Entity FrameWork

WhatIsEf ?

چهارچوبی برای دسترسی به داده است که به صورت lightweight و CrossPlatform بوده و برای کار با .net طراحی شده است و یک ORM را ارائه می دهد Object Relational Mapper و به توسعه دهندگان امکان دسترسی به داده را می دهد.

WhatIsOrm ?

اختصار Object Relational Mapper است که یک تکنیک برنامه نویسی که به برنامه نویس ها کمک می کند که Data را بین سیستم های incompatible تبدیل کنند (ناسازگار) به خصوص بین سیستم های oop مثل c# و relational database مثل sql server , my sql

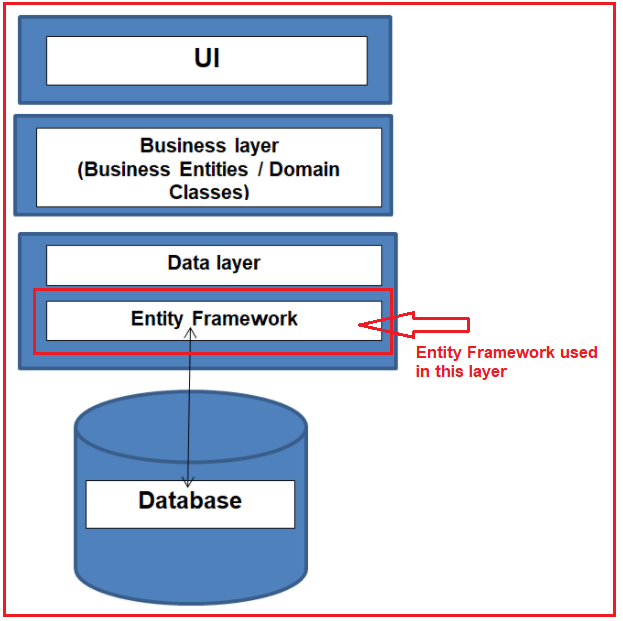
این امکان را به توسعه دهندگان می دهد که بر اساس قوانین object با Data کار کنند به جای table , columns به این معنی که orm به صورت خودکار کلاس ها را بر اساس database table می سازد همچنین برعکس این نیز صادق است یعنی می تواند sql لازم برای ایجاد DataBase بر اساس Class ها را نیز تولید کند.

از جمله کارکرد های آن به ساده سازی عملیات crud می توان اشاره کرد یا کار های مثل openingConnection یا executing command یا مدریت transaction ها و اطمینان از data integrity

نکته : به کلاس های در سطح اپ خود domain object یا business object می گویند.

نکته : از ef می توانیم در DataBase های non-relational نیز استفاده کرد.

نکته : قبل از نسخه ی .net 3.5 توسعه دهندگان از Ado.net استفاده می کردند

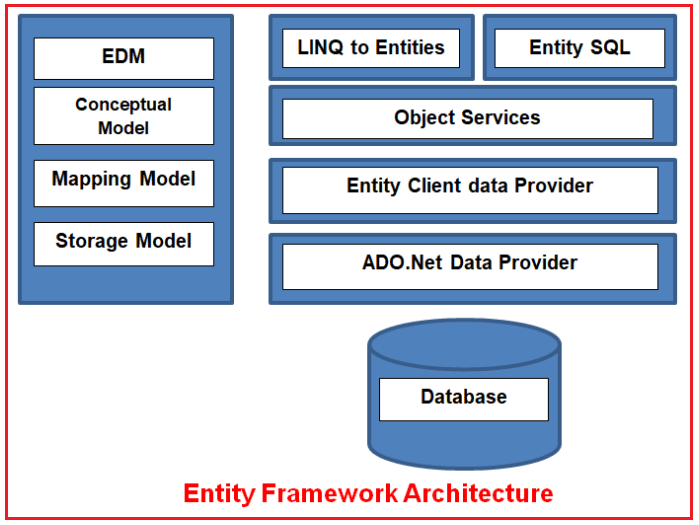


Ef داده های ذخیره شد در property های مربوط به کلاس ها را در DataBase ذخیره می کند

**Entity Framework Architecture**

**معماری entity framework به صورت :**

1. **The Entity Data Model**
2. **LINQ to Entities**
3. **Entity SQL**
4. **The Object Services Layer**
5. **Entity Client Data Provider**
6. **ADO.NET Data Provider**

****

**EDM (Entity Data Model):**

طرحواره منطقی یا رابطه ای انتزاعی است که و این مدل را از طریق یک رویکرد سه لایه نمایش می دهد.

