 必須為**公開型別**
 - 不能標示 sealed
 - 不能為抽象類別
 - 兩種屬性類型：
 - **純量屬性** (儲存DB的資料)
 - **導覽屬性** (用來**關聯**其他實體的資料)
 - **導覽屬性**必須為 public 或 virtual
 - 每個**集合屬性**必須為 **ICollection<T>** 型別

EF 的實體物件類型 (2)

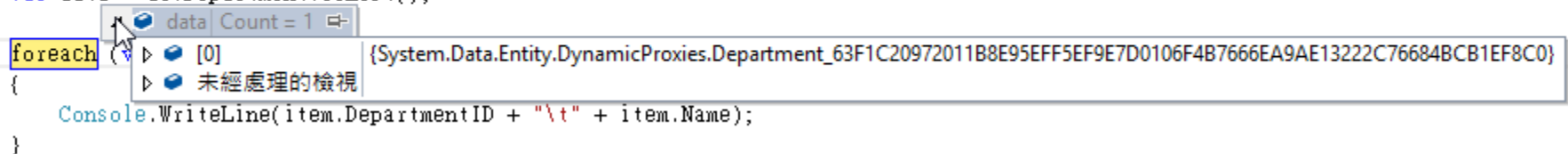
- POCO 代理物件 (Dynamic Proxy)
 - 動態代理實體是在 runtime 動態產生的物件
 - *db.Configuration.ProxyCreationEnabled* 預設為 *true*
 - 主要特性：讓設定 *virtual* 的導覽屬性支援 延遲載入

```
var db = new ContosoUniversityEntities();

//db.Configuration.ProxyCreationEnabled = false;

var data = db.Department.ToList();

foreach (var item in data)
{
    Console.WriteLine(item.DepartmentID + "\t" + item.Name);
}
```



[Entity Framework Working with Proxies](#)

EF 的實體物件之間的關聯

- 一對一 (One-to-One relationship)
- 一對多 (One-to-Many Relationship)
- 多對多 (Many-to-Many Relationship)
 - 資料庫中的多對多關聯表格將不會出現在 EF 設計工具中

EF 的實體物件之間的關聯

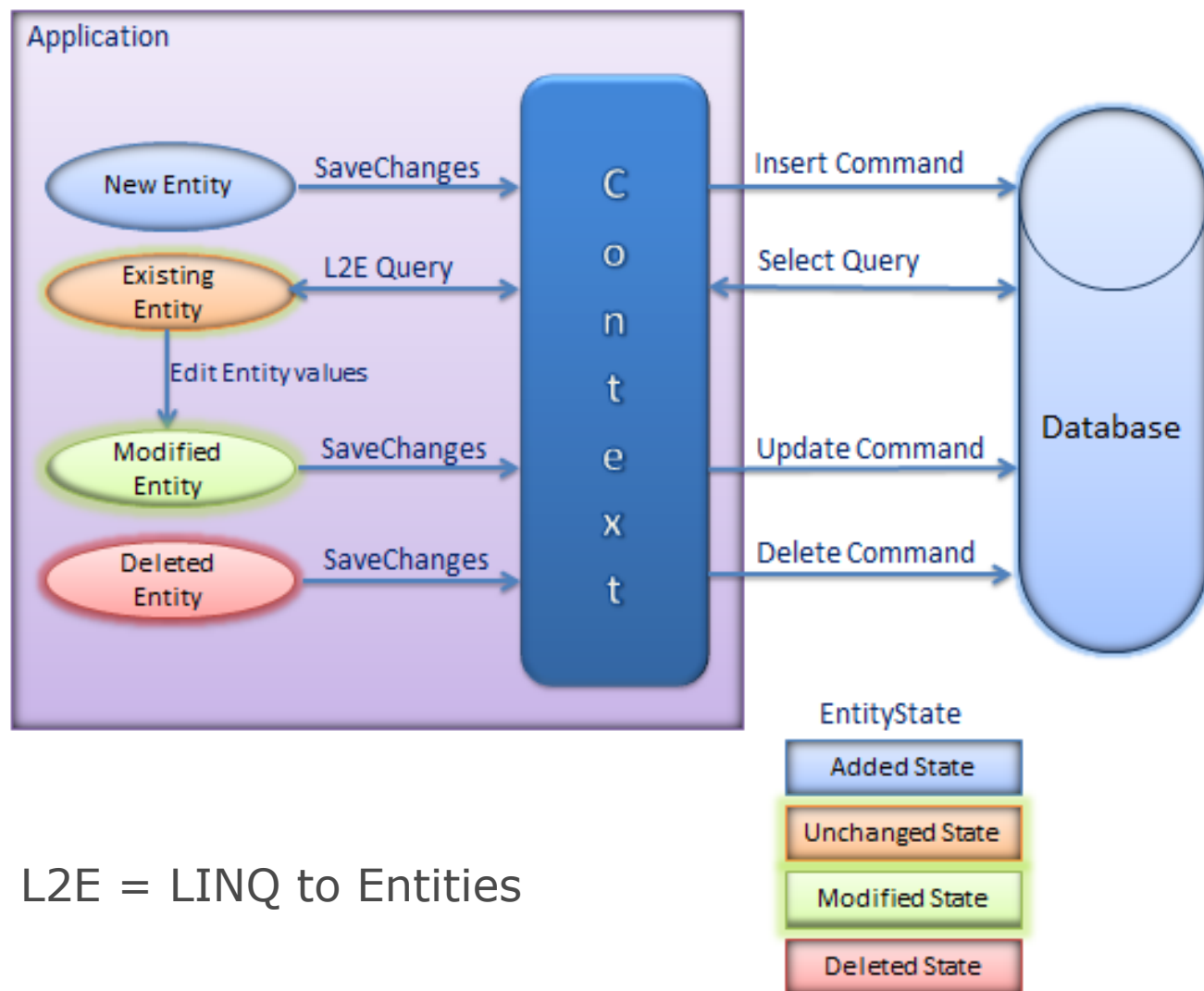


看看哪個表格不見了？

EF 執行生命週期

- 常見的 CRUD 資料操作 (Create, Read, Update, Delete)
- EF 執行生命週期裡，都是透過 DbContext 操作
- 每個被取出的實體 (Entity) 都擁有 **EntityState** 實體狀態
- System.Data.Entity.EntityState (列舉型別)
 - Unchanged
 - Added
 - Deleted
 - Modified
 - Detached
- 變更追蹤 (Change Tracking)
 - DbContext 會自動追蹤實體狀態變更 (透過POCO代理物件)

EF 執行生命週期



EF 如何查詢資料

- LINQ to Entities (可透過 LINQPad 練習)

```
using (var db = new SchoolDBEntities())
{
    var data = from p in db.Students where p.Id == 1 select p;
}
```

- Entity SQL

- <http://www.entityframeworktutorial.net/Querying-with-EDM.aspx>

- Native SQL

```
using (var db = new SchoolDBEntities())
{
    var studentName = db.Database.SqlQuery<Student>("SELECT studentid,
studentname, standardId FROM Student WHERE studentname='Bill'")
        .FirstOrDefault();
}
```


EF 如何新增資料

```
var db = new ContosoUniversityEntities();
```

```
db.Department.Add()
```

Department DbSet<Department>.Add(Department entity)

