Accelebrate®

ACCELERATED LEARNING, CELEBRATED RESULTS®

Entity Framework Core

https://www.accelebrate.com
(877) 849-1850 ❖ info@accelebrate.com

Customized Technical Training

Accelerate
ACCELERATED LEARNING, CELEBRATED RESULTS®

On-Site, Customized Private Training

Don't settle for a one-size-fits-all class! Let Accelerate tailor a private class to your group's goals and experience. Classes can be delivered at your site or online (or a combination of both) worldwide. Visit our web site at **https://www.accelerate.com** and contact us at **sales@accelerate.com** for details.

Public Online Training

Need to train just 1-3 people? Attend one of our regularly scheduled, live, instructor-led public online classes. Class sizes are small (typically 3-6 attendees) and you receive just as much hands-on time and individual attention from your instructor as our private classes. For course dates, times, outlines, pricing, and registration, visit **https://www.accelerate.com/public-training-schedule.**

Newsletter

Want to find out about our latest class offerings? Subscribe to our newsletter **https://www.accelerate.com/newsletter**.

Blog

Get insights and tutorials from our instructors and staff! Visit our blog, **https://www.accelerate.com/blog** and join the discussion threads and get feedback from our instructors!

Learning Resources

Get access to learning guides, tutorials, and past issues of our newsletter at the Accelerate library, **https://www.accelerate.com/library**.

Call us for a training quote!

877 849 1850

Accelerate, Inc. was founded in 2002 with the goal of delivering **private training** that rapidly achieves participants' goals. Each year, our experienced instructors deliver hundreds of classes online and at client sites all over the US, Canada, and abroad. We pride ourselves on our **instructors' real-world experience** and ability to **adapt the training** to your team and their objectives. We offer a wide range of topics, including:

- Angular, React, and Vue
- JavaScript
- Data Science using R, Python, & Julia
- Excel Power Query
- Power BI & Tableau
- .NET & VBA programming
- SharePoint & Microsoft 365
- DevOps & CI/CD
- iOS & Android Development
- PostgreSQL, Oracle, and SQL Server
- Java, Spring, and Groovy
- Agile & IT Leadership
- Web/Application Server Admin
- HTML5 & Mobile Web Development
- AWS, Azure, and Cloud Computing
- Adobe & Articulate
- Docker, Kubernetes, Ansible, & Git
- Software Design and Testing
- AND MORE *(see back)*

"World class training with experts who know their field and have a passion for teaching. Our team is walking away with a wealth of knowledge that we look forward to bringing back to our daily work."

— Keith, Silicon Valley Bank

Like us on Facebook + Follow us on Twitter + Watch us on YouTube

Visit our website for a complete list of courses!

Adobe & Articulate
Adobe Captivate
Adobe Presenter
Articulate Storyline / Studio
Camtasia
RoboHelp

AWS, Azure, & Cloud
AWS
Azure
Cloud Computing
Google Cloud
OpenStack
VMware

Big Data
Alteryx
Apache Spark
Teradata
Snowflake SQL

Data Science and RPA
Blue Prism
Data Literacy
Django
Julia
Machine Learning
MATLAB
Minitab
Python
R Programming
SPSS
Tableau
UiPath

Database & Reporting
BusinessObjects
Crystal Reports
Excel Power Query
MongoDB
MySQL
NoSQL Databases
Oracle
Oracle APEX
Power BI
PivotTable and PowerPivot
PostgreSQL
SQL Server
Vertica Architecture & SQL

DevOps, CI/CD & Agile
Agile
Ansible
Chef
DEI
Docker
Git
Gradle Build System
IT Leadership
ITIL
Jenkins
Jira & Confluence
Kubernetes
Linux
Microservices
OpenShift
Red Hat
Software Design

Java
Apache Maven
Apache Tomcat
Groovy and Grails
Hibernate
Java & Web App Security
JavaFX
JBoss
Oracle WebLogic
Scala
Selenium & Cucumber
Spring Boot
Spring Framework

JS, HTML5, & Mobile
Angular
Apache Cordova
CSS
D3.js
Flutter
HTML5
iOS/Swift Development
JavaScript
MEAN Stack
Mobile Web Development
Node.js & Express
React & Redux
Svelte
Swift
Xamarin
Vue

