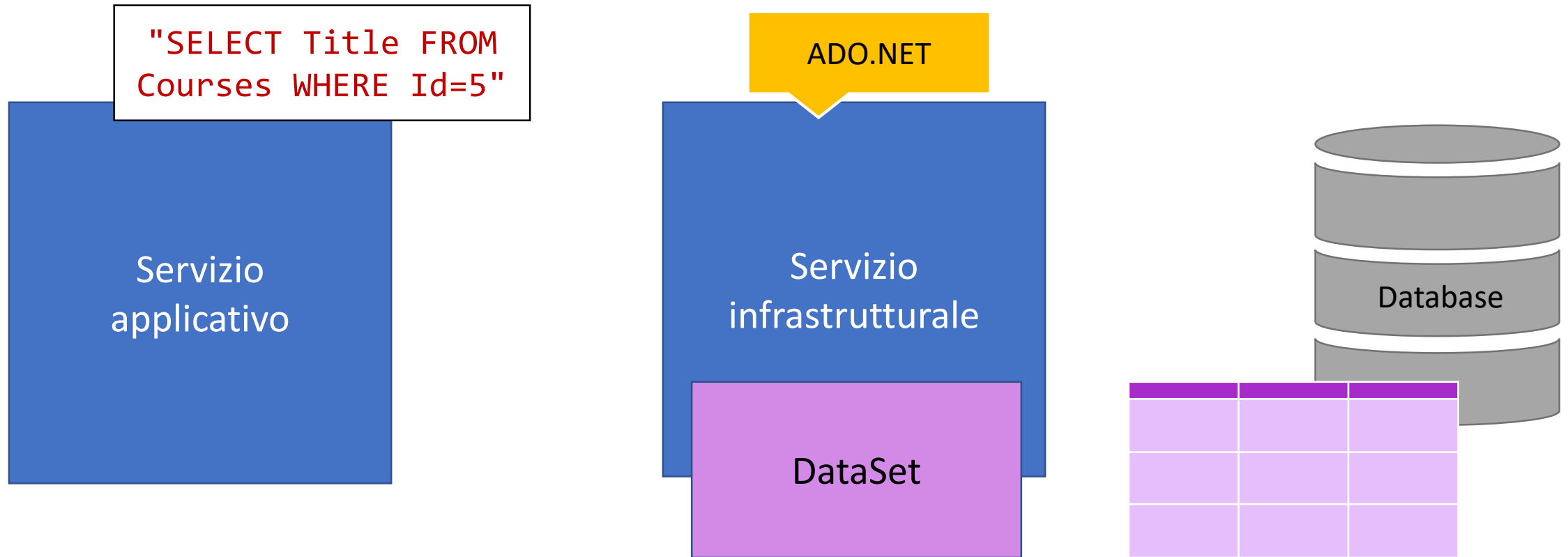


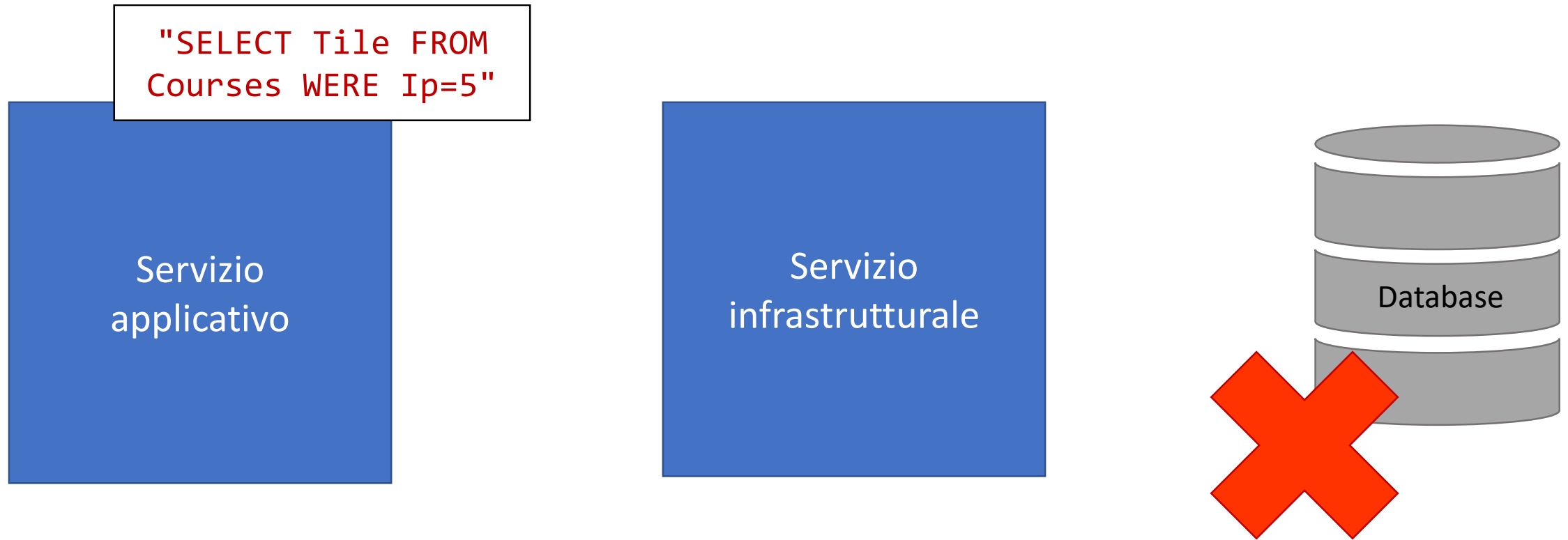
Sezione 11

Accedere al database con Entity Framework Core

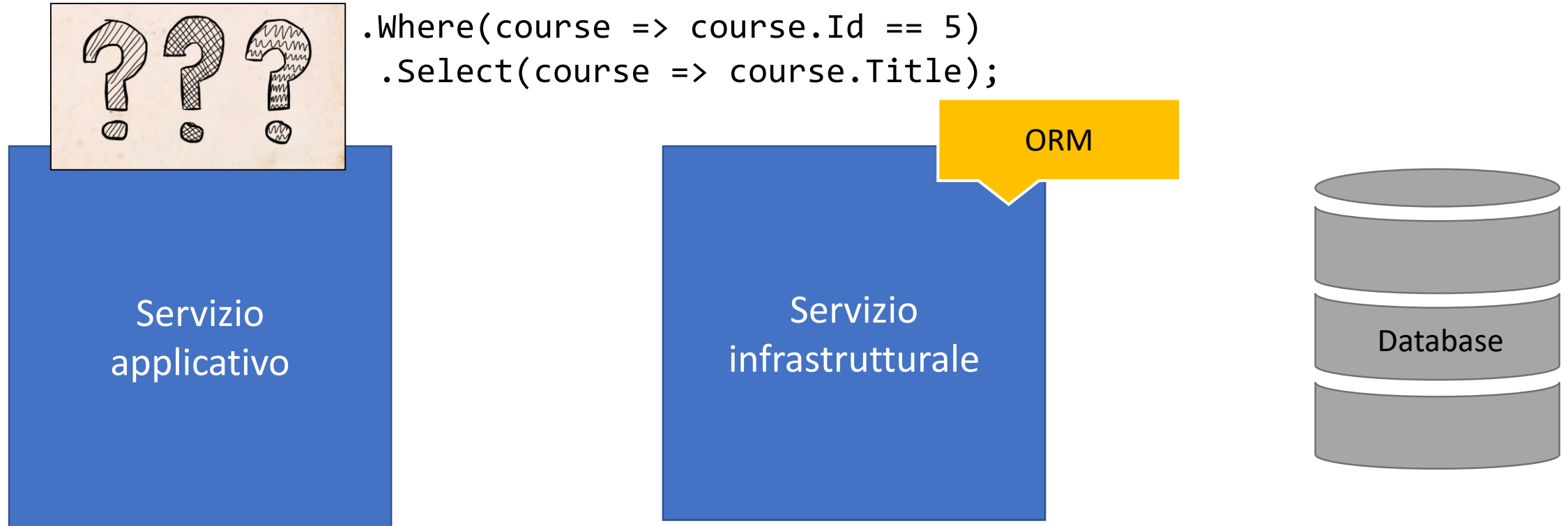
Interrogare il database con ADO.NET



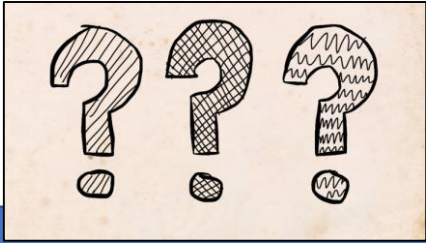
Interrogare il database con ADO.NET



ORM: un altro modo di interrogare il database



ORM: un altro modo di interrogare il database



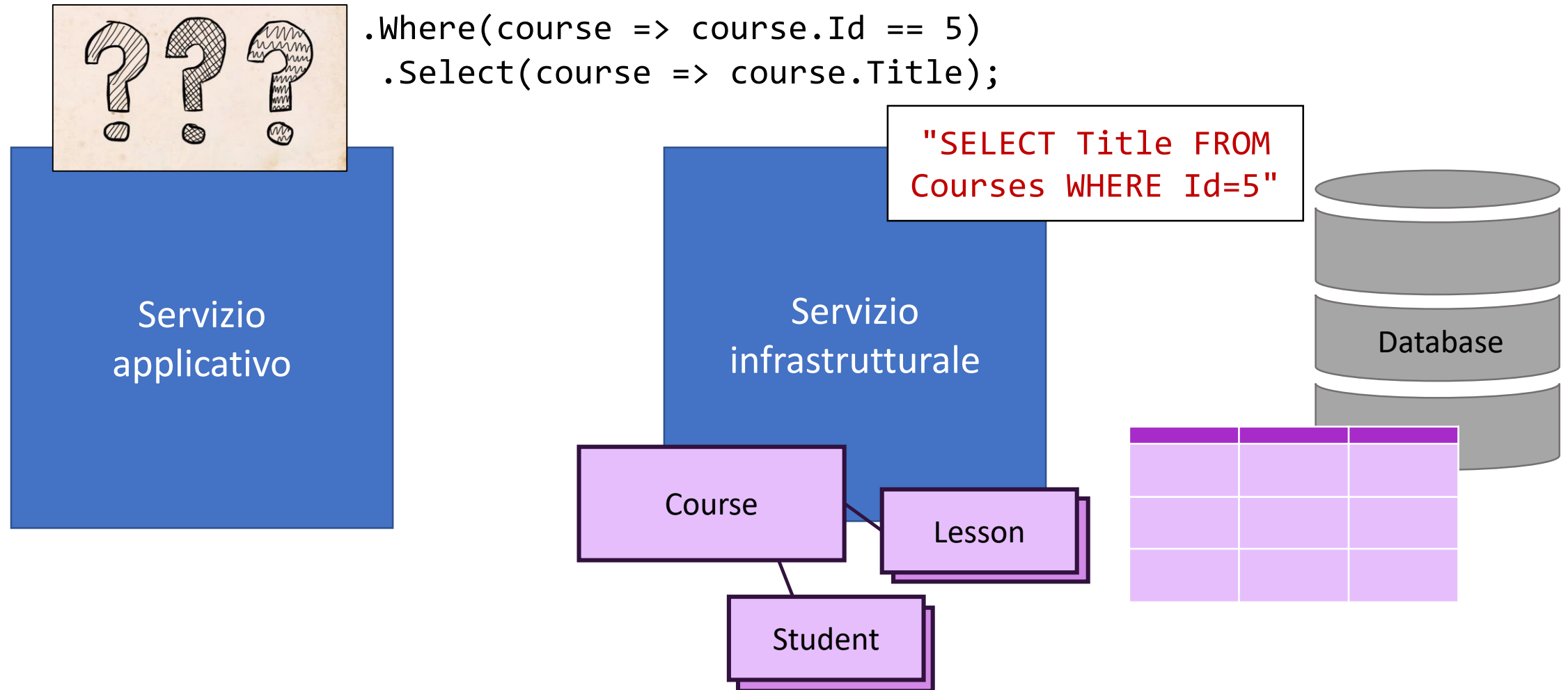