將給定的實體加入至 Added 狀態集合的基礎內容中，好讓呼叫 SaveChanges 時可將它插入資料庫中。

entity: 要加入的實體。

```
db.Department.Add(new Department() {  
    DepartmentID = 1,  
    Name = "RD"  
});
```

```
db.SaveChanges();
```

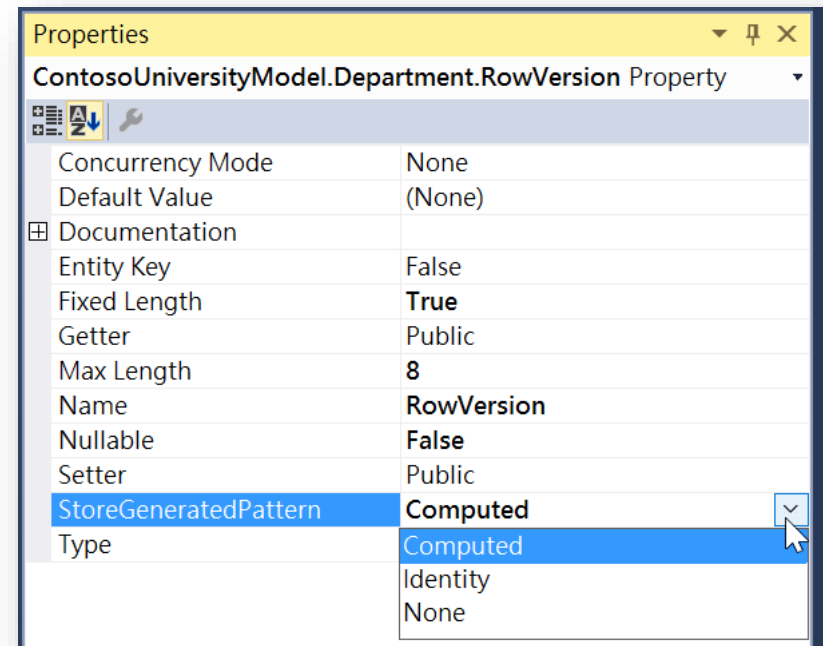
關於 StoreGeneratedPattern

- [關於 StoreGeneratedPattern 列舉類型](#)

- Computed 在插入和更新時都會產生值。
- Identity 在插入時會產生值，並在更新時保持不變。
- None 表示並非伺服器所產生屬性的值。此為預設值。

- 相關連結

- [Set Identity and Computed Properties in Entity Framework Without Triggers](#)
- [What is the recommended identity generation approach in Entity framework? - Stack Overflow](#)



EF 如何更新資料

- 依賴變更追蹤機制

```
var db = new ContosoUniversityEntities();  
  
var item = db.Department.Find(1);  
  
item.Name = "RD";  
  
db.SaveChanges();
```

- `db.SaveChanges();`

EF 如何刪除資料

- 透過 DbContext 提供的 Entity 屬性更新

db.Department.Remove()

Department DbSet<Department>.Remove(Department entity)

將給定的實體標記為 Deleted，好讓呼叫 SaveChanges 時可從資料庫中將它刪除。請注意，此實體必須存在於某個其他狀態的內容中，然後才會呼叫此方法。

entity: 要移除的實體。

// db.Configuration.ProxyCreationEnabled = false;

db.Department.RemoveRange()

IEnumerable<Department> DbSet<Department>.RemoveRange(IEnumerable<Department> entities)

從要置於 Added 狀態之每個實體的集合基礎內容中移除給定的實體集，讓呼叫 SaveChanges 時可將它從資料庫中刪除。

entities: 要刪除之實體的集合。

// db.Configuration.ProxyCreationEnabled = false;

常見的 Entity Framework 查詢

- 標示**粗體**的都是「**預先載入**」的方法
 - Restriction
 - [Where](#)
 - Projection
 - [Select](#)
 - Ordering
 - [OrderBy](#)
 - [OrderByDescending](#)
 - Elements:
 - [First](#)
 - [FirstOrDefault](#)
 - [Last](#)
 - [LastOrDefault](#)
 - [Single](#)
 - [SingleOrDefault](#)
 - Partitioning
 - [Take](#)
 - [TakeWhile](#)
 - [Skip](#)
 - [SkipWhile](#)
 - Conversion
 - [ToArray](#)
 - [ToList](#)
 - [ToDictionary](#)
 - [ToLookup](#)
 - Qualifiers
 - [Any](#)
 - [All](#)
 - [Contains](#)
 - Aggregation:
 - [Count](#)
 - [LongCount](#)
 - [Sum](#)
 - [Min](#)
 - [Max](#)
 - [Average](#)
 - Join
 - [Join](#)
 - Grouping
 - [GroupBy](#)
 - Set
 - [Distinct](#)

更多 Entity Framework 查詢技巧

- 標示**粗體**的都是「**預先載入**」的方法
 - Restriction: [Where](#), [OfType](#)
 - Projection: [Select](#), [SelectMany](#)
 - Ordering: [OrderBy](#), [OrderByDescending](#), [ThenBy](#), [ThenByDescending](#), [Reverse](#)
 - Join: [Join](#), [GroupJoin](#)
 - Grouping: [GroupBy](#)
 - Set: [Zip](#), **[Distinct](#)**, [Union](#), [Intersect](#), [Except](#)
 - Aggregation: **[Aggregate](#)**, **[\[Count\]](#)**, **[LongCount](#)**, **[Sum](#)**, **[Min](#)**, **[Max](#)**, **[Average](#)**
 - Partitioning: [Take](#), [TakeWhile](#), [Skip](#), [SkipWhile](#)
 - Concatenating: [Concat](#)
 - Conversion: **[ToArray](#)**, **[ToList](#)**, **[ToDictionary](#)**, **[ToLookup](#)**, [\[Cast\]](#)
 - Equality: **[SequenceEqual](#)**
 - Elements: **[\[First\]](#)**, **[\[FirstOrDefault\]](#)**, **[\[Last\]](#)**, **[\[LastOrDefault\]](#)**, **[\[Single\]](#)**, **[\[SingleOrDefault\]](#)**, **[\[ElementAt\]](#)**, **[\[ElementAtOrDefault\]](#)**, **[DefaultIfEmpty](#)**
 - Generation: [Range](#), [Repeat](#), **[Empty](#)**
 - Qualifiers: **[Any](#)**, **[All](#)**, **[\[Contains\]](#)**



Advanced Topic for Entity Framework

Entity Framework 進階議題

深入了解 DbSet<T> 類別

方法名稱	回傳型別	說明
Find	實體物件	傳入 P.K. 參數，回傳單一筆實體物件
Add	實體物件	將一個 POCO 物件加入 DbSet 並將物件標示為 Added 狀態
Remove	實體物件	將傳入的實體物件標示為 Deleted 狀態
Create	POCO 物件	建立一筆 POCO 物件 (不會自動加入 DbSet 裡)
AsNoTracking	DbQuery<T>	回傳的實體物件會停用變更追蹤 (針對唯讀資料的情境會提升效能) ※ 實體物件將不會快取在 DbContext 物件服務中！
Attach	實體物件	將不受變更追蹤的物件加入變更追蹤並標示為 Unchanged 狀態
Include	DbQuery<T>	宣告一併載入其他關連實體的物件
SqlQuery	DbSqlQuery<T>	執行任意 SQL 查詢並回傳含有變更追蹤的實體物件 ※ 可搭配 AsNoTracking 回傳不含變更追蹤的實體物件

