.NET Conf 2024

Le novità di Entity Framework Core 9

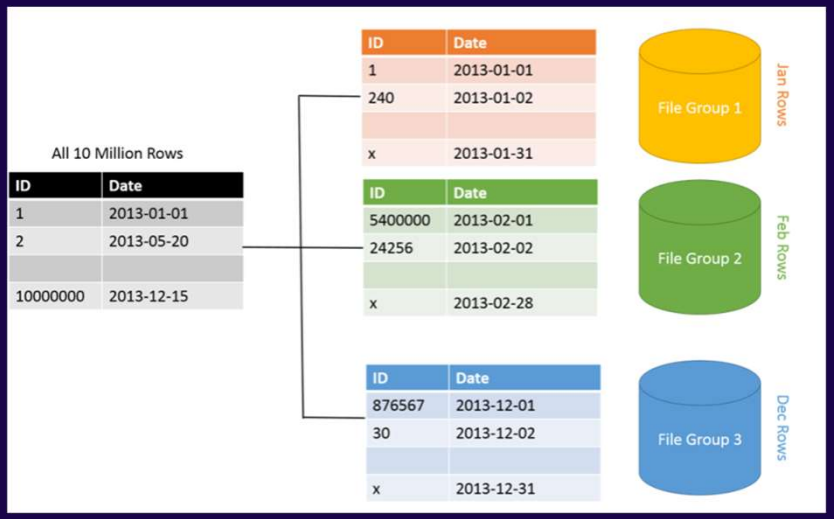
Giampaolo Tucci

Entity Framework: Novità in .NET9

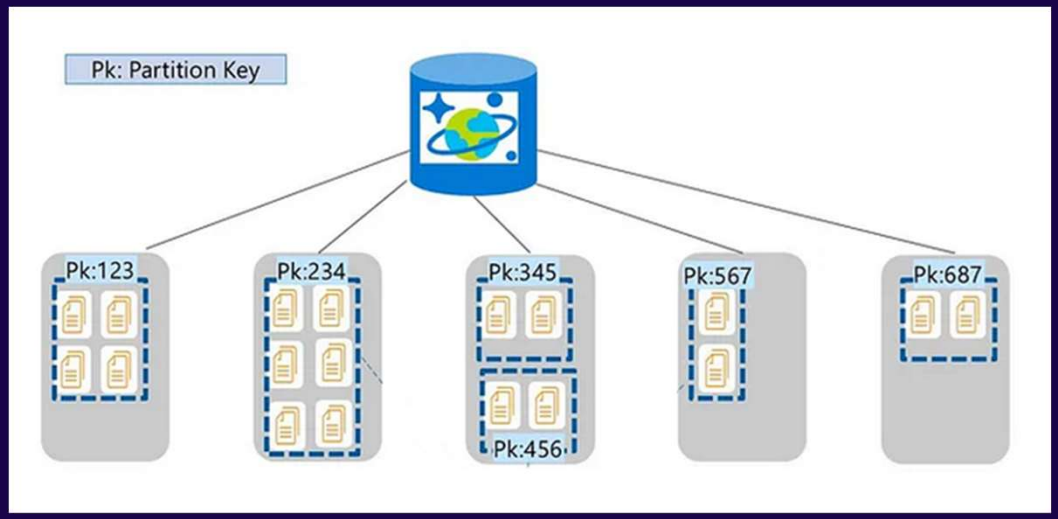
- **Enhanced support for Azure Cosmos DB**
  - Improvements querying with partition keys
  - Point Read
  - Vector Similarity Search (preview)
- **Migration & Seeding**
- **Support for .NET NativeAOT (preview)**

CosmosDB for NOSQL & Partition Keys 1/3

Sharding in Sql Server



Sharding in CosmosDB



CosmosDB for NOSQL & Partition Keys 2/3

Query/PartitionKey

```
var res1 = await context.ToDoItems
                .Where(x => x.PartitionKey == "partitionTest" &&
                      x.Title.StartsWith("ciao"))
                 .ToListAsync();

var res2 = await context.ToDoItems
                 .WithPartitionKey("partitionTest")
                 .Where(x => x.Title.StartsWith("ciao"))
                 .ToListAsync();
```

Partition keys: Compatible Types

```
modelBuilder.Entity<ToDoItem>()
                .ToContainer("ToDoItems")
             .HasPartitionKey(c => c.PartitionKey)
             .HasNoDiscriminator()
             .HasKey(c => c.Id);

public class ToDoItem
{
   …..
    public Guid PartitionKey { get; set; }
….
```