Microsoft & .NET
.NET Core
ASP.NET
Azure DevOps
C#
Design Patterns
Entity Framework Core
IIS
Microsoft Dynamics CRM
Microsoft Exchange Server
Microsoft 365
Microsoft Power Platform
Microsoft Project
Microsoft SQL Server
Microsoft System Center
Microsoft Windows Server
PowerPivot
PowerShell
VBA
Visual C++/CLI
Visual Studio
Web API

Security
.NET Web App Security
C and C++ Secure Coding
C# & Web App Security
Linux Security Admin
Python Security
Secure Coding for Web Dev
Spring Security

SharePoint
Power Automate & Flow
SharePoint Administrator
SharePoint Developer
SharePoint End User
SharePoint Online
SharePoint Site Owner

SQL Server
Azure SQL Data Warehouse
Business Intelligence
Performance Tuning
SQL Server Administration
SQL Server Development
SSAS, SSIS, SSRS
Transact-SQL

Teleconferencing Tools
Adobe Connect
GoToMeeting
Microsoft Teams
WebEx
Zoom

Web/Application Server
Apache httpd
Apache Tomcat
IIS
JBoss
Nginx
Oracle WebLogic

Other
C++
Go Programming
Project Management
Regular Expressions
Ruby on Rails
Rust
Salesforce
XML

Visit ***www.accelebrate.com/newsletter*** *to sign up and receive our* **newsletters** *with information about* **new courses**, **free webinars**, **tutorials**, *and* **blog articles**.

Call us for a training quote! 877 849 1850 (US/Canada) or +1 678 648 3133

Entity Framework Core

Agenda

- Introduction
- LINQ
- DbContext
- Entity Data Model
- Inheritance
- Querying Data
- Modifying Data
- EF Migrations (time permitting)
- Q/A and Code Review (after lunch on day 3)

Entity Framework Core

Introduction

- Evolution of the .NET Platform
- Evolution of Entity Framework
- .NET SDK and Runtimes
- Visual Studio and VS Code
- Components of an EF Query

2002 .NET 1.0, C# 1.0, Visual Studio .NET

2003 .NET 1.1, Visual Studio 2003

2004 Mono 1.0

2005 .NET 2.0, C# 2.0, Visual Studio 2005
Generics, Nullable Value Types

2006 .NET 3.0, Mono 1.2
WPF, WCF, WF

2007 .NET 3.5, C# 3.0, Visual Studio 2008
LINQ, Anonymous Types, Lambda Expressions, Extension Methods, Implicit Typing

2008 Entity Framework 1.0

2009 ASP.NET MVC 1.0

2010 .NET 4.0, C# 4.0, ASP.NET MVC 2, Visual Studio 2010
Named / Optional Arguments, Dynamic Binding

2012 .NET 4.5, C# 5.0, Mono 3.0, ASP.NET MVC 4, Visual Studio 2012
Asynchronous Members (async / await)

2013 .NET 4.5.1, ASP.NET MVC 5, Visual Studio 2013
SignalR 1.0

2015 .NET 4.6, C# 6.0, Mono 4.0, Visual Studio 2015, Visual Studio Code 1.0
Expression Bodied Members, Null Propagator, String Interpolation

2016 Xamarin Acquisition, .NET Core 1.0, .NET Standard 1.0
Entity Framework Core 1.0

2017 .NET 4.7, .NET Core 2.0, C# 7.0, Visual Studio 2017
ASP.NET Razor Pages, Out Variables, Tuples, Ref Locals and Returns

2018 GitHub Acquisition, .NET Standard 2.0
Blazor Server

2019 .NET 4.8, .NET Core 3.0, C# 8.0, Visual Studio 2019
gRPC, Default Interface Methods, Using Declarations, Nullable Reference Types

2020 .NET 5, C# 9.0
Blazor WebAssembly, Records, Init Only Setters, Top-Level Statements

