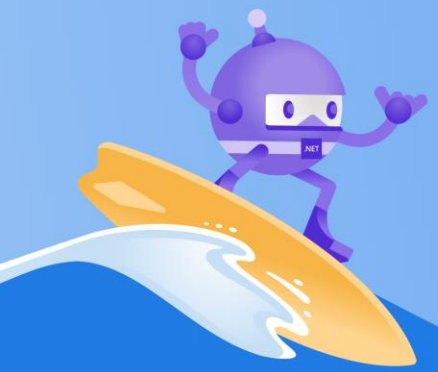
Microsoft STUDY4 Build School

.NET Conf

探索 .NET 新世界



Microsoft STUDY4 Build School

Entity Framework Core 5.0

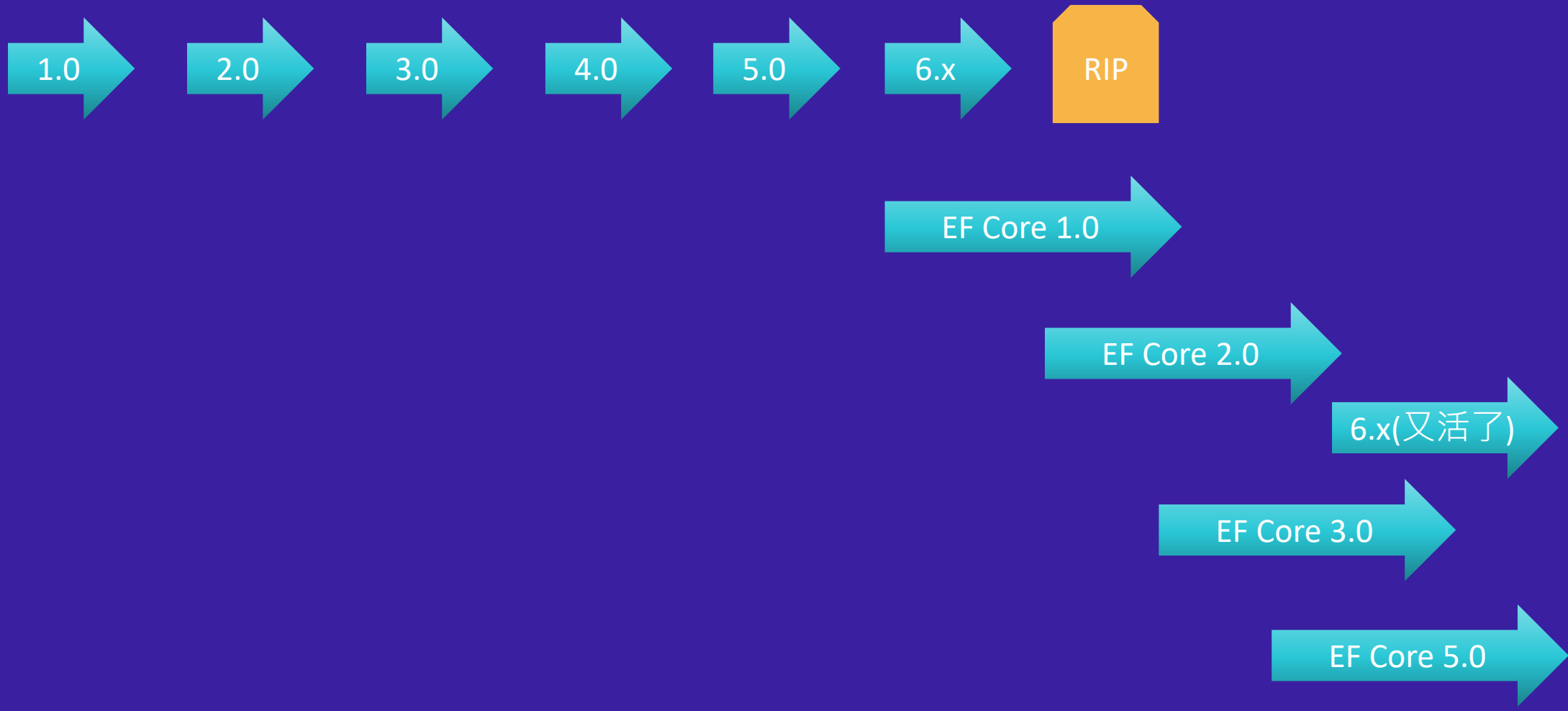
黃忠成

Who am I

- 技術顧問
- SkillTree 講師
  - LINQ 課程
  - DI、EF、FP、非同步 課程

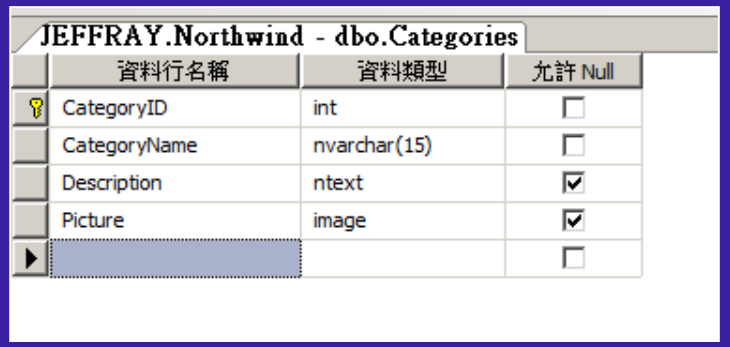
- FB