Hierarchical partition keys

```
public class UserSession
{
    // Item ID
    public Guid Id { get; set; }

    // Partition Key
    public string TenantId { get; set; }
    public Guid UserId { get; set; }
    public int SessionId { get; set; }

    // Other members
    public string Username { get; set; }
}
```

```
modelBuilder.Entity<UserSession>()
                .ToContainer("UsersSession")
             .HasPartitionKey(c => new { c.TenantId, c.UserId, c.SessionId })
             .HasNoDiscriminator()
             .HasKey(c => c.Id);
```

CosmosDB for NOSQL & Partition Keys 3/3

Point Read

```
var res = await context.ToDoItems
              .FirstOrDefaultAsync(x => x.PartitionKey == "partitionTest" &&
                                        x.Id == guidTest);
```

EF v8

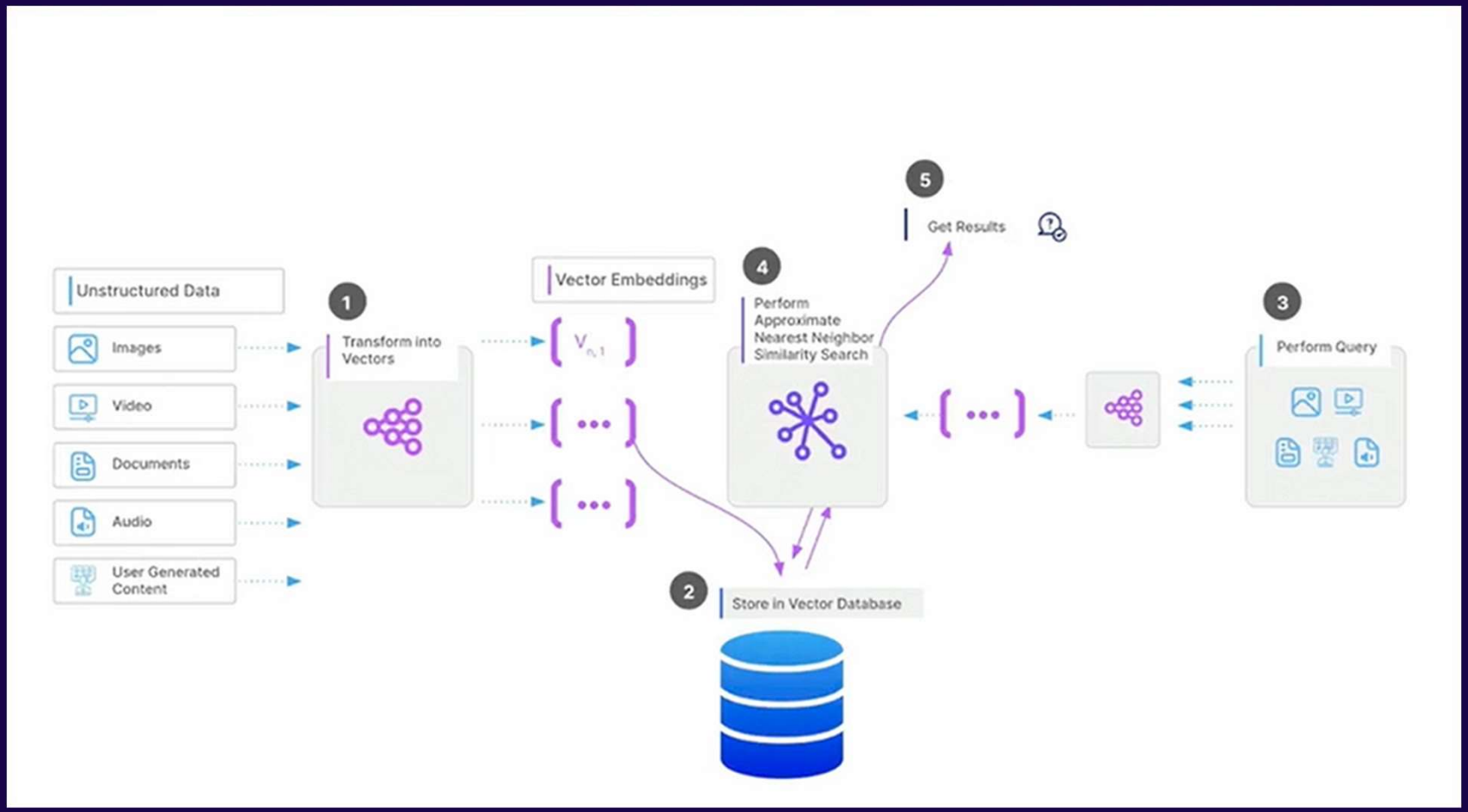
Executed ReadNext (1749.0602 ms, 2.79 RU) ActivityId='7370199d-482a-4623-9e83-95f622dc4c7b', Container='ToDoItems', Partition='partitionTest', Parameters=[@__guidTest_0='5702dbdc-be24-41b1-9207-525c4909a42c']

```
SELECT c
FROM root c
WHERE (c["Id"] = @__guidTest_0)
OFFSET 0 LIMIT 1
```

