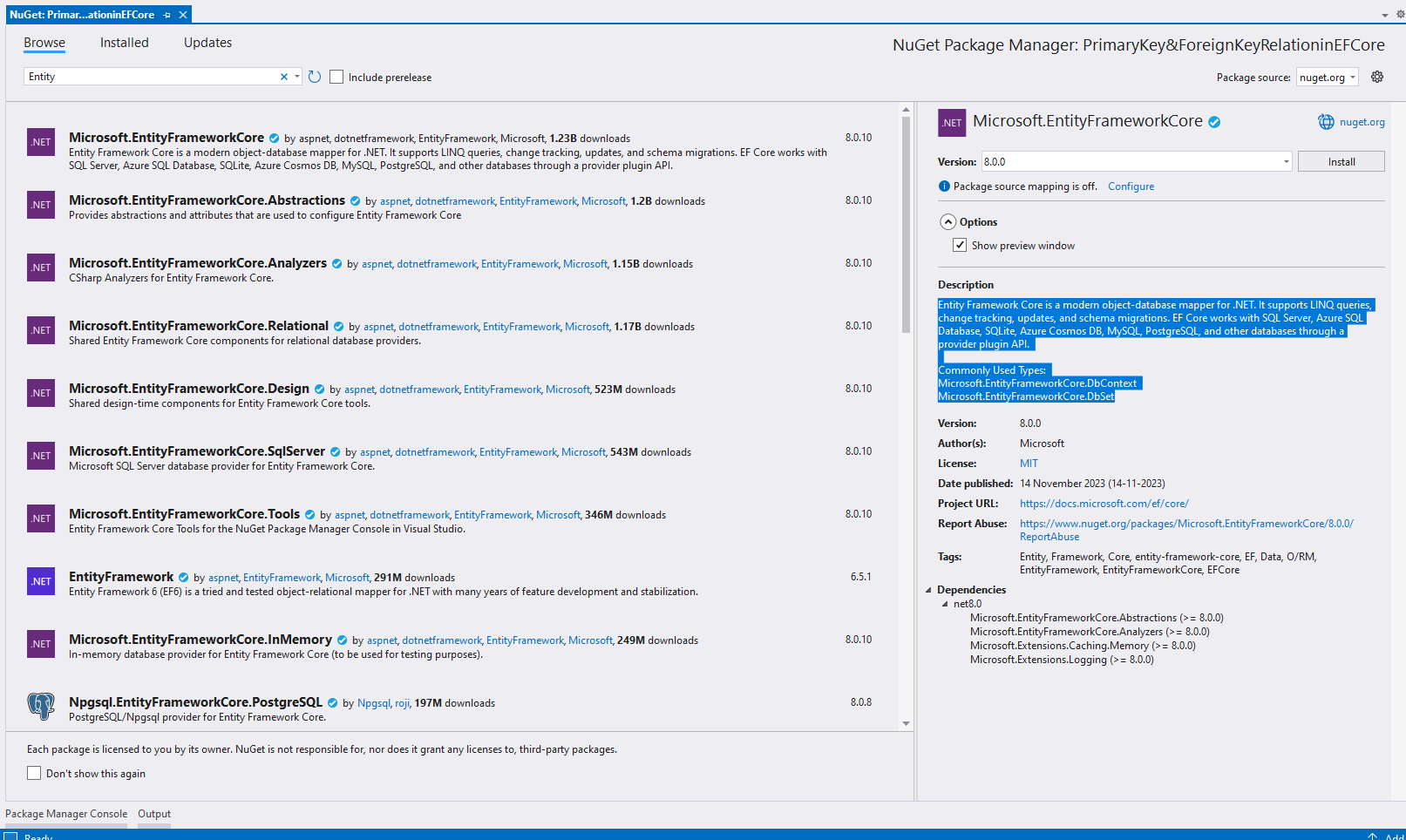
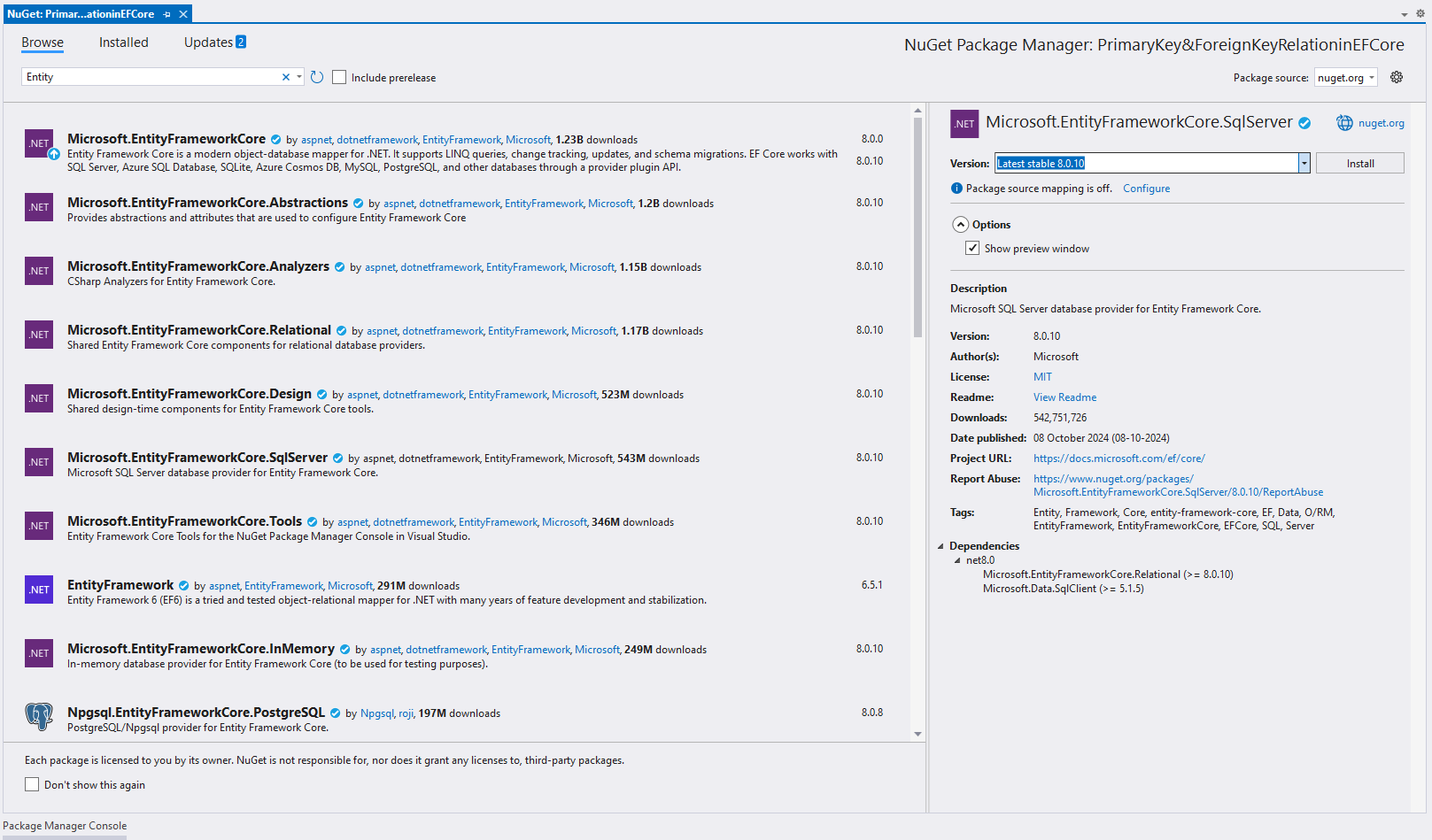
