ENTITY FRAMEWORK CORE

PITFALLS!

BY KHALID @BUHAKMEH

follow me on Twitter @buhakmeh

WHO AM I?

- Khalid Abuhakmeh
  - Twitter: @buhakmeh
- ASP.NET Professional
- .NET Developer Advocate
- Technical Writer
  - abuhakmeh.com
- Meme Artisan
- Photoshop Hobbiest

It Depends

THINK ABOUT WHETHER IT APPLIES TO YOU

WHAT IS EF CORE?

follow me on Twitter @buhakmeh

DEVELOPER'S POINT OF VIEW

```
public class EntertainmentDbContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder options)
        => options.UseSqlite("Data Source=entertainment.db");

    public DbSet<Production> Productions { get; set; }
    public DbSet<Movie> Movies { get; set; }
    public DbSet<Series> Series { get; set; }
    public DbSet<Rating> Ratings { get; set; }
    public DbSet<Character> Characters { get; set; }
    public DbSet<Actor> Actors { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        seed data

        base.OnModelCreating(modelBuilder);
    }
}
```

```
(base) ef-core-entertainment on  main [!?] via Ⓒ base via .NET 6.0.100-preview.3.21202.5
→ dotnet ef --help
Entity Framework Core .NET Command-line Tools 5.0.0-rc.2.20475.6

Usage: dotnet ef [options] [command]

Options:
  --version         Show version information
  -h|--help         Show help information
  -v|--verbose      Show verbose output.
  --no-color        Don't colorize output.
  --prefix-output   Prefix output with level.

Commands:
  database    Commands to manage the database.
  dbcontext   Commands to manage DbContext types.
  migrations  Commands to manage migrations.

Use "dotnet ef [command] --help" for more information about a command.
```

JOURNEY THROUGH THE JUNGLE

- Environment
- Design Decisions
- Querying
- Internals
- Deployment

ENVIRONMENT

follow me on Twitter @buhakmeh

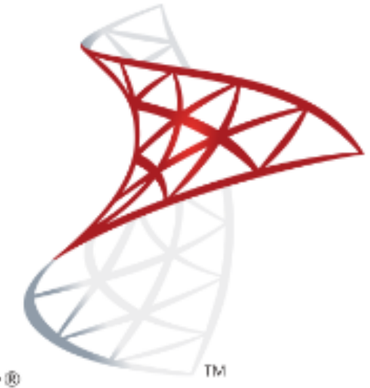
INSTALL TOOLS *Locally* VS. GLOBALLY

follow me on Twitter @buhakmeh

INSTALL LOCALLY

1. `dotnet new tool-manifest`
   - Create a `.config/dotnet-tools.json` file
2. `dotnet tool install dotnet-ef`
   - exclude the `--global` flag

Not ALL DATABASES ARE THE SAME



Microsoft® SQL Server®

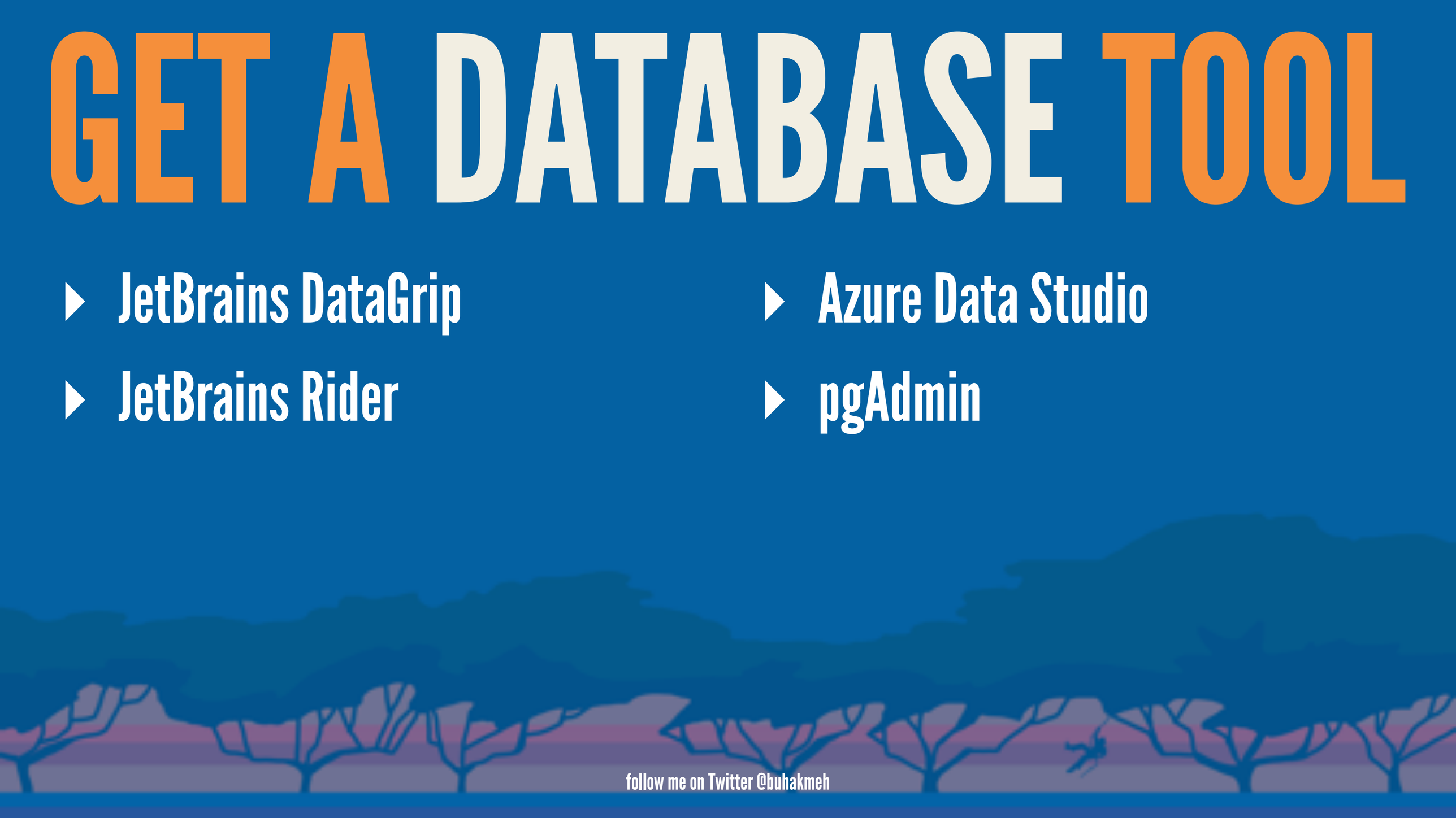
MySQL®

SQLite

IBM DB2

GET A DATABASE TOOL

- JetBrains DataGrip
- JetBrains Rider
- Azure Data Studio
- pgAdmin



DESIGN DECISIONS

follow me on Twitter @buhakmeh

USING A NON-CONVENTIAL NAMING SCHEME

SAVE YOURSELF THE HEADACHE

- Snake case - automobile_manufacturers
- Kebab case - railroad-systems
- prefixes - tblCrocodiles