深入了解 DbEntityEntry<T> 類別

- DbEntityEntry

- 幫助你取得**實體物件**的

- 實體物件 (Entry) → **object**
 - 實體狀態 (State) → EntityState
 - 原始內容 (OriginalValues) → DbPropertyValues
 - 目前內容 (CurrentValues) → DbPropertyValues
 - 重新載入資料庫資料 → **Reload()**

- 範例程式

```
DbEntityEntry ce = db.Entry(course);  
if (ce.State == System.Data.Entity.EntityState.Modified)  
{  
    course.ModifiedOn = DateTime.Now;  
}
```

深入了解 DbPropertyValues 類別

- DbPropertyValues

- 幫助你取得或設定**實體物件**的特定屬性

- PropertyNames 所有屬性清單
 - GetValue<TValue>(屬性名稱) 取得特定屬性值
 - SetValues(object) 設定新的屬性值

※ 可以傳入匿名型別物件，他會自動比對屬性名稱

- 範例程式

```
DbEntityEntry ce = db.Entry(course);
if (ce.State == System.Data.Entity.EntityState.Modified)
{
    var orig = ce.OriginalValues.GetValue<int>("Credits");
    ce.CurrentValues.SetValues(new { Credits = 2 });
}
```

深入了解變更追蹤機制

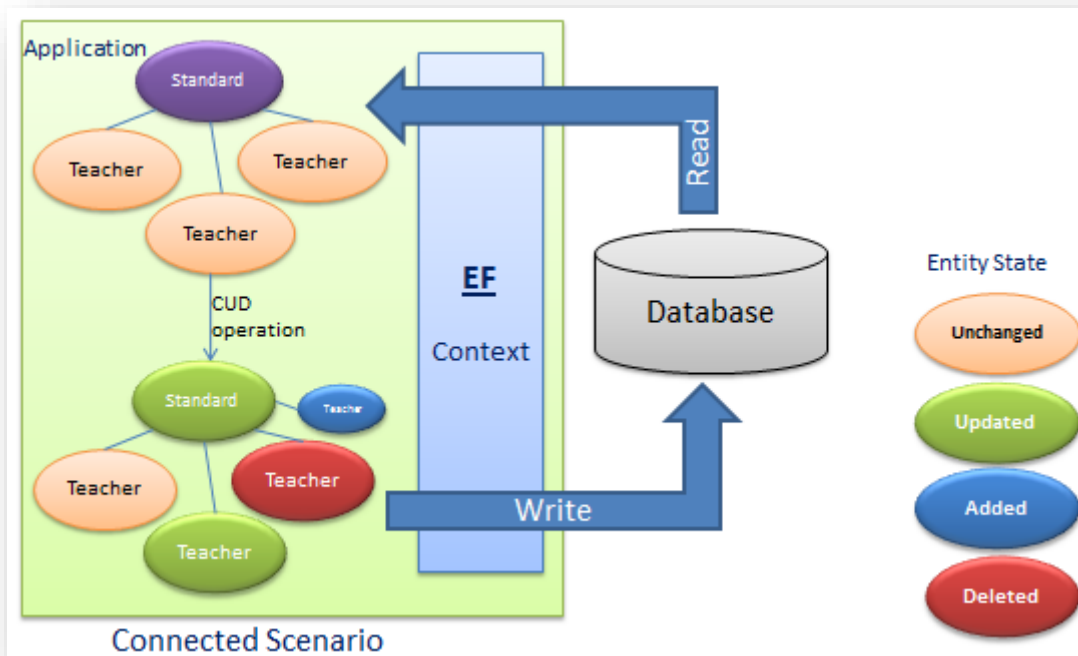
3 個參考 | 0 位作者 | 0 項變更

```
public partial class ContosoUniversityEntities : DbContext
{
    1 個參考 | 0 位作者 | 0 項變更
    public override int SaveChanges()
    {
        var entities = this.ChangeTracker.Entries();

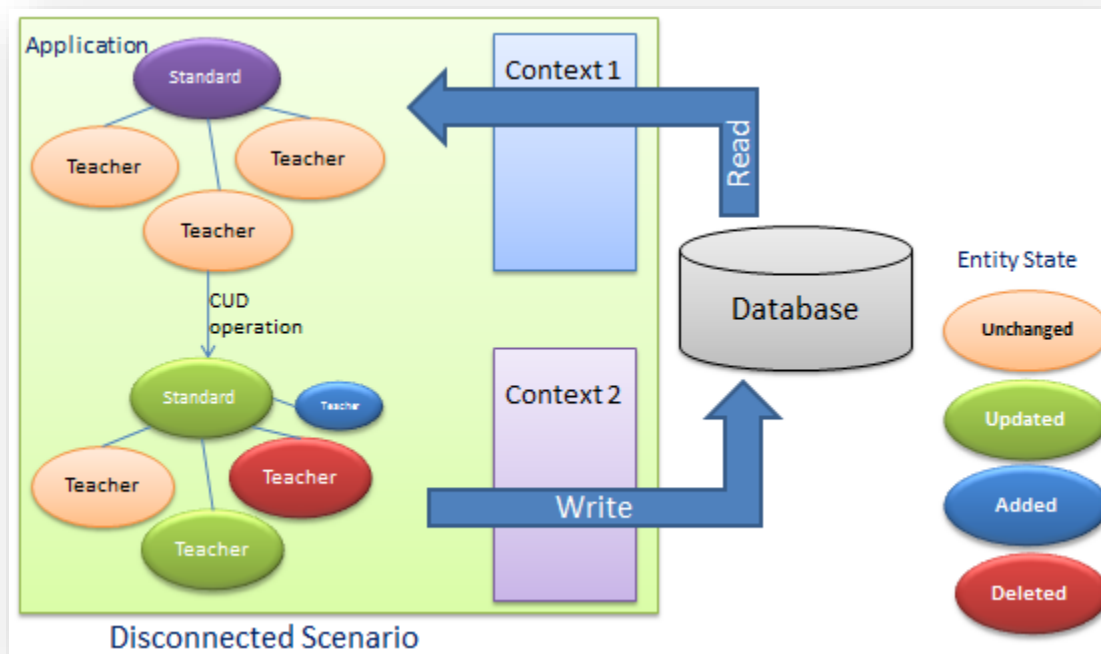
        foreach (var entry in entities)
        {
            Console.WriteLine("Entity Name: {0}", entry.Entity.GetType().FullName);
            Console.WriteLine("Status: {0}", entry.State);

            if (entry.State == EntityState.Modified)
            {
                entry.CurrentValues.SetValues(new { ModifiedOn = DateTime.Now });
            }
        }
        return base.SaveChanges();
    }
}
```

連線模式 v.s. 離線模式



連線模式 v.s. 離線模式



Entity Framework 連線模式

- 連線模式下會自動偵測實體物件變更
 - 可關閉自動偵測變更
 - `db.Configuration.AutoDetectChangesEnabled = false;`
 - 手動偵測變更的方法
 - `db.ChangeTracker.DetectChanges();`
- 透過 `ToList()` 建立的新集合
 - 無法偵測 **Added** 與 **Deleted** 狀態 (無法變更追蹤)
 - 僅能自動偵測實體物件的 **Modified** 狀態

