Platforms

.NET Framework (Console, Winform, WPF, ASP.NET)

.NET Core (Console, ASP.NET Core)

Mono & Xamarin (in-progress)

UWP (in-progress)

EF Core - Working with DbContext

Create and use DbContext object

```
using (var context = new SchoolContext())
{
    //work with context here
}
```

Create Operation

```
context.Add<Student>(newStudentObj);
context.SaveChanges()
//or
context.Students.Add(newStudentObj);
context.SaveChanges();
// or
context.Entry(newStudentObj).State = EntityState.Added;
context.SaveChanges();
```

Update Operation

```
context.Update<Student>(StudentObj);
context.SaveChanges();
//or
context.Entry(studentObj).State = EntityState.Modified;
context.SaveChanges();
```

Delete Operation

```
Context.Remove(studentObj);
context.SaveChanges();
//or
context.Entry(studentObj).State = EntityState.Deleted;
context.SaveChanges();
```

Get the current State of an entity

```
varstate = context.Entry<Student>(studentObj).State;
```

Execute raw SQL query for entity types

```
varsList = context.Students.FromSql($"Select * from
Student where StudentName='name'").ToList<Student>();
```

Execute raw SQL commands

```
int noOfRowsAffected =
context.Database.ExecuteSqlCommand("CUD command");
```

Explicitly load navigation /reference entity

```
context.Entry(student).Reference(s => s.Grade).Load();
```

Explicitly Load Collection

```
context.Entry(student).Collection(s=>s.Courses).Load();
```

Find by PrimaryKey

```
var student = context.Students.Find(1);
```

Disable automatic detect changes

```
context.ChangeTracker.AutoDetectChangesEnabled = false
```

Disable Query Tracking

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

Disable Automatic Detect Changes

```
context.ChangeTracker.AutoDetectChangesEnabled = false;
```

Eager Loading

```
context.Students.Where(s => s.FirstName == "Bill")
            .Include(s => s.Grade).FirstOrDefault();
```

Multi Level Eager Loading

```
context.Students.Where(s => s.FirstName == "Bill")
            .Include(s => s.Grade)
              .ThenInclude(g => g.Teachers);
```

EF Core Conventions

Schema

dbo

Table Name

Same as DbSet<TEntity> property name in the context class.

Primary key Name

1) Id
2) <Entity Class Name> Id (case insensitive)

e.g. Id or StudentId property becomes primary key of Student by default.

Foreign key property Name

EF Core API will create a foreign key column for each reference navigation property in an entity with one of the following naming patterns.

1. <Reference Navigation Property Name>Id
2. <Reference Navigation Property Name><Principal Primary Key Property Name>

Null column

All reference type properties and nullable primitive properties.

NotNull Column

PrimaryKey properties and non-nullable primitive properties (int, decimal, datetime, float etc.)

Index

Clustered index on PrimaryKey columns.
Non-clustered index on Foreignkey columns.

Properties mapping to DB

By default all properties will map to database.

Cascade delete

Enabled by default.

EF Core Fluent API - Entity Type Configurations

ToTable() - Maps an entity to database table

```
modelBuilder.Entity<Student>().ToTable("StudentInfo");

modelBuilder.Entity<Student>()
            .ToTable("StudentInfo", "dbo");
```

ToTable() – Maps two entities to one database table

```
modelBuilder.Entity<Student>().ToTable("Student");

modelBuilder.Entity<StudentDeail>()
.ToTable("Student");
```

HasKey() - Configures Primary Key(s)

```
modelBuilder.Entity<Student>().HasKey(s => s.StudId);
```

HasAlternateKey() - Configures an alternate key in the EF model

```
modelBuilder.Entity<Student>().HasAlternateKey(s=>s.Id)
```

HasIndex() - Configures an index on the specified properties

```
modelBuilder.Entity<Student>().HasIndex(s => s.Id)
```

HasOne() - Configures the One part of the relationship

```
modelBuilder.Entity<Student>().HasOne(s => s.Grade)
```

