



Entity Framework Core

Version EF Core 9 · verified 2026-07-29

A practical guide to EF Core — DbContext, querying, tracking vs no-tracking, saving changes, and migrations.

Contents

Mental model	2
DbContext & Models	2
Querying	3
Tracking vs No-Tracking	4
Saving changes	5
Migrations	6
Further reading	6

Mental model

EF Core maps C# classes to database tables and translates LINQ queries into SQL. It uses a `DbContext` to manage a unit of work. When you read entities, EF tracks them in memory, meaning it observes changes you make to the C# objects. When you call `SaveChangesAsync()`, it generates the `INSERT`, `UPDATE`, or `DELETE` statements required to persist those changes back to the database.

DbContext & Models

API	Use
<code>DbContext</code>	coordinates EF functionality
<code>DbSet<T></code>	table in the database
<code>OnConfiguring</code>	set provider (no DI)
<code>OnModelCreating</code>	configure mapping explicitly
<code>[Key]</code>	annotation for primary key

The `DbContext` is registered in DI as scoped by default. It is not thread-safe: never run multiple EF operations in parallel on the same context instance.

```
public class AppDb : DbContext
{
    public AppDb(DbContextOptions<AppDb> o)
        : base(o) {}
}
```

```
public DbSet<User> Users { get; set; } = null!;  
}  
  
public class User  
{  
    public int Id { get; set; } // PK by convention  
    public string Name { get; set; } = "";  
}
```

Tip: properties named `Id` or `<EntityName>Id` are automatically configured as primary keys.

Querying

API	Use
<code>.Where(u => u.Id == 1)</code>	filters the query
<code>.Select(u => u.Name)</code>	limits columns returned
<code>.Include(u => u.Posts)</code>	eagerly load related data
<code>.FirstOrDefaultAsync()</code>	fetch one row, or null
<code>.ToListAsync()</code>	fetch all matching rows

Queries are written in LINQ and translated to SQL. Execution is deferred until a terminal operator like `ToListAsync()` or `FirstOrDefaultAsync()` is called.

```
var activeUsers = await db.Users
    .Where(u => u.IsActive)
    .OrderBy(u => u.Name)
    .ToListAsync();
```

Gotcha: if a .NET method cannot be translated to SQL, EF Core throws an exception at runtime. Always evaluate queries to SQL before calling arbitrary logic.

Tracking vs No-Tracking

API	Use
tracked query (default)	for entities you will update
.AsNoTracking()	read-only, saves memory
.AsNoTrackingWithIdentityResolution()	keep identity map

By default, EF tracks entities. This adds memory and CPU overhead. For queries where you only intend to read data (like returning a JSON response), use `AsNoTracking()` to bypass the change tracker and improve performance.

```
// Read-only query, no tracking overhead
var names = await db.Users
    .AsNoTracking()
    .Where(u => u.Age > 20)
```

```
.Select(u => u.Name)
.ToListAsync();
```

Saving changes

API	Use
<code>db.Add(e) / e.Add(e)</code>	track new entity
<code>db.Remove(e)</code>	mark entity for deletion
<code>db.Update(e)</code>	mark all properties modified
<code>SaveChangesAsync()</code>	persist to database

You do not need to call `Update()` if an entity was queried with tracking. Simply modify the property, and `SaveChangesAsync()` will detect the change and issue an `UPDATE` statement only for the modified columns.

```
var user = await db.Users.FindAsync(1);
if (user != null)
{
    user.Name = "New Name";
    // Update() is NOT needed here
    await db.SaveChangesAsync();
}
```

Gotcha: `Update()` marks *every* property as modified. Only use it for disconnected entities (e.g., coming from an API payload) that were not tracked by EF.

Migrations

Command	Action
dotnet ef migrations add N	create migration N
dotnet ef database update	apply to database
dotnet ef database update N	revert to migration N
dotnet ef migrations remove	delete last unapplied
dotnet ef migrations script	generate SQL script

Migrations evolve the database schema to keep it in sync with the C# model. Never modify a migration file that has already been applied to a production database.

```
# Add a new migration called "AddEmailColumn"
dotnet ef migrations add AddEmailColumn

# Apply pending migrations to the database
dotnet ef database update
```

Further reading

- [EF Core Documentation](#)
- [Querying Data](#)
- [Saving Data](#)
- [Tracking vs. No-Tracking Queries](#)