Entity Framework 離線模式

- 困難點

- 已離線的實體物件並沒有「實體狀態」可參考

- 解決方法

- 透過 Add, Attach 或 Entry 方法加入變更追蹤
- 透過設定適當的 EntityState 狀態

`db.Entry(disconnectedEntity).State =`

- `EntityState.Added` 所有導覽屬性也會一併新增
- `EntityState.Modified` 所有導覽屬性不會一併更新
- `EntityState.Deleted` 所有導覽屬性不會一併刪除
- `EntityState.Unchanged`

Entity Framework 離線模式

- 離線模式的新增方法 (以下兩個語法相等)
 - `db.Course.Add(newCourse);`
 - `db.Entry(newCourse).State = EntityState.Added;`

Entity Framework 離線模式

- 離線模式的更新方法

`db.Entry(course).State = EntityState.Modified;`

```
Student stud;
//1. Get student from DB
using (var ctx = new SchoolDBEntities())
{
    stud = ctx.Students.Where(s => s.StudentName == "New Student1").FirstOrDefault<Student>();
}

//2. change student name in disconnected mode (out of ctx scope)
if (stud != null)
{
    stud.StudentName = "Updated Student1";
}

//save modified entity using new Context
using (var dbCtx = new SchoolDBEntities())
{
    //3. Mark entity as modified
    dbCtx.Entry(stud).State = System.Data.Entity.EntityState.Modified;

    //4. call SaveChanges
    dbCtx.SaveChanges();
}
```

Entity Framework 離線模式

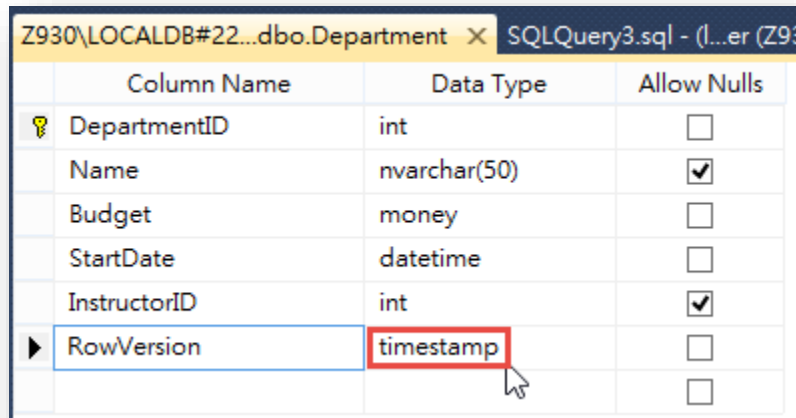
- 離線模式的刪除方法

```
db.Entry(courseToDelete).State = EntityState.Deleted;
```

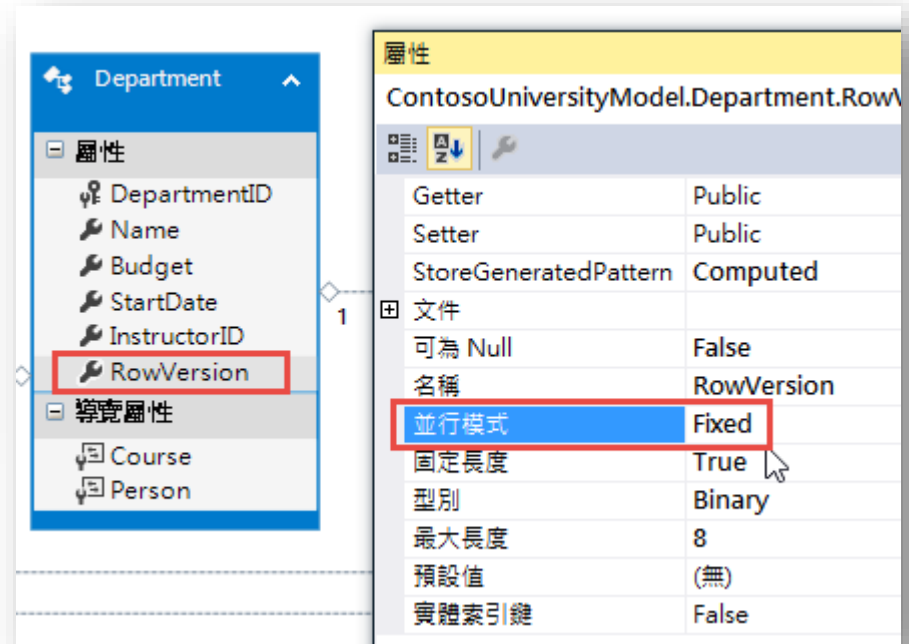
上述 `courseToDelete` 只要有 `EntityKey` 屬性就可以刪除！

並行模式 (Concurrency Mode)

- Entity Framework 內建支援樂觀並行控制 (Optimistic Concurrency)
 - 資料表：擁有一個 RowVersion 欄位 (timestamp)
 - EDMX：在 RowVersion 屬性設定 Fixed 並行模式
 - 若發生併行衝突，會引發 **DbUpdateConcurrencyException**



Column Name	Data Type	Allow Nulls
DepartmentID	int	<input type="checkbox"/>
Name	nvarchar(50)	<input checked="" type="checkbox"/>
Budget	money	<input type="checkbox"/>
StartDate	datetime	<input type="checkbox"/>
InstructorID	int	<input checked="" type="checkbox"/>
RowVersion	timestamp	<input type="checkbox"/>



屬性	
ContosoUniversityModel.Department.RowVersion	
Getter	Public
Setter	Public
StoreGeneratedPattern	Computed
文件	
可為 Null	False
名稱	RowVersion
並行模式	Fixed
固定長度	True
型別	Binary
最大長度	8
預設值	(無)
實體索引鍵	False