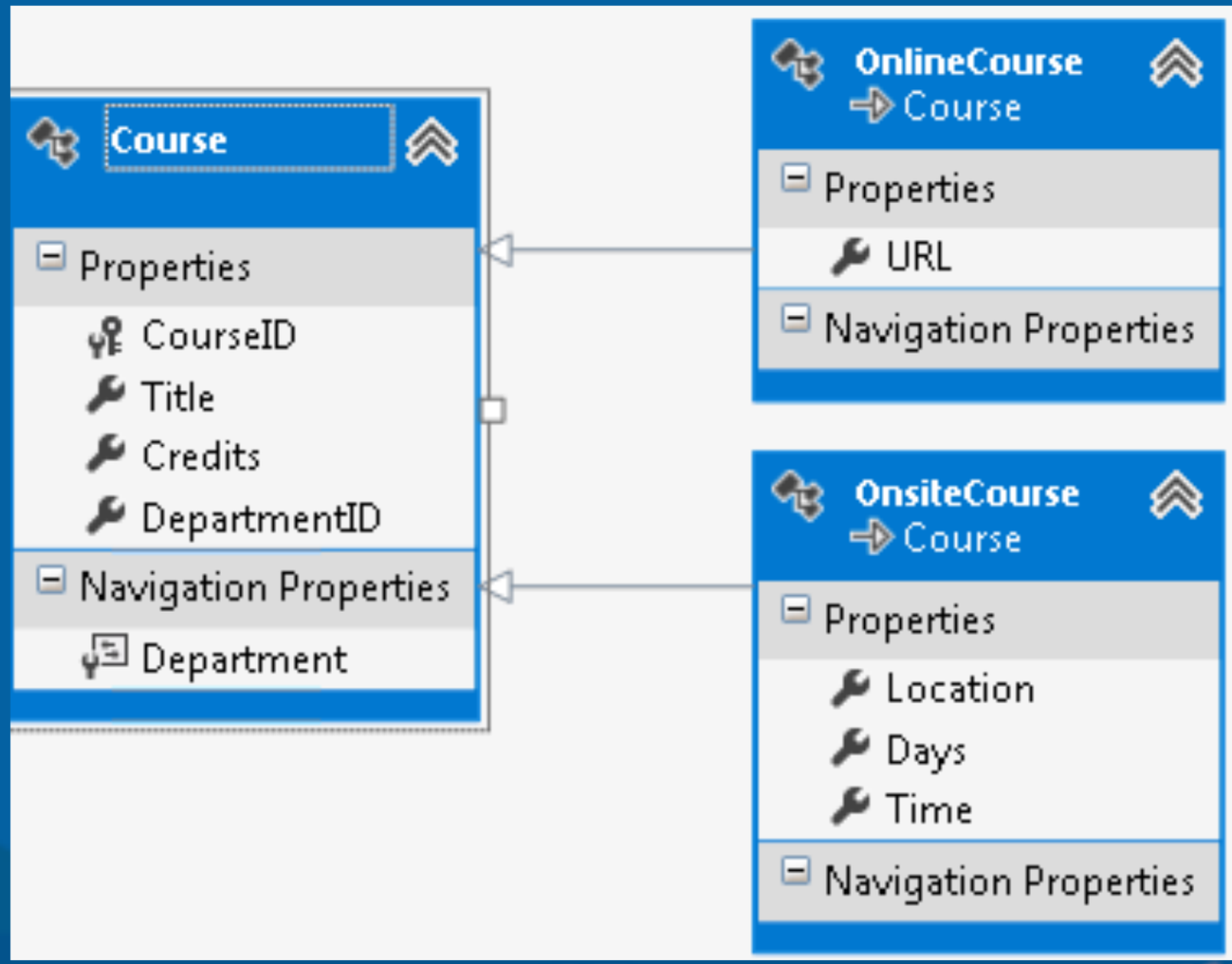
FORGETTING REVERSE ENGINEERING

```
dotnet ef dbcontext scaffold
    "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Chinook"
    Microsoft.EntityFrameworkCore.SqlServer
```

DON'T USE

TABLE-PER-TYPE

(NOPE, STOP, PLEASE...)