2021 .NET 6, C# 10.0, Visual Studio 2022
.NET MAUI

2008 Entity Framework 1.0 (.NET 3.5 SP1)

2010 Entity Framework 4.0 (.NET 4.0)

2011 .Entity Framework 4.1
Code First

2012 Entity Framework 4.3.1
Migrations

2012 Entity Framework 5.0 (.NET 4.5)
Many-to-Many Improvements, Split Queries, TPT Mapping

2013 Entity Framework 6.0
Open Source, Out-of-Band from .NET

2016 Entity Framework Core 1.0
Completely New Codebase, More Lightweight and Extensible

2021 Entity Framework Core 6.0

Introduction

Evolution of the .NET Platform

- .NET 1.0 had the potential to be cross-platform but was only officially supported on Windows
- Current version of this variant is 4.8 and now referred to as ".NET Framework"
- Will be supported for many years to come (end of support for .NET 3.5 SP1 is October 2028)

Introduction

Evolution of the .NET Platform

- In 2016, Microsoft introduced a new variant of .NET called .NET Core
- Many components were completely rewritten
- Fully supported on Windows, macOS, and Linux
- Included a subset of the functionality provided by .NET Framework
  - Focused on web-based workloads (web UIs and services)

Introduction

Evolution of the .NET Platform

- The version of .NET Core after 3.1 became the "main line" for .NET and was labeled .NET 5.0
- Supports development of Windows Forms and WPF applications that run on Windows
- The ASP.NET framework in .NET still includes the name "Core" to avoid confusion with previous versions of ASP.NET MVC

Introduction

Evolution of the .NET Platform

- The entire .NET platform is made available as open-source
- Community contributions are encouraged via pull requests
  - Thoroughly reviewed and tightly controlled by Microsoft

`github.com/dotnet`

Introduction

.NET SDKs and Runtimes

- .NET Runtime
  - Different version for each platform
  - Provides assembly loading, garbage collection, JIT compilation of IL code, and other runtime services
  - Includes the dotnet tool for launching applications
- ASP.NET Core Runtime
  - Includes additional packages for running ASP.NET Core applications
  - Reduces the number of packages that you need to deploy with your application

Introduction

.NET SDKs and Runtimes

- .NET SDK
  - Includes the .NET runtime for the platform
  - Additional command-line tools for compiling, testing, and publishing applications
  - Contains everything needed to develop .NET applications (with the help of a text editor)

Introduction

.NET SDKs and Runtimes

- Each version of .NET has a lifecycle status
  - Current – Includes the latest features and bug fixes but will only be supported for a short time after the next release
  - LTS (Long-Term Support) – Has an extended support period
  - Preview – Not supported for production use
  - Out of support – No longer supported

`dotnet.microsoft.com/download`

Introduction

Visual Studio and Visual Studio Code

- Visual Studio is available for Windows and macOS
  - Full-featured IDE
- Visual Studio Code is available for Windows, macOS, and Linux
  - Includes IntelliSense and debugging features
  - Thousands of extensions are available for additional functionality

`visualstudio.microsoft.com`

Introduction

Visual Studio and Visual Studio Code

- JetBrains also offers an IDE for .NET development called Rider
- Available for Windows, macOS, and Linux
- Includes advanced capabilities in the areas of refactoring, unit testing, and low-level debugging

`www.jetbrains.com/rider`

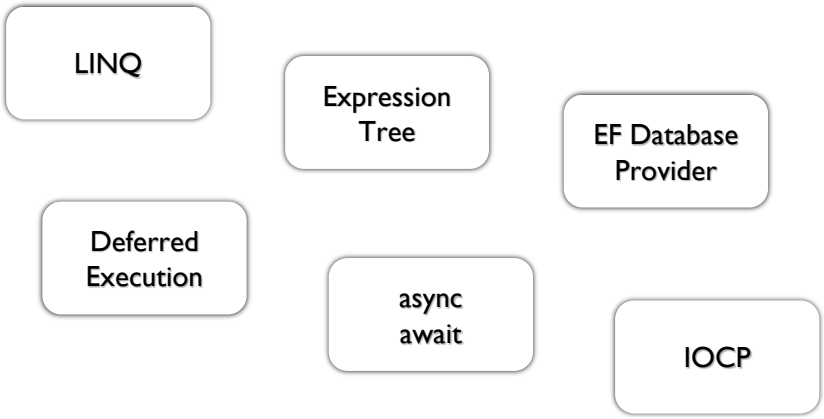
Introduction

Components of an EF Query

- Working with database data with Entity Framework requires several separate .NET technologies to work together
- Gaining a deeper understanding of these individual technologies can be helpful
  - Identifying bottlenecks
  - Optimizing performance
  - Improved maintainability
  - Efficient troubleshooting

