Entity Framework Core - Practice Sheet

10 Q&A on migrations, relationships, loading, concurrency, and testing

Q1. What is the Code-First approach in Entity Framework Core?

A: You define C# entity classes and a DbContext with DbSet<T> properties. EF Core derives the database schema from the model. dotnet ef migrations add creates migration files, and dotnet ef database update applies them no hand-written SQL needed.

Q2. How do you configure DbContext for dependency injection in ASP.NET Core?

A: Call builder.Services.AddDbContext<AppDbContext>(opts => opts.UseSqlServer(connectionString)) in Program.cs. ASP.NET Core injects a Scoped DbContext per request. Avoid Singleton DbContext because it's not thread-safe.

Q3. How do you define a one-to-many relationship using Fluent API?

A: In OnModelCreating: modelBuilder.Entity<Order>().HasMany(o => o.Items).WithOne(i => i.Order).HasForeignKey(i => i.OrderId).OnDelete(DeleteBehavior.Cascade). Fluent API takes precedence over Data Annotations and keeps entity classes clean.

Q4. What is the difference between eager, lazy, and explicit loading?

A: Eager: Include(o => o.Items) adds a JOIN in the query. Lazy: accessing a navigation property on a proxy triggers an extra SQL query automatically (requires UseLazyLoadingProxies). Explicit: context.Entry(order).Collection(o => o.Items).Load() fires one SQL on

Q5. How do EF Core migrations work and how do you roll one back?

A: dotnet ef migrations add Name generates a C# migration class with Up() and Down() methods. dotnet ef database update applies pending migrations in order. To roll back: dotnet ef database update PreviousMigrationName, which runs Down() of the removed migration.

Q6. How do you execute raw SQL safely in EF Core?

A: context.Database.ExecuteSqlRaw('UPDATE Orders SET Status=0 WHERE Id={0}', id) uses parameterized binding. Use FromSqlRaw to return entities: context.Orders.FromSqlRaw('SELECT * FROM Orders WHERE Status={0}', status). Never concatenate untrusted strings.

Q7. What is optimistic concurrency and how do you implement it in EF Core?

A: Add a [ConcurrencyCheck] property or a [Timestamp] byte[] property to the entity. EF Core includes the original value in WHERE status=@original in UPDATE statements. If another transaction modified the row, the WHERE clause matches 0 rows and EF Core throws

Q8. What is the Repository pattern over EF Core and when is it worth it?

A: A Repository wraps DbContext with a domain-specific interface (IUserRepository). Benefits: testability via in-memory implementations and hiding EF details. Drawbacks: duplicates IQueryable composition that DbContext already provides. Use it for complex domain models; skip it

Q9. What are compiled queries in EF Core and when do they help?

A: EF Core translates LINQ to SQL on every query call, including query plan compilation. EF.CompileQuery((AppDb db, int id) => db.Users.Where(u => u.Id == id)) precompiles the plan. Useful in hot paths where the same parameterized query runs thousands of times per second.

Q10. How do you test EF Core with an in-memory provider vs SQLite in-memory?

A: UseInMemoryDatabase is not a relational DB it ignores SQL constraints and cascades. SQLite in-memory (UseSqlite('Data Source=:memory:')) is a real SQL engine supporting constraints, migrations, and relational behavior, making it more reliable for integration tests.