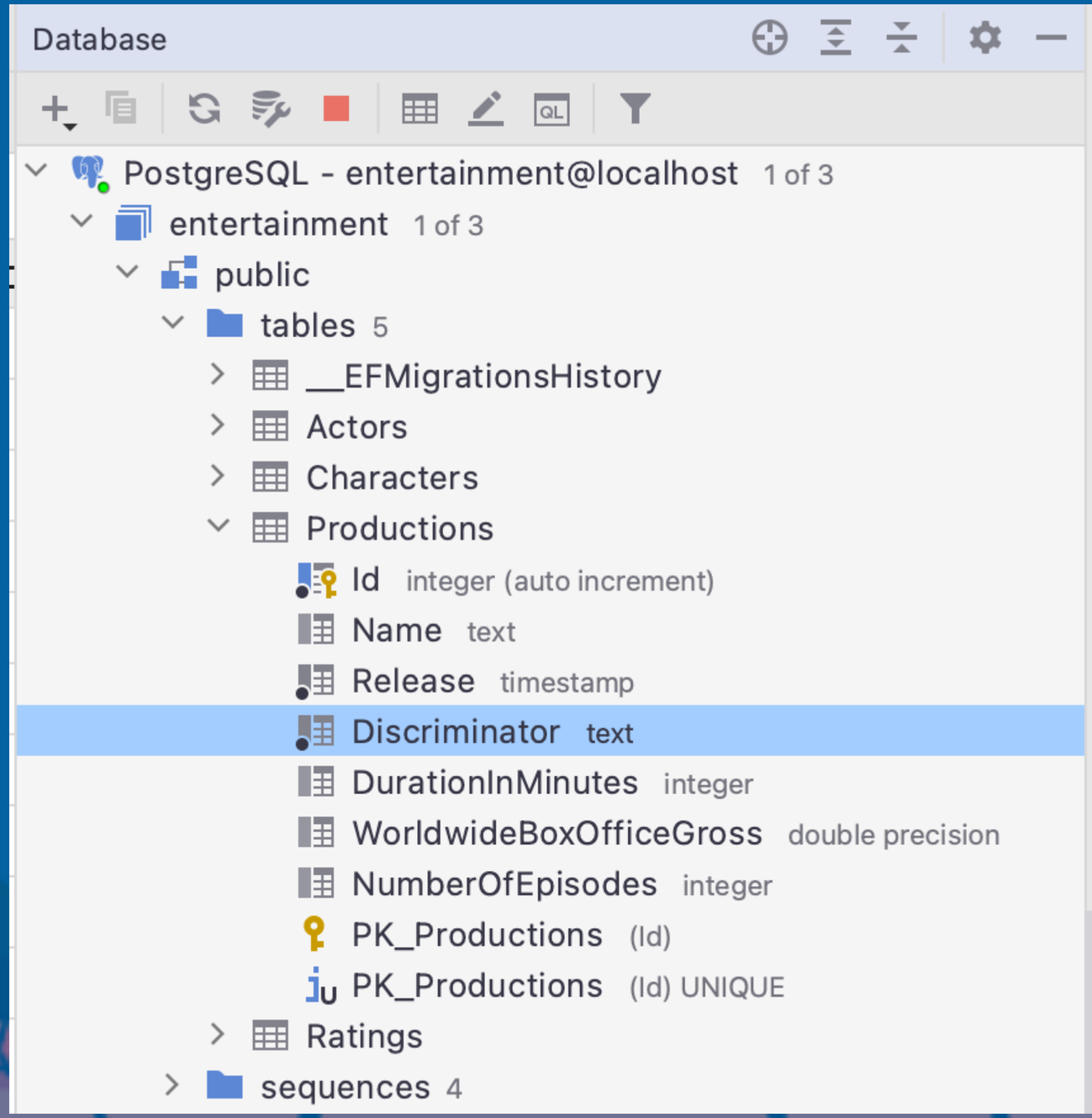


TPT & SQL GENERATION

```
// Course & Departments +
// OnlineCourses, OnsiteCourses,
// PrivateCourses, SelfGuidedCourses
var sql  =
    database
        .Courses
        .Include(c ⇒ c.Department)
        .ToQueryString();
```

```
SELECT "c"."CourseId",
       "c"."Credits",
       "c"."DepartmentId",
       "c"."Title",
       "o"."Url",
       "o0"."Days",
       "o0"."Location",
       "o0"."Time",
       "p"."Instructor",
       "p"."Student",
       "s"."NumberOfSections",
       CASE
           WHEN "o"."CourseId" IS NOT NULL THEN 'OnlineCourse'
           END AS "Discriminator",
       "d"."DepartmentId",
       "d"."CourseId"
FROM "Courses" AS "c"
         LEFT JOIN "OnlineCourses" AS "o" ON "c"."CourseId" = "o"."CourseId"
         LEFT JOIN "OnsiteCourses" AS "o0" ON "c"."CourseId" = "o0"."CourseId"
         LEFT JOIN "PrivateCourses" AS "p" ON "c"."CourseId" = "p"."CourseId"
         LEFT JOIN "SelfGuidedCourses" AS "s" ON "c"."CourseId" = "s"."CourseId"
         LEFT JOIN "Departments" AS "d" ON "c"."DepartmentId" = "d"."DepartmentId"
```

USE
TABLE-PER-HIERARCHY
(SIMILAR ENTITIES IN THE SAME TABLE)



follow me on Twitter @buhakmeh

TEST DATA IN SEED DATA

```
// actors
modelBuilder.Entity<Actor>().HasData(new Actor[]
{
    new Actor { Id = 1, Name = "Robert Downey Jr." },
    new Actor { Id = 2, Name = "Chris Evans" },
    new Actor { Id = 3, Name = "Danai Guira" },
    new Actor { Id = 4, Name = "Donald Glover" },
    new Actor { Id = 5, Name = "Beyoncé" },
    new Actor { Id = 6, Name = "Donny Yen" },
    new Actor { Id = 7, Name = "Will Smith" },
    new Actor { Id = 8, Name = "Maggie Smith" },
```

```
migrationBuilder.InsertData(
    table: "Actors",
    columns: new[] { "Id", "Name" },
    values: new object[] { 1, "Robert Downey Jr." });

migrationBuilder.InsertData(
    table: "Actors",
    columns: new[] { "Id", "Name" },
    values: new object[] { 18, "Melissa McBride" });
```

follow me on Twitter @buhakmeh

QUERYING

follow me on Twitter @buhakmeh

TREATING ENTITIES

exactly LIKE C# OBJECTS

```
6 usages  2 inheritors  Khalid Abuhakmeh *  4 exposing APIs
public abstract class Production
{
    12 usages
    public int Id { get; set; }
    18 usages
    public string Name { get; set; }
    16 usages
    public DateTime Release { get; set; }

    public List<Character> Characters { get; set; } = new();
    2 usages
    public List<Rating> Ratings { get; set; } = new();

    // Doh!
    new *
    public double AverageRating ⇒
        Ratings.Average(selector: r :Rating ⇒ r.Stars);
}
```

follow me on Twitter @buhakmeh

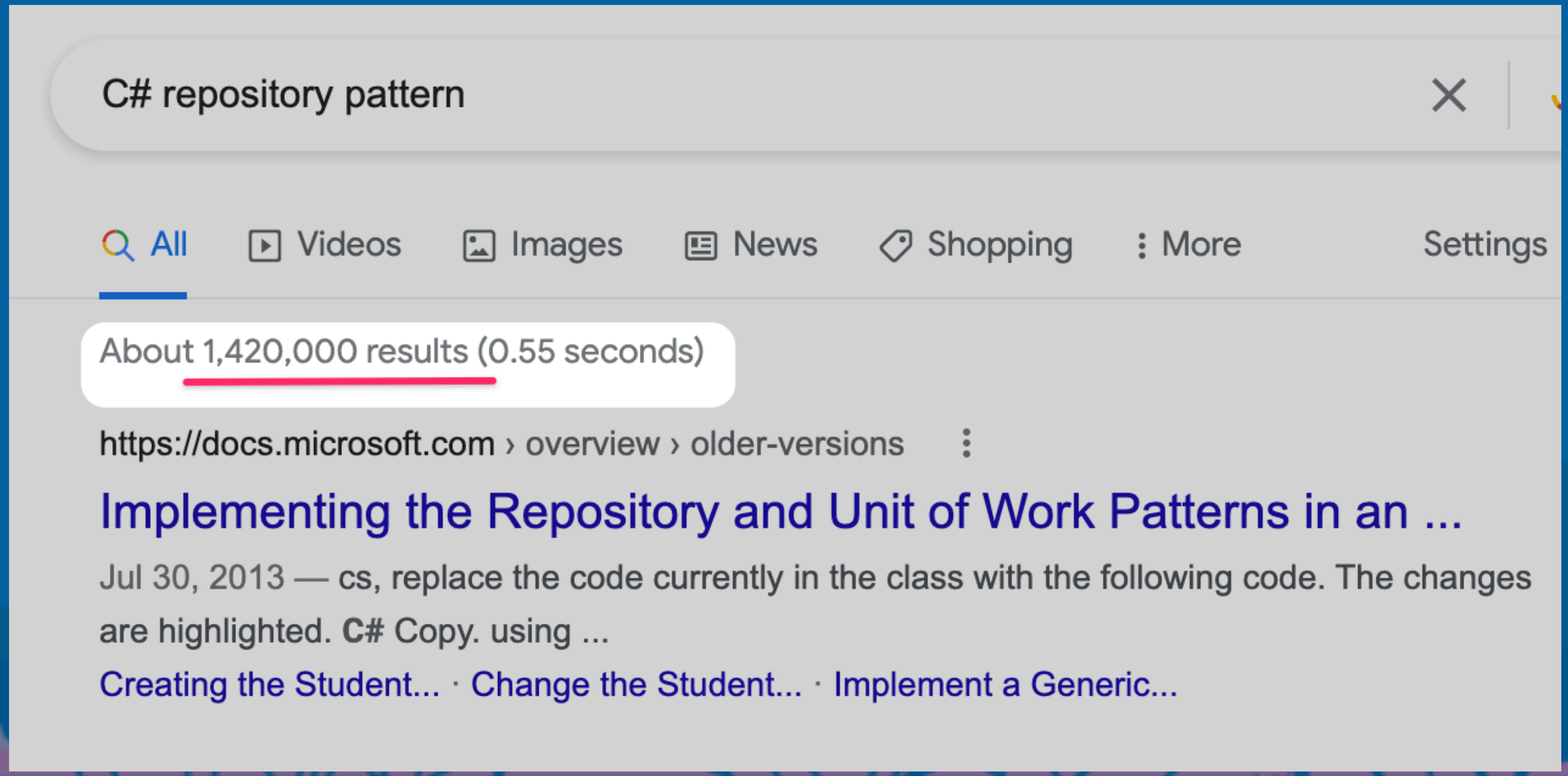
WRAPPING DBCONTEXT IN A REPOSITORY FOR EACH ENTITY

