多奇·教育訓練

ASP.NET Core 5 開發實戰

資料存取篇 (Entity Framework Core)

多奇數位創意有限公司

技術總監 黃保翕（Will 保哥）

https://blog.miniasp.com

About Entity Framework Core

關於 Entity Framework Core

什麼是 Entity Framework

- 一個 ORM 框架
- 可大幅減少開發時期資料存取的程式碼
- 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

什麼是 Entity Framework Core

- 2016 年微軟將 Entity Framework 6 全部重寫 (打掉重練)
  - https://github.com/aspnet/EntityFrameworkCore
  - **資料庫提供者** (Database Providers) 完全以 NuGet 發行
    - 內建 SQL Server, SQLite, InMemory, Cosmos 四種
  - **跨平台**與**開放原始碼** (EF Core) (EF6)
- 大部分 APIs 都跟 Entity Framework 6 一模一樣！
  - EF6 & EF Core Feature Comparsion
- 支援兩種開發模型 (開發工作流程)
  - **Code Only** (DB First) 與 **Code First**

如何選擇 EF6 或 EF Core

- **新專案，選 Entity Framework Core**
  - 你的應用程式是 **.NET Core** 且需要**跨平台**支援
  - EF Core 可以支援你的應用程式所需的資料存取能力
    - 如果有缺少必要的功能，就必須思考是否採用
    - 請參考 **EF Core Roadmap** 看未來是否會支援！
- **新專案，選 EF6**
  - 你的應用程式是 **.NET Framework 4.0+** 與 **Windows** 平台
  - EF6 可以支援你的應用程式所需的資料存取能力
- **舊專案**
  - **不建議立刻升級**，除非有**立竿見影**的效果
  - 簡單說：**EF6 無法直升 EF Core，只能將程式碼逐步轉移過去！**

關於 ORM 與 DDD

- ORM (Object Relational Mapping)
  - 透過「**物件導向思維**」來管理「**關聯式資料**」
  - 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**
- DDD (Domain Driven Design)
  - 介面展示層 (Presentation Layer)　負責**顯示**和**接受輸入**
  - 應用程式層 (Application Layer)　僅包含**流程控制邏輯**
  - 領域模型層 (Domain Layer)　包含整個應用的所有**商業邏輯**
  - 基礎結構層 (Infrastructure Layer)　提供整個應用程式的**基礎服務**

ASP.NET Core 的 DDD 範例專案

- Clean Architecture
  - A starting point for Clean Architecture with ASP.NET Core
  - Clean Architecture - Visual Studio Marketplace (VS 專案範本)

- eShopOnWeb
  - Microsoft eShopOnWeb ASP.NET Core Reference Application
  - Tour of Microsoft's Reference ASP NET Core App eShopOnWeb
  - Architect Modern Web Applications with ASP.NET Core and Azure (免費電子書)
  - .NET 微服務：容器化 .NET 應用程式的架構 (電子書正體中文翻譯)

Entity Framework Core Quick Start

Entity Framework Core 新手上路

使用 .NET CLI 初始化 EF Core 設定

- 建立專案範本
  - `dotnet new console -n efc3 && cd efc3`
- 安裝 EF Core 5.0 的 NuGet 套件
  - `dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 5.0.6`
  - `dotnet add package Microsoft.EntityFrameworkCore.Design --version 5.0.6`
  - `dotnet add package Microsoft.EntityFrameworkCore.Tools --version 5.0.6`
- 其他 EF Core 資料庫提供者 (Database Providers)
  - `dotnet add package Microsoft.EntityFrameworkCore.InMemory --version 5.0.6`
  - `dotnet add package Microsoft.EntityFrameworkCore.Sqlite --version 5.0.6`
  - `dotnet add package Microsoft.EntityFrameworkCore.Cosmos --version 5.0.6`
  - `dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL --version 5.0.6`
  - `dotnet add package MySql.EntityFrameworkCore --version 5.0.3.1`
  - `dotnet add package Oracle.EntityFrameworkCore --version 5.21.1`
    - Announcing ODP.NET 19.10 Release: New .NET 5 and Bulk Copy Support
    - Introducing ODP.NET 21c: EF Core 5, JSON, and More

使用 PowerShell 初始化 EF Core 設定

- 安裝 EF Core 5.0 的 NuGet 套件 (Package Manager Console Refs)
  - `Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 5.0.6`
  - `Install-Package Microsoft.EntityFrameworkCore.Design -Version 5.0.6`
  - `Install-Package Microsoft.EntityFrameworkCore.Tools -Version 5.0.6`
- 其他 EF Core 資料庫提供者 (Database Providers)
  - `Install-Package Microsoft.EntityFrameworkCore.InMemory -Version 5.0.6`
  - `Install-Package Microsoft.EntityFrameworkCore.Sqlite -Version 5.0.6`
  - `Install-Package Microsoft.EntityFrameworkCore.Cosmos -Version 5.0.6`
  - `Install-Package Npgsql.EntityFrameworkCore.PostgreSQL -Version 5.0.6`
  - `Install-Package MySql.EntityFrameworkCore -Version 5.0.3.1`
  - `Install-Package Oracle.EntityFrameworkCore -Version 5.21.1`

建立模型類別與資料內容類別

- 建立**模型類別** (Todo.cs)
  - 可使用 `prop` 程式碼片段
  - 請建立兩個屬性即可（`int Id , string Item`）

- 建立**資料內容類別** (TodoContext.cs)
  - 可使用 `ef-dbcontext` 程式碼片段

- 建立**資料內容工廠類別** (TodoContextFactory.cs)
  - 可使用 `ef-dbcontext-factory` 程式碼片段
  - 只有 **非 ASP.NET Core 專案** 才需要特別建立**資料內容工廠類別**
    - 請參考 Design-time DbContext Creation 文件說明