Introduction

Components of an EF Query



Entity Framework Core

LINQ

- Introduction
- Method Syntax vs. Query Syntax
- Query Compilation
- Query Execution
- Deferred Execution
- LINQ to Objects
- LINQ Query Execution Pattern
- Expression Trees

LINQ

Introduction

- Language-Integrated Query (LINQ) integrates query capabilities directly into the C# language
- Avoids using different query languages for each type of data source (SQL databases, XML, web services, etc.)
  - Seen by C# as opaque strings without type checking or IntelliSense support
- Implemented as a collection of extension methods
  - Many basic methods defined on IEnumerable
  - Functionality for remote data sources defined by IQueryable
  - Other methods added by provider-specific types like DbSet

LINQ

Method Syntax vs. Query Syntax

- Writing a LINQ query by invoking the extensions methods directly is referred to as method syntax or a method-based query

```
var evens = nums.Where(n => n % 2 == 0).OrderBy(n => n);
```

- Alternatively, a query can be defined using the LINQ declarative query syntax
  - Uses keywords that map to their associated methods
  - Translated by the compiler into method calls

```
var evens = from n in nums
            where (n % 2 == 0)
            orderby n
            select n;
```


LINQ

Method Syntax vs. Query Syntax

- There is no semantic or performance difference between the two different forms
  - Both compile to the same code that is used to create a query object
- Any query that can be expressed by using query syntax can also be expressed by using method syntax
- Some query operations, such as Count or Max, have no equivalent query expression clause and must therefore be expressed as a method call

LINQ

Query Compilation

- IEnumerable queries are compiled to delegates
- IQueryable queries are compiled to expression trees

LINQ

Query Execution

- Query objects that return a single value are executed immediately
- Query objects that return a collection do not store the resulting collection, but the steps to produce the results when needed
  - Query is not executed until you iterate over the query variable, for example, in a foreach statement
  - This is referred to as deferred execution

LINQ

Deferred Execution

- Deferred execution allows for better efficiency and query composition
- Care must be used to identify the correct place to perform error handling and performance benchmarking

LINQ

LINQ to Objects

- LINQ to Objects refers to the use of LINQ queries with an IEnumerable collection directly, without the use of an intermediate LINQ provider or API such as Entity Framework
- Delegates are used for execution that is very similar to how you would do it previously with simple iteration
- Can be more concise and readable
- Provides advanced capabilities with a minimum of application code (e.g., AsParallel)

LINQ

LINQ Query Execution Pattern

- Entity Framework's LINQ APIs accept Expression Trees as the arguments for the LINQ Query Expression Pattern
- Enables Entity Framework to translate the query you wrote in C# into SQL that executes in the database engine

LINQ

Expression Trees

- An Expression Tree is a data structure that defines code
- Based on the same structures that a compiler uses to analyze code and generate the compiled output
- An existing Expression Tree can easily be modified or extended prior to execution
- A provider like Entity Framework can perform a wide variety of optimizations to an Expression Tree before execution
  - Elimination of unnecessary operations
  - Resequencing of operations
  - Splitting of the query into multiple individual queries

Entity Framework Core

DbContext

- Introduction
- Database Schema
- Initialization
- DbContext Pooling

DbContext

Introduction

- DbContext represents a combination of the Unit Of Work and Repository patterns
- Can be used to query a database and group together changes that will be written back to the database as a unit
- Typically used with a derived type that contains a collection of DbSet<T> properties for the root entities of the model
- Entity Framework does not query the database to determine the structure of the model
  - Based on a combination of default conventions and information provided to EF by you

DbContext

Database Schema

- Entity Framework does not query the database to determine the database structure
- EF maintains an in-memory representation of the database structure
  - Referred to as the Entity Data Model
  - Constructed using a set of conventions
  - Additional configuration can be provided to supplement and/or override what is discovered by convention

DbContext

Initialization

- All DbSet properties are initialized when the instance of the derived class is created
- Can be suppressed via the SuppressDbSetInitialization attribute
- If nullable reference types are enabled, DbSet properties can be initialized using the Set( ) method of DbContext or set to null with the null-forgiving operator

```
public DbSet<Customer> Customers => Set<Customer>();
public DbSet<Order> Orders => Set<Order>();
```

```
public DbSet<Customer> Customers => null!;
public DbSet<Order> Orders => null!;
```

DbContext

Initialization

- A DbContext is designed to be used for a single unit-of-work
- The lifetime of a DbContext instance is usually very short
- DbContext is not thread-safe
  - A single DbContext instance should not be used by multiple threads at the same time

DbContext

Initialization

- DbContext will invoke the OnConfiguring method during initialization
- Can be overridden for custom initialization
  - Connection string, model mappings, etc.