1. The Conceptual Model (C- Space),
2. Mapping model (C-S Space)
3. Storage model (S – Space)

**Conceptual Model:**

شامل مدل های ما classes یا entity به همراه روابط بین آن ها است و بستگی به طراحی دیتا بیس ما دارد.

**Mapping Model**

این وظیفه را دارد که مشخص کند چگونه conceptual model به storage mode مپ شود. کار مپ کردن مدل business به همراه روابط بین آن ها که در لایه conceptual است به سطح table ها و روابط منطقی بین آن ها است.

**Storage Model / Logical Model**

مدل طراحی دیتابیس است که شامل table , views, keys , sp و ... روابط بین آن ها است که شامل 3 نوع فایل xml است C-Space, C-S Space, and the S-Space respectively

**.CSDL (Conceptual Schema Definition Language)**:

برای مپ کردنtype entity با استفاده از conceptual model است.

**.MSL (Mapping Schema Language)**

برای مپ کردن logical mode به conceptual model کاربرد دارد.

**.SSDL (Store Schema Definition Language)**

برای مپ کردن schema information of logical layer کاربرد دارد

**LINQ to Entities:**

برای کوئری نوشتن بر روی object model است و entity برمیگرداند که در conceptual model تعریف شده اند.

**Entity SQL:**

زبان کوئری نویسی دیگری است که به صورت داخلی به کوئری دیتابیس تبدیل می شود.

**Object Service**

در بسیاری از موارد باید با entity های که به صورت in-memroy هستند کار کنیم با استفاده از object service

**کارکرد ها :**

کاربرد های مثل انجام عملیات های مثل add , update , delete , و ... است.

لایه orm که نتیجه داده ها را به نمونه های object ما تبدیل می کند.

قابلیت استفاده از امکاناتی مثل primary key و changeTracking و ... را به توسعه دهندگان می دهد

**سرویس های ارائه شده توسط object service**

1. **Change tracking**
2. **Lazy loading**
3. **Inheritance**
4. **Optimistic concurrency**
5. **Merging data**
6. **Identity resolution**
7. **Support for querying data using Entity SQL and LINQ to Entities**

**Entity Client Data Provider:**

کار اصلی این لایه تبدیل linq به sql query است و با ado.net تعامل دارد که کار ارسال و بازیابی داده از database را بر عهده دارد.

**ADO.Net Data Provider:**

لایه ای است که برای تعامل با DataBase طراحی شده است.

**Entity Framework Context Class**

Ef به ما این امکان را می دهد که عملیات crud را بر روی داده ها با استفاده از CLR یا Common language runtime انجام بدهیم که به عنوان Entity یا .net class شناخته می شوند و ef کلاس ها را و روابط بین آن ها را به مدل Data Base مپ می کند.

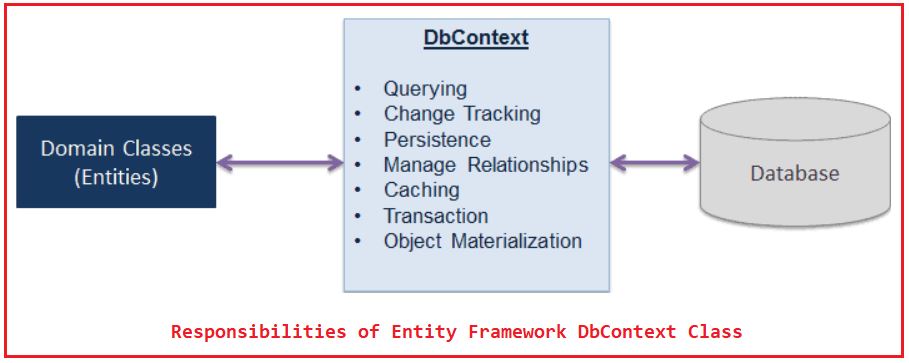
**Context**

کلاس اصلی جهت برقرارای ارتباط با database کلاس Context است. این یک کلاس معمولی بوده که از DbContext باید ارث بری کند. این کلاس یک session مرتبط با database را ارائه می دهد.