```
.Where(course => course.Ip == 5)  
.Select(course => course.Title);
```

Servizio
applicativo

Servizio
infrastrutturale

Database

ORM: un altro modo di interrogare il database

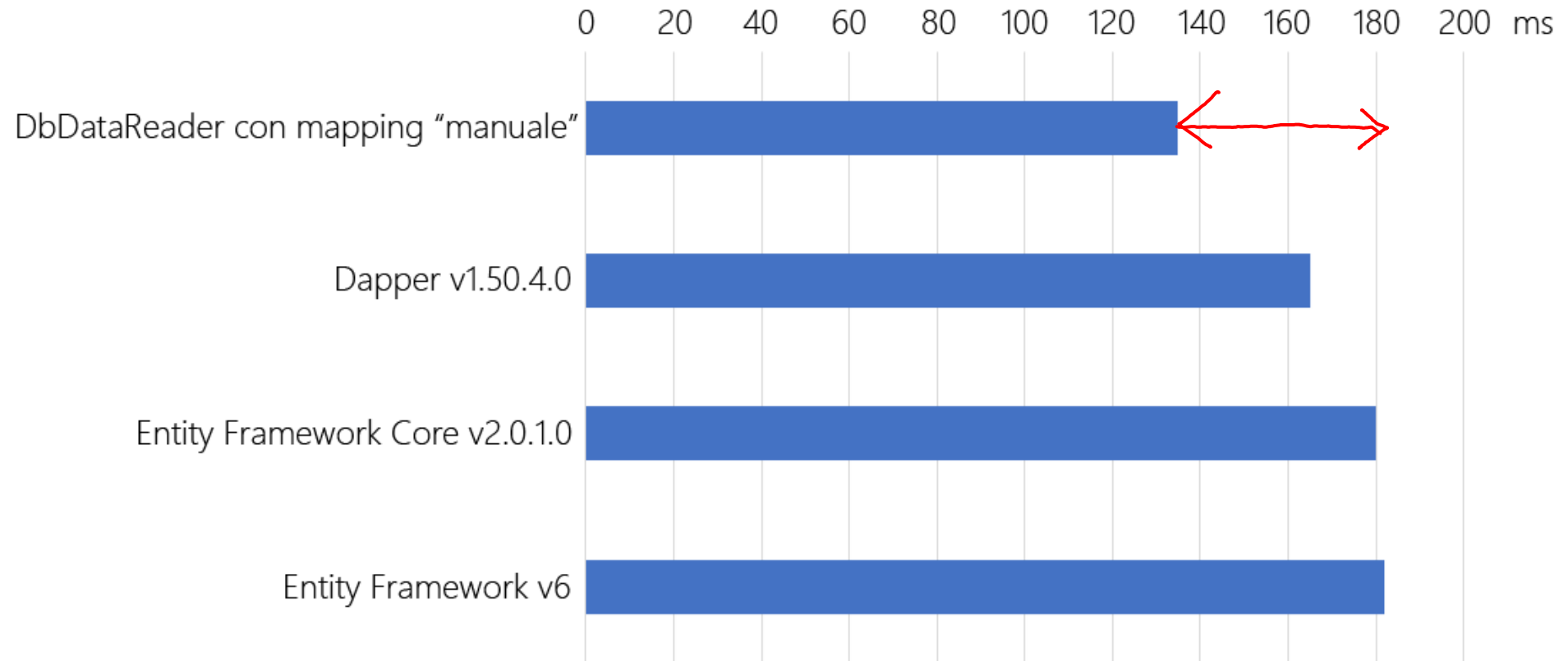


ORM

Object / Relational Mapper

Costo prestazionale di un ORM

Esempio: tempo in ms richiesto per estrarre e materializzare 31.000 righe, senza change tracking (meno è meglio)