  - 風雪之閣
    https://www.facebook.com/cooldotnet

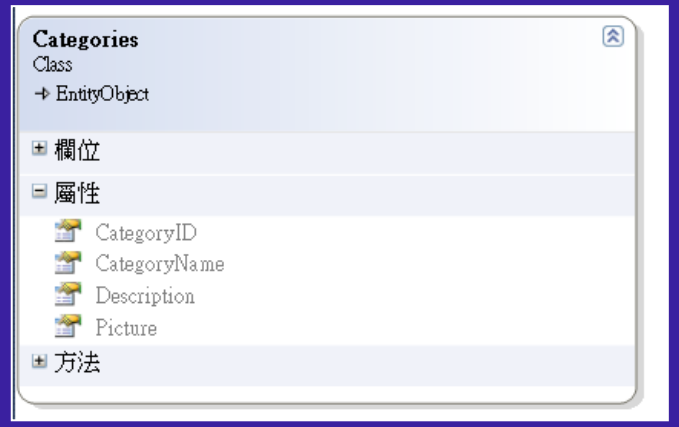
The History of Entity Framework



如果你不知道這是什麼?

- 資料表映射成物件，反之亦然





- 查詢式對應成物件

```
SELECT [Id], [NAME],Address FROM Customers
```

```
public partial class Customers
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
}
```