簡介 POCO (Plain Old CLR Object)

- 在 ORM 領域中經常被使用 POCO 當作資料模型的定義
- 用**最簡單**且**無負擔**的方式表達資料模型
- 可當 DTO(Data Transfer Object) 或 DAO(Data Access Object) 使用
  - 序列化 (Serialize)
  - 反序列化 (Deserialize)

建立 Blog 實體模型

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public int Rating { get; set; }
    public List<Post> Posts { get; set; }
}
```

- Conventions in Entity Framework Core

建立 Post 實體模型

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

- Conventions in Entity Framework Core

建立 DbContext 資料內容類別

```
public class BlogContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    public BlogContext (DbContextOptions<BlogContext> options)
        : base (options) { }
}
```

Design-time DbContext Creation - EF Core | Microsoft Docs

認識資料庫移轉

- 傳統資料庫應用開發
  - 主要採用 Database First 開發流程
  - 資料庫版本控管不易 (較容易被忽略)
  - 版控中的 **程式碼** 與 **資料庫結構描述** 無法有效同步
- 採用 EF Core 的資料庫應用開發
  - 可選用 **Database First** 或 **Code First** 開發流程
  - 可完全交由 EF Core 管理**模型變更歷程**與**資料庫變更指令碼**
  - 可透過 dotnet-ef 工具自動管理資料庫移轉作業
  - 版控中的 **程式碼** 與 **資料模型** (**資料庫結構描述**) 可有效同步 (由**程式碼**來對**資料庫結構**進行**版本控制**)

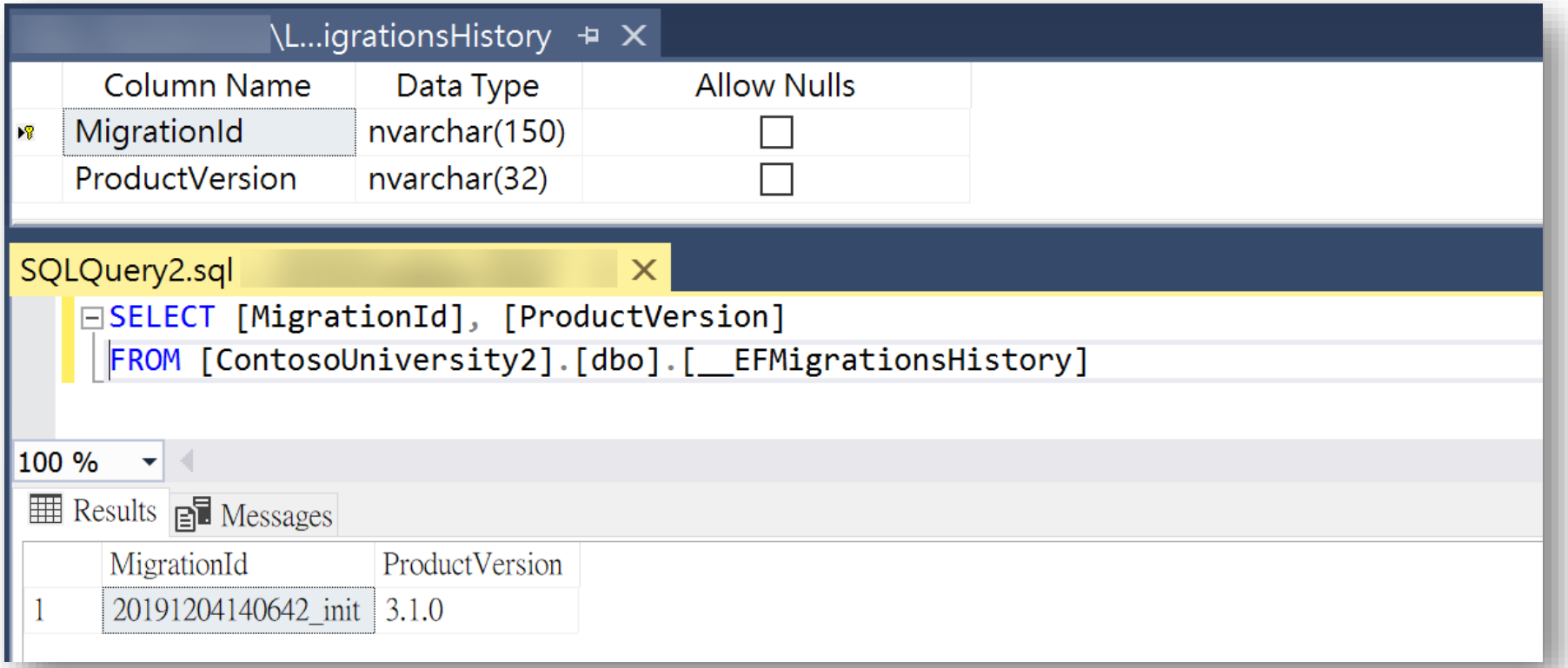
資料庫移轉與初始化資料庫 (.NET CLI)

- 新增資料庫移轉設定 (執行完畢請查看 **Migrations** 資料夾)
  - `dotnet ef migrations add init`
- 移除資料庫移轉設定 (預設會刪除最後一個移轉設定，但會比對資料庫中的紀錄)
