Frequently asked interview questions on Entity Framework Core for a candidate with 8+ years of experience, along with comprehensive answers to help in preparation:

**### 1. \*\*What is Entity Framework Core and how does it differ from Entity Framework 6?\*\***

\*\*Answer:\*\*

Entity Framework Core (EF Core) is a lightweight, extensible, and cross-platform version of Entity Framework. Key differences include:

- \*\*Cross-Platform:\*\* EF Core supports Windows, Linux, and macOS, unlike EF 6 which is primarily Windows-based.

- \*\*Performance:\*\* EF Core has improved performance due to its lightweight architecture.

- \*\*LINQ Queries:\*\* EF Core provides better support for LINQ queries.

- \*\*Modeling:\*\* EF Core introduces new features like shadow properties, field mapping, and owned entity types.

- \*\*Migrations:\*\* EF Core has a more flexible migrations system, allowing for more granular control over database schema changes.

**### 2. \*\*What are some key features introduced in EF Core 5.0 and 6.0?\*\***

\*\*Answer:\*\*

\*\*EF Core 5.0:\*\*

- \*\*Table-per-type (TPT) inheritance:\*\* Support for TPT inheritance strategy.

- \*\*Filtered Include:\*\* Ability to filter related data using LINQ in Include statements.

- \*\*Split Queries:\*\* Ability to split queries to avoid the cartesian explosion problem.

- \*\*Many-to-Many Relationships:\*\* Simplified configuration for many-to-many relationships.

\*\*EF Core 6.0:\*\*

- \*\*Performance Improvements:\*\* Various enhancements for query performance.

- \*\*Temporal Tables:\*\* Support for SQL Server temporal tables.

- \*\*Compiled Models:\*\* Improved startup performance by using precompiled models.

- \*\*LINQ Enhancements:\*\* Additional LINQ operators and improvements.

**### 3. \*\*How would you configure a many-to-many relationship in EF Core?\*\***

\*\*Answer:\*\*

In EF Core, many-to-many relationships can be configured using either the fluent API or data annotations. For example, using the fluent API:

```csharp

public class Student

{

public int StudentId { get; set; }

public string Name { get; set; }

public ICollection<Course> Courses { get; set; }

}

public class Course

{

public int CourseId { get; set; }

public string Title { get; set; }

public ICollection<Student> Students { get; set; }

}

public class SchoolContext : DbContext

{

public DbSet<Student> Students { get; set; }

public DbSet<Course> Courses { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Student>()

.HasMany(s => s.Courses)

.WithMany(c => c.Students)

.UsingEntity<Dictionary<string, object>>(

"StudentCourse",

j => j.HasOne<Course>().WithMany().HasForeignKey("CourseId"),

j => j.HasOne<Student>().WithMany().HasForeignKey("StudentId")

);

}

}

```

**### 4. \*\*What is shadow property in EF Core and how is it useful?\*\***

\*\*Answer:\*\*

Shadow properties are properties that are not defined in your entity classes but are defined in the EF Core model. They are useful for storing data that should not be directly exposed in your entity classes.

Example of shadow property configuration:

```csharp

modelBuilder.Entity<Blog>()

.Property<DateTime>("LastUpdated");

```

You can access and modify shadow properties using the `ChangeTracker` API:

```csharp

var blog = context.Blogs.First();

var lastUpdated = context.Entry(blog).Property("LastUpdated").CurrentValue;

context.Entry(blog).Property("LastUpdated").CurrentValue = DateTime.Now;

```

**### 5. \*\*How does EF Core handle concurrency conflicts?\*\***

\*\*Answer:\*\*

EF Core handles concurrency conflicts using a technique called optimistic concurrency control. It involves checking whether the data has been modified since it was last read before applying updates. This is typically done using a concurrency token, often a timestamp or version number.

Example configuration with a concurrency token:

```csharp

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

[ConcurrencyCheck]

public int Stock { get; set; }

}

public override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted\_Connection=True;");

}

```

Handling concurrency exceptions:

```csharp

try

{

context.SaveChanges();

}

catch (DbUpdateConcurrencyException ex)

{

foreach (var entry in ex.Entries)

{

if (entry.Entity is Product)

{

var proposedValues = entry.CurrentValues;

var databaseValues = entry.GetDatabaseValues();

foreach (var property in proposedValues.Properties)

{

var proposedValue = proposedValues[property];

var databaseValue = databaseValues[property];

// Decide how to resolve conflict

proposedValues[property] = databaseValue;

}

entry.OriginalValues.SetValues(databaseValues);

}

}

context.SaveChanges();

}

```

**### 6. \*\*What is the purpose of `DbContext` in EF Core?\*\***

\*\*Answer:\*\*

The `DbContext` class serves as the primary class for interacting with the database using Entity Framework Core. It:

- Manages database connections.

- Provides methods for querying and saving data.

- Tracks changes to entities.

- Configures the model through the `OnModelCreating` method.

- Manages migrations and schema changes.

**### 7. \*\*How do you implement a custom convention in EF Core?\*\***

\*\*Answer:\*\*

Custom conventions in EF Core can be implemented by overriding the `OnModelCreating` method in your `DbContext`:

```csharp

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

foreach (var entity in modelBuilder.Model.GetEntityTypes())

{

// Apply a custom convention to all string properties

var stringProperties = entity.GetProperties()

.Where(p => p.ClrType == typeof(string));

foreach (var property in stringProperties)

{

property.SetMaxLength(200);

}

}

}

```

**### 8. \*\*How do you handle lazy loading in EF Core?\*\***

\*\*Answer:\*\*

Lazy loading in EF Core can be enabled by installing the `Microsoft.EntityFrameworkCore.Proxies` package and configuring your `DbContext` to use lazy loading proxies:

```csharp

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder

.UseLazyLoadingProxies()

.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted\_Connection=True;");

}

public class Blog

{

public int BlogId { get; set; }

public string Name { get; set; }

public virtual ICollection<Post> Posts { get; set; }

}

```

**### 9. \*\*Explain the difference between `AsNoTracking` and tracking queries in EF Core.\*\***

\*\*Answer:\*\*

`AsNoTracking` queries are read-only and do not track the returned entities in the `DbContext`'s change tracker. This improves performance for read-only scenarios where the entities do not need to be updated.

```csharp

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

```

Tracking queries, on the other hand, track the returned entities, so any changes made to them are detected and can be persisted to the database.

```csharp

var blogs = context.Blogs.ToList();

```

**### 10. \*\*How do you configure database seeding in EF Core?\*\***

\*\*Answer:\*\*

Database seeding in EF Core can be configured in the `OnModelCreating` method:

```csharp

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Product>().HasData(

new Product { ProductId = 1, Name = "Laptop", Price = 999.99m },

new Product { ProductId = 2, Name = "Smartphone", Price = 499.99m }

);

}

```

Ensure migrations are applied to seed the data:

```bash

dotnet ef migrations add SeedDatabase

dotnet ef database update

```

These questions and answers should provide a comprehensive review for a candidate preparing for an interview focused on Entity Framework Core.