follow me on Twitter @buhakmeh

```
public class CharacterRepository
    : IRepository<Character>
{
    private readonly EntertainmentDbContext database;

    public CharacterRepository(EntertainmentDbContext database)
    {
        this.database = database;
    }

    public IList<Character> All()
    {
        throw new System.NotImplementedException();
    }

    public Character FindById(int id)
    {
        throw new System.NotImplementedException();
    }

    public Task Update(Character entity)
    {
        throw new System.NotImplementedException();
    }
}
```

```
1 usage  1 inheritor
public interface IRepository<T>
{
    1 implementation
    IList<T> All();
    1 implementation
    T FindById(int id);
    1 implementation
    Task Update(T entity);
}
```

REPOSITORY PATTERN POSTS



QUERYING ALL THE DATA!

```
// 2. Highest Rated productions
var highestRated :Task<List<...>> = database // EntertainmentDbContext
    .Productions // DbSet<Production>
    .Select(x :Production ⇒ new
    {
        id = x.Id,
        name = x.Name,
        avg = x.Ratings.Average( selector: r :Rating ⇒ r.Stars),
        type = x.GetType().Name
    }) // IQueryable<{id,name,...}>
    .OrderByDescending(x :{id,name,...} ⇒ x.avg)
    // Oops! // IOrderedQueryable<{id,name,...}>
    .ToListAsync();
    // Use Skip & Take
```

RUNNING MULTIPLE QUERIES

Use the `Include` keyword to reduce network roundtrips.

```
var movies = database
    .Movies
    .Include(m ⇒ m.Characters)
    .OrderByDescending(x ⇒ x.WorldwideBoxOfficeGross);
```

NOT RUNNING MULTIPLE QUERIES

```sql
SELECT p."Id", p."Discriminator", p."Name", p."Release",
       p."DurationInMinutes", p."WorldwideBoxOfficeGross", c."Id",
       c."ActorId", c."Name", c."ProductionId"
FROM "Productions" AS p
         LEFT JOIN "Characters" AS c ON p."Id" = c."ProductionId"
WHERE (p."Discriminator" = 'Movie') AND (p."Id" = 1)
ORDER BY p."Id", c."Id"
```

3 rows CSV

```
1,Movie,Avengers: Endgame,2019-04-26 00:00:00.000000,181,2797800564,1,1,Tony Stark,1
1,Movie,Avengers: Endgame,2019-04-26 00:00:00.000000,181,2797800564,2,2,Steve Rogers,1
1,Movie,Avengers: Endgame,2019-04-26 00:00:00.000000,181,2797800564,3,3,Okoye,1
```

```sql
Select * from "Productions" p
    where p."Discriminator" = 'Movie' and p."Id" = 1;
```

1 row Tx: Auto DDL

```
1,Avengers: Endgame,2019-04-26 00:00:00.000000,Movie,181,2797800564,
```

```sql
Select C.*  from "Characters" C
    inner join "Productions" P on P."Id" = C."ProductionId"
    where p."Discriminator" = 'Movie' and p."Id" = 1;
```

entertainment.public.Characters 2 | entertainment.public.Characters

3 rows Tx: Auto DDL

	"Id"	"ProductionId"	"Name"	"ActorId"
1	1	1	Tony Stark	1
2	2	1	Steve Rogers	2
3	3	1	Okoye	3

follow me on Twitter @buhakmeh

NOT RUNNING MULTIPLE QUERIES

CONTINUED

```
var split = database   split: "SELECT p."Id", p."Discriminator",
    .Movies
    .AsSplitQuery()
    .Include(m ⇒ m.Characters)
    .ThenInclude(c ⇒ c.Actor)
    .OrderByDescending(x ⇒ x.WorldwideBoxOfficeGross)
    .ToQueryString();
```

```
SELECT p."Id", p."Discriminator", p."Name", p."Release",
  p."DurationInMinutes", p."WorldwideBoxOfficeGross"
FROM "Productions" AS p
WHERE p."Discriminator" = 'Movie'
ORDER BY p."WorldwideBoxOfficeGross" DESC, p."Id"

This LINQ query is being executed in split-query mode. The SQL
 shown is for the first query to be executed. Additional
queries may also be executed depending on the results of the
first query.
```