  - `dotnet ef migrations remove`
- 產生資料庫移轉變更指令碼
  - `dotnet ef migrations script <FROM> <TO> -o output.sql`
    - 其中 `<FROM>` 可以是 `0` (代表從最早的版本開始產生指令碼)
- 更新資料庫 (若資料庫不存在則會自動建立新資料庫)
  - `dotnet ef database update -v` (套用變更到目前最新版本)
  - `dotnet ef database update 0` (回復所有資料庫移轉變更)
  - `dotnet ef database update <TO>` (套用變更到特定版本)
- 刪除資料庫
  - `dotnet ef database drop`

資料庫移轉與初始化資料庫 (PowerShell)

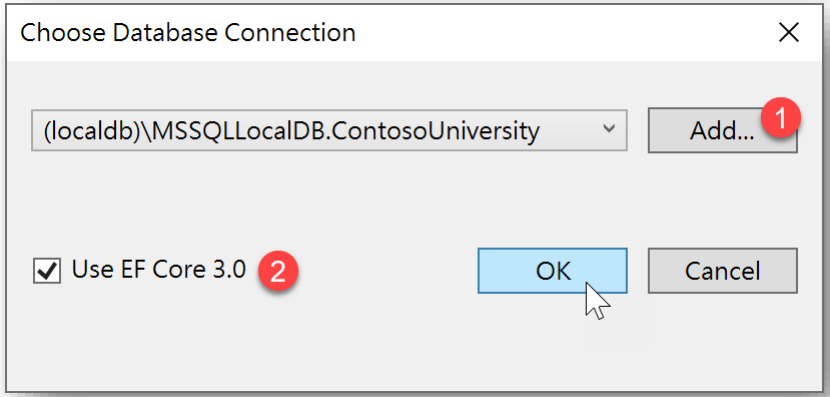
- 新增資料庫移轉設定 (執行完畢請查看 **Migrations** 資料夾)
  - `Add-Migration Init`
- 移除資料庫移轉設定
  - `Remove-Migration`
- 產生資料庫移轉變更指令碼
  - `Script-Migration -From <FROM> -To <TO> -Output <FILENAME>.sql -Verbose`
    - 其中 `<FROM>` 可以是 `0` (代表從最早的版本開始產生指令碼)
- 更新資料庫 (若資料庫不存在則會自動建立新資料庫)
  - `Update-Database -Verbose`
  - `Update-Database -Migration <TO>`
    - 其中 `<TO>` 是指你希望更新資料庫到哪個版本
- 刪除資料庫
  - `Drop-Database`

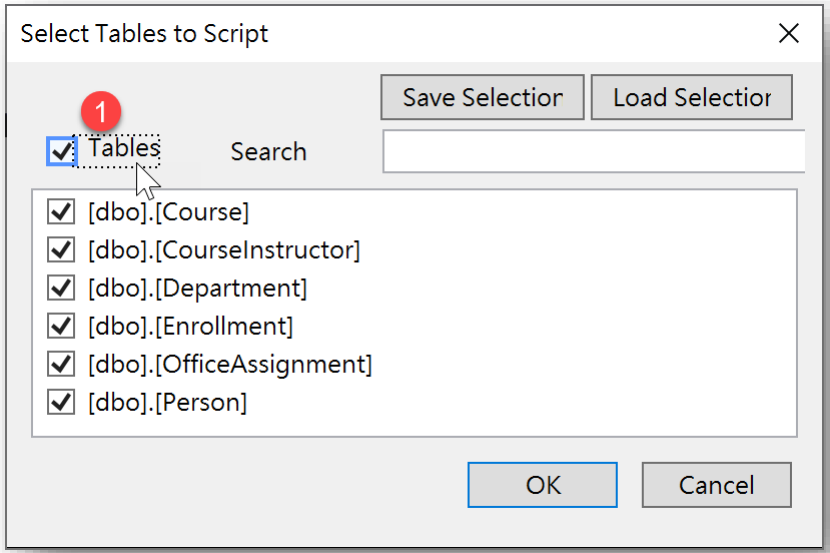
認識 __EFMigrationsHistory 資料表

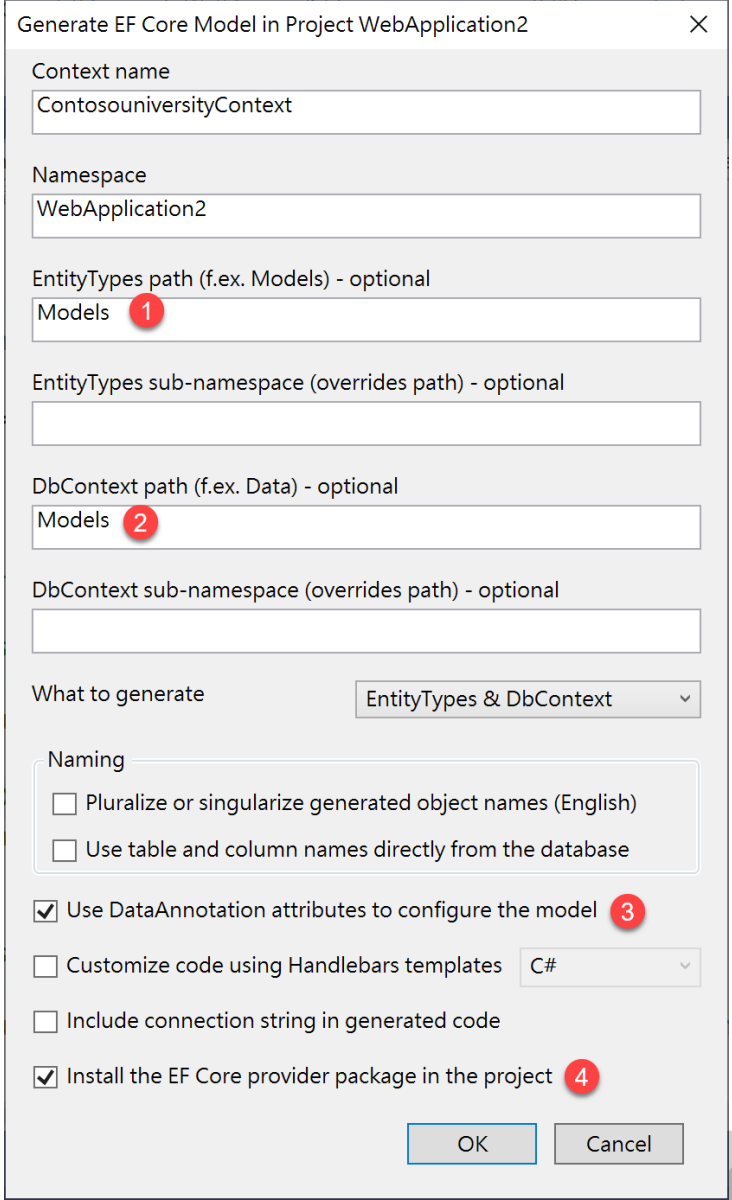


用 VS2019 快速產生 EF Core 所需模型

- 安裝 EF Core Power Tools 擴充套件 (須重開 Visual Studio 2019)
- 建立 **ASP.NET Core Web 應用程式** 專案
  - 選擇 **Web 應用程式 (模型-檢視-控制器)** 專案範本
- 使用資料庫反向工程建立資料庫實體模型
  - [方案總管] / [專案] / 滑鼠右鍵 / [EF Core Power Tools] / [Reverse Engineer]
  - 新增資料庫連線 / 勾選 [**Use EF Core 5**]
  - 選取要產生程式碼的資料庫表格 (Select Tables to Script)