یک instance از dbContext می تواند ترکیبی از الگو های unitOfWork و Repository باشد و برای کوئری زدن به database کاربرد داشته باشد.

**What are the Responsibilities of DbContext in Entity Framework?**

در واقع یک پل میان domain class و database است



1. تبدیل کوئری linq به sql و ارسال آن ها به database
2. ثبت وقایع یا change tracking بر روی entity ها بعد از این که دیتا از دیتابیس fetch میشود
3. انجام عملیات های crud
4. ارائه fist-level-caching به صورت default
5. ارائه ویژگی transaction
6. مدیریت relation های میان entity ها در حالت code first و dbFirst
7. تبدیل داده های خام دریافت شده از database به entityObjects

**Entities in Entity Framework**

هر چیزی که در کلاس dbContext به صورت یک property و به شکل public DbSet<User> User {get;set;} تعریف شده باشد را می توانیم entity بنامیم.

Ef هر کدام از entity ها را به یک جدول در دیتابیس مپ می کند و هر کدام از property های مربوط به آن entity را به یک column در دیتابیس مپ می کند.

**انواع Property در یک entity کلاس**

**Scalar property**

در واقع property های است که نوع داده آن از نوع Primitive است مثل int string یا انواع nullable ها

**Navigation property**

این نوع property ها ارائه دهنده یک relationship میان entity ها هستند که می توانند از 2 نوع زیر باشند.

**Reference navigation property**

اگر یک entity دارای یک property از نوع یک entity دیگر باشد از این نوع است که بیان گر رابطه ی 1 به 1 در Ef است در کلاس Employee یک کلاس از نوع Department وجود دارد.

**public virtual Department Department { get; set; }**

**Collection navigation property**

اگر دارای یک Collection از نوع یک entity به عنوان property در یک entity دیگر باشیم که بیان گر رابطه ای one to many یا یک به چند است

**public virtual ICollection<Employee> Employees { get; set; }**

**Types of Entities in Entity Framework**

انواع entity در ef

**Poco entities**

مخفف plain old clr objects که می توان به عنوان یک domain object موجود در Data model باشد.

این نوع کلاس مپ می شوند به entity های تعریف شده در dataModel. این نوع entity یک کلاس است **که به هیچ یک از framework base class ها وابسته یا Depend نیست.**

این نوع تمام عملیات های insert update delete را پشتیبانی می کند درست همانند entity های generate شده توسط Data Model.

**Sample**

**public class Employee**

**{**

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

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

**public Nullable<int> Salary { get; set; }**

**public Nullable<int> DepartmentId { get; set; }**

**public virtual Department Department { get; set; }**

**}**

**Dynamic Proxy Entities**

زمانی که از یک موجودیت poco یک instance می سازیم به عنوان یک proxy عمل می کند برای آن entity. همچنین می توان گفت که در زمان runtime پروکسی مانند یک wrapper یا کلاس پوششی برای موجودیت poco خواهد بود

**Requirements of poco proxy entity**

کلاس باید public باشد.

نباید sealed باشد

نباید Abstract باشد

تمامی navigation property ها باید public و virtual باشند.

تمام کالکشن ها باید از نوع ICollection<> باشد.

در context مقدار proxyCreationEnabled باید مقدار false داشته باشد.

**public** **class** Employee

**{**

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

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

**public** Nullable**<int>** Salary **{** **get**; **set**; **}**

**public** Nullable**<int>** DepartmentId **{** **get**; **set**; **}**

**public** **virtual** Department Department **{** **get**; **set**; **}**

**}**

**نکته : فقط با یک تغییر virtual و حذف آن کلاس از حالت dynamic proxy به حالت poco تبدیل می شود**

**When to use Dynamic Proxy in Entity Framework?**

زمانی که به lazyLoad یا Change tracking نیاز داشته باشیم.

**Lazy**

به صورت پیش فرض در Ef فعال است به این صورت که related ها را تا زمانی که درخواست مستقیم ندهیم load نمی کند.