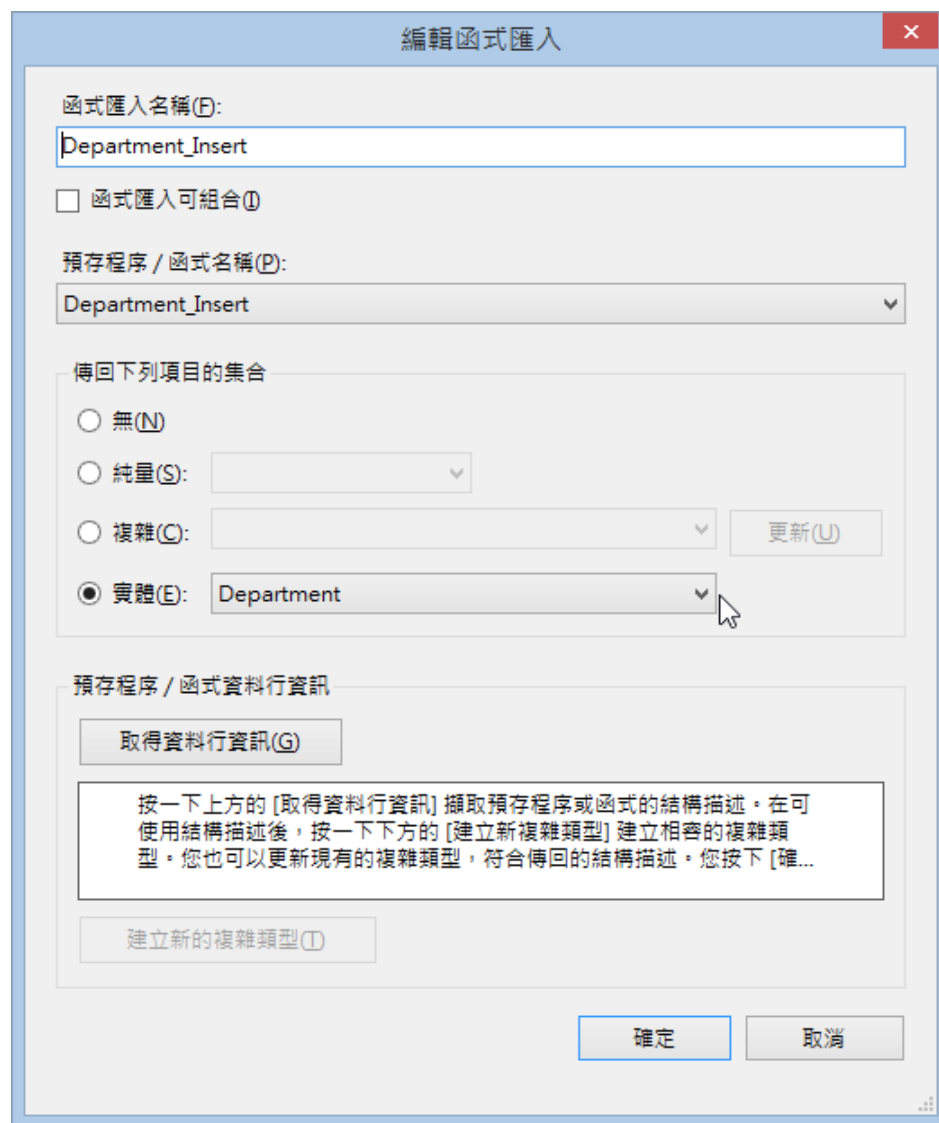
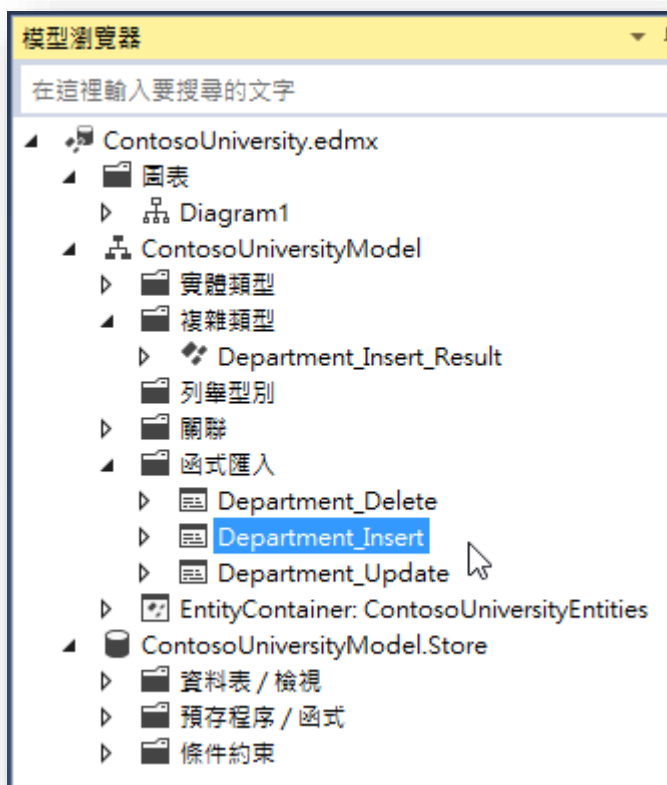
預存程序

- 從資料庫更新模型
- 模型瀏覽器
 - 函式匯入
 - 可自訂回傳型別
 - 複雜類型
- 支援 Table-Valued Function (TVFs)

```
USE [SchoolDB]
GO
/***** Object: UserDefinedFunction [dbo].[GetCourseListByStudentID] */
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [dbo].[GetCourseListByStudentID]
(
    -- Add the parameters for the function here
    @studentID int
)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    select c.courseid, c.coursename, c.Location, c.TeacherId
    from student s left outer join studentcourse sc on sc.studentid = s.studentid
    where s.studentid = @studentID
)
```

```
using (var ctx = new SchoolDBEntities())
{
    //Execute TVF and filter result
    var courseList = ctx.GetCourseListByStudentID(1).Where(c => c.Location == "City1")
        .ToList<GetCourseListByStudentID_Result>();

    foreach (GetCourseListByStudentID_Result cs in courseList)
        Console.WriteLine("Course Name: {0}, Course Location: {1}",
            cs.CourseName, cs.Location);
}
```