DbContext

DbContext Pooling

- Creating and disposing of a DbContext does not involve a database operation
  - Should not have a noticeable impact on performance
- Some internal services and objects are created during initialization and that can have an impact in high-performance scenarios
- EF Core can pool instances to reduce initialization overhead
  - Underlying database connection pooling is independent of DbContext pooling

DbContext

DbContext Pooling

- AddDbContextPool service registration method can be used to enable DbContext pooling

```
builder.Services.AddDbContextPool<MyContext>(options => ...);
```

- OnConfiguring will only be invoked once
  - Should not contain any information that might change during the lifetime of the application

Entity Framework Core

Entity Data Model

- Introduction
- Model Builder API
- Model Types
- Tables, Views, and Functions
- Entity Properties
- Nullability
- Keys
- Shadow Properties
- Relationships
- Concurrency

Entity Data Model

Introduction

- EF Core uses a set of conventions to build a model based on the shape of the entity classes
- Attributes in System.ComponentModel.DataAnnotations.Schema can be applied to entity types to define mappings
- Use of the ModelBuilder API (Fluent API) in OnModelCreating is the most powerful method of configuration
  - Has the highest precedence and will override conventions and data annotations

```
modelBuilder.Entity<Customer>()
            .Property(c => c.LastName)
            .HasColumnName("last_name")
            .HasColumnType("varchar(200)")
            .IsRequired();
```

Entity Data Model

Model Builder API

- Instead of placing all configuration in OnModelCreating, the configuration for a specific entity can be separated into a class that implements IEntityTypeConfiguration<TEntity>

```
public class CustomerEntityConfig : IEntityTypeConfiguration<Customer> {
  public void Configure(EntityTypeBuilder<Customer> builder) {
    builder.Property(c => c.LastName).HasColumnName("last_name");
  }
}
```

- Invoke the separate Configure method from OnModelCreating

```
new CustomerEntityConfig().Configure(modelBuilder.Entity<Custoemr>());
```


Entity Data Model

Model Types

- By default, EF will add types to the EDM that are...
  - Included as DbSet properties
  - Specified during the execution of OnModelCreating
  - Any types found by recursively exploring the navigation properties of other discovered entity types

Entity Data Model

Model Types

- Types can be explicitly excluded from the model if desired

```
[NotMapped]
public class BlogMetadata
{
  public DateTime LoadedFromDatabase { get; set; }
}
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Ignore<BlogMetadata>();
}
```


Entity Data Model

Tables, Views, and Functions

- By convention, each entity type will be set up to map to a database table with the same name as...
  - The name of the DbSet property that exposes the entity
  - If no DbSet property exists, the class name is used

```
public DbSet<Customer> Customers;
```

Entity Data Model

Tables, Views, and Functions

- Entity types can be mapped to database views using the Fluent API

```
modelBuilder.Entity<Blog>().ToView("blogsView");
```

- If mapped to both a view and a table, the view mapping will be used for queries and the table mapping will be used for updates
  - Supported in version 5.0 and later

Entity Data Model

Tables, Views, and Functions

- ToFunction( ) method can be used to map an entity to a table-valued function (TVF)

```
modelBuilder.Entity<BlogWithMultiplePosts>()
  .HasNoKey()
  .ToFunction("BlogsWithMultiplePosts");
```

Entity Data Model

Entity Properties

- By convention, all public properties with a getter and a setter will be included in the model
- You can configure a property with an exact database-specific type

```
[Column(TypeName = "varchar(200)")]
public string LastName { get; set; }
```


Entity Data Model

Nullability

- A property is considered optional if it valid for it to contain null
- When mapped to a relational database...
  - Required property is created as a non-nullable column
  - Optional property is created as a nullable column

Entity Data Model

Keys

- A key serves as a unique identifier for an entity
- Most entities will have a single key that maps to the concept of a primary key in the database
  - By convention, a property named Id or <type name>Id will be configured as the primary key of an entity