**changeTracking**

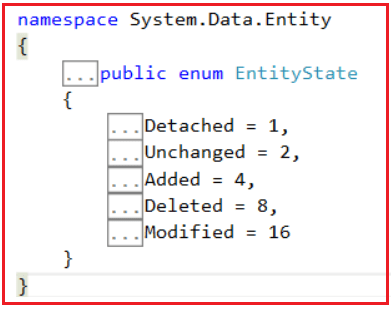
**به صورت خودکار ef از automic change tracking of loaded entites پشتیبانی می کند در طول زمان context life time و کلاس DbChangeTracker به صورت خودکار اطلاعات entity های track شده را می دهد**

**Entity States in Entity Framework**

چرخه حیاط یا life cycle در ef فرایندی را توصیف می کند که در آن یک entity ایجاد اضافه تغییر و حذف می شود.

یک entity در طول عمر خود دارای state های مختلفی است و ef کار نگه داری state هر یک از entity ها را در طور life time خود انجام می دهد.

هر entity دارای یک state بر مبنای عملی که روی ان انجام می شود از طریق کلاس context که خود دارای یک enum است.



نکته : یک entity همیشه در یکی از State ها قرار دارد.

**Added**: The entity is marked as added. The entity is being tracked by the context but does not yet exist in the database.

**Deleted**: The entity is marked as deleted. The entity is being tracked by the context and exists in the database, but has been marked for deletion from the database the next time SaveChanges is called.

**Modified**: The entity has been modified. The entity is being tracked by the context and exists in the database, and some or all of its property values have been modified.

**Unchanged**: The entity hasn’t been modified. The entity is being tracked by the context and exists in the database, and its property values have not changed from the values in the database.

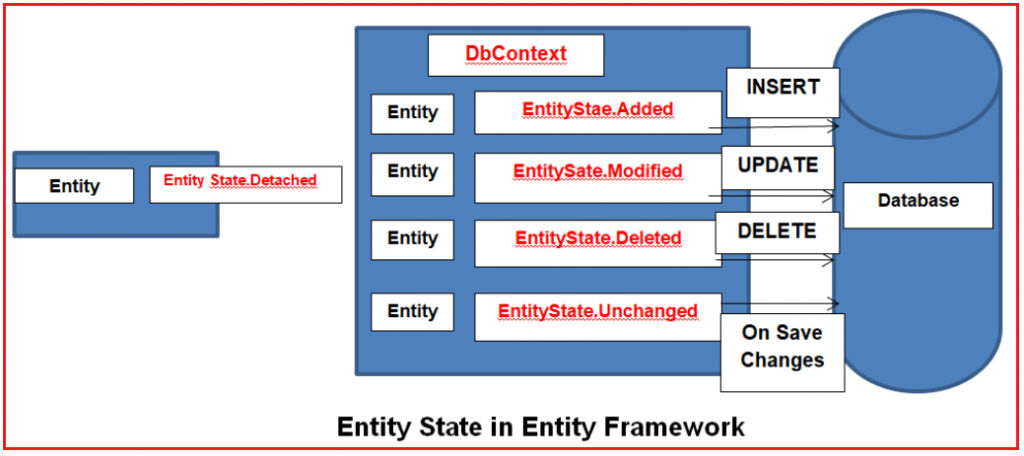
**Detached**: The entity is not being tracked by the context. An entity is in this state immediately after it has been created with the new operator or with one of the System.Data.Entity.DbSet Create methods

**changeTracking**

شی context نه تنها رفرنس تمامی entity ها را در خود نگه داری می کند بلافاصله بعد از این که از database بازخوانی می شوند بلکه کار ردیابی یا tracking و نگه داری state و تغییرات اعمال شده بر روی هر یک از property های entity را نیز انجام می دهد.

**Entity Lifecycle in Entity Framework**

تغییر حالت یک Entity از **حالت unchanged به حالت modified** تنها حالت تغییر وضعیت است که به صورت **automatic توسط context** کلاس مدیریت می شود.



