**ORM - Object Relational Mapping**

Applikasiyalarda olan datalar fiziki olaraq database’lərdə saxlanılır. Beləliklə yazılan app ilə database arasında daim bir bağlantı üzərindən əlaqə saxlanılır. Çünki app’in əldə etdiyi xarici datalar, database’də saxlanıla bilsin və ya istənilən zaman o datalar üzərində müəyyən əməliyyatlar aparıla bilsin. Bu səbəbdən app ilə database arasında dediyimiz bağlantı yaradılır və o bağlantı üzərindən data transferi həyata keçir. Yazılan kodlar, bu bağlantı üzərindən database’ə anlayacağı dildən sorğular göndərir və database də bu sorğulara əsasən müəyyən datalar üzərində əməliyyatlar həyata keçirir. Bu sorğular əsasən SQL dilində olur. ORM’ə olan ehtiyac elə burada yaranır.

Ado.Net vasitəsi ilə yazılmış app’lərin daxilində məlum olduğu kimi sql sorğuları yer almaqdadır. Kodların arasında yer alan bu Query’lər isə ümumi olaraq kodun çirklənməsinə səbəb olur. Bu şəkildə yazılan kodlar SQL’ə olan bağlılığı artırmaqda, idarə oluna bilməsini isə çətinləşdirməkdədir. Əlavə olaraq komandamıza gələcək olan yeni üzvümüz yazılan bu kodları başa düşməklə yanaşı həmçinin SQL dilini bilməli, onu anlamalı və tətbiq etməyi bacarmalıdır. Bu isə əlavə resursların israfına gətirib çıxarır.

Uzun sözün qısası, kod daxilində SQL kodları yazmaq və Database’dən qayıdan cavabı manual olaraq parse etmək böyük proyektlərdə böyük problemlərə səbəb ola bilər. Bu problemlər aşağıdaki şəkildədir.

\*Kodun çirklənməsinə səbəb olur.

\*Development və support resursu daha çox olacaq bir proyektə səbəb olacaq.

\*Database’ə bağlılıq yaradır.

\*Kompleks sorğuları manual olaraq yazmaq lazım olur.

\*Developer tərəfindən SQL məsuliyyəti də gözlənilir.

\*Database sorğuları nəticəsində gələn dataları manual olaraq parse edilir. Sorğu zamanı table və ya column kimi asılılıqlar olduğu kimi gələn dataların da eyni şəkildə asılılıqları olacaqdır.

\*Database daxilində baş vermiş dəyişikliklərə əsasən kod təkrardan review edilməlidir.

\*Kodun inkşafı zamanı bütün database əməliyyatları, yazılan proqramlaşdırma dilinin və OOP’in avantajını istifadə etmədən icra edilir.

ORM nədir?

Software və database arasındaki bağlantı üzərindən sorğular vasitəsi ilə data transferini OOP istifadə edərək təmin olunmasını və beləliklə kodun da, developer’in də SQL’ə asılılığı olmadan sürətli və asanlıqla əməliyyatlar həyata keçirə biləcəyi bir anlayışdır.

**Entity Framework**

Migration,MigrationBuilder,CustomSQL

Biz demək olar hər şeyi C# kodları vasitəsi ilə həyata keçirə bilirik. Lakin bir sıra məqamlarda, zamandan qazanmaq və gələcəyə yönümlü yazmaq üçün yenə də SQl query’lərdən istifadə edirik.

Bu əməliyyat Migration yaradan zaman icra edilir. Ilk öncə boş migration yaratmalı oluruq. Hər migration, Migration class’dan miras alır və müvafiq olaraq update-migration və remove-migration’a uyğun olaraq Up və Down method’ları generate olunur. Hər bir method MigrationBuilder type parameter qəbul edir. Hər iki method daxilində bu parametr’ə aid olan .Sql(string query) method tətbiq edilərək yazmış olduğumuz query icra edilir.

View in Entity Framework

Entity Framework vasitəsi ilə biz həmçinin View yaradıb, View’dan gələn dataları qəbul edə bilərik. Bunun üçün ilk öncə boş migration yaradılmalıdır. Daha sonra Up method daxilində icra etməli olduğumuz query’ni migrationBuilder.Sql(query); method’u vasitəsi ilə işə icra edirik. Down method daxilində isə drop əməliyyatı üçün lazım olan query yazılır və bu query də yuxarıda yazdığımız method ilə icra edilir.

Daha sonra yerinə yetirməli olduğumuz bir sıra əməliyyatlar da mövcuddur. Bildiyimiz kimi Entity Framework vasitəsi ilə hər hansısa SQL table yaratmaq üçün DbSet<TEntity> property’ə ehtiyac duyulur. View üçün də bu proses keçərlidir.