  - 設定**命名空間**、輸出的**資料夾路徑**、額外設定的程式碼產生器選項
- 執行結果
  - 會自動安裝 Microsoft.EntityFrameworkCore.SqlServer 套件
  - 會自動產生 **efpt.config.json** 設定檔 (用來儲存選項設定)


Choose Database Connection
(localdb)\MSSQLLocalDB.ContosoUniversity
Add...
1
Use EF Core 3.0
2
OK
Cancel


Select Tables to Script
Save Selection
Load Selection
1
Tables
Search
[dbo].[Course]
[dbo].[CourseInstructor]
[dbo].[Department]
[dbo].[Enrollment]
[dbo].[OfficeAssignment]
[dbo].[Person]
OK
Cancel


Generate EF Core Model in Project WebApplication2
Context name
ContosouniversityContext
Namespace
WebApplication2
EntityTypes path (f.ex. Models) - optional
Models
1
EntityTypes sub-namespace (overrides path) - optional
DbContext path (f.ex. Data) - optional
Models
2
DbContext sub-namespace (overrides path) - optional
What to generate
EntityTypes & DbContext
Naming
Pluralize or singularize generated object names (English)
Use table and column names directly from the database
Use DataAnnotation attributes to configure the model
3
Customize code using Handlebars templates
C#
Include connection string in generated code
Install the EF Core provider package in the project
4
OK
Cancel

用命令列工具快速產生 EF Core 所需模型

- Getting Started with EF Core on ASP.NET Core with an Existing Database
- **.NET Core CLI**