**Unchanged State**

**نکته: property** مربوط به یک entity تغییر نکرده است از زمانی که از database داده retrieved شده است در این حالت زمانی که saveChange فراخوانی شود این entity **توسط context نادیده گرفته می شود** **یا ignore**

**وضعیت unchanged وضعیت پیش فرض در این حالت ها است. یعنی یک entity را دریافت کرده و کاری نکینم و save کنیم.**

**Attaching an Existing Entity to the context**

اگر یک entity داشته باشیم که اطمینان داریم در database ما موجود است اما توسط context رهگیری نمی شود track می توانیم به context بگوییم که entity را رهگیری کند با استفاده از Attach() بر روی همان dbSet در این حالت به صورت unchanged قرار می گیرد.

//Creating an Existing Department Entity

Department ExistingDepartment = new Department**()**

**{**

ID = 1,

Name = "IT",

Location = "BBSR"

**}**;

//Attaching the Entity to the Context

context.Departments.Attach**(**ExistingDepartment**)**;

//Or you can also attach an existing entity as follows

//context.Entry(ExistingDepartment).State = EntityState.Unchanged;

**Detached State of an Entity in Entity Framework**

می توانیم وضعیت یک entity را به حالت detached در بیاوریم در صورتی که این کار را انجام دهیم آن entity توسط chageTracker دنبال نمی شود

//Changing the Entity State to Detached

context.Entry**(**firstDepartment**)**.State = EntityState.Detached;

**Added State of an Entity in Entity Framework**

زمانی که با استفاده از متد Add مربوط به context یک entity جدید را add می کنیم در واقع state آن را به حالت Added تغییر داده ایم. Add به معنای این است که Entity در context وجود دارد اما در database وجود ندارد و زمانی که saveChange ما invoke شد یک کوئری insert ایجاد می شود و زمانی که عملیات با موفقیت انجام شود و داده insert شود state مربوط به entity به حالت unchanged تغییر پیدا میکند

**Modified State of an Entity in Entity Framework**

یک entity زمانی در حالت modified قرارمیگیرد که یکی از scalar property های آن تغییر پیدا کرده باشد و به این معنی است که entity تغییر کرده است اما نه در database و همچنین به این معنی است که entity در database موجود است و بعد saveChange کوئری update ایجاد می شود و state به حالت unchanged تغییر پیدا میکند.

نکته : دستور update فقط برای property های که تغییر داشته اند ایجاد می شود.

**Deleted State of an Entity in Entity Framework**

زمانی که بر روی یک entity متد remove را فراخونی می کنیم به حالت deleted تغییر State می دهد و از context حذف می شود و زمانی که saveChange فراخوانی می شود از database هم ردیف مربوطه حذف می شود این State به این معنی است که دیگر در context وجود ندارد و در Db وجود دارد البته حذف از context زمانی صورت می گیرد که حذف از db با موفقیت انجام شود و به حالت detached در می آید

**What SaveChanges Method Does in Entity Framework?**

اگر entity ها در حالت unchanged باشد هیچ کاری نمی کند.

اگر entity ها در حالت Added باشد کوئری insert برای هر entity می سازد

اگر entity ها در حالت modified باشند کوئری update برای هر کدام می سازد

اگر entity ها در حالت delete باشند از context به حالت detached می روند و بعد موفقیت کار در Db از context حذف شده و دیگر state آن نگه داری نمی شود

بعد از انجام عملیات db در مورد 1 تا 3 به حالت unchanged تغییر وضعیت می دهند.

**Development Approach with Entity Framework**

3 رویکرد متفاوت برای کار با DataBase که توسط ef تعریف می شوند به شرح زیر است.

Database first

Code first

Model first

**Database-First Approach**

این حالت زمانی که است که database ما وجود داشته باشد و ما کلاس context را بر اساس database موجود توسط edm wizard ایجاد می کنیم. مناسب زمانی که db از قبل وجود دارد.

**موارد کاربرد:**

یک database به ما ارث رسیده باشد

زمانی که توسعه دیتابیس توسط تیم دیگری انجام می شود

زمانی که اپلیکیشن ما از نوع داده محور است data centric

**Code-First Approach**

مناسب زمانی که database از قبل وجود ندارد در این حالت entity یا domain class ها را ایجاد می کنیم در context معرفی می کنیم و با دستورات migration دیتابیس را ایجاد می کنیم.

مناسب برای اپلیکیشن های domain centric و آن های که ابتدا domain class را تعریف می کنند. در حالت های که ddd داریم مناسب است.

**مناسب زمان های که:**

منطقی سمت دیتابیس وجود ندارد