HasMany() - Configures the Many part of the relationship

```
modelBuilder.Entity<Student>().HasMany(s => s.Courses)
```

OwnsOne() - Configures a relationship where the target entity is owned by this entity

```
modelBuilder.Entity<Student>()
            .OwnsOne(p => p.HomeAddress)
            .OwnsOne(p => p.PermanentAddress);
```

EF Core Fluent API – Property Configuration

HasColumnType - Configures a column data type

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasColumnType("varchar");
```

HasColumnName - Configures a Column name

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasColumnName("Name");
```

HasDefaultValue - Configures a default value

```
modelBuilder.Entity<Student>()
            .Property(s => s.PendingFees)
            .HasDefaultValue(100);
```

HasComputedColumnSql - Configures a computed column

```
modelBuilder.Entity<Student>()
            .Property(s => s.PendingFees)
            .HasComputedColumnSql("Sql here");
```

IsRequired - Configures a Null column

```
modelBuilder.Entity<Student>().Property(s =>
s.DoB).IsRequired(false);
```

IsRequired - Configures a NotNull column

```
modelBuilder.Entity<Student>()
            .Property(s => s.DoB)
            .IsRequired();
```

HasMaxLength - Configures maximum Length for string column

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasMaxLength(50);
```

IsUnicode - Configures a Unicode string column

```
modelBuilder.Entity<Student>()
            .Property(s =>    s.StudentName)
            .IsUnicode();
//or
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .IsUnicode(false);
```

IsConcurrencyToken – Configures a concurrency property

```
modelBuilder.Entity<Student>()
            .Property(p => p.StudentName)
            .IsConcurrencyToken();
//or

modelBuilder.Entity<Student>()
            .Property(p => p.RowVersion)
            .IsRowVersion();
```

HasField - Configures a backing field

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasField("_StudentName");
```

EF Core Fluent API – Relationship Configurations

One-to-zero-or-one

```
modelBuilder.Entity<Student>()
            .HasOne<StudentAddress>(s =>
                                    s.Address)
            .WithOne(ad => ad.Student)
```

One-to-Many

```
modelBuilder.Entity<Student>()
    .HasOne<Grade>(s => s.Grade)
    .WithMany(g => g.Students)
```

Many-to-Many

```
modelBuilder.Entity<StudentCourse>()
            .HasKey(sc => new {
                        sc.StudentId,
                        sc.CourseId
              });

modelBuilder.Entity<StudentCourse>()
    .HasOne<Student>(sc => sc.Student)
    .WithMany(s => s.StudentCourses);

modelBuilder.Entity<StudentCourse>()
    .HasOne<Course>(sc => sc.Course)
    .WithMany(s => s.StudentCourses);
```

Entity Framework Core Cheat Sheet

Features	EF Core
DB-First - Command line	1.0
DB-First -VS Wizard	X
Model-First	X
Code-First	1.0
DbContext & DbSet	1.0
LINQ-to-Entities	1.0
ChangeTracker	1.0
Automated Migration	X
Code-based Migration	1.0
Graphical Visualization of EDM	X
EDM Wizard	X
Querying using EntitySQL	X
Table per hierarchy	1.0
Table per type	X
Table per concrete class	X
Many-to-Many without join entity	X
Entity Splitting	X
Spatial Data	X
Lazy loading	2.1
Stored procedure mapping with entity for CUD operation	X
Seed Data	2.1
Complex Type/Owned types	X
Table splitting	2.0
Field Mapping	1.1
Shadow properties	1.0
Alternate Keys	1.0
Identity Key Generation	1.0
Global query filter	2.0
DB Scalar function	2.0
Mixed client/server evaluation	1.0
Eager Loading	1.0
Proxy Entity	X
Interception	X
Simple Logging	X
GroupBy Transaction	2.1
Raw SQL Queries: Entity Types	1.0
Raw SQL Queries: non-entity types	2.1
DbContext Pooling	2.0
Data annotations	1.0
Fluent API	1.0
Model Format: EDMX (XML)	X