Əlavə olaraq FluentApi vasitəsi ilə yaranacaq olan SQL obyektinin table yox, view olduğunu göstərməliyik. Context class’ın OnModelCreating(ModelBuilder modelBuilder) method’u daxilində modelBuilder.Entity<ViewEntity>().HasNoKey().ToView(“viewName”) vasitəsi ilə

və ya

FluentApi’ın separate class halında yəni IEntityTypeConfiguration<TEntity> interface’dən miras alan class’ın Configure(EntityTypeBuilder<TEntity> builder) method’u daxilində builder.HasNoKey().ToView(“viewName”) method işə salınaraq hər iki yol ilə də View create ola bilər.

View’dan gələn datalar, Table’larda olduğu kimi context obyekti vasitəsi ilə Entity adıdan müraciət edilir.

Owned type

**Database First**

Database first anlayışı o deməkdir ki, bizim daha öncədən Databazamız dizayn edilib və table’lar artıq yaradılıb. Hazır olan table’ları biz EntityFramework vasitəsi ilə modelləşdirməli və entity’lərimizi əldə etməliyik. Bunun üçün bir sıra Package’lərə ehtiyac vardır.

**Packages For Database First**

Package Manager Console(PMC) istifadə ediriksə,

Microsoft.EntityFrameworkCore.Tools və Microsoft.EntityFrameworkCore.[Provider] package’ləri proyektə əlavə edilməlidir.

Dotnet CLI istifadə ediriksə,

Microsoft.EntityFrameworkCore.Design və Microsoft.EntityFrameworkCore.[Provider] package’ləri proyektə əlavə edilməlidir.

**Scaffold**

PMC vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

Scaffold-DbContext ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider]

Dotnet CLI vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

dotnet ef dbcontext scaffold ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider]

**Xüsusi Table’ların gətirilməsi**

Bu kodları run etdikdən sonra default olaraq database daxilində yerləşən bütün table’lar modelləşdirilir və Project’ə add olunur. Lakin biz sadəcə seçili table’ların modelləşdirilməsini və əlavə edilməsini istəyiriksə aşağıdaki ardıcıllığı izləməliyik:

PMC vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

Scaffold-DbContext ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] -Tables TableName1, …, TableNameN

Dotnet CLI vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

dotnet ef dbcontext scaffold ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] --table TableName1, …, --table TableNameN

**DbContext adının dəyişdirilməsi**

Scaffold vasitəsi ilə əldə etdiyimiz DbContext class’ı Database adı ilə və Context sonluğu ilə yaranır. Biz bu adı dəyişdirmək istəsək,

PMC vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

Scaffold-DbContext ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] -Context ContextName

Dotnet CLI vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

dotnet ef dbcontext scaffold ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] --context ContextName

**Path və Namespace**

Əldə olunan Entity’lər və DbContext class’ı, default olaraq proyektin özünə modellənir və namespace’ini istifadə edir. Əgər əldə olunan Entity’lərin və DbContext class’ın başqa bir path’də yer almağını istəsək,

**Path**

PMC vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

Scaffold-DbContext ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] -ContextDir FolderForContextClass --OutputDir FolderForEntities

Dotnet CLI vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

dotnet ef dbcontext scaffold ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] --context-dir FolderForContextClass --output-dir FolderForEntities

**Namespace**

PMC vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

Scaffold-DbContext ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] -Namespace NamespaceForEntities --ContextNamespace NamespaceForContext

Dotnet CLI vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

dotnet ef dbcontext scaffold ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] --namespace NamespaceForEntities --context-namespace NamespaceForContext

**Force**

Database’də baş verən dəyişiklikləri project’ə yansıtmaq üçün Scaffold prosesini təkrarlamaq lazımdır. Lakin bu zaman Entity’lərin zatən yarandığı ilə bağlı error alacayıq. Error almadan dəyişiklikləri əldə etmək üçün Scaffold prosesinin sonunda Force keyword’dən istifadə etmək kifayətdir.

PMC vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

Scaffold-DbContext ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] -Force

Dotnet CLI vasitəsi ilə bu proses aşağıdaki command vasitəsilə olur:

dotnet ef dbcontext scaffold ‘connectionString’ Microsoft.EntityFrameworkCore.[Provider] -- force

**Migration**

PCM vasitəsi ilə Migration əlavə olunma prosesi: add-migration “migrationName”

Dotnet CLI vasitəsi ilə Migration əlavə olunma prosesi: dotnet ef migrations add “migrationName”