EF Core 5.0 Supported Platforms

- .NET Core 3.1
- .NET 5
- Blazor

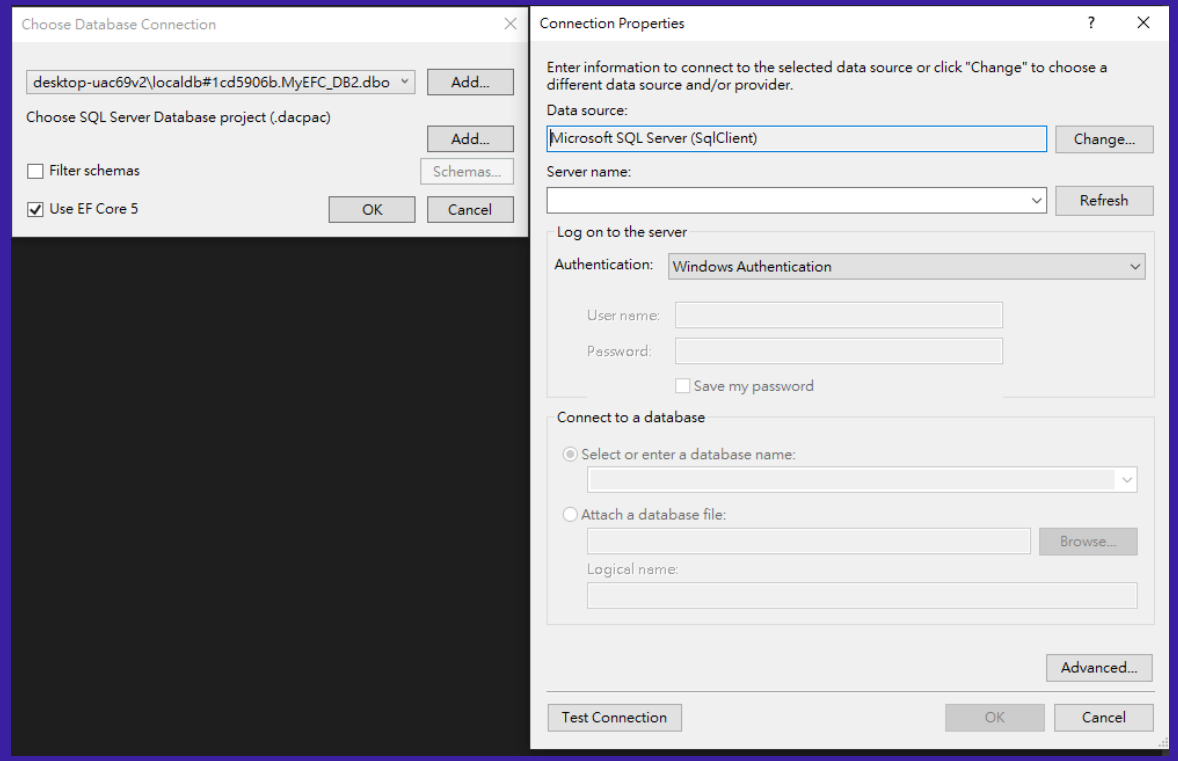
EF Core 開始的改變

- EF Core 開始大量的接受 Pull-Request
  - 接受來自社群的實務建議
  - 整合來自社群的貢獻
- 這代表什麼?
  - EF Core 越來越貼近開發者的真實需求
  - 架構上由 規格化 走向 實務導向
  - 跨資料庫遷移由 顯性 變成 隱性

圖片連結:http://www.ifuun.com/a2018092016265758/

The Design-time Tool

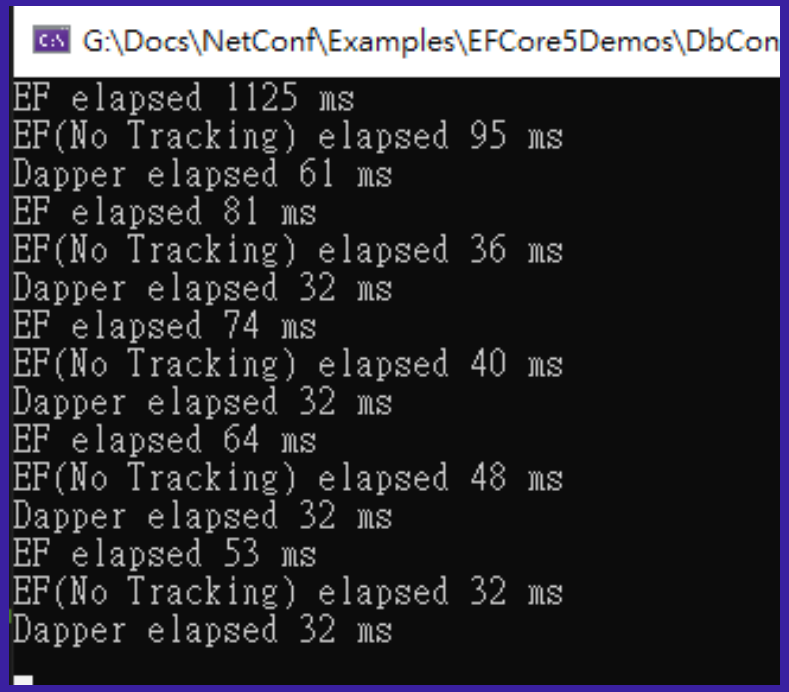
- EF Core Power Tools
  - https://marketplace.visualstudio.com/items?itemName=ErikEJ.EFCorePowerTools



ORM vs Micro ORM

- ORM
  - Mapping
  - Query Translate
  - Tracking
  - Relations
- Micro-ORM
  - Mapping
  - Simple CRUD Plug-in

Performance?
Dapper vs EF Core 5.0



What's DbContext

- 一個連線
- 一群資料表
- 管理已取出的 Entities
- 追蹤已取出的 Entities 變動
- 需要 Dispose 來清除已取出的 Entities
- 讓 DbContext 維持短暫生命週期，不要共用

Object Tracking

- 所有已取出的 Entitles 都是由 DbContext 所管理
  - 預設的 Tracking 機制是 Context-Aware
    - 逐筆比對
  - 如果你不需要 修改 功能，那麼請愛用 AsNoTracking
- 新功能 – 透過 ChangeTracker 清除已儲存的 Entities