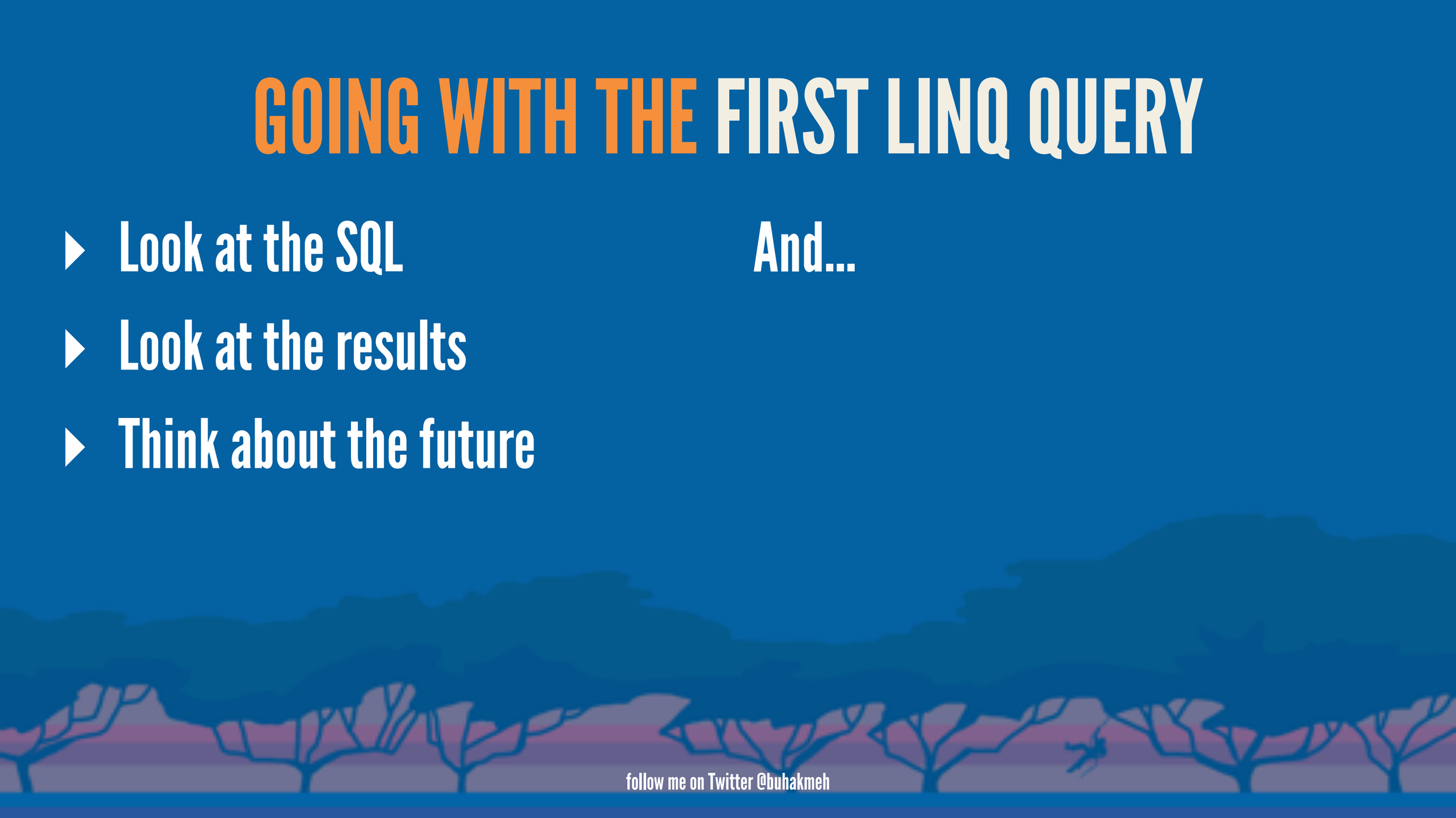
FORGETTING ABOUT PROJECTIONS

```
var actorsPlayingThemselves:IQueryable<{character,actor}> =
    database.Characters// DbSet<Character>
        .Where(c:Character ⇒ c.Name == c.Actor.Name)// IQueryable<Character>
        .Select(c:Character ⇒ new
        {
            character = c.Name,
            actor = c.Actor.Name
        });
```

GOING WITH THE FIRST LINQ QUERY

- Look at the SQL
- Look at the results
- Think about the future

And...



EF CORE TRANSLATES WHAT YOU WRITE!

```
var gnarly = database
    .Productions
    .Select(m ⇒
        new
        {
            m.Name,
            characters = m.Characters.Select(c ⇒ new
            {
                c.Id,
                c.Name,
                actorId = c.Actor.Id,
                actorName = c.Actor.Name,
                averageProductionRating = c.Production.Ratings.Average(x ⇒ x.Stars)
            }).ToList()
        }
    )
    .OrderBy(x ⇒ x.characters.First().averageProductionRating)
    .Skip(1)
    .Take(100);
```

```
.param set @__p_1 100
.param set @__p_0 1

SELECT "t"."Name",
       "t"."Id",
       "t0"."Id",
       "t0"."Name",
       "t0"."actorId",
       "t0"."actorName",
       "t0"."averageProductionRating",
       "t0"."Id0"
FROM (
        SELECT "p0"."Name",
               "p0"."Id",
               (
                   SELECT (
                            SELECT AVG(CAST("r"."Stars" AS REAL))
                            FROM "Ratings" AS "r"
                            WHERE "p"."Id" = "r"."ProductionId")
                   FROM "Characters" AS "c"
                            INNER JOIN "Productions" AS "p" ON "c"."ProductionId" = "p"."Id"
                   WHERE "p0"."Id" = "c"."ProductionId"
                   LIMIT 1) AS "c" FROM "Productions" AS "p0"
ORDER BY (
    SELECT (
    SELECT AVG (CAST ("r"."Stars" AS REAL))
    FROM "Ratings" AS "r"
    WHERE "p"."Id" = "r"."ProductionId")
    FROM "Characters" AS "c"
    INNER JOIN "Productions" AS "p" ON "c"."ProductionId" = "p"."Id"
    WHERE "p0"."Id" = "c"."ProductionId"
    LIMIT 1)
    LIMIT @__p_1
OFFSET @__p_0 ) AS "t" LEFT JOIN (
SELECT "c0"."Id", "c0"."Name", "a"."Id" AS "actorId", "a"."Name" AS "actorName", (
    SELECT AVG (CAST ("r0"."Stars" AS REAL))
    FROM "Ratings" AS "r0"
    WHERE "p1"."Id" = "r0"."ProductionId") AS "averageProductionRating", "p1"."Id" AS "Id0", "c0"."ProductionId"
FROM "Characters" AS "c0"
    INNER JOIN "Actors" AS "a"
ON "c0"."ActorId" = "a"."Id"
    INNER JOIN "Productions" AS "p1" ON "c0"."ProductionId" = "p1"."Id"
    ) AS "t0" ON "t"."Id" = "t0"."ProductionId"
ORDER BY "t"."c", "t"."Id", "t0"."Id", "t0"."actorId", "t0"."Id0"
```

follow me on Twitter @buhakmeh

INTERNALS

follow me on Twitter @buhakmeh

DON'T ENABLE LAZY LOADING

```
foreach (var rating in database.Ratings) {
    // Aggggggggggh!
    // if lazy-loading was enabled
    // the call to rating.Production would
    // trigger a database call for each new
    // production instance.
    Console.WriteLine($"{rating.Production.Name}: {rating.Stars}");
}
```

TRACKING ALL ENTITIES

```
// tracking entities
// and managing state
var movies = database
    .Movies
    .OrderByDescending(x ⇒ x.WorldwideBoxOfficeGross);

// reading rows and
// mapping data as is
movies = database
    .Movies
    .AsNoTracking()
    .OrderByDescending(x ⇒ x.WorldwideBoxOfficeGross);

// track uniqueness of
// entities by id and store
// them in memory once
movies = database
    .Movies
    .AsNoTrackingWithIdentityResolution()
    .OrderByDescending(x ⇒ x.WorldwideBoxOfficeGross);
```

