Entity Framework Core Cheat Sheet

DbContext · Migrations · LINQ · Relationships

SETUP & DBCONTEXT

```
dotnet add package
Microsoft.EntityFrameworkCore.SqlServer
dotnet add package
Microsoft.EntityFrameworkCore.Tools
public class AppDbContext : DbContext {
public DbSet<User> Users { get; set; }
public DbSet<Order> Orders { get; set; }
protected override void
OnConfiguring(DbContextOptionsBuilder o)
=> o.UseSqlServer(connectionString);
}
```

ENTITY CONFIG

```
// Data Annotations:
[Table("users")] [Key] [Required]
[MaxLength(100)] [Column("full_name")]
// Fluent API (in OnModelCreating):
modelBuilder.Entity<User>().HasKey(u =>
u.Id);
modelBuilder.Entity<User>().Property(u
=> u.Email)
.IsRequired().HasMaxLength(200).IsUnicode(false);
```

CRUD OPERATIONS

```
// Create:
db.Users.Add(new User { Name = "Alice"
});
await db.SaveChangesAsync();
// Read:
var user = await db.Users.FindAsync(id);
var users = await db.Users.Where(u =>
u.Active).ToListAsync();
// Update: load  modify  SaveChanges()
// Delete: Remove(entity)  SaveChanges()
```

MIGRATIONS

```
dotnet ef migrations add InitialCreate
dotnet ef database update
dotnet ef migrations remove  (last
uncommitted)
dotnet ef database update
20240101_AddEmail  (target)
dotnet ef database update 0  (rollback
all)
dotnet ef migrations script  (generate
SQL)
```

LINQ QUERIES

```
db.Users.Where(u => u.Role == "Admin")
.OrderBy(u => u.Name)
.Select(u => new { u.Id, u.Name })
.Skip(20).Take(10)
.ToListAsync()
db.Users.AnyAsync(u => u.Email == email)
db.Users.CountAsync(u => u.Active)
```

RELATIONSHIPS

```
// One-to-many:
public class User { public
ICollection<Order> Orders { get; set; }
}
public class Order { public int UserId {
get; set; }
public User User { get; set; } }
// Config:
modelBuilder.Entity<Order>().HasOne(o =>
o.User)
.WithMany(u => u.Orders).HasForeignKey(o
=> o.UserId);
```

EAGER/LAZY/EXPLICIT LOADING

```
// Eager:
db.Orders.Include(o => o.User).Include(o
=> o.Items).ToList()
// Lazy (requires proxies + virtual nav
props):
var user = await db.Users.FindAsync(id);
var orders = user.Orders; // lazy loaded
// Explicit:
await db.Entry(user).Collection(u =>
u.Orders).LoadAsync();
```

TRACKING vs NO-TRACKING

```
// Tracked (default)  change detection
runs on SaveChanges
var user = await db.Users.FindAsync(id);
user.Name = "New"; await
db.SaveChangesAsync(); // auto-UPDATE
// No-tracking (read-only, faster):
db.Users.AsNoTracking().Where(u =>
u.Active).ToListAsync()
```

Use AsNoTracking() for pure read operations

TRANSACTIONS

```
using var tx = await
db.Database.BeginTransactionAsync();
try {
await db.SaveChangesAsync();
await db.SaveChangesAsync(); // second
operation
await tx.CommitAsync();
} catch { await tx.RollbackAsync();
throw; }
// SaveChanges is a single implicit
transaction
```

RAW SQL

```
var users = db.Users.FromSqlRaw(
"SELECT * FROM users WHERE status =
{0}", status);
db.Database.ExecuteSqlRaw(
"UPDATE users SET active = 1 WHERE id =
{0}", id);
// Interpolated (parameterized
automatically):
db.Users.FromSqlInterpolated($"SELECT *
FROM users WHERE id={id}");
```

COMMON ANNOTATIONS

```
[Key]
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
[Required] [MaxLength(200)]
[Column("col_name")]
[Table("table_name")]
[Index(nameof(Email), IsUnique=true)]
[NotMapped]  // exclude property from DB
[Timestamp]  // optimistic concurrency
token
[ForeignKey(nameof(UserId))]
```

COMMON GOTCHAS

DbContext is not thread-safe one per request (Scoped)
N+1 queries always Include() navigation properties
Mixing tracked and untracked entities causes conflicts
Cascade delete enabled by default for required relationships
Complex queries: check generated SQL with .ToQueryString()