Change Proxies

- EF Core 3.1 開始支援 另一種較有效率的 Tracking 模式
  - Change Proxies

Simple Logging

- EF 5 回到原先的模式，提供簡單的 Logging 機制

```
0 references
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseSqlServer(
        connectionString: @"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=MYEFC_DB2;Integrated Security=True;Connect Timeout=30");
    optionsBuilder.LogTo(Console.WriteLine);
}
```

Savepoint

- 這個機制主要用於 Transaction 中包含多個 SaveChanges 時
  - 經典流程是
  - 建立交易
  - 修改資料 A-> SaveChanges
  - 建立 Savepoint A
  - 修改資料 B-> SaveChanges
  - 發生錯誤
  - 恢復回 Savepoint A
  - 最終只有 修改資料 A 被儲存

DbContextFactory

- 提供在 Dependency Injection 的環境下，建立短暫的 DbContext 的一致性寫法

Query Type

- EF Core 2.0 開始支援 Query Type，也就是純查詢的 Entities 映射，這些不會被 DbContext 追蹤
  - RIP in EF Core 3.0
- EF Core 5.0 重新支援，並允許設定 SQL 指令

```
modelBuilder.Entity<Customers>().ToSqlQuery("SELECT * FROM Customers -- i am custom sql");
```

Update/Query Mapping

- 現在你可以映射 Entities 到 View，但更新至 Table

```
modelBuilder.Entity<Customers>().ToTable("Customers").ToView("Customers17View");
```

Compute Columns

- 支援映射自資料庫的計算欄位
  - 這需要資料庫支援

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
      modelBuilder.Entity<Orders>(entity =>
      {
            entity.Property(e => e.Id).ValueGeneratedNever();
            entity.Property(e => e.ProductName).HasMaxLength(50);
            entity.Property(e => e.Total).HasComputedColumnSql("Price * Qty", true);
      });
      OnModelCreatingPartial(modelBuilder);
}
```

Global Filter

- 全域性的過濾條件
  - 從 EF Core 3 開始提供，這通常用於實作限定視界或是軟刪除

```
modelBuilder.Entity<Products>().HasQueryFilter(m => EF.Property<bool>(m, propertyName: "isDeleted") == false);
```

Property Bag

- 現在你可以使用 Dictionary 做為 Entity Type

```
0 references
static void QueryData()
{
    var ctx = new MyDbContext();
    var r = ctx.Customers.FirstOrDefault(a => a[key: "NAME"] == "tom12");
    Console.WriteLine(r[key: "NAME"]);
}

1 reference
static void QueryData2(string key, string value)
{
    var ctx = new MyDbContext();
    var r = ctx.Customers.FirstOrDefault(a => a[key] == value);
    Console.WriteLine(r[key: "NAME"]);
}
```

Split Query

- EF Core 預設會產生單一 SQL 來處理查詢
  - 這在複雜的映射查詢時會產生效能問題
  - Cartesian Explosion
- 5.0 提供了 Split Query 的選項

```
static void QueryDataSplit()
{
    var ctx = new MyDBContext();
    var r = ctx.Orders.AsSplitQuery().Include(b => b.Details).ToList();
}
```

近一點看 Cartesian Explosion

SQLQuery2.sql - (...C69V2\code6 (71))* | SQLQuery1.sql - (...C69V2\code6 (53))*

```
SELECT * FROM Orders
LEFT JOIN Order_Details ON Order_Details.OrdersId = Orders.Id
```

100 %

結果 訊息

	Id	OrderDate	CustomerName	Id	Product	Qty	Price	OrdersId
1	1	0001-01-01 00:00:00.0000000	code6421	1	C#	1	1200.00	1
2	1	0001-01-01 00:00:00.0000000	code6421	2	VBNET	1	1700.00	1

SQLQuery2.sql - (...C69V2\code6 (71))* | SQLQuery1.sql - (...C69V2\code6 (53))*

```
SELECT "a"."Id", "a"."CustomerName"
FROM "Orders" AS "a"
ORDER BY "a"."Id"

SELECT "a0"."Id", "a0"."OrdersId", "a0"."Product", "a"."Id"
FROM "Orders" AS "a"
INNER JOIN "Order_Details" AS "a0" ON "a"."Id" = "a0"."OrdersId"
ORDER BY "a"."Id"
```

100 %

結果 訊息

	Id	CustomerName
1	1	code6421

	Id	OrdersId	Product	Id
1	1	1	C#	1
2	2	1	VBNET	1

Filter Include

- 你可以在 Include 的時候加上過濾條件