EF v9

Executed ReadItem (255 ms, 1 RU) ActivityId='42c30985-d39e-453d-8f96-be484edac064', Container='ToDoItems', Id='5702dbdc-be24-41b1-9207-525c4909a42c', Partition='["partitionTest"]'

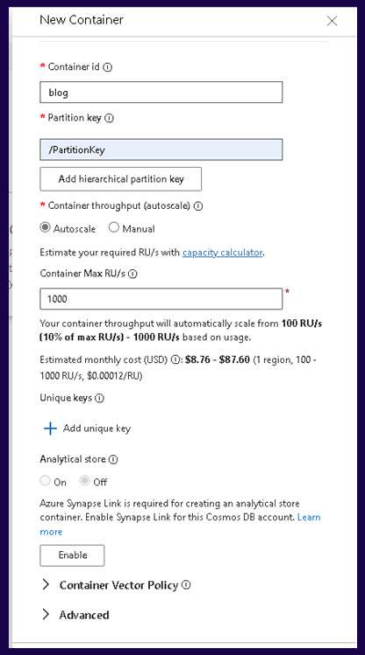
CosmosDB for NOSQL like Vector DB 1/3

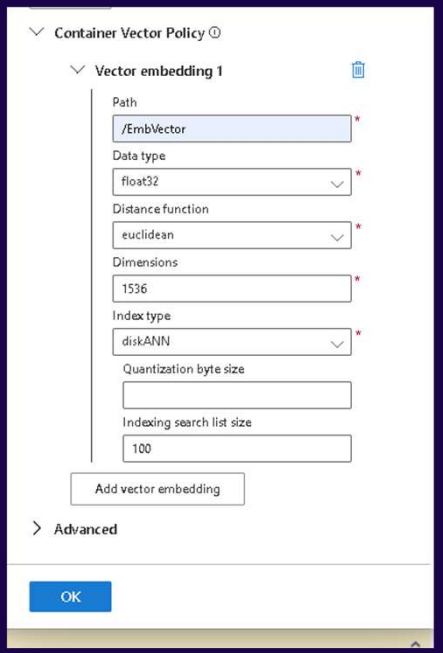


CosmosDB for NOSQL like Vector DB 2/3

Abilitazione Feature in Azure

Creazione Container





CosmosDB for NOSQL like Vector DB 3/3

Definition

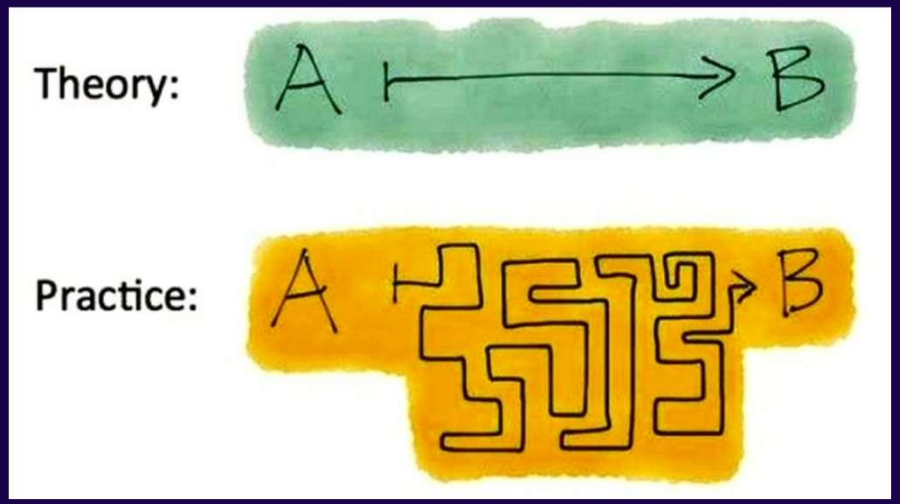
```
modelBuilder.Entity<Blog>()
                    .ToContainer("Blog")
                    .HasPartitionKey(c => c.PartitionKey)
                    .HasNoDiscriminator()
                    .HasKey(c => c.Id);

#pragma warning disable EF9103
modelBuilder.Entity<Blog>()
                     .Property(b => b.EmbVector)
                     .IsVector(distanceFunction: DistanceFunction.Cosine, dimensions: 1536);
#pragma warning restore EF9103
```