<https://weblogs.asp.net/fbouma/net-micro-orm-fetch-benchmark-results-and-the-fine-details>

Vantaggi e svantaggi di usare EFCore

VANTAGGI

- ✓ Query fortemente tipizzate
- ✓ Migliore disaccoppiamento
- ✓ Lavoriamo con un modello a oggetti
- ✓ E' molto facile persistere le modifiche

SVANTAGGI

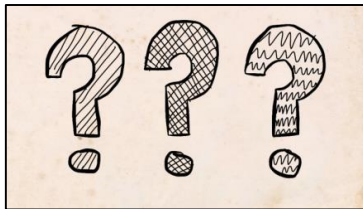
- ✗ Performance leggermente inferiori
- ✗ Rischiamo di inviare query inefficienti
- ✗ Dobbiamo reimparare a fare le query

Vantaggi e svantaggi di usare EFCore

“

Secondo me SQL era più semplice

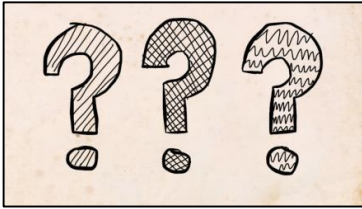
”



`.FromSql("SELECT * FROM Courses")`

Prima le basi

query LINQ



```
.Where(course => course.Id == 5)  
.Select(course => course.Title);
```

Lambda expression

Espressioni C#

```
string name = "Mario";
```

```
DateTime dob = new DateTime(2000, 12, 25);
```

```
bool canDrive = dob.AddYears(18) <= DateTime.Today;
```

espressioni

Funzioni (metodi)

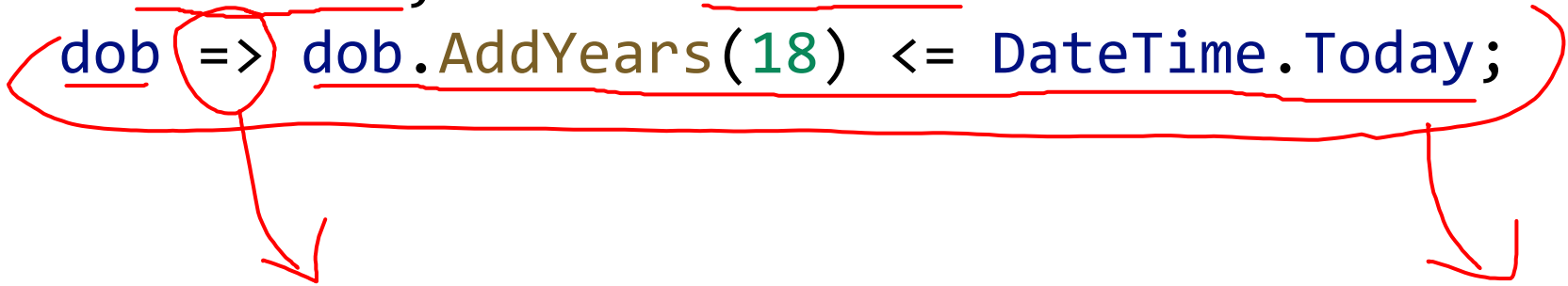
```
public bool CanDrive (DateTime dob) {  
    return dob.AddYears(18) <= DateTime.Today;  
}
```

Funzioni (metodi)

```
public bool CanDrive (DateTime dob, int legalAge = 18) {  
    return dob.AddYears(legalAge) <= DateTime.Today;  
}
```

Lambda expression

```
Func<DateTime, bool> canDrive =  
    dob => dob.AddYears(18) <= DateTime.Today;
```



Operatore lambda

*lambda expression
(è una funzione anonima)*

Lambda expression

```
Func<DateTime, bool> canDrive =  
    (dob) => {  
        bool value = dob.AddYears(18) <= DateTime.Today;  
        return value;  
    };
```

Se usiamo le graffe, allora ci va il return (proprio come in una normale funzione)

Le parentesi tonde sono necessarie solo in presenza di zero o più di 1 argomento

Lambda expression

```
Func<DateTime, int, bool> canDrive =  
    (dob, legalAge) => {  
        bool value = dob.AddYears(legalAge) <= DateTime.Today;  
        return value;  
    };
```

```
Func<DateTime, int, bool> canDrive =  
    (dob, legalAge) => dob.AddYears(legalAge) <= DateTime.Today;
```

Quanti tipi di lambda expression esistono?

Func<T, TResult>

Func<T1, T2, T3, TResult>

Func<T1, T2, TResult>

Func<TResult>

Func<T1, T2, T3, T4, TResult>

Quanti tipi di lambda expression esistono?

Action<T>

Action<T1, T2, T3>

Action<T1, T2>

Action

Action<T1, T2, T3, T4>

Quanti tipi di lambda expression esistono?

`Func<T1, T2, TResult>`

Accetta 2 parametri di tipo T1 e T2
L'ultimo type parameter indica il tipo
restituito dalla funzione

`Action<T1, T2>`

Accetta 2 parametri di tipo T1 e T2
La funzione non restituisce
nulla (void)

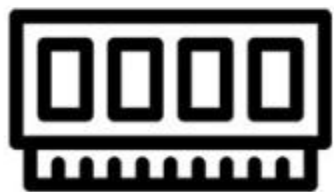


FILE

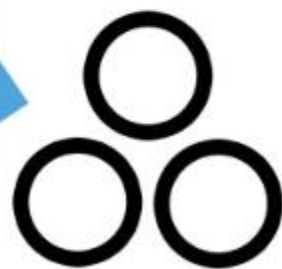


DB

LINQ



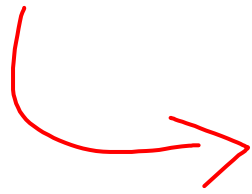
OGGETTI
IN MEMORIA



ALTRA
SORGENTE DATI

LINQ

- Offre un insieme di ~~metodi~~ **extension method** come `Select`, `Where`, `OrderBy` per interrogare elenchi;
- A molti di essi possiamo fornire delle lambda expression come argomento;
- Un elenco è una qualsiasi classe che implementi l'interfaccia **`IEnumerable<T>`**.