```
dotnet ef dbcontext scaffold
"Server=(localdb)\MSSQLLocalDB;Database=Blogging;Trusted_Connection=True"
Microsoft.EntityFrameworkCore.SqlServer -o Models
```

- **Windows PowerShell (Visual Studio 2019 套件管理器主控台)**

```
Scaffold-DbContext
"Server=(localdb)\MSSQLLocalDB;Database=Blogging;Trusted_Connection=True"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

- **EntityFrameworkCore.Generator (.NET Global Tool)**

```
efg generate -p SqlServer -c
"Server=(localdb)\MSSQLLocalDB;Database=Blogging;Trusted_Connection=True"
```

注意：此工具會將實體模型輸出到當下目錄的 **Data** 子資料夾！

EntityFrameworkCore.Generator

- 快速上手
  - `dotnet tool install --global EntityFrameworkCore.Generator`
  - `mkdir efcg && cd efcg`
  - `dotnet new console`
  - `dotnet user-secrets init --id 50e6f746-1a06-4c3d-85e6-adc5aaf8d41f`
  - **`efg initialize -c`** `"Server=(localdb)\MSSQLLocalDB;Database=ContosoUniversity;Trusted_Connection=True" -p SqlServer --id 50e6f746-1a06-4c3d-85e6-adc5aaf8d41f`
  - `dotnet user-secrets list`
  - `dotnet add package FluentValidation`
  - `dotnet add package AutoMapper`
  - **`efg generate`**
- 相關連結
  - GitHub / NuGet / Documentation / 完整 generation.yml 設定檔範例

基礎 EF Core 資料操作：新增資料

```
using (var db = new BlogContext())
{   // db.Database.EnsureCreated();
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}

using (var db = new BlogContext())
{
    var blog = new Blog {
        Url = "http://example.com",
        Posts = new List<Post>() { new Post { Title = "Post Title" } }
    };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

基礎 EF Core 資料操作：查詢資料

```
using (var db = new BlogContext())
{
    var blog = db.Blogs.Find(1);
}

using (var db = new BlogContext())
{
    var blogs = db.Blogs.Where(b => b.Rating > 3).OrderBy(b => b.Url);
}

using (var db = new BlogContext())
{
    var blogs = from b in db.Blogs where b.Rating > 3 orderby b.Url;
}
```

EF Core 資料查詢的運作方式

- https://docs.microsoft.com/zh-tw/ef/core/querying/how-query-works
- 查詢執行
  - 呼叫 LINQ 運算子時，只會在**記憶體中**建立一個**表達查詢**的物件
  - 只有在**取得查詢結果**時，才會將產生的 SQL 傳送到資料庫執行
    - 透過 foreach 跑迴圈取得所有結果
    - 執行像是 ToList、ToArray、Single、Count 諸如此類的方法
    - 透過資料繫結綁定查詢到 UI 控制項中 (WinForm, WPF, ...)
- 查詢的生命週期
  - **LINQ 查詢**由 EF Core **編譯**起來 (**編譯結果**會被**快取**)
  - 將**編譯好的 LINQ 查詢**提交給 EF Core **資料庫提供者**解析與執行 (取回資料)
  - 替**每筆資料**建立**實體物件** (entity instance) 並**快取**在 DbContext 物件中

複雜查詢運算子

- Join
- GroupJoin
- SelectMany
- GroupBy
- LEFT JOIN

基礎 EF Core 資料操作：附加資料