follow me on Twitter @buhakmeh

CONSTANTS
IN EXPRESSIONS

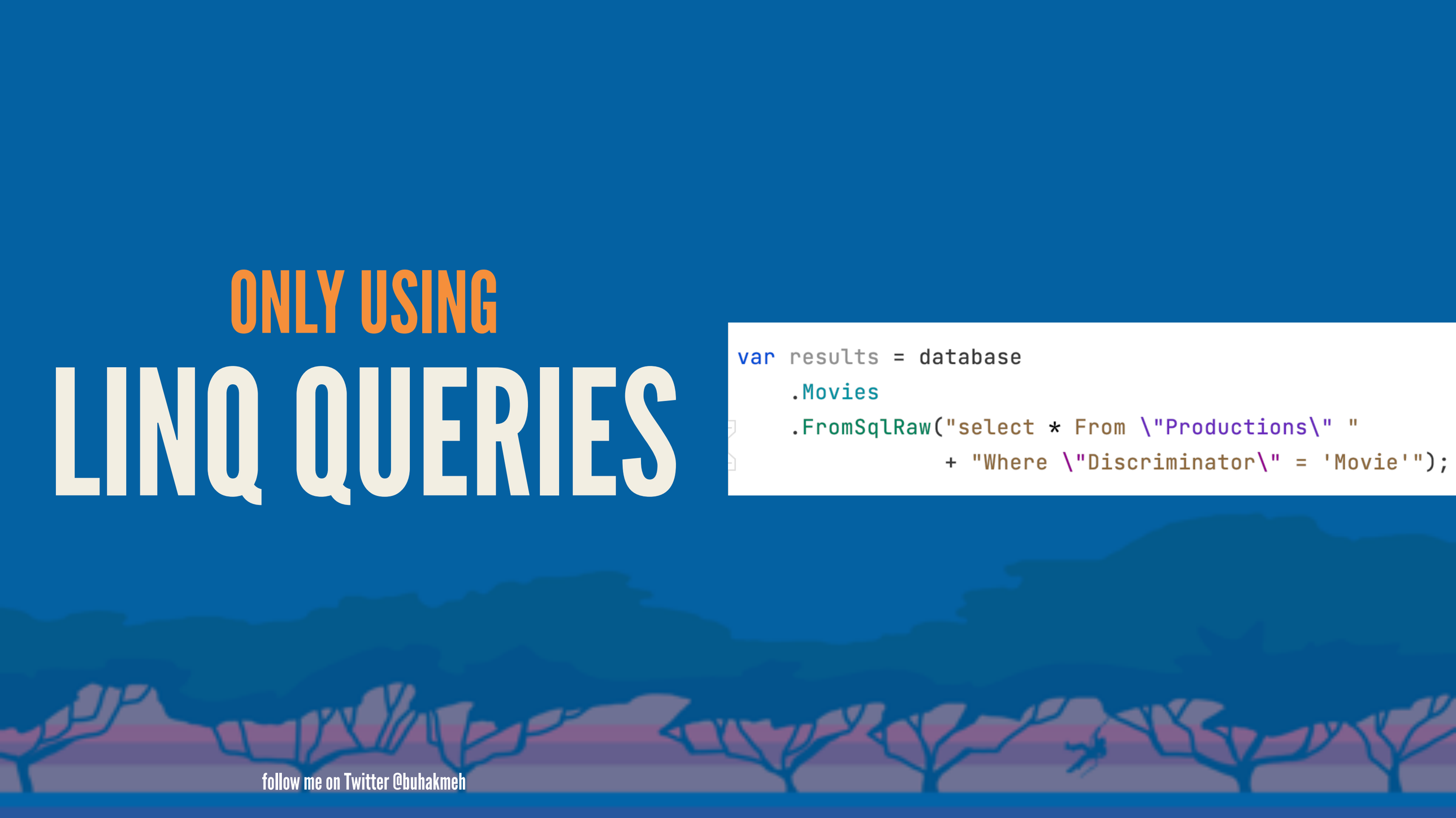
follow me on Twitter @buhakmeh

```
// two different expressions
var first = database
    .Movies
    .First(m ⇒ m.Id == 1);

var second = database
    .Movies
    .First(m ⇒ m.Id == 2);

// same expression for two
// different queries
var id = 1;
first = database
    .Movies
    .First(m ⇒ m.Id == id);

id = 2;
second = database
    .Movies
    .First(m ⇒ m.Id == id);
```

ONLY USING

LINQ QUERIES

```
var results = database
    .Movies
    .FromSqlRaw("select * From \"Productions\" "
                + "Where \"Discriminator\" = 'Movie'");
```



follow me on Twitter @buhakmeh

INTRODUCING SQL INJECTION BY ACCIDENT

```
// Safe
string id = "1";
var results = database
    .Movies
    .FromSqlInterpolated($"Select * From Movies where id = {id}")
    .ToQueryString();

// SQL Injection Attack -- Ooops
string id = "1";
var query = $"Select * From Movies where id = {id}";
var results = database
    .Movies
    .FromSqlRaw(query)
    .ToQueryString();
```

follow me on Twitter @buhakmeh

QUERY CACHING

> **Note**
>
> EF Core's **event counters** report the Query Cache Hit Rate. In a normal application, this counter reaches 100% soon after program startup, once most queries have executed at least once. If this counter remains stable below 100%, that is an indication that your application may be doing something which defeats the query cache - it's a good idea to investigate that.

Console Copy

```
dotnet counters monitor Microsoft.EntityFrameworkCore -p <PID>
```

`dotnet-counters` will now attach to your running process and start reporting continuous counter data:

Console Copy

```
Press p to pause, r to resume, q to quit.
 Status: Running

[Microsoft.EntityFrameworkCore]
    Active DbContexts                                      1
    Execution Strategy Operation Failures (Count / 1 sec)  0
    Execution Strategy Operation Failures (Total)          0
    Optimistic Concurrency Failures (Count / 1 sec)        0
    Optimistic Concurrency Failures (Total)                0
    Queries (Count / 1 sec)                                1
    Queries (Total)                                      189
    Query Cache Hit Rate (%)                             100
    SaveChanges (Count / 1 sec)                            0
    SaveChanges (Total)                                    0
```

LONG-LIVED DBCONTEXT

```
var database =
    new EntertainmentDbContext();
```

USE DBCONTEXT POOLING

```
services
   .AddDbContextPool<BloggingContext>(
     options ⇒ options
       .UseSqlServer(connectionString)
     );
```

COMPILED MODEL

How does 10x performance sound to you? Our team created a sample project with a `DbContext` that contains **449 entity types, 6,390 properties and 720 relationships**. I wrote a console app that loops several times, creates a new instance of a `DbContext` and loads a set of entities with no filters or ordering. The start-up time for the first run consistently takes around **two seconds** on my laptop, with subsequent cached instances weighing in at about 1.5 seconds. Here's the output from a run:

```
$ dotnet run -c Release
Model has:
  449 entity types
  6390 properties
  720 relationships
Instantiating context...
It took 00:00:02.1603163.
Instantiating context...
It took 00:00:01.6268628.
Instantiating context...
It took 00:00:01.7144346.
Instantiating context...
It took 00:00:01.6090380.
Instantiating context...
It took 00:00:01.7049987.
```

After testing the baseline application, I used the new [EF Core tools Command Line Interface (CLI)] feature to optimize the `DbContext`:

```
dotnet ef dbcontext optimize -output-dir MyCompiledModels --namespace MyCompiledMode
```

The tool gave me instructions to add a single line of code to my `DbContext` configuration:

```
options.UseModel(MyCompiledModels.BlogsContextModel.Instance);
```

I made the update and re-ran the code to receive a **10x performance gain** with the initial model taking **257ms** to complete. The cached model reduced additional calls to just **10ms**.

```
$ dotnet run -c Release
Model has:
  449 entity types
  6390 properties
  720 relationships
Instantiating context...
It took 00:00:00.2573627.
Instantiating context...
It took 00:00:00.0132345.
Instantiating context...
It took 00:00:00.0119556.
Instantiating context...
It took 00:00:00.0101717.
Instantiating context...
It took 00:00:00.0139057.
```