È un elenco se lo possiamo scorrere col foreach

LINQ

Elenco (non esaustivo) degli extension methods

TIPICI DEL MONDO RELAZIONALE

Where, Select e SelectMany
OrderBy e OrderByDescending
Join e GroupBy
Skip e Take

AGGREGAZIONE

Max, Min e Average
Sum, Count, All e Any
Aggregate e Distinct

INSIEMISTICI

Concat e Union
Except e Intersect
Contains
Zip

ESTRAZIONE DI SINGOLI OGGETTI

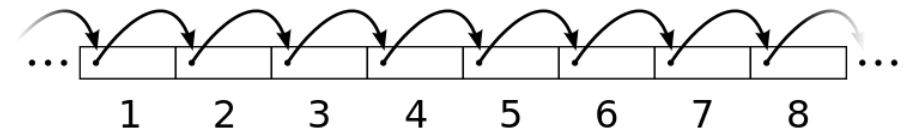
First e Last
Single
ElementAt

IEnumerable<T>

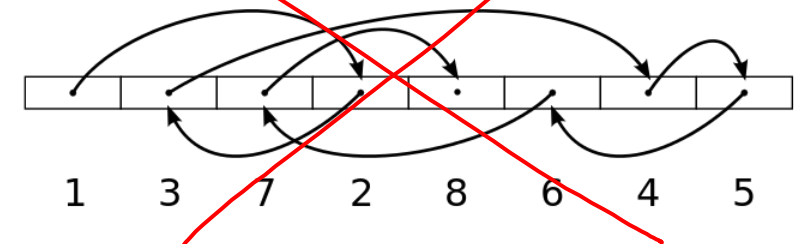
Le classi che implementano IEnumerable<T> permettono:

- Di accedere sequenzialmente (cioè di scorrere in avanti);
- Di leggere l'elemento corrente;
- Di ripartire da capo.

Accesso sequenziale



~~Accesso casuale~~



Where

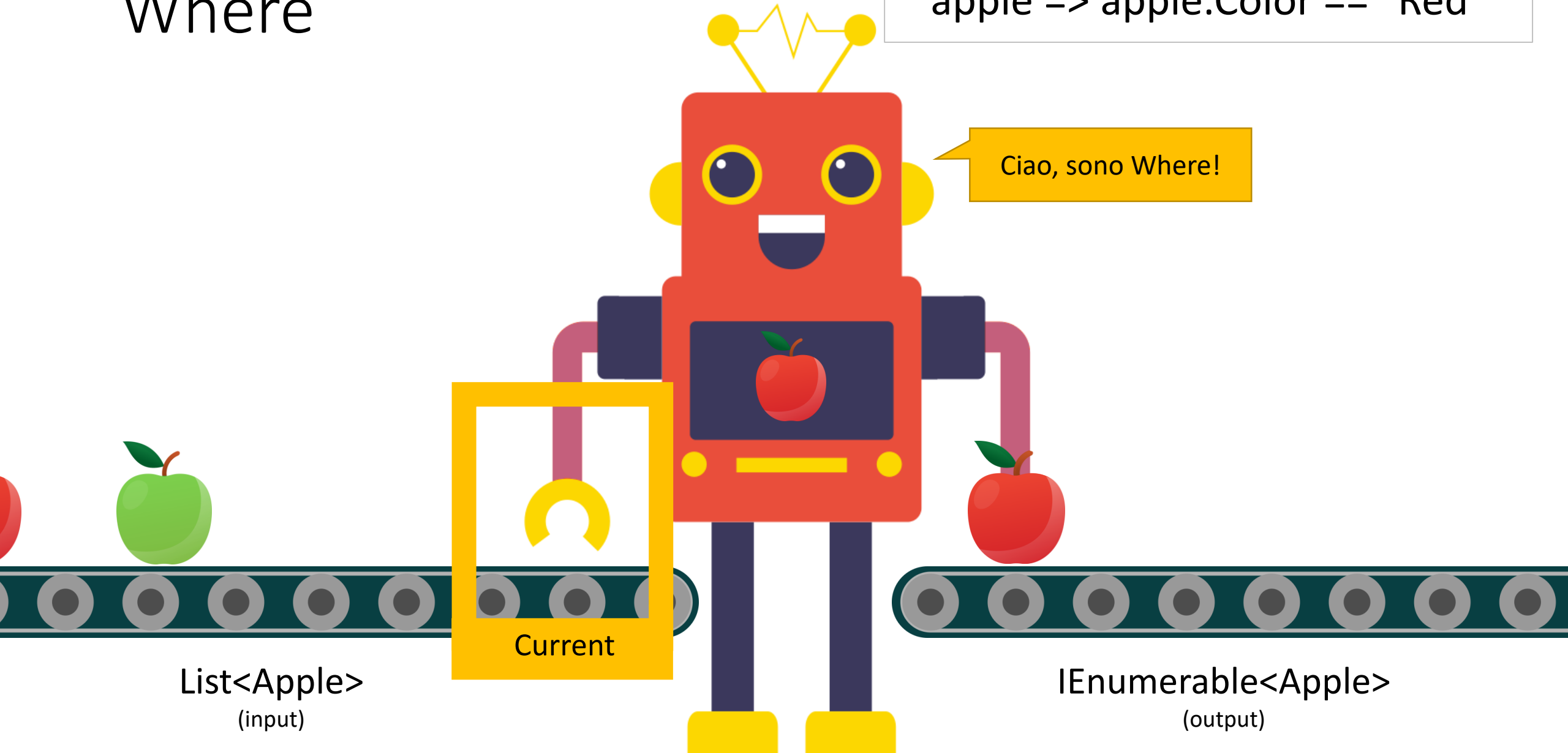
```
apple => apple.Color == "Red"
```

Ciao, sono Where!