```
using (var db = new BlogContext())
{
    var blog = new Blogs {
        BlogId = 1,
        Url = "https://blog.miniasp.com",
        Rating = 5
    };

    db.Attach(blog); // 將 blog 加入 EF 物件快取中（不透過資料庫取出資料）
}
```

基礎 EF Core 資料操作：更新資料

```
using (var db = new BlogContext())
{
    var blog = db.Blogs.Find(1);
    blog.Rating++;
    db.SaveChanges();
}

using (var db = new BlogContext())
{
    var blog = new Blogs { BlogId = 1 };
    db.Update(blog); // 告訴 EF Core 這筆 blog 需要被更新
    db.SaveChanges();
}
```

基礎 EF Core 資料操作：刪除資料

```
using (var db = new BloggingContext())
{
    var blog = db.Blogs.Find(1);
    db.Blogs.Remove(blog);
    db.SaveChanges();
}

using (var db = new BloggingContext())
{
    var blog = new Blogs { BlogId = 1 );
    db.Blogs.Remove(blog);
    db.SaveChanges();
}
```

關於 EF Core 資料庫提供者的限制

- SQLite EF Core Database Provider Limitations
  - Modeling limitations
    - Schemas
    - Sequences
  - Query limitations
    - DateTimeOffset
    - Decimal
    - TimeSpan
    - UInt64
  - Migrations limitations
    - 有許多 Schema 操作都不支援，使用時要特別注意！
- EF Core Azure Cosmos DB Provider Limitations

Entity Framework Core Concepts

Entity Framework Core 核心觀念

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) 與 交易處理 (Transactions)
  - 依據實體物件的狀態 (EntityState) 執行資料的新增、更新、刪除等操作
- 物件快取 (Object 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) (Model Building)
  - 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)

EF 的實體物件之間的關聯 (Relationships)

- 一對多（One-to-Many Relationship）
  - 認識「**導覽屬性**」(Navigation Properties)
  - Managing One To Many Relationships With EF Core
  - EF Core One To Many Relationships Conventions
  - Configuring One To Many Relationships in EF Core
- 多對多（Many-to-Many Relationship）
  - EF Core **不支援**多對多關聯、只能建立兩個**一對多**關聯
  - Configuring Many To Many Relationships in EF Core
- 一對一（One-to-One relationship）
  - EF Core One To One Relationships Conventions
  - Configuring One To One Relationships In EF Core

與實體關聯有關的專有名詞

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

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

主體實體 (principal entity)

主索引鍵 (Principal key)

集合導覽屬性
(Collection navigation property)

相依實體 (Dependent entity)

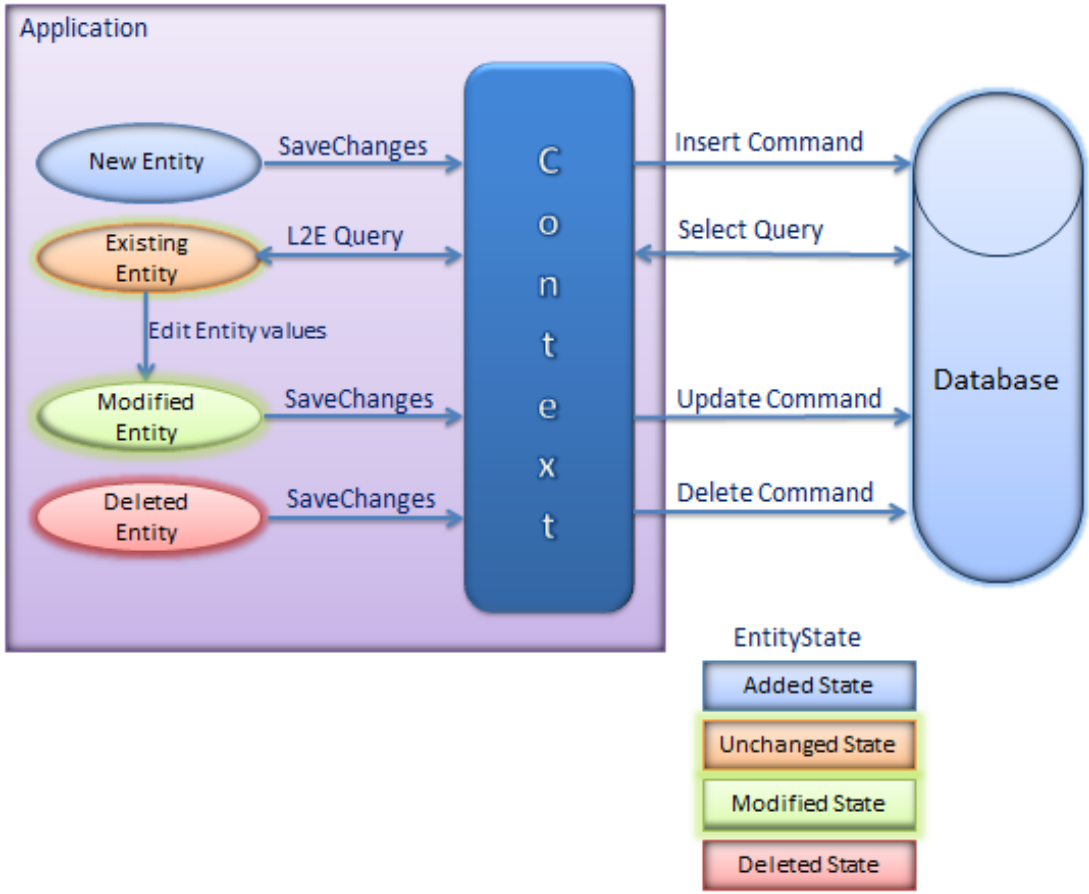
外部鍵 (Foreign key)

參考導覽屬性
(Reference navigation property)

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
- 每個被取出的物件實體 (Entity) 都擁有 EntityState 實體狀態
- Microsoft.EntityFrameworkCore.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

EF 執行生命週期



Entity Framework Core Deep Dive

Entity Framework Core 深入探討

深入了解 DbSet<T> 類別

方法名稱	回傳型別	說明
`Find` `FindAsync`	`TEntity`	傳入 P.K. 參數，回傳單一筆實體物件

`Add`
`AddAsync`
`AddRange`
`AddRangeAsync`
	`EntityEntry<TEntity>`	將一個 POCO 物件加入 DbSet 並將物件標示為 **Added** 狀態

`Remove` `RemoveRange`	`EntityEntry<TEntity>`	將傳入的實體物件標示為 **Deleted** 狀態
`Update` `UpdateRange`	`EntityEntry<TEntity>`	將不受變更追蹤的物件加入變更追蹤並標示為 Modified 狀態
`Attach` `AttachRange`	`EntityEntry<TEntity>`	將不受變更追蹤的物件加入變更追蹤並標示為 Unchanged 狀態

https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbset-1?view=efcore-5.0

深入了解 EntityEntry<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) → PropertyValues
    - 目前內容 (CurrentValues) → PropertyValues
    - 重新載入資料庫資料 → Reload()
  - 範例程式