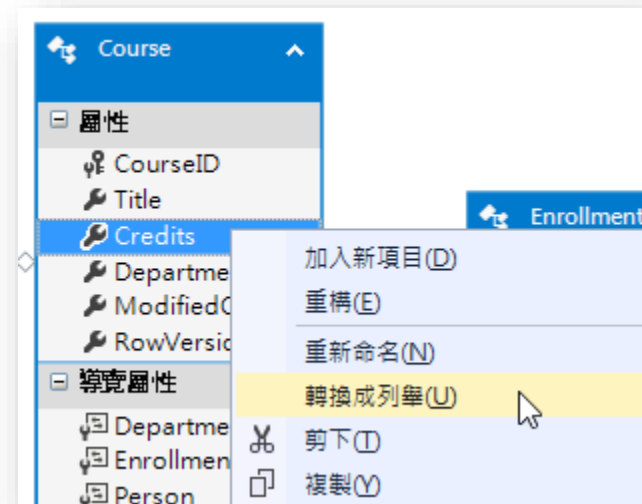
預存程序對應

- 讓實體模型的 CUD 對應指定的預存程序

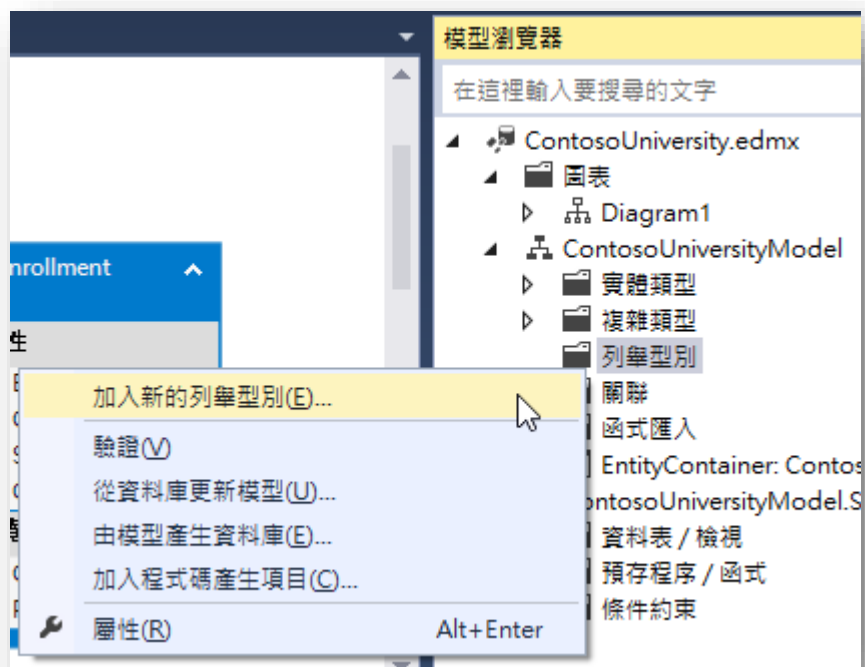
對應詳細資料 - Department		
參數 / 資料行	運算子	屬性
函式		
使用 Department_Insert 插入		
參數		
@ Name : nvarchar	←	Name : String
@ Budget : money	←	Budget : Decimal
@ StartDate : datetime	←	StartDate : DateTime
@ InstructorID : int	←	InstructorID : Int32
結果資料行繫結		
<加入結果繫結>		
使用 Department_Update 更新		
參數		
@ DepartmentID : int	←	DepartmentID : Int32
@ Name : nvarchar	←	Name : String
@ Budget : money	←	Budget : Decimal
@ StartDate : datetime	←	StartDate : DateTime
@ InstructorID : int	←	InstructorID : Int32
@ RowVersion_Original : timestamp	←	RowVersion : Binary
結果資料行繫結		
<加入結果繫結>		
使用 Department_Delete 刪除		
參數		
@ DepartmentID : int	←	DepartmentID : Int32
@ RowVersion_Original : timestamp	←	RowVersion : Binary

使用列舉型別

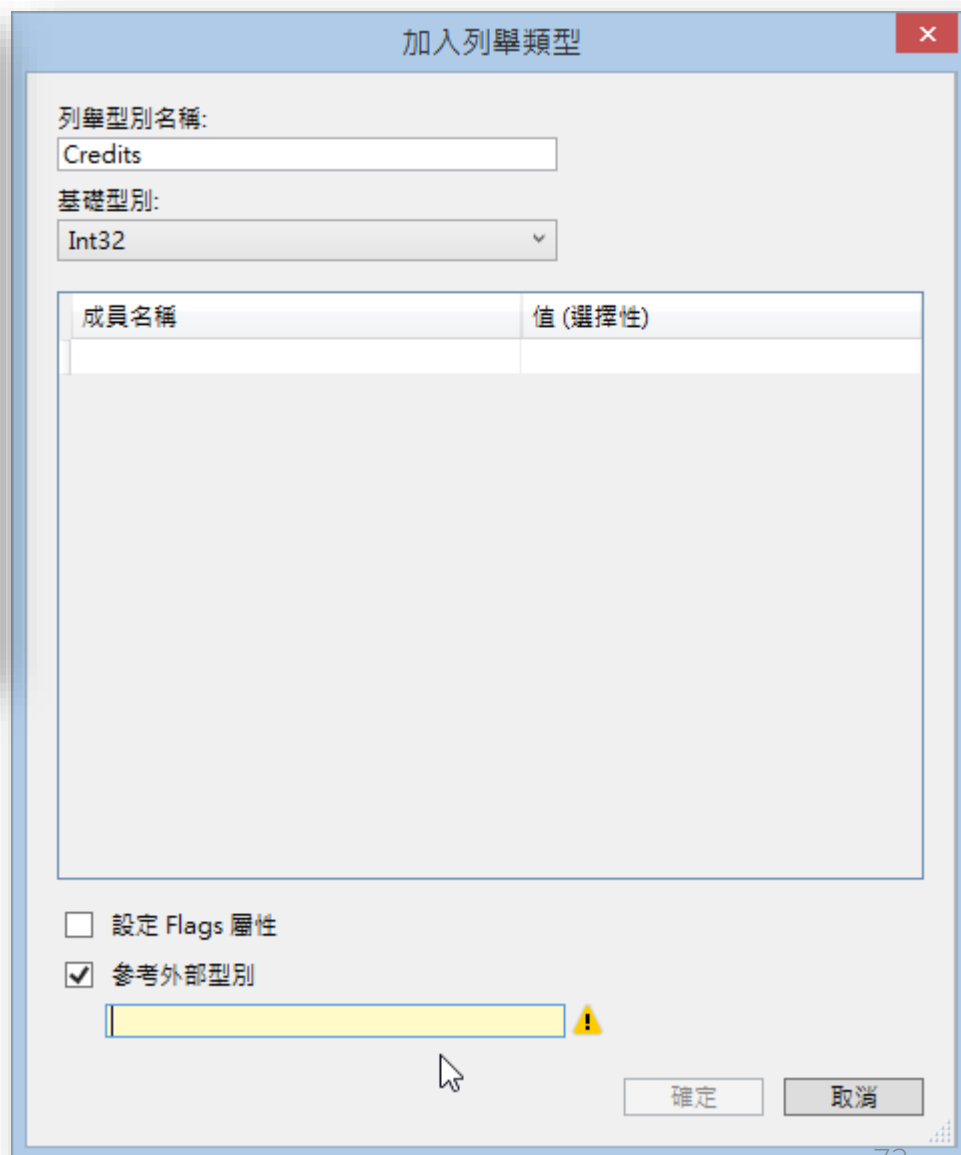
- 列舉支援型別
 - Int16
 - Int32
 - Int64
 - Byte
 - SByte
- 三種建立列舉的方法
 - 轉換成列舉 (如下圖)
 - 自行加入新的列舉型別
 - 使用現有的 Enum 型別



自行加入新的列舉型別



使用現有的 Enum 型別



預先載入 (Eager Loading)

- 載入關聯表格
 - `db.Course.Include("Person")`
 - `db.Course.Include(s => s.Person)`
- 載入多層關聯表格
 - `db.Course.Include("Person.OfficeAssignment")`
 - `db.Course.Include(s => s.Person.OfficeAssignment)`
- 注意事項
 - 使用 lambda 語法必須引用 `System.Data.Entity` 命名空間
`using System.Data.Entity;`

延遲載入 (Lazy Loading)

- EF 預設啟用延遲載入

```
using (var ctx = new SchoolDBEntities())
{
    //Loading students only
    IList<Student> studList = ctx.Students.ToList<Student>();

    Student std = studList[0];

    //Loads Student address for particular Student only (seperate SQL query)
    StudentAddress add = std.StudentAddress;
}
```

- 關閉延遲載入
 - 關閉所有實體物件的延遲載入
 - `db.Configuration.LazyLoadingEnabled = false;`
 - 也可將這段寫在 `DbContext` 類別的建構式裡
 - 將導覽屬性的 `virtual` 關鍵字刪除

延遲載入 (Lazy Loading)

- 關閉延遲載入後也可手動載入資料
 - 載入導覽屬性

```
db.Entry(course).Reference(s => s.Department).Load();
```

- 載入導覽屬性集合

```
db.Entry(course).Collection(s => s.Person).Load();
```

執行任意 SQL 指令

- 針對特定實體進行查詢
 - `db.Course.SqlQuery("select * from xxx");`
- 針對 ViewModel 進行查詢
 - `db.Database.SqlQuery("select * from xxx");`
- 執行無回傳值的 DDL/DML
 - `db.Database.ExecuteSqlCommand("delete from dbo.Course where CourseID<50");`