```
var r = ctx.Orders.Include(a => a.OrderDetails.Where(b => b.Product.Contains("Go"))).ToList();
```

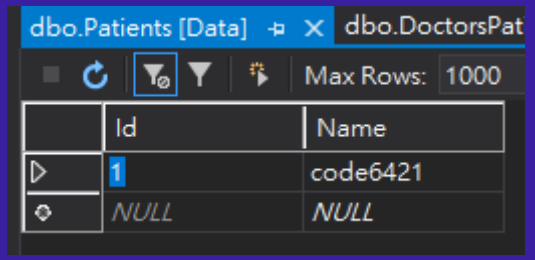
- 需要的時候也可以搭配 Split Query

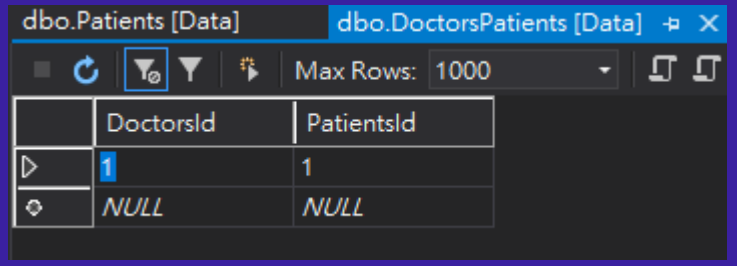
多對多映射

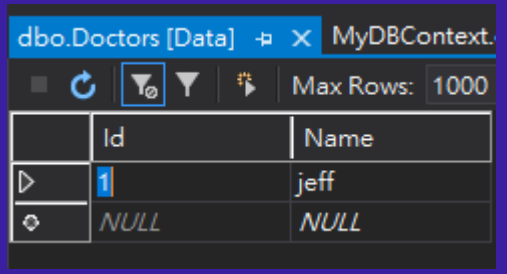
- EF Core 5.0 簡化了多對多設定
  - 如果需要的話，你還是可以進行完整的細部映射設定

```
public class Doctors
{
    public int Id { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Patients> Patients { get; set; }
    …..
}

public class Patients
{
    public int Id { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Doctors> Doctors { get; set; }
    …..
}
```







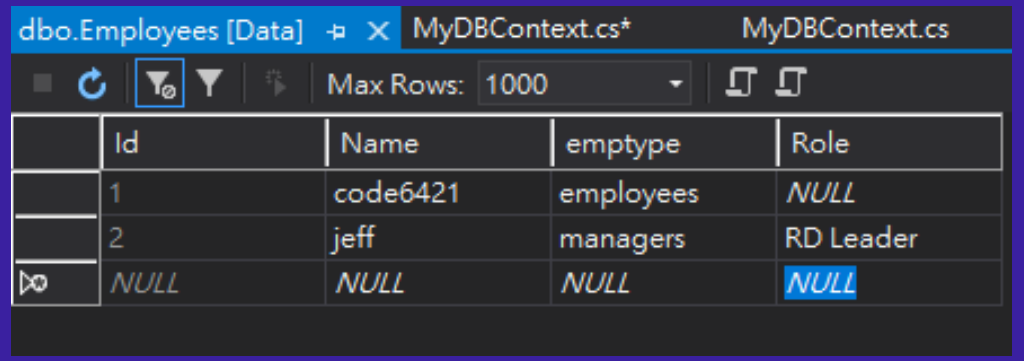
Inheritance

- EF Core 5 支援兩種繼承
  - TPH -> 單一資料表涵蓋所有繼承體
    - 這會在該資料表產生一個辨識欄位
  - TPT -> 每個資料表代表一個繼承體
    - 這通常是 DBA 能接受的方式，無辨識欄位，每個繼承體就是一個資料表

Inheritance-TPH

- 一個 資料表 包含所有繼承體系