```
EntityEntry ce = db.Entry(course);
if (ce.State == EntityState.Modified) {
    course.ModifiedOn = DateTime.Now;
}
```

深入了解 PropertyValues 類別

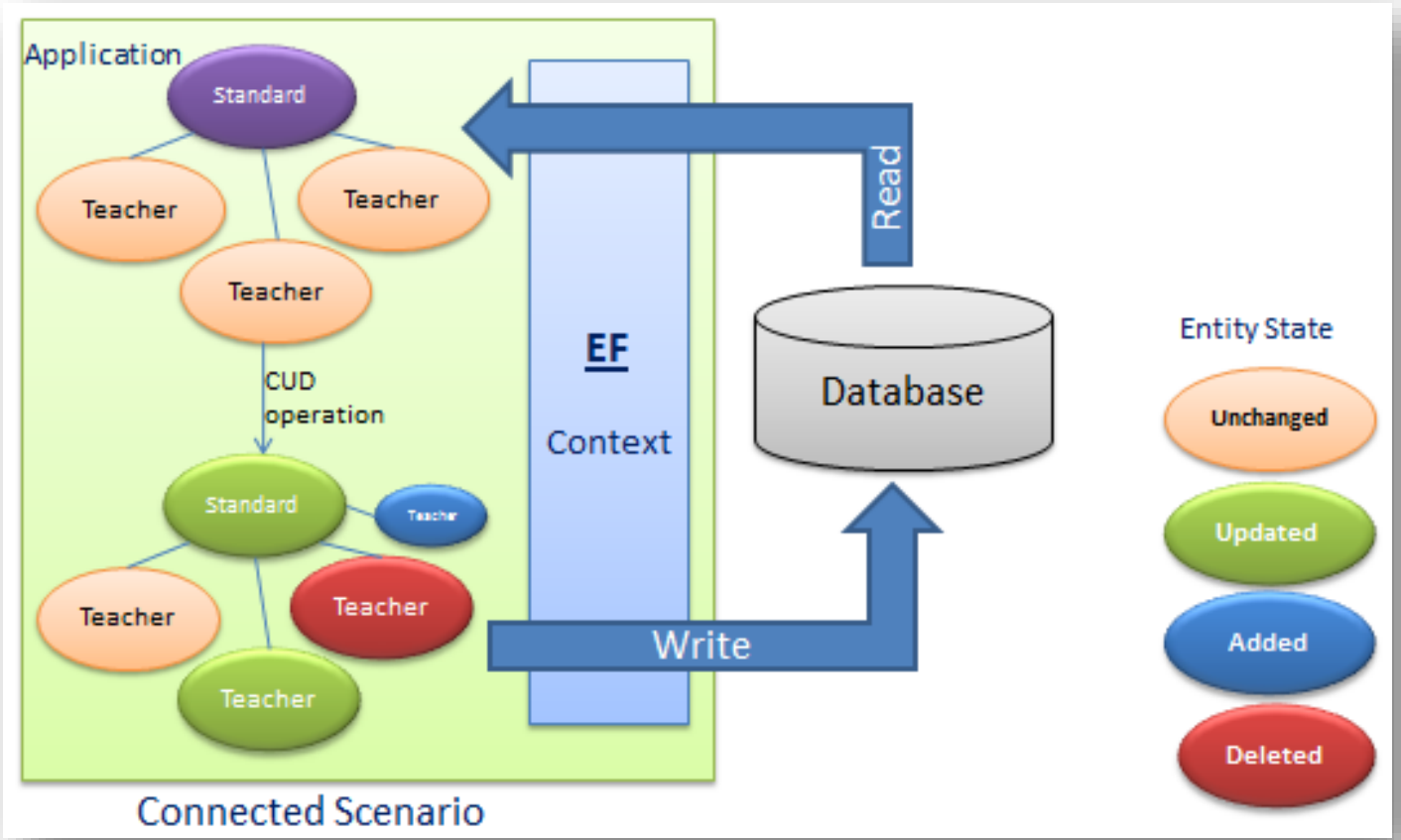
- PropertyValues
  - 幫助你取得或設定**實體物件**的特定屬性
    - Properties　　所有屬性清單
    - GetValue<TValue>(propName)　　取得特定屬性值
    - SetValues(object)　　設定新的屬性值
  - 範例程式

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

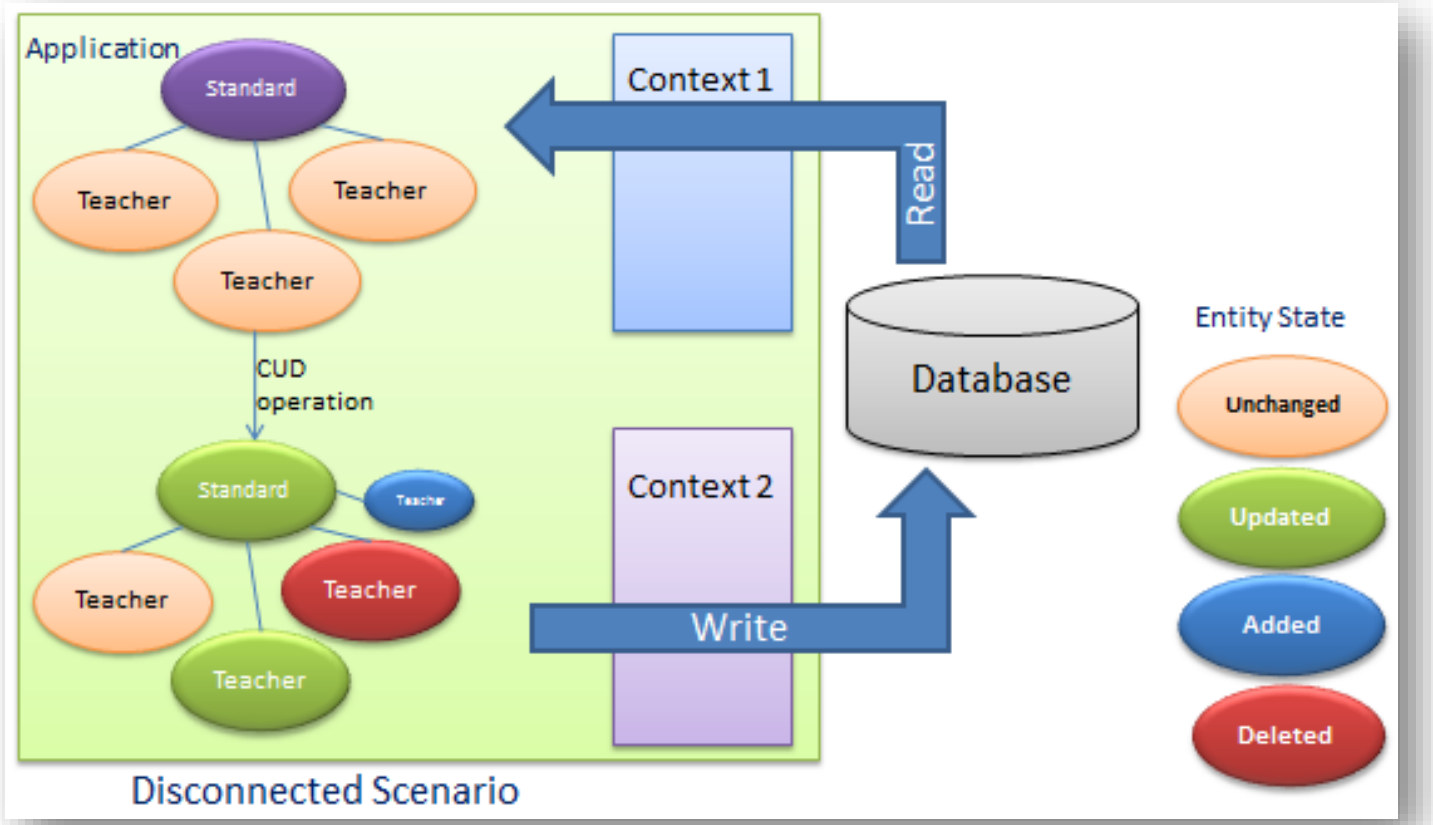
深入了解變更追蹤機制