Entity Data Model

Keys

- EF also supports...
  - Keyless entities
  - Entities with additional keys (alternate keys)
  - Composite keys

```
modelBuilder.Entity<Car>()
            .HasKey(c => new { c.State, c.LicensePlate });
```

Entity Data Model

Keys

- For non-composite numeric and GUID primary keys, EF Core will set up for value generation
  - For example, a numeric primary key in SQL Server is automatically set up to be an IDENTITY column
- Database provider may automatically set up value generation for some property types, but others may require you to manually set up how the value is generated
- Depending on the database provider being used, values may be generated client side by EF or in the database
- If generated by the database server, the client-side value will be updated during SaveChanges( )

Entity Data Model

Shadow Properties

- Shadow properties are properties that aren't defined in your entity class but are in the Entity Data Model
  - Maintained by the Change Tracker
- Most often used for foreign key properties
  - EF will introduce a shadow property when a relationship is discovered but no foreign key property is found

Entity Data Model

Relationships

- A relationship defines how two entities relate to each other
- In a relational database, this is represented by a foreign key constraint
- By default, a relationship will be created when there is a navigation property discovered on a type
  - A property is considered a navigation property if the type cannot be mapped as a scalar type by the current database provider

Entity Data Model

Relationships

- By convention, cascade delete will be set to Cascade for required relationships and ClientSetNull for optional relationships
- Many-to-many relationships require a collection navigation property on both sides
  - Join table is used in the database
  - EF creates an entity type to represent the join table known as the join entity type

Entity Data Model

Concurrency

- Database concurrency refers to situations in which multiple processes or users access or change the same data in a database at the same time
- Concurrency control refers to specific mechanisms used to ensure data consistency in presence of concurrent changes
- By default, EF Core does use a transaction when performing an update but implemets a Last in Wins approach

Entity Data Model

Concurrency

- EF Core provides a way to implement optimistic concurrency control by using concurrency tokens
- Use TimestampAttribute or IsRowVersion to identify the property to be used as a concurrency token
  - Some difference can exist between different database providers

Data Entity Model

Concurrency

- When an update or delete operation is performed, the value of the concurrency token is checked
  - Check performed as part of the WHERE clause
- If the value is different than the one originally read, a DbUpdateConcurrencyException is thrown

Entity Framework Core

Inheritance

- Introduction
- Table-Per-Hierarchy (TPH)
- Table-Per-Type (TPT)
- Table-Per-Concrete-Type (TPC)

Inheritance

Introduction

- EF Core can map a .NET type hierarchy to a database
- EF Core will not automatically scan for base or derived types
  - All individual types must be explicitly specified as part of the model

```
internal class MyContext : DbContext {
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<RssBlog> RssBlogs { get; set; }
}

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

public class RssBlog : Blog {
    public string RssUrl { get; set; }
}
```


Inheritance

Table-Per-Hierarchy (TPH)

- By default, EF Core maps inheritance using the table-per-hierarchy (TPH) pattern
- Single table is used to store the data for all types in the hierarchy
- A discriminator column is used to identify which type each row represents
  - Can be a shadow property or mapped to a regular .NET property in the entity

Inheritance

Table-Per-Hierarchy (TPH)

- When querying for a derived type, results will be automatically filtered via the discriminator column
- When querying for a base type, no filtering is performed

Inheritance

Table-Per-Type (TPT)

- In the TPT mapping pattern, all the types are mapped to individual tables
  - Enabled by explicitly specifying a table name for each entity
- Tables for derived types include a foreign key that joins the derived table with the base table
  - Will typically result in inferior performance when compared with TPH

Inheritance

Table-Per-Concrete-Type (TPC)

- TPC is supported by EF6 (.NET Framework)
- Not yet supported by EF Core

Entity Framework Core

Querying Data

- Introduction
- Client vs. Server Evaluation
- Change Tracking
- Loading Related Data
- Split Queries
- Pagination
- Raw SQL Queries

Querying Data

Introduction

- EF Core uses LINQ to query data from a database using a derived context and an entity data model
- EF Core passes a representation of the LINQ query to the database provider which translates it into a database-specific query language
- Queries are always executed against the database even if the entities in the result already exist in the context