```
modelBuilder.Entity<Employees>().HasDiscriminator<string>(
     "emptype").HasValue<Employees>(
     "employees").HasValue<Managers>("managers");
```

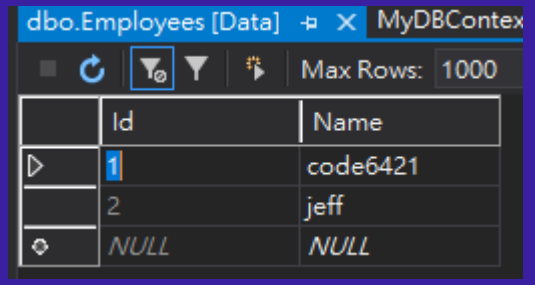


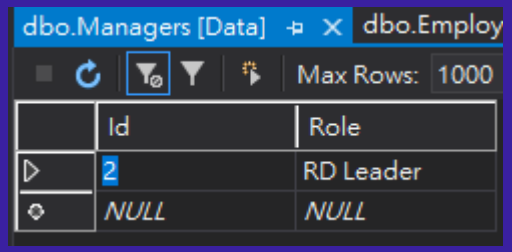
Inheritance-TPT

- 每個繼承體都會有各自的資料表

```
modelBuilder.Entity<Employees>().ToTable("Employees");

modelBuilder.Entity<Managers>().ToTable("Managers");
```

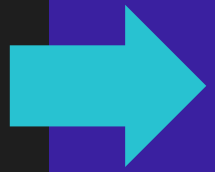




User Function Mapping

- 映射 使用者定義函式

```
CREATE FUNCTION[dbo].[MySum]
(
    @param1 int,
    @param2 int
)
RETURNS INT AS
BEGIN
   RETURN @param1 + @param2
END
```



```
[DbFunction("MySum")]
public int MySum(int x, int y) => x + y;
```

Table Value Function

- 映射 TVF 至 Entities

```
CREATE FUNCTION [dbo].[GetCustomerByName] ( @name nvarchar(120) )
RETURNS @report TABLE ( Id int, [Name] nvarchar(120) ) AS BEGIN
INSERT @report
SELECT Id, [Name] FROM Customers WHERE [Name] Like @name
RETURN END
```

```
modelBuilder.Entity<CustomerReport>().HasNoKey();
modelBuilder.HasDbFunction(() => GetCustomerByName(default));
```

SaveChanges Interceptor

- 以往，我們都是覆載DbContext.SaveChanges 來攔截儲存動作
- EF Core 5 開始，添加兩種方式
  - SavingChanges、SavedChanges 事件
  - SaveChangesInterceptor

Performance Tips

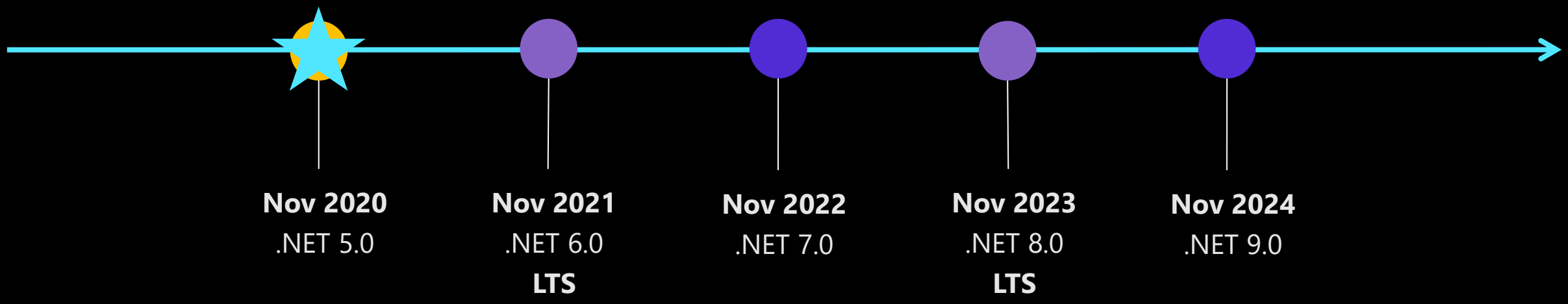
- DbContext is Cheap
- Use Find instead of Where
- Don’t query twice
- Separate the Query and Update with DbContext and AsNoTracking
- You don’t need use a framework only

Microsoft STUDY4 Build School

Sample Code

https://qrgo.page.link/eD6DE

.NET Schedule



- .NET 5.0 released today!
- Major releases every year in November
- LTS for even numbered releases
- Predictable schedule, minor releases as needed

Microsoft STUDY4 Build School

Thanks for joining!

Ask questions on Twitter using #dotNETConf

.NET Conf 2020

特別感謝

91APP Technical Network

Red Hat

KKTIX

JET BRAINS

HackMD

MVP Microsoft® Most Valuable Professional

Microsoft

Build School

STUDY4 為學習而生

以及各位參與活動的你們