```
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 離線模式的困難點

- 主要困難之處
  - 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 離線模式的操作方法

- Disconnected entities
- 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;
- 離線模式的更新方法
  - db.Entry(course).State = EntityState.Modified;
- 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！

EF Core 3.0 無索引鍵實體類型

- 在 EF Core 2.1 時稱為 **Query Types** (查詢類型)
- 從 EF Core 3.0 開始則稱為 **Keyless Entity Types**
  - 可以用來對應到一個**資料庫檢視表** (View)
  - 可以用來對應到一個**沒有主索引鍵**的類別 (Keyless entity types)
- 應用情境
  - 你需要執行 原始 SQL 查詢 (Raw SQL Queries) 並對應到特定實體模型
  - 你需要對應實體模型到一個不含主索引鍵的資料庫檢視表 (Views)
  - 你需要對應實體模型到一個沒有定義主索引鍵的資料表 (Tables)
  - 你需要對應實體模型到 **`OnModelCreating()`** 定義的 LINQ 查詢

EF Core 3 紀錄機制 (Logging)

- EF Core 3+ 的紀錄機制會自動採用 ASP.NET Core 的紀錄設定
- 任何**非 ASP.NET Core 應用程式**，都需要實作 ILoggerFactory

```
public static readonly ILoggerFactory MyLoggerFactory =
  LoggerFactory.Create(builder => {
    builder.AddConsole();
  });
```

- 然後將其註冊到 **DbContextOptionsBuilder** 裡面

```
services.AddDbContext<ContosoUniversityContext>(options =>
  options
    .UseLoggerFactory(MyLoggerFactory)
    .UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

- 文章：ASP.NET Core 如何紀錄 EF Core 5.0 自動產生的 SQL 命令

紀錄過濾器 (Log filtering)

```
public static readonly ILoggerFactory MyLoggerFactory =
    LoggerFactory.Create(builder =>
    {
        builder
            .AddFilter((category, level) =>
                category == DbLoggerCategory.Database.Command.Name
                && level == LogLevel.Information)
            .AddConsole();
    });
```

載入關聯資料的三種方法

- https://docs.microsoft.com/en-us/ef/core/querying/related-data
- **預先載入** (Eager loading)
  - 在**第一次查詢資料**時，就**直接載入**所有**關聯表格**的資料

```
var blogs = context.Blogs.Include(blog => blog.Posts).ToList();
```

- **明確載入** (Explicit loading)
  - 當有需要的時候，你可以**明確載入**特定**導覽屬性**關聯下的資料

```
context.Entry(blog).Collection(b => b.Posts).Load();
context.Entry(blog).Reference(b => b.Owner).Load();
```

- **延遲載入** (Lazy loading)
  - 當透過**導覽屬性**讀取關聯資料時，會**自動載入**關聯資料 (需額外設定)

```
dotnet add package Microsoft.EntityFrameworkCore.Proxies
.AddDbContext<BloggingContext>(b => b.UseLazyLoadingProxies() …… );
```

  - 需特別注意 Related data and serialization 的問題！

交易處理技巧 (Using Transactions)

- 預設 EF Core 就會在每次執行 **SaveChanges()** 時進行交易處理
- 並非所有資料庫提供者都支援交易處理
- 主要的交易處理方式
  - 在多個 **SaveChanges()** 之間進行交易處理 (說明)
  - 透過外部的 **DbTransactions** 進行交易處理 (說明)
  - 在多個 **DbContext** (資料內容類別) 之間進行交易處理 (說明)
  - 透過 **System.Transactions** 的 **TransactionScope** 進行交易處理 (說明)
    - .NET Core 2.1 之後才有此能力
    - 但不支援任何**分散式交易**處理能力
    - 不是所有資料庫提供者都可以進行此類交易

在多個 SaveChanges() 之間進行交易處理

```
using (var context = new BloggingContext())
{
    using (var transaction = context.Database.BeginTransaction())
    {
        try
        {
            context.Blogs.Add(new Blog { Url = "https://dot.net" });
            context.SaveChanges();

            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/" });
            context.SaveChanges();

            transaction.Commit();
        }
        catch (Exception)
        {
            // TODO: Handle failure
        }
    }
}
```

效能調校技巧

- Raw SQL Queries
  - 支援 **實體類型** 與 **無索引鍵實體類型** (例如：**檢視表** 或 **預存程序**)
  - 透過 FromSqlRaw / FromSqlInterpolated 方法執行參數化查詢 (範例)
  - 透過 Database.ExecuteSqlCommand 可執行任意 SQL 並回傳影響筆數
- Asynchronous Queries
  - **await** context.Blogs.ToListAsync();
- Asynchronous Saving
  - **await** context.SaveChangesAsync();
- 使用 AsNoTracking() 查詢資料並停用追蹤變更 (**唯讀資料**)
  - var blogs = context.Blogs.AsNoTracking().ToList();
  - context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
- 透過 Query Tags 標記產生的 SQL 查詢語法

不重複資料識別解析 (Identity resolution)

- EF Core 3.0 對於 Identity resolution 跟前版有很大的不同
  - 只要是 **Keyless Entity Types** 就永遠**不會有變更追蹤**
    - 只要**沒有變更追蹤**，就不會有 **Identity resolution** 的問題！
    - 所有 EntityKey 重複的資料，也都會輸出唯一的實體物件
  - 以下問題不存在了
    - 解開Entity Framework資料重複之謎
    - Entity Framework的View Primary Key 錯置疑雲
- EF Core 3.0 對於 LINQ 查詢效能遠勝 EF Core 2.2 以前的版本
  - 大幅減少 **物件實體化** (Object Materialization) 的次數
  - 即便透過 Select 投射出非實體模型的物件，一樣有變更追蹤
  - 從 EF Core 3.0 開始不支援用戶端評估！

What's New in Entity Framework Core 5.0

Entity Framework Core 5.0 全新特性

What's new in Entity Framework Core

- Entity Framework Core 5
  - Announcing the Release of EF Core 5.0
  - Plan for Entity Framework Core 5.0
  - What's New in EF Core 5.0

- Entity Framework Core 3
  - New features in Entity Framework Core 3.0
  - Breaking changes included in EF Core 3.0
  - Entity Framework Core 3.0: A Foundation for the Future

相關連結

- EF Core | Microsoft Docs
- EF Core Tools & Extensions
- Entity Framework Core Database Providers
- EF Core Power Tools (Visual Studio 2017, 2019)
  - https://github.com/ErikEJ/EFCorePowerTools
- Entity Framework Core Documentation And Tutorials
- Entity Framework Core Tutorials

聯絡資訊

The Will Will Web

網路世界的學習心得與技術分享

http://blog.miniasp.com/

Facebook

Will 保哥的技術交流中心

http://www.facebook.com/will.fans

Twitter

https://twitter.com/Will_Huang

多奇·教育訓練

THANK YOU!

Q&A