Ricerca similarità per vettore

```
var blogs = await context.Blog
        .OrderBy(s =>
            EF.Functions
              .VectorDistance(s.EmbVector, floatArray)
            ).Take(5)
            .ToListAsync();
```

```
public class Blog
{
    public int Id { get; set; }
    public string Titolo { get; set; }
    public float[] EmbVector { get; set; }
    public string PartitionKey { get; set; }
}
```

Migration – La Teoria vs la pratica



La teoria

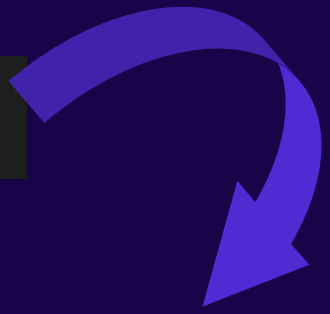
Applying database migration is part of application deployment, not application startup

……La pratica

```
using var context = new ToDoItemDbContext();
await context.Database.MigrateAsync();
```

Migration & Locking: whaat ?

```
await context.Database.MigrateAsync();
```



Sql Server-way

```
...
EXEC @result = sp_getapplock @Resource = '__EFMigrationsLock', @LockOwner = 'Session', @LockMode = 'Exclusive';
…..

EXEC @result = sp_releaseapplock @Resource = '__EFMigrationsLock', @LockOwner = 'Session';
…
```

….e se il database non supporta alcun meccanismo di lock ?
Uso tabella *__EFMigrationLock*

Seeding: “old way “ 1/2

```
public virtual DataBuilder<TEntity> HasData(params TEntity[] data)
{
   return HasData((IEnumerable<object>)data);
}
```

Esempio

```
modelBuilder.Entity<ToDoItem>().HasData(

      new ToDoItem() {
            Id = new Guid("{E96600FA-5C32-4B01-8637-59B93960D870}"),
            Description="Partecipare all'evento .NET 9 di .NET Liguria",
            Title="Test",
            Done = false
      }

);
```

Seeding: “old way” 2/2

Codice che genera errore

```
modelBuilder.Entity<ToDoItem>().HasData(

      new ToDoItem() {
            Description="Partecipare all'evento .NET 9 di .NET Liguria",
            Title="Test",
            Id = Guid.NewGuid(),
            Done = false
      }
);
```

The model for context 'DbContext' has pending changes.
Add a new migration before updating the database.
This exception can be suppressed or logged by passing event ID 'RelationalEventId.PendingModelChangesWarning' to the 'ConfigureWarnings' method in 'DbContext.OnConfiguring' or 'AddDbContext'.

Soluzione work-around

```
optionsBuilder.ConfigureWarnings(w => w.Ignore(RelationalEventId.PendingModelChangesWarning));
```