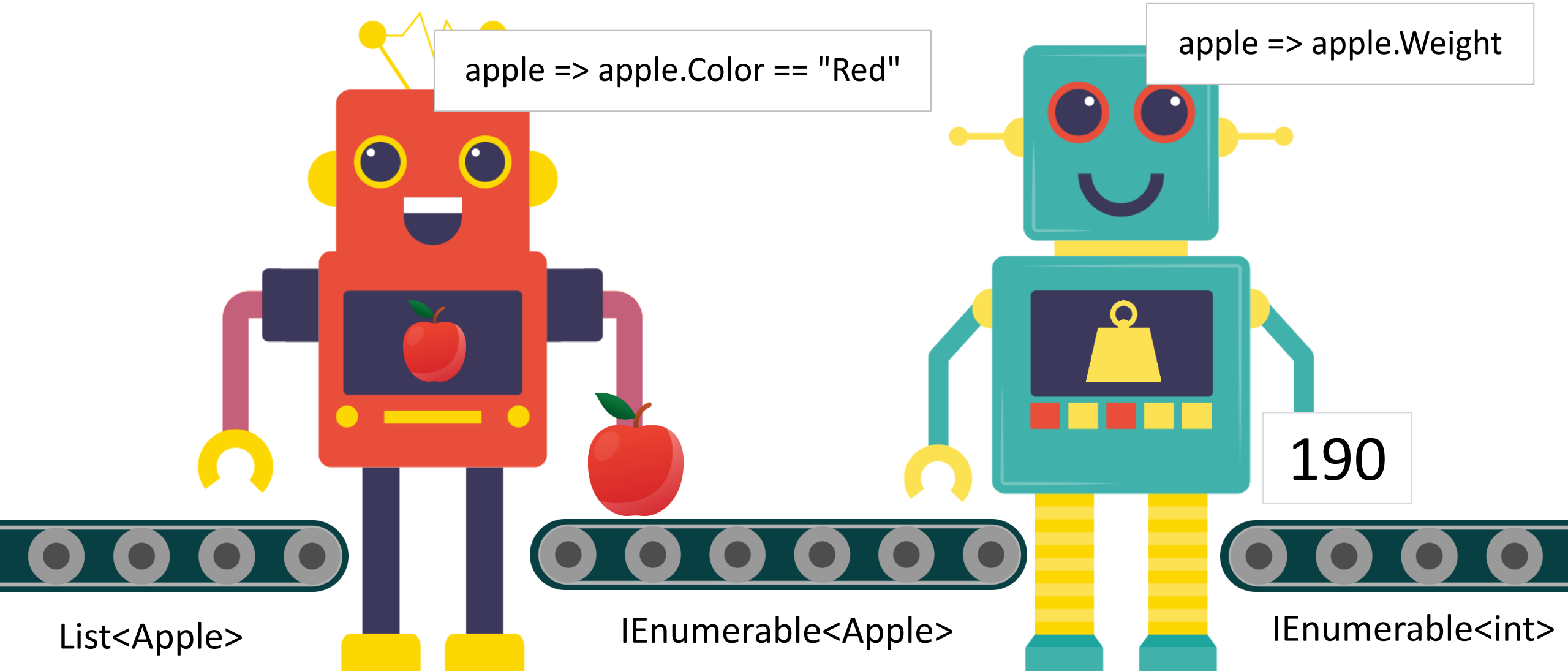
Current

List<Apple>
(input)

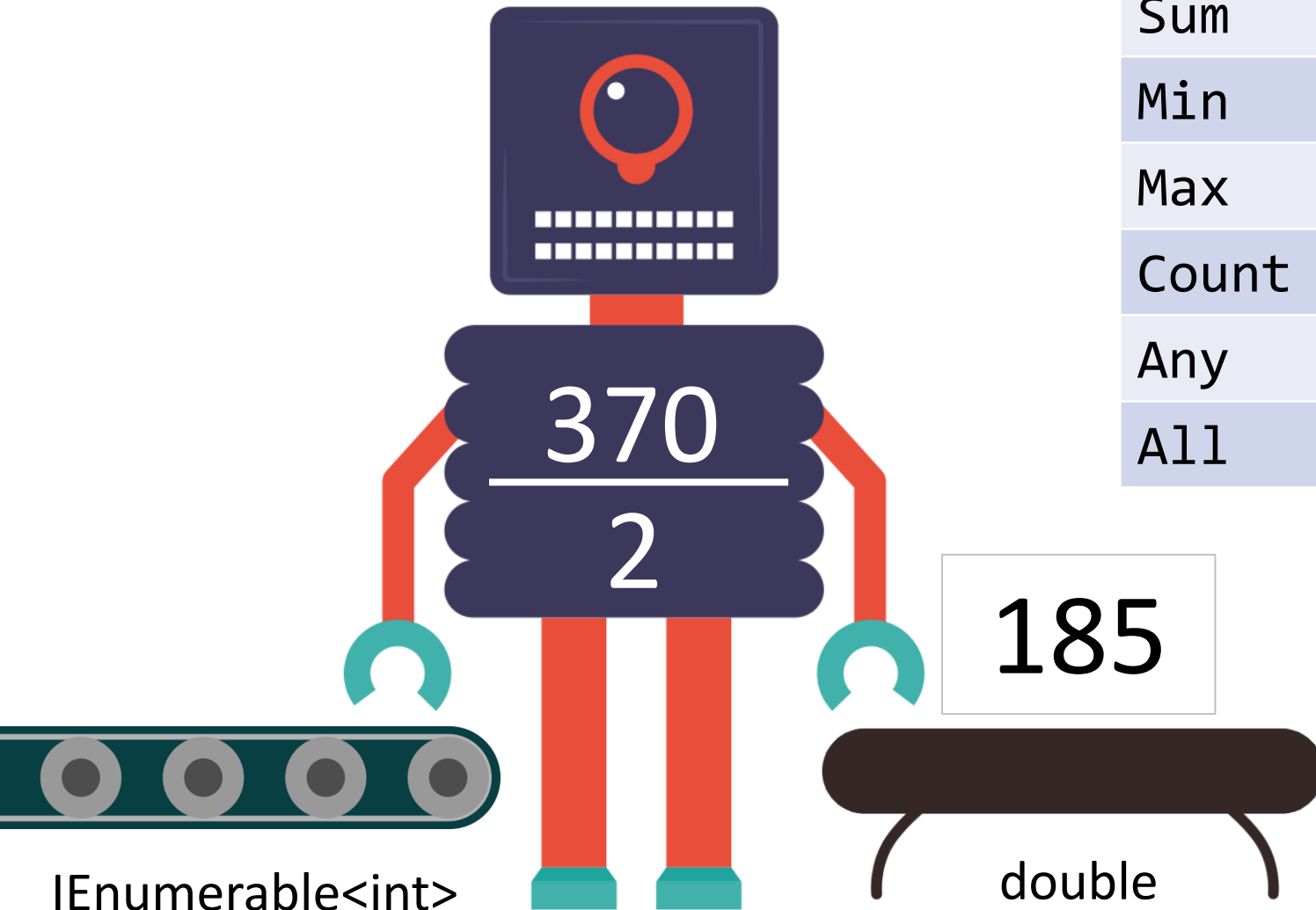
IEnumerable<Apple>
(output)



Where e Select in combinazione



Average



Extension method	Scopo
Average	Calcolare la media
Sum	Calcolare il totale
Min	Ottenere il valore minimo
Max	Ottenere il valore massimo
Count	Contare il numero
Any	Almeno uno è conforme
All	Tutti sono conformi

Con e senza LINQ

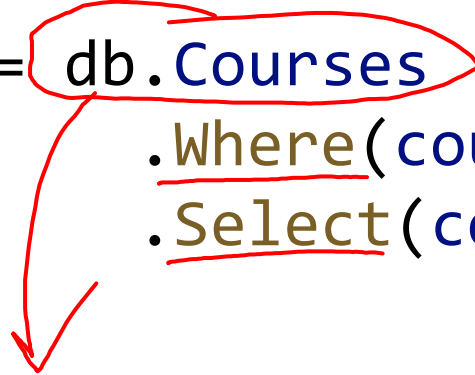
```
double average = apples
    .Where(apple => apple.Color == "Red")
    .Select(apple => apple.Weight)
    .Average();
```

```
int count = 0;
int sum = 0;
foreach (var apple in apples)
{
    if (apple.Color == "Red")
    {
        count++;
        sum += apple.Weight;
    }
}
double avg = count > 0 ? sum/(double)count : 0;
```

DbSet<T>

È l'oggetto a cui rivolgiamo le nostre query LINQ.

```
var bestCourses = db.Courses
                    .Where(course => course.Rating >= 4.5)
                    .Select(course => course.Title);
```



DbSet<Course>

DbSet<T> implementa il pattern *repository*

Consente tutte le operazioni CRUD (Create, Read, Update e Delete)

```
db.Courses.Add(course);
```

```
db.Courses.Remove(course);
```

```
db.Courses.Update(course);
```

DbContext

```
public class MyCourseDbContext : DbContext
{
    public DbSet<Course> Courses { get; set; }
    public DbSet<Lesson> Lessons { get; set; }

    //Qui configurazione e mapping
}
```

Classi di entità

Fanno parte del modello concettuale

Per iniziare con Entity Framework Core...

- ...per prima cosa procuriamoci il provider da NuGet per la tecnologia database che abbiamo scelto di usare.