Querying Data

Client vs. Server Evaluation

- EF Core will attempt to perform as much work as possible on the database side (sever evaluation)
  - Helps to avoid bringing unnecessary data to the client
- Some query operations cannot be translated into a task the database can perform but could be done via client evaluation
  - EF Core 3.0 and earlier would do this automatically whenever possible
  - Can potentially have a very significant performance impact

Querying Data

Client vs. Server Evaluation

- In newer versions of EF Core, client-evaluation will still be used in the top-level projection
  - Example – C# method call in the Select( )
- All other uses of client evaluation will throw a runtime exception before being performed
  - Example – C# method call in the Where( )
- Client evaluation can be performed explicitly by using a method like ToList( ) before performing the operation

Querying Data

Change Tracking

- If an entity is tracked, any changes to the entity will be persisted to the database during SaveChanges( )
- Tracking is enabled for all queries by default
- Keyless entities are never tracked

Querying Data

Change Tracking

- Tracking can be disabled for an individual query or for an entire context
- Faster execution since there is no need to set up the change tracking information
- Should be used whenever retrieving entities that will not be updated via EF

```
var blogs = context.Blogs.AsNoTracking().ToList();
```

```
context.ChangeTracker.QueryTrackingBehavior =
                      QueryTrackingBehavior.NoTracking;

var blogs = context.Blogs.ToList();
```


Querying Data

Change Tracking

```
protected override void OnConfiguring(...)
{
  optionsBuilder
    .UseSqlServer(@"Server=...")
    .UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking);
}
```

Querying Data

Change Tracking

- Tracking queries use identity resolution
  - Will return the same entity instance from the change tracker if it is already being tracked
- No-tracking queries will return a new instance of the entity even when the same entity is contained in the result multiple times
- Starting with EF Core 5.0, identity resolution can be used independent of change tracking

```
var blogs = context.Blogs
            .AsNoTrackingWithIdentityResolution()
            .ToList();
```


Querying Data

Loading Related Data

- There are three common O/RM patterns used to load related entities
  - Eager loading – Related data is loaded from the database as part of the initial query
  - Explicit loading – Related data is explicitly loaded from the database at a later time
  - Lazy loading – Related data is transparently loaded from the database when a navigation property is accessed

Querying Data

Loading Related Data

- Eager loading can be performed via the Include and ThenInclude methods

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

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

```
var blogs = context.Blogs
              .Include(blog => blog.Posts)
              .ThenInclude(post => post.Author)
              .Include(blog => blog.Posts)
              .ThenInclude(post => post.Tags);
```


Querying Data

Split Queries

- It is recommended to not use Include( ) alone for a collection navigation property
  - Could result in poor performance from a "cartesian explosion" from executing the join query
- A split query should be used
  - Feature added in EF Core 5.0 and enhanced in EF Core 6.0

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

Querying Data

Split Queries

- A split query will use two separate SQL statements to avoid duplicate data in the result set
- EF Core will generate a warning when it detects a query that loads multiple collections but does not explicitly specify AsSingleQuery or AsSplitQuery

Querying Data

Pagination

- A common way to implement pagination is to use offset pagination via the LINQ Skip and Take functions

```
var position = 20;
var nextPage = context.Posts
                .OrderBy(b => b.PostId)
                .Skip(position)
                .Take(10);
```

- Can be inefficient and a different approach is recommended when possible

Querying Data

Pagination

- An alternative approach for pagination is keyset pagination
- Uses a WHERE clause to skip rows, instead of an offset
  - Client must remember the last entry fetched in the sorted sequence

```
var lastId = 55;
var nextPage = context.Posts
                .OrderBy(b => b.PostId)
                .Where(b => b.PostId > lastId)
                .Take(10);
```

- Does not provide for random access where a user can jump to a specific page

Querying Data

Pagination

- If performing keyset pagination, a second property may need to be used if duplicate rows might exist for the first property

```
var lastDate = new DateTime(2020, 1, 1);
var lastId = 55;
var nextPage = context.Posts
  .OrderBy(b => b.Date)
  .ThenBy(b => b.PostId)
  .Where(b => b.Date > lastDate ||
        (b.Date == lastDate && b.PostId > lastId))
  .Take(10);
```