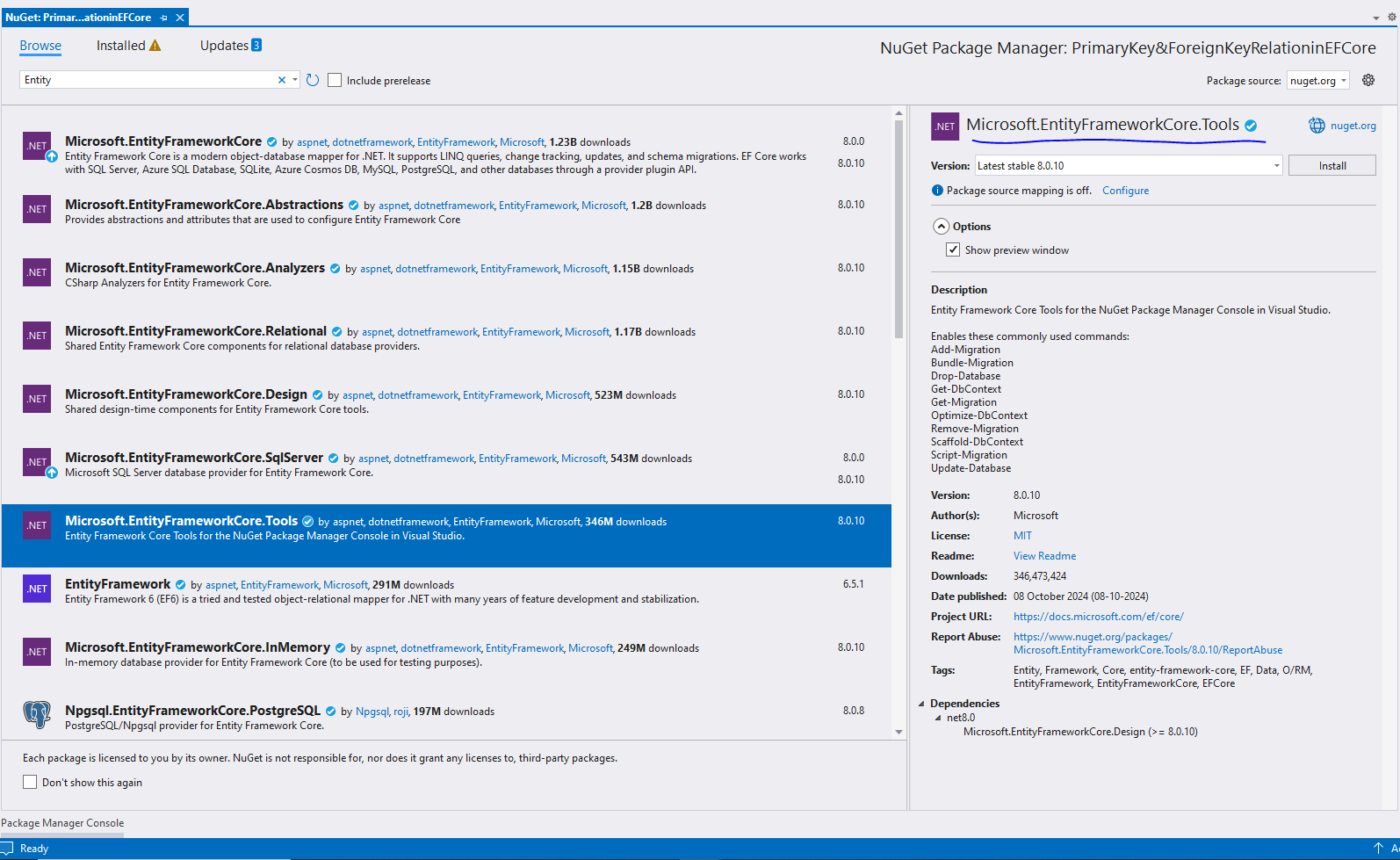
**PRIMARY KEY AND FOREIGN KEY RELATION ESTABLISH IN ENTITY FRAMEWORK CORE**