驗證實體物件

- 在 DbContext 類別覆寫 ValidateEntity 方法

```
protected override System.Data.Entity.Validation.DbEntityValidationResult ValidateEntity
(System.Data.Entity.Infrastructure.DbEntityEntry entityEntry, IDictionary<object, object> items)
{
    if (entityEntry is Course)
    {
        if (String.IsNullOrEmpty(entityEntry.CurrentValues.GetValue<string>("Title")))
        {
            var list = new List<System.Data.Entity.Validation.DbValidationError>();
            list.Add(new System.Data.Entity.Validation.DbValidationError("Title", "Title 欄位必填"));

            return new System.Data.Entity.Validation.DbEntityValidationResult(entityEntry, list);
        }
    }
    return base.ValidateEntity(entityEntry, items);
}
```

驗證實體物件

- 在應用程式中檢查驗證錯誤的方式

```
try
{
    using (var ctx = new SchoolDBEntities())
    {
        ctx.Students.Add(new Student() { StudentName = "" });
        ctx.Standards.Add(new Standard() { StandardName = "" });

        ctx.SaveChanges();
    }
}
catch (DbEntityValidationException dbEx)
{
    foreach (DbEntityValidationResult entityErr in dbEx.EntityValidationErrors)
    {
        foreach (DbValidationError error in entityErr.ValidationErrors)
        {
            Console.WriteLine("Error Property Name {0} : Error Message: {1}",
                              error.PropertyName, error.ErrorMessage);
        }
    }
}
```

驗證實體物件

- 在 SaveChanges() 時，關閉驗證的方式

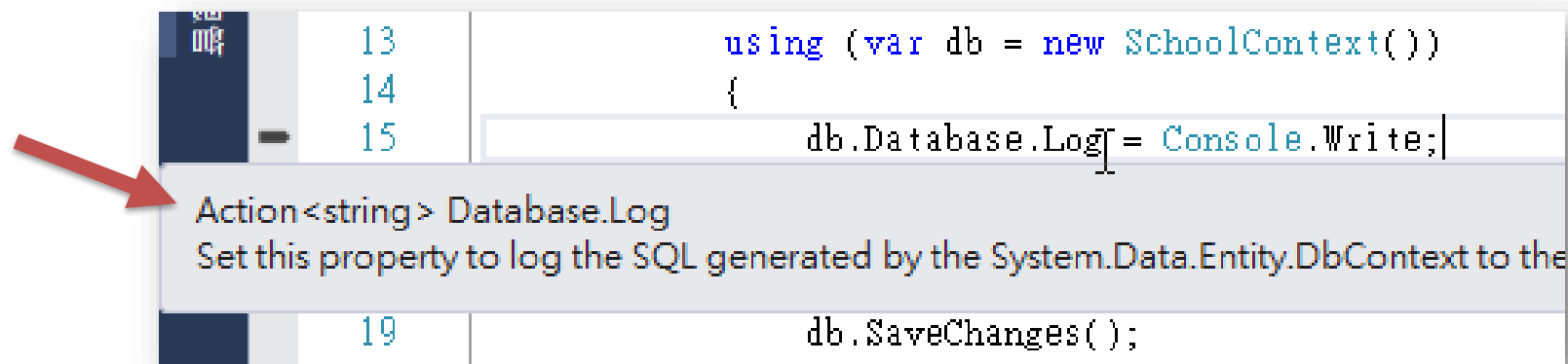
```
db.Configuration.ValidateOnSaveEnabled = false;
```

關於 SQL Server 檢視表 (Views)

- 解決 SQL Server 檢視表 (Views) 無法匯入 EDMX 的問題
 - <http://blog.miniasp.com/post/2013/11/07/Entity-Framework-and-Primary-Keys-on-Views.aspx>
- 解開Entity Framework資料重複之謎
 - <http://blog.darkthread.net/post-2012-06-29-ef-duplicate-data-when-incorrect-pk.aspx>

追蹤 EF 對 DB 下達的 SQL 指令

- 擷取 Database 這端所有執行紀錄
 - `db.Database.Log = Console.Write;`



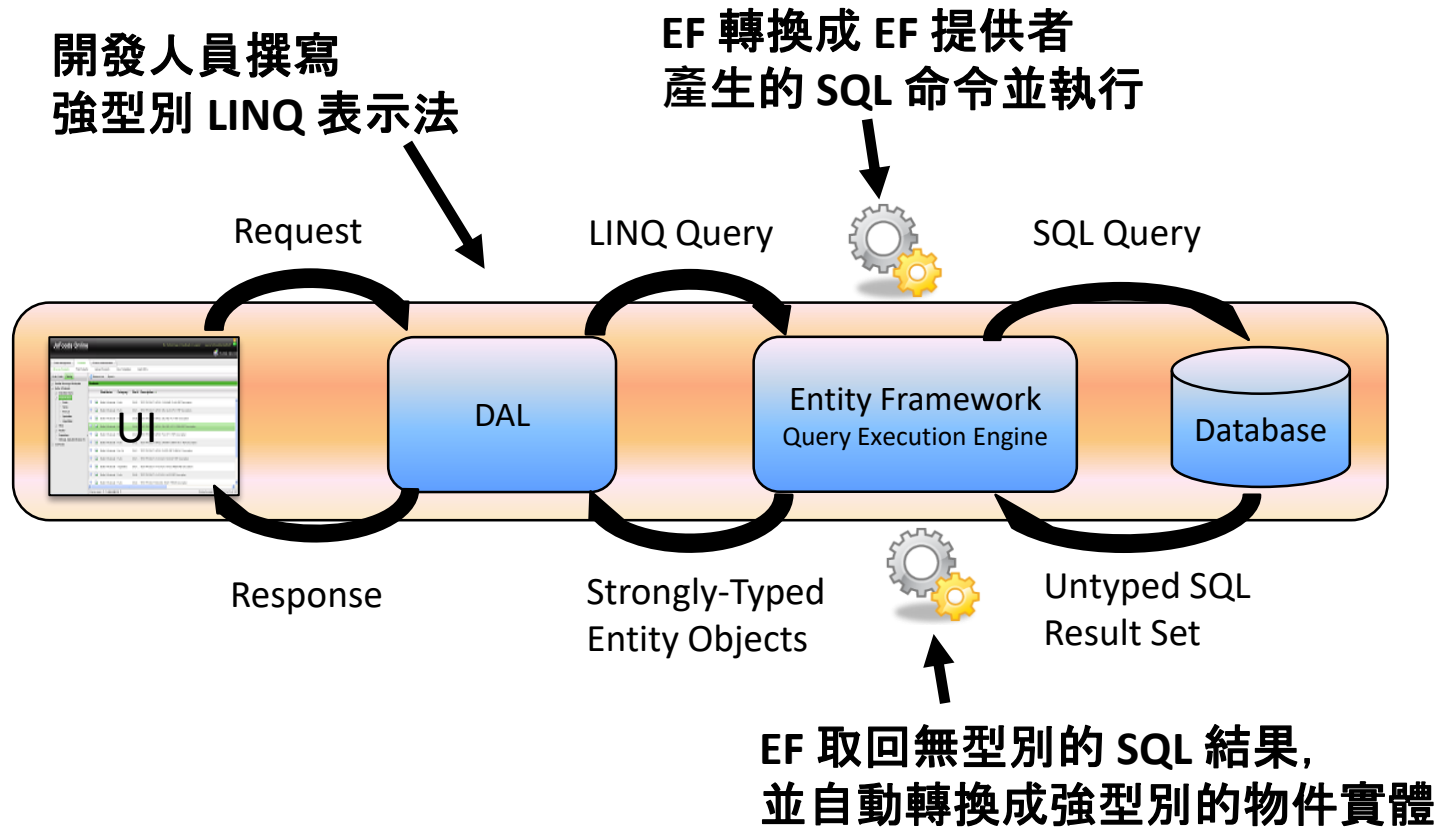
- `db.Database.Log =`
 `(message) => Trace.WriteLine(message);`