Querying Data

Raw SQL Queries

- EF Core allows the use of raw SQL for returning regular entity types or keyless entity types
- Useful for queries that cannot be expressed with LINQ or that leverage database-provider specific functionality
- Also good for situations where inefficient SQL is generated by the database provider

```
var blogs = context.Blogs
  .FromSqlRaw("SELECT * FROM dbo.Blogs")
  .ToList();
```


Querying Data

Raw SQL Queries

- The SQL provided can be used to execute a stored procedure

```
var blogs = context.Blogs
  .FromSqlRaw("EXECUTE dbo.GetMostPopularBlogs")
  .ToList();
```

Querying Data

Raw SQL Queries

- FromSqlRaw provides several ways to pass input parameters

```
.FromSqlRaw("EXECUTE dbo.GetBlogsForUser {0}", user)
```

```
.FromSqlInterpolated($"EXECUTE dbo.GetBlogsForUser {user}")
```

```
.FromSqlRaw("EXECUTE dbo.GetBlogsForUser @user", user)
```

```
.FromSqlRaw("EXECUTE dbo.GetBlogsForUser @filterByUser=@user", user)
```

- All of these provide protection from SQL injection

Querying Data

Raw SQL Queries

- DO NOT pass a constructed string to FromRawSql

```
var sql = $"EXECUTE dbo.GetBlogsForUser {user}";
var blogs = context.Blogs.FromSqlRaw(sql);
```

```
.FromSqlRaw($"EXECUTE dbo.GetBlogsForUser {user}")
```

- This approach is vulnerable to SQL injection

Querying Data

Raw SQL Queries

- It is possible to compose on top of an initial raw SQL query

```
var blogs = context.Blogs
  .FromSqlInterpolated($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
  .Where(b => b.Rating > 3)
  .OrderByDescending(b => b.Rating);
```

- EF Core will use the raw SQL as a subquery
- SQL Server does not support composing over stored procedure calls

Querying Data

Raw SQL Queries

- There are some limitations when using raw SQL queries
  - SQL query must return data for all properties of the entity type
  - Column names in the result must match the column names the entity properties are mapped to
  - SQL query cannot contain related data (Include method can be used in combination with a raw SQL query)

Entity Framework Core

Modifying Data

- Introduction
- Adding Data
- Updating Data
- Deleting Data
- Transactions

Modifying Data

Introduction

- The ChangeTracker can be used to track and apply changes to the database
  - The database provider is responsible for translating the changes into database-specific operations
- Sometimes, a hybrid approach is appropriate
  - Non-tracking queries used for retrievals and updates applied outside of EF

Modifying Data

Adding Data

- Use the DbSet.Add method to add a new entity
- Entity will be inserted into the database when you call SaveChanges

```
var blog = new Blog { Url = "http://example.com" };
context.Blogs.Add(blog);
context.SaveChanges();
```

- Added to the ChangeTracker once saved

Modifying Data

Updating Data

- EF will automatically detect changes made to an existing entity that is tracked by the context

```
var blog = context.Blogs.First();
blog.Url = "http://example.com/blog";
context.SaveChanges();
```

Modifying Data

Deleting Data

- Use the DbSet.Remove method to delete instances of your entity classes
  - If it already exists in the database, it will be deleted during SaveChanges
  - If has been added but not yet saved, it will simply be removed from the context

```
var blog = context.Blogs.First();
context.Blogs.Remove(blog);
context.SaveChanges();
```


Modifying Data

Transactions

- By default, if the database provider supports transactions, all pending operations executed by SaveChanges will be applied within a single transaction
- You can use the DbContext.Database API to explicitly begin, commit, and rollback transactions
  - Database providers that do not support transactions may throw an exception or no-op

```
using var context = new BloggingContext();
using var transaction = context.Database.BeginTransaction();

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

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

  var blogs = context.Blogs
      .OrderBy(b => b.Url)
      .ToList();

  // Commit transaction if all commands succeed
  // Transaction will auto-rollback when disposed if either commands fails
  transaction.Commit();
}
catch (Exception) {
    // TODO: Handle failure
}
```


Modifying Data

Cross-Context Transactions

- It is possible to share a transaction across multiple context instances
- Context must share both a DbConnection and a DbTransaction
  - Pass the same DbConnection to the constructor of each context
- It is also possible to share a transaction between different data access technologies (e.g., EF and SqlClient)