Seeding: la “recommended way “

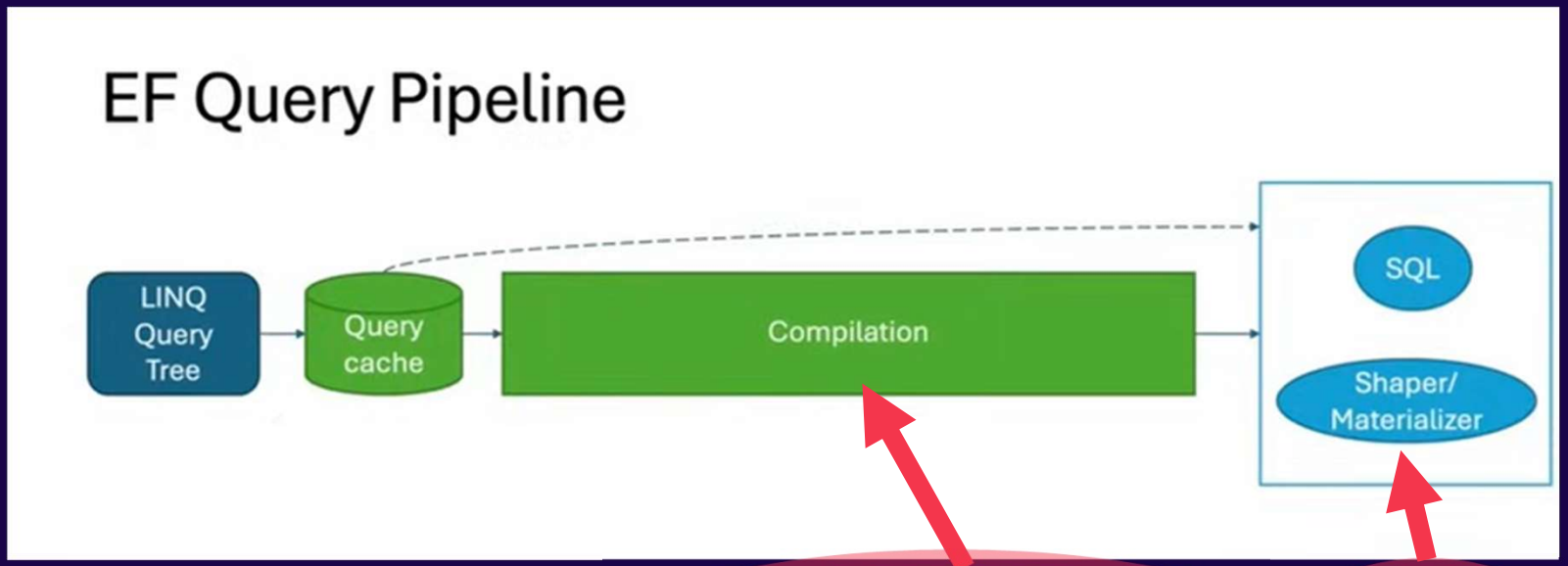
```
optionsBuilder
    .UseSqlServer(«ConnectionString»)
    .UseAsyncSeeding(async (dbctx,_,ct) => {
        var todoItemDbSet=dbctx.Set<ToDoItem>();
        var toDoItem = await todoItemDbSet.FirstOrDefaultAsync(x => x.Title == "Partecipare all'evento di DotNetLiguria");
        if (toDoItem == null)
        {
            await todoItemDbSet.AddAsync(new ToDoItem { Title = "Partecipare all'evento .NET9 di DotNetLiguria",
                                                        Description = "Partecipare all'evento .NET9 di DotNetLiguria",
                                                        Done = false }
                                        );
            await dbctx.SaveChangesAsync();
        }
    }).UseSeeding((dbctx, _) => {
        var todoItemDbSet = dbctx.Set<ToDoItem>();
        var toDoItem = todoItemDbSet.FirstOrDefault(x => x.Title == "Partecipare all'evento .NET9 di DotNetLiguria");
        if (toDoItem == null)
        {
            todoItemDbSet.Add(new ToDoItem { Title = "Partecipare all'evento .NET9 di DotNetLiguria",
                                             Description = "Partecipare all'evento .NET9 di DotNetLiguria",
                                             Done = false });
            dbctx.SaveChanges();
        }

    })
    .LogTo(Console.WriteLine, (id, lev) => id == RelationalEventId.CommandExecuting)
    .EnableSensitiveDataLogging()
    .EnableDetailedErrors();
```

AOT & Entity Framework

Native AOT: What is ?

- Compile .NET applications ahead-of-time directly to final binary
- No more just-in-time compilation
- Small, self-contained binaries that have smaller memory footprints and are easier to deploy
- Running applications in environments where just-in-time compilation isn't supported
- ...and ...what about Entity Framework ??

Query Precompilation !




```
var listResult = dbcontext.Blog.Where(x=>x.Titolo == «EF Core9»).ToListAsync();.
```

C# interceptors: ehm... Cioè ???

- Native AOT si basa su C# Interceptors
- C# Interceptors <> Entity Framework Core Interceptors
- Introdotti in C# 12/.NET 8
- Un Interceptor è un metodo che permette di sostituirsi ad un'altro a "compile-time"
- Lo scopo principale è quello di generare metodi sostitutivi (scritti da un source-generator) rispetto a quelli dichiarati.

Il comando

```
dotnet ef dbcontext optimize --precompile-queries –nativeaot
```

Materializer

Legacy
.UseModel(MyCompiledModels.BlogsContextModel.Instance)

In EF9 NON più necessario

Query Generator

Legacy
EF.Compile

Additional code

Questionario