*.NET 6 Preview 5: bit.ly/3wGiBWO

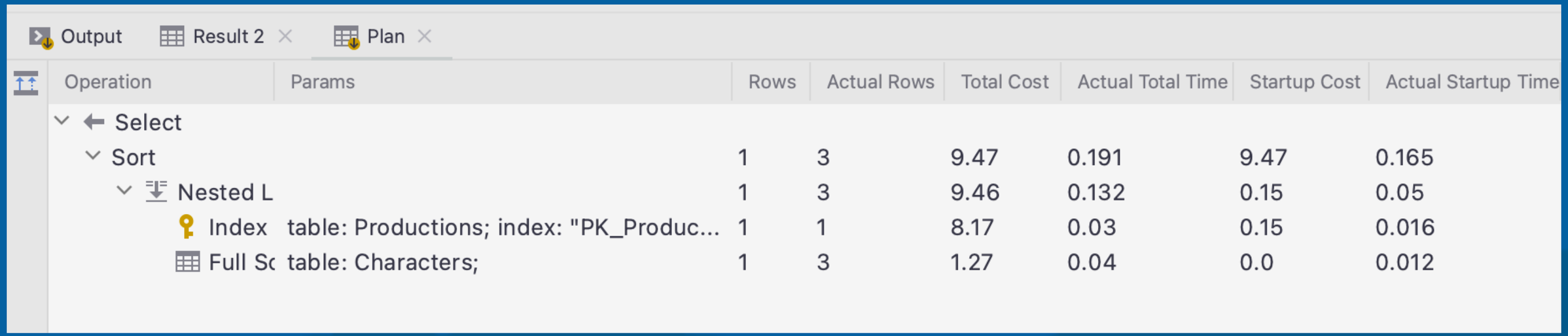
follow me on Twitter @buhakmeh

DEPLOYMENT

(BEFORE & AFTER)

follow me on Twitter @buhakmeh

Not MEASURING QUERY PERFORMANCE



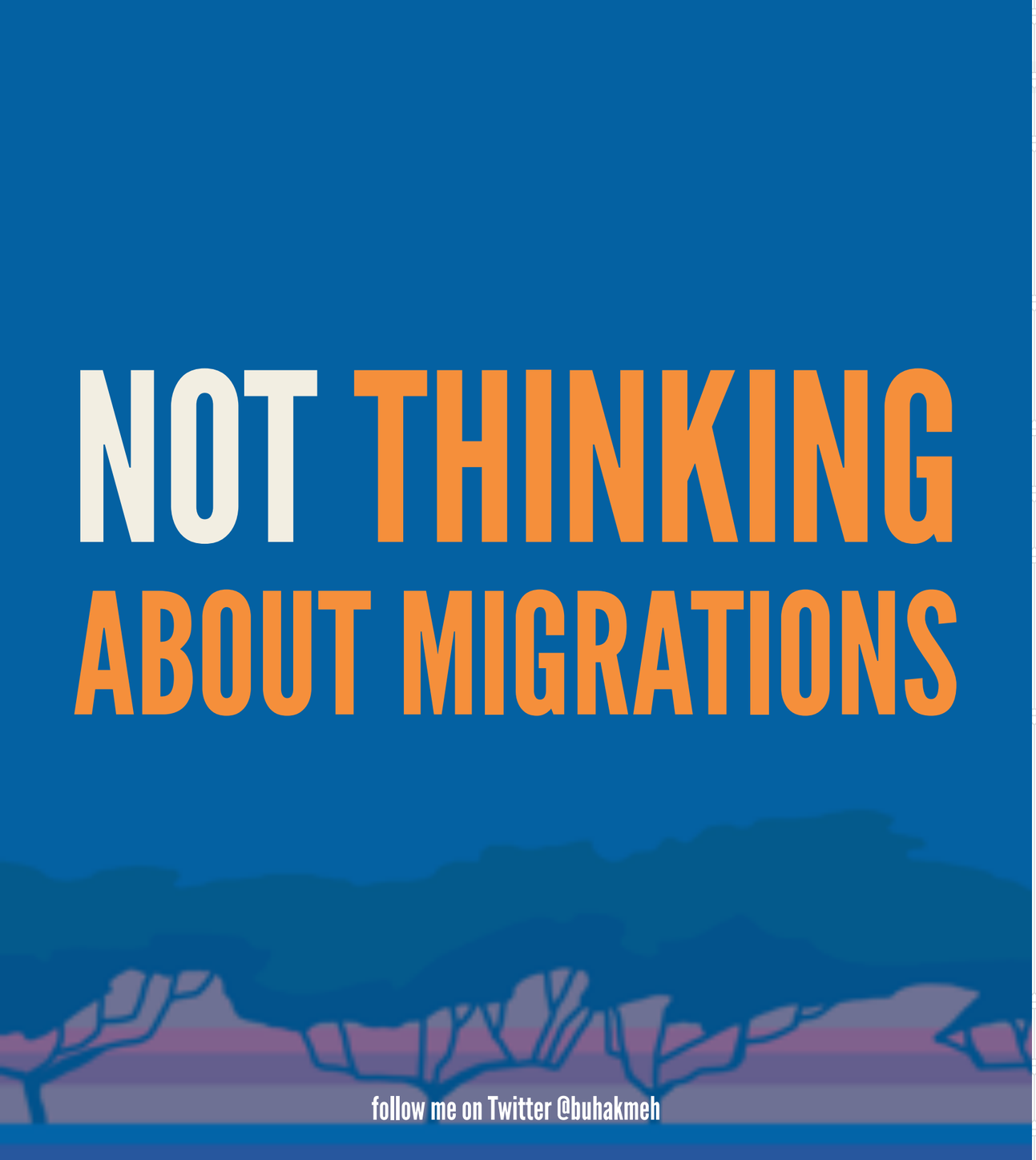
Output | Result 2 | Plan

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Sort		1	3	9.47	0.191	9.47	0.165
Nested L		1	3	9.46	0.132	0.15	0.05
Index	table: Productions; index: "PK_Produc...	1	1	8.17	0.03	0.15	0.016
Full Sc	table: Characters;	1	3	1.27	0.04	0.0	0.012