Performance Tuning for Entity Framework

Entity Framework 效能調校

預設的查詢過程



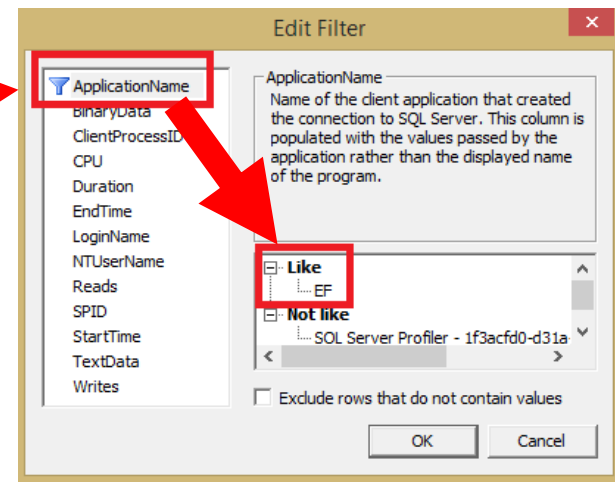
查詢效能

- 你寫的 SQL 效能好嗎？
 - 不要假設 LINQ 產生的語法比你自己寫的好
 - 看起來像 SQL 但執行起來不見得像 SQL
 - EF 幫你取得資料，但不保證有效率
- 你必須對 LINQ Queries 進行剖析 (Profiling)
 - 你 LINQ 的寫法可能對查詢效能有重大影響
 - EF 知道你的模型，但不見得了解你的模型
 - 你幾乎無法從 LINQ 表示法中得知查詢的效能
- 所有的真相都在 SQL 執行計畫中
 - 我們來看看吧...

剖析查詢效能的檢核清單

- 在連接字串設定 Application Name 屬性

```
<connectionStrings>  
  <add name="IndustrialStrengthEfContext"  
        connectionString="Data Source=.;  
        Application Name=EF;  
        Initial Catalog=IndustrialStrengthI  
        Integrated Security=True;  
        MultipleActiveResultSets=True"  
        providerName="System.Data.SqlClient  
</connectionStrings>
```



- 執行 SQL Trace
- 執行 Query Analyzer
- 設定 SET STATISTICS TIME ON/OFF
- 檢視 實際查詢計畫 (Actual Execution Plan)

預存程序

- Entity Framework 偏好自動產生 SQL 語法
 - 預設行為 – 開發人員撰寫 LINQ 語法 → EF 自動產生 SQL
- 預存程序永遠是個選項
 - 透過 EDMX 可以很方便地將預存程序對應成一個簡單的方法(Methods)
- 預存程序的使用方式跟 EF 的開發體驗完全一樣
 - 自動完成 ADO.NET 的底層操作
 - 回傳強型別的實體物件
 - 一樣有變更追蹤機制
- 重要觀念
 - 用對的工具/程式/語言/框架，來解決對的問題！

效能調校最佳實務

- 早期分析/頻繁分析
 - 緩慢的操作/瓶頸/複雜查詢
- 必要動作
 - 連接字串的 Application Name 屬性一定要設
 - Set Statistics Time (On/Off)
 - Actual Execution Plan 與 Estimated Execution Plan
- 別害怕預存程序
 - EF 支援預存程序！
 - 客戶經常混合 EF 查詢與預存程序在應用程式中
 - 這在大型應用程式中是非常常見的應用方法

唯讀查詢

- 停用變更追蹤

- 只要讀取的資料不會被更新，只要加上

```
var customers = ctx.Customers.AsNoTracking().ToList();
```

- AsNoTracking()

- 僅取得資料
 - 不會有查詢快取或變更追蹤操作
 - 大幅降低執行時間與記憶體消耗
 - 當取得大量資料時可大幅提升執行效能
 - 僅影響當下的查詢

非同步查詢

- EF 6 提供非同步操作

- 大多 EF/IQueryable<T> 方法都提供了 async 方法
- 可套用在 C# 5.0 的 Async/Await 設計樣式

SaveChangesAsync()	FirstOrDefaultAsync()	ToListAsync()
LoadAsync()	SumAsync()	MaxAsync
ExecuteSqlCommandAsync()	AnyAsync()	FindAsync()

- 快速且流暢的經驗

- 提升用戶端回應性/伺服器延展性
- 執行長時間查詢時可避免影響到主要執行執行緒的運作
- 啟用 async 操作不需要管理背景執行緒

SQL Trace Filtering

The screenshot shows the 'Trace Properties' dialog box with the 'Events Selection' tab active. The dialog has a title bar with a close button. Below the tabs, there is a descriptive text: 'Review selected events and event columns to trace. To see a complete list, select the "Show all events" and "Show all columns" options.' A table lists various events with checkboxes for each column. 'RPC:Completed' and 'SQL:BatchCompleted' are selected and highlighted with red boxes. Below the table, a description for 'SQL:BatchCompleted' is shown. At the bottom right, there are buttons for 'Show all events', 'Show all columns', 'Column Filters...', 'Organize Columns...', 'Run', 'Cancel', and 'Help'.

Events	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration	ClientProcess
<input type="checkbox"/> Security Audit									
<input type="checkbox"/> Audit Login	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					<input type="checkbox"/>
<input type="checkbox"/> Audit Logout		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Sessions									
<input type="checkbox"/> ExistingConnection	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Stored Procedures									
<input checked="" type="checkbox"/> RPC:Completed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> SQL:BatchCompleted	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> SQL:BatchStarting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					<input type="checkbox"/>

SQL:BatchCompleted
Occurs when the Transact-SQL statement has completed.

LoginName (no filters applied)
Name of the login of the user (either SQL Server security login or the Windows login credentials in the form of DOMAIN \Username).

☐ Show all events
☐ Show all columns

Column Filters...
Organize Columns...

Run Cancel Help

其他效能追蹤工具

- SQL Profiler (SQL Server)
- IntelliTrace (Visual Studio)
- Red Gate Database Profiler
- EF Profiler (Hibernating Rhinos)
- ORM Profiler (Solution Design, Inc.)
- MiniProfiler-EF

其他效能調校文章

- [Performance Considerations for EF 4, 5, and 6](#)
- [Performance Considerations \(Entity Framework\)](#)
- [Tips to improve Entity Framework Performance](#)
- [Entity Framework Performance and What You Can Do About It](#)
- [SQL Server Performance EF Performance Optimization](#)

聯絡資訊

- The Will Will Web

記載著 Will 在網路世界的學習心得與技術分享

- <http://blog.miniasp.com/>

- Will 保哥的技術交流中心 (臉書粉絲專頁)

- <http://www.facebook.com/will.fans>

- Will 保哥的推特

- https://twitter.com/Will_Huang