Tecnologia database	Nome del pacchetto NuGet
SQLite	<code>Microsoft.EntityFrameworkCore.Sqlite</code>
SQL Server	<code>Microsoft.EntityFrameworkCore.SqlServer</code>
MySQL	<code>Pomelo.EntityFrameworkCore.MySql</code>
Oracle	<code>Devart.Data.Oracle.EFCore</code>
PostgreSQL	<code>Devart.Data.PostgreSql.EFCore</code>

Creare il modello concettuale da un db esistente

```
dotnet ef dbcontext scaffold "Data Source=Data/MyCourse.db"
Microsoft.EntityFrameworkCore.Sqlite --output-dir
Models/Entities --context-dir
Models/Services/Infrastructure --context MyCourseDbContext
```

Classi di entità

- Si mappano dal metodo `OnModelCreating` del `DbContext`;
- Le mappiamo con `modelBuilder.Entity<T>(...)`;
- Dovrebbero contenere logica applicativa:
 - Validazione dei dati;
 - Mutamento dello stato interno;
 - Creazione di relazioni.
- Possono contenere proprietà complesse e di navigazione.

Owned types
(nel gergo di EFCore)

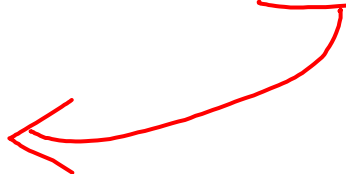
Relazioni

Owned types

- Sono classi che ci aiutano a tenere coesi più valori;
 - Come la classe Money, che ha le proprietà Currency e Amount;
- Non sono entità (cioè NON hanno un ID – ovvero un'identità);
- Si mappano con `entity.OwnsOne(...)`

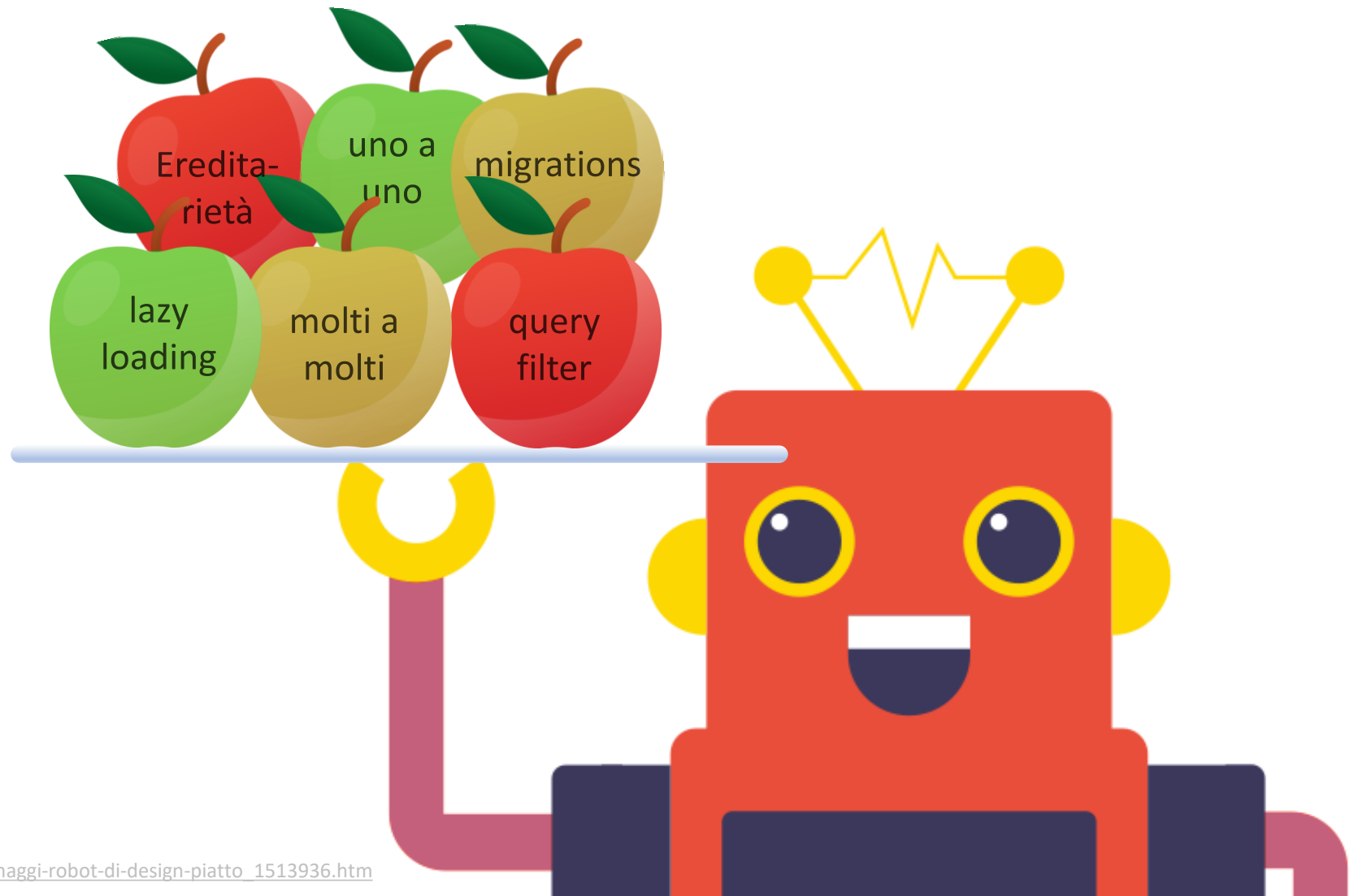
Relazioni uno-a-molti

```
public class Course
{
    //...
    public ICollection<Lesson> Lessons {get;set;}
}
```

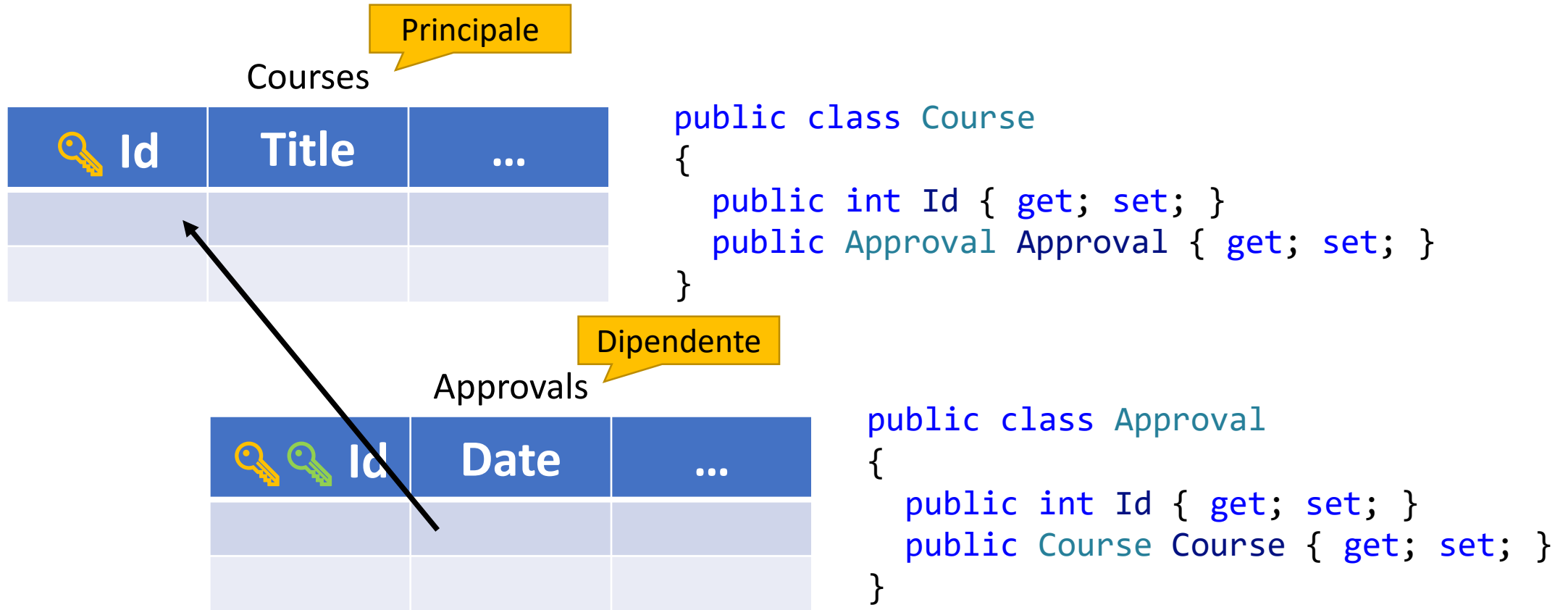


- Ci permettono di usare proprietà di navigazione;
- Si mappano con `entity.HasMany(...)`
`.WithOne(...)`
`.HasForeignKey(...);`

Entity Framework Core è molto vasto



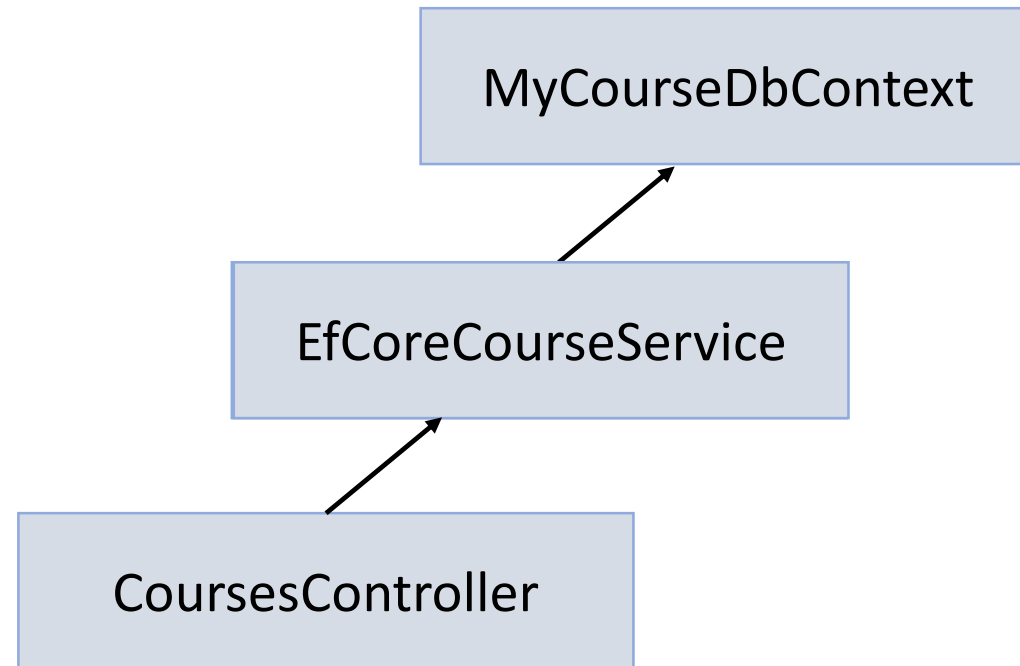
Relazione uno-a-molti



 Chiave primaria

 Chiave esterna

Torniamo ai nostri servizi...

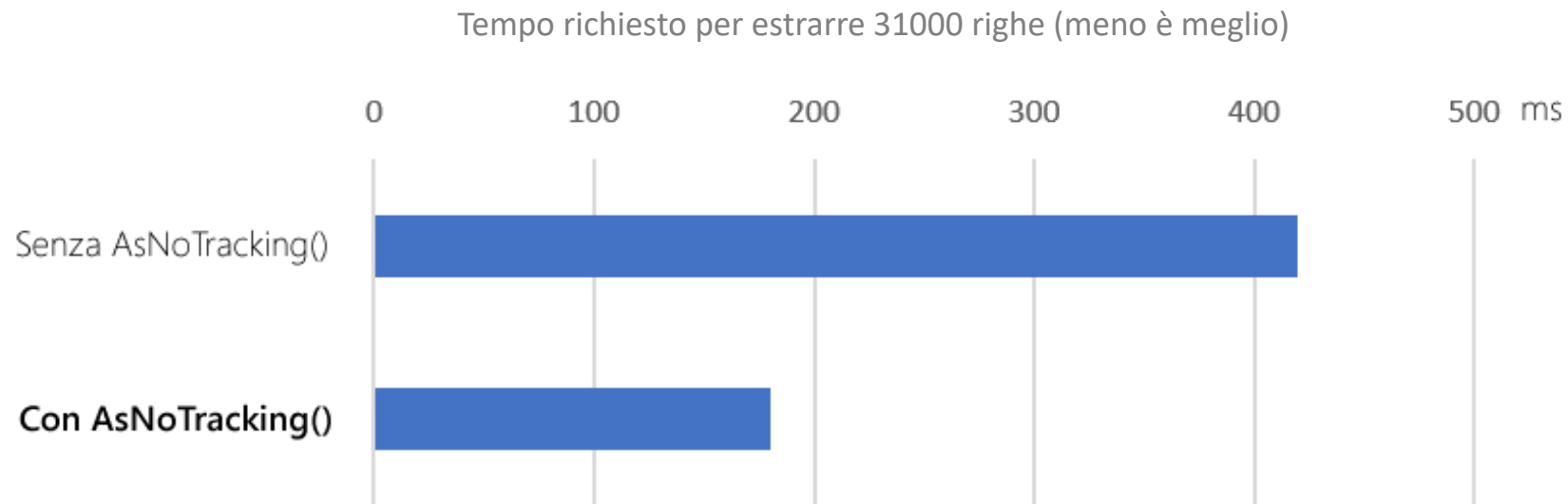


Performance



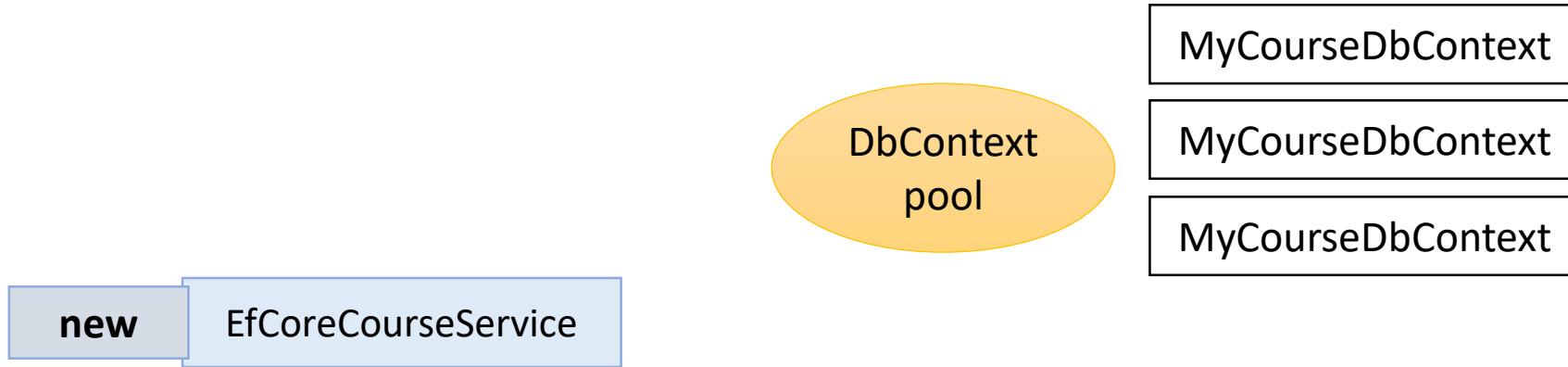
dbContext.Courses.AsNoTracking()

- Si mette quando vogliamo solo leggere i dati;
- Indica a EFCore che vogliamo rinunciare al suo "change tracker", così da avere miglioramenti prestazionali.