FORGETTING INDEXED COLUMNS

follow me on Twitter @buhakmeh

NOT THINKING ABOUT MIGRATIONS

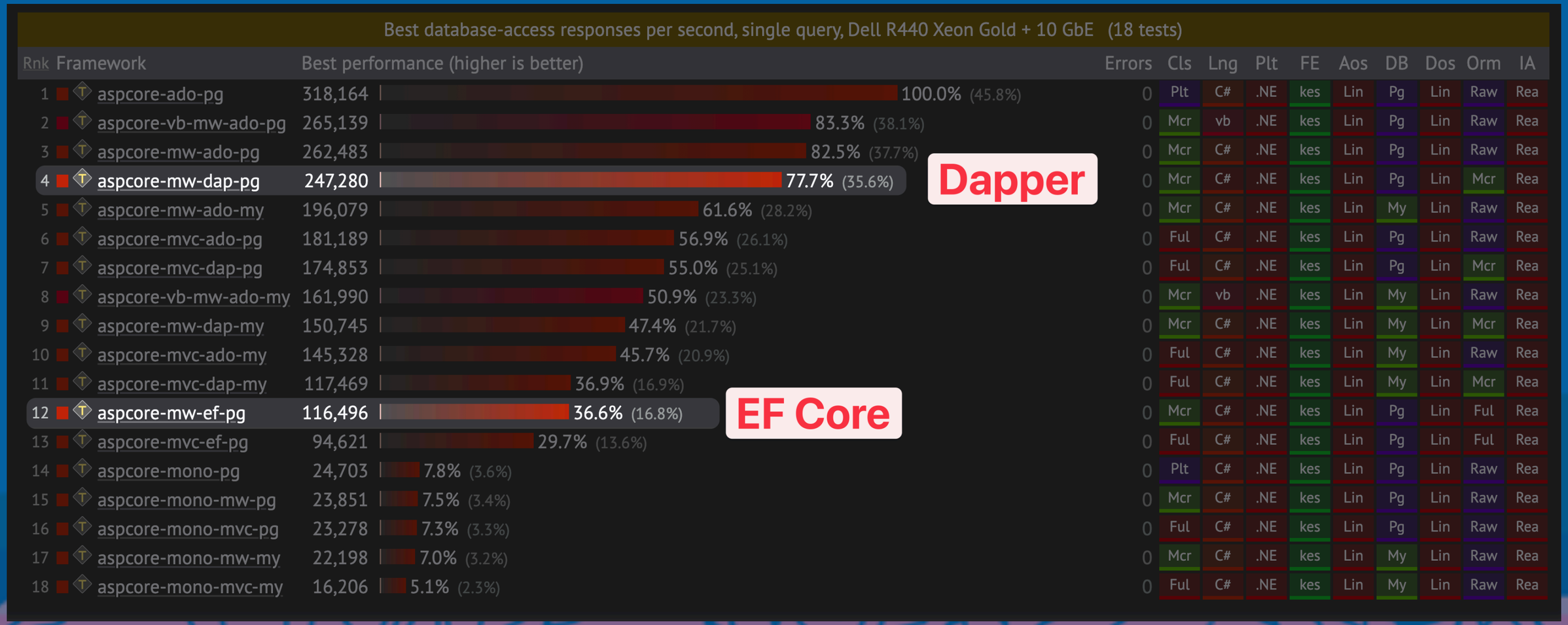



```
namespace Entertainment.Migrations
{
    public partial class InitialMigration : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Actors",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenera
                    Name = table.Column<string>(type: "text", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Actors", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Productions",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenera
                    Name = table.Column<string>(type: "text", nullable: true),
                    Release = table.Column<DateTime>(type: "timestamp without time zone
                    Discriminator = table.Column<string>(type: "text", nullable: false)
                    DurationInMinutes = table.Column<int>(type: "integer", nullable: tr
                    WorldwideBoxOfficeGross = table.Column<double>(type: "double precis
                    NumberOfEpisodes = table.Column<int>(type: "integer", nullable: tru
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Productions", x => x.Id);
                });
```

USING EF CORE IN THE FIRST PLACE

follow me on Twitter @buhakmeh

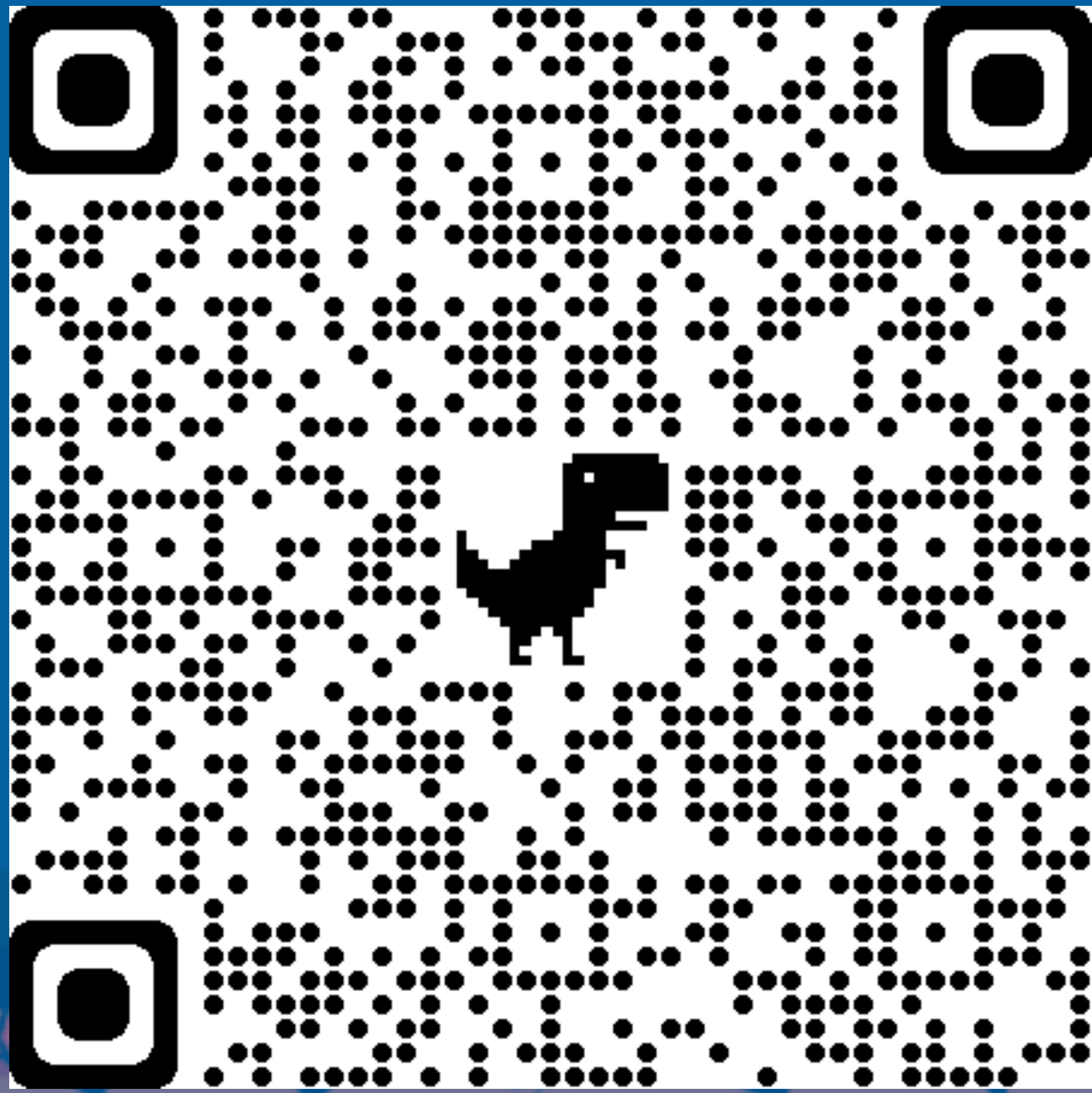
BENCHMARKS



follow me on Twitter @buhakmeh

MORE

ENTITY FRAMEWORK CORE 5 – PITFALLS TO AVOID AND IDEAS TO TRY



follow me on Twitter @buhakmeh

THANKS
& QUESTIONS

follow me on Twitter @buhakmeh