**\*\*\*\*\*\*\*\*\*You need Install below packages First in your project\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***







**============================================================================**

**Note: In** **Entity Framework Core (EF Core)**, you can define **primary keys** and **foreign keys** using

(1) **Data annotations.** **(Mainly focus this approach)**

(2) **Fluent API**. **(Just Check How it is working, Don’t Focus More)**

**Below are the examples for both approaches.**

1. **Using Data annotations(Mainly Focus This approach)**

In this approach .you can use attributes directly on the entity classes.

**Example:**

**using Microsoft.EntityFrameworkCore;**

**using System.ComponentModel.DataAnnotations;**

**using System.ComponentModel.DataAnnotations.Schema;**

**public class Author**

**{**

**[Key] // Primary Key**

**public int AuthorId { get; set; }**

**public string Name { get; set; }**

**// Navigation property**

**public List<Book> Books { get; set; } = new();**

**}**

**public class Book**

**{**

**[Key] // Primary Key**

**public int BookId { get; set; }**

**public string Title { get; set; }**

**// Foreign Key**

**public int AuthorId { get; set; }**

**// Navigation property**

**[ForeignKey("AuthorId")]**

**public Author Author { get; set; }**

**}**

**public class AppDbContext : DbContext**

**{**

public AppDbContext()

{

}

public AppDbContext(DbContextOptions<AppDbContext> options)

: base(options)

{

}

**public DbSet<Author> Authors { get; set; }**

**public DbSet<Book> Books { get; set; }**

**}**

**Goto appsettings.json (add below connection string)**

"ConnectionStrings": {

"CFAConnection": "Server=DESKTOP-P2V677R;Database=PrimaryKey&ForeignKeyRelationinEFCore;Trusted\_Connection=True;TrustServerCertificate=True;MultipleActiveResultSets=True;"

}

**Go to Program.cs(add below lines to inbuilt decency container).**

**builder.Services.AddDbContext<AppDbContext>(options =>**

**options.UseSqlServer(builder.Configuration.GetConnectionString("CFAConnection")));**

**Nex go tool=>nugget packagemager =>click on that =>you will navigate to nugget package manager window**

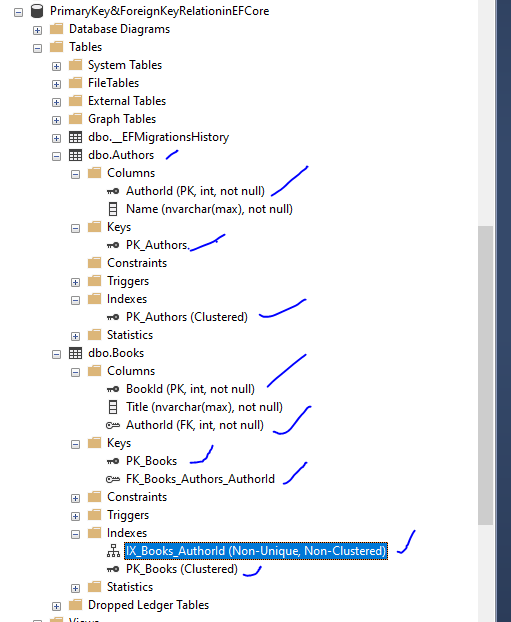
**PM>ADD-MIGRATION INITIAL**

**PM>UPDATE-DATABASE**

**\*\*\*\*AFTER SUCCESSFULLY EXECUTING THE ABOVE 2 COMMANDS IT WILL GENERATE THE DATABASE.**

**Explanation**

1. **[Key]**: Marks the primary key for the table.
2. **[ForeignKey]**: Specifies the foreign key relation with another entity (AuthorId in Book refers to the Author).