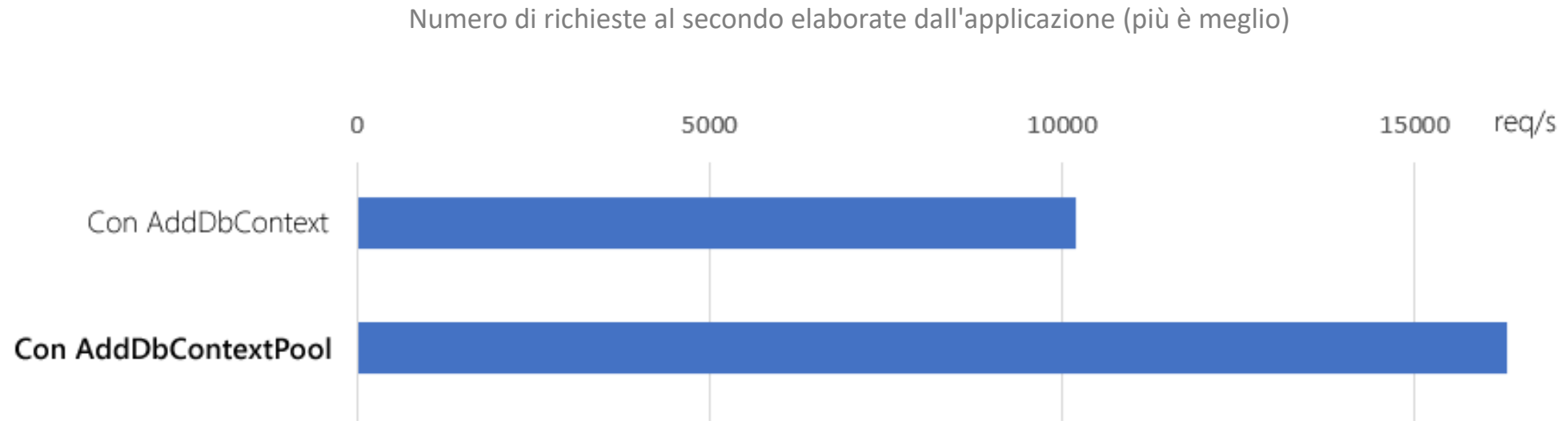
<https://weblogs.asp.net/fbouma/net-micro-orm-fetch-benchmark-results-and-the-fine-details>

```
services.AddDbContextPool<T>();
```



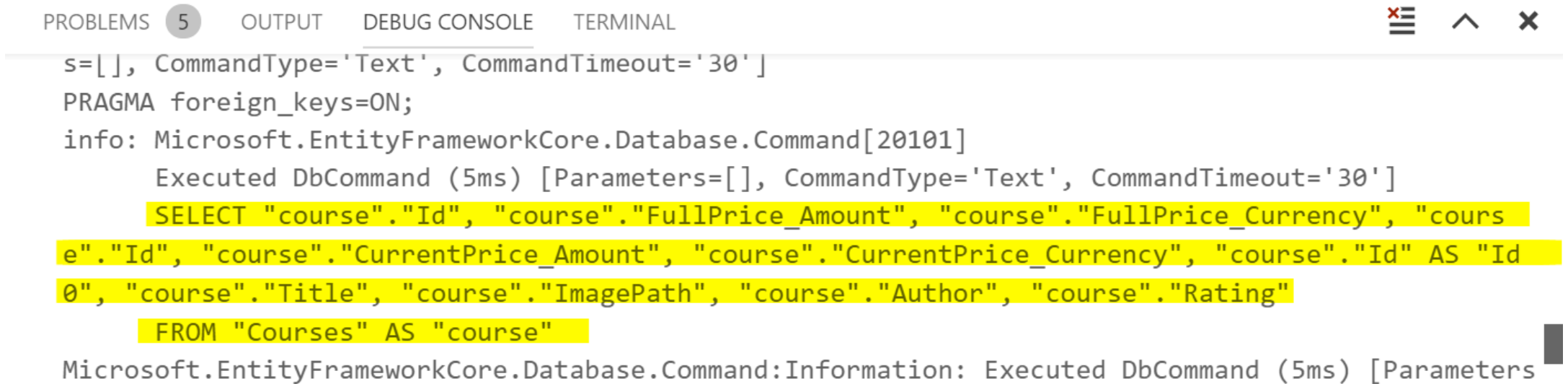
services.AddDbContextPool<T>();

- Serve a creare DbContext in maniera proattiva, così che siano subito pronti all'uso. (Un po' come le connessioni in un connection pool).



Come faccio a vedere le query?

- Vengono loggate nella Debug Console



The screenshot shows the Visual Studio interface with the 'DEBUG CONSOLE' tab selected. The console displays the following log output:

```
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL
s=[], CommandType='Text', CommandTimeout='30']
PRAGMA foreign_keys=ON;
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT "course"."Id", "course"."FullPrice_Amount", "course"."FullPrice_Currency", "course"."Id", "course"."CurrentPrice_Amount", "course"."CurrentPrice_Currency", "course"."Id" AS "Id0", "course"."Title", "course"."ImagePath", "course"."Author", "course"."Rating"
      FROM "Courses" AS "course"
Microsoft.EntityFrameworkCore.Database.Command:Information: Executed DbCommand (5ms) [Parameters=]
```

Deferred execution

Le query SQL vengono inviate solo quando è effettivamente necessario;

- Cioè quando vogliamo leggere i risultati.

IQueryable<Course>

```
var query = dbContext.Courses.Where(course => course.Rating > 4);
```

La query SQL non viene inviata qui...

IEnumerable<Course>

```
var results = await query.ToListAsync();
```

← ...ma qui sì...

```
foreach (var result in query)
```

← ...e anche qui!

```
{  
    //...  
}
```

Sarebbe stato meglio fare un ciclo su **results**, perché è una lista di oggetti in memoria, già ottenuti dal db grazie a **ToListAsync**.

IQueryable<T>

- Permette di interrogare collezioni di elementi che non esistono ancora nella memoria dell'applicazione;
- Quando usiamo gli extension method di LINQ su oggetti che implementano IQueryable<T>, stiamo costruendo un "albero di espressioni".
 - Le nostre lambda expression non vengono di fatto eseguite, ma vengono usate per costruire questo "albero di espressioni";
 - Il provider LINQ (es. quello per Sqlite) legge l'"albero di espressioni" dalla proprietà Expression dell'IQueryable<T> e lo converte in SQL.

IQueryable<T> ed expression trees

```
var query = dbContext.Courses.Where(course => course.Rating > 4);
```