**2nd way Using Fluent API(just check.don’t focus more on this one)**

In Fluent API, relationships and keys are configured inside the OnModelCreating method in the DbContext.

**Example:**

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

public class Author

{

public int AuthorId { get; set; }

public string Name { get; set; }

// Navigation property

public List<Book> Books { get; set; } = new();

}

public class Book

{

public int BookId { get; set; }

public string Title { get; set; }

// Foreign Key

public int AuthorId { get; set; }

// Navigation property

public Author Author { get; set; }

}

public class AppDbContext : DbContext

{

public DbSet<Author> Authors { get; set; }

public DbSet<Book> Books { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("YourConnectionString");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Primary Key

modelBuilder.Entity<Author>()

.HasKey(a => a.AuthorId);

modelBuilder.Entity<Book>()

.HasKey(b => b.BookId);

// Foreign Key Relationship

modelBuilder.Entity<Book>()

.HasOne(b => b.Author) // Each Book has one Author

.WithMany(a => a.Books) // Each Author has many Books

.HasForeignKey(b => b.AuthorId); // Foreign key in Book entity

}

}

**Explanation**

1. **HasKey**: Defines the primary key.
2. **HasOne / WithMany**: Configures the one-to-many relationship between Book and Author.
3. **HasForeignKey**: Specifies the foreign key in the dependent entity (Book).

**Migration and Database Update**

After defining the entities, follow these steps:

1. Run the following commands in the **Package Manager Console**:

PM>Add-Migration InitialCreate

PM>Update-Database

1. This will generate a migration and update the database with the new schema.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Entity Framework core new Migrations\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

In **Entity Framework Core** using the **Code-First approach**, if you need to add a **new entity class** or make any schema changes after the initial migration and database creation, you can follow these steps. EF Core allows you to easily evolve the database schema with migrations.

## **Steps to Add a New Entity Class to Database**

1. **Create the New Entity Class**  
   Add a new entity class to your project that reflects the new requirements.
2. **Register the New Entity in DbContext**  
   Update the **DbContext** class to include the new entity.
3. **Generate a New Migration**  
   Use the EF Core CLI or Package Manager Console to create a new migration for the changes.
4. **Update the Database**  
   Apply the migration to update the database schema.

### ****Example Walkthrough****

#### 1. Add a New Entity Class

Suppose you want to add a Publisher entity related to the existing Book entity.

public class Publisher

{

public int PublisherId { get; set; } // Primary Key

public string Name { get; set; }

// Navigation Property

public List<Book> Books { get; set; } = new();

}

public class Book

{

public int BookId { get; set; }

public string Title { get; set; }

// Foreign Key to Publisher

public int PublisherId { get; set; }

public Publisher Publisher { get; set; } // Navigation Property

}

#### 2. Update the DbContext Class

Register the Publisher entity in your DbContext:

public class AppDbContext : DbContext

{

public DbSet<Book> Books { get; set; }

public DbSet<Publisher> Publishers { get; set; } // New DbSet for Publisher

}

#### 3. Generate a New Migration

Open **Package Manager Console** or **Terminal** and run the following command:

PM>Add-Migration AddPublisherEntity (Here you need to give new migration name).

This will create a new migration file in your Migrations folder, reflecting the changes (new Publisher table and foreign key in Book).

#### 4. Apply the Migration to the Database

Now, run the following command to update your database:

**PM>Update-Database**

This will apply the migration and create the new **Publisher** table with the relationship to the **Book** table.

**Verifying the Changes**

After running the migration, your database should have:

* A new Publishers table.
* A foreign key column (PublisherId) in the Books table.

### ****Optional: Rolling Back Migrations (if needed)****

If something goes wrong, you can **revert a migration** using:

**PM>Remove-Migration**

This will undo the last migration that was added but not yet applied to the database.

**Summary**

* Add the new entity class and relationships.
* Update the DbContext to include the new DbSet and configurations.
* Create a migration to reflect the changes.
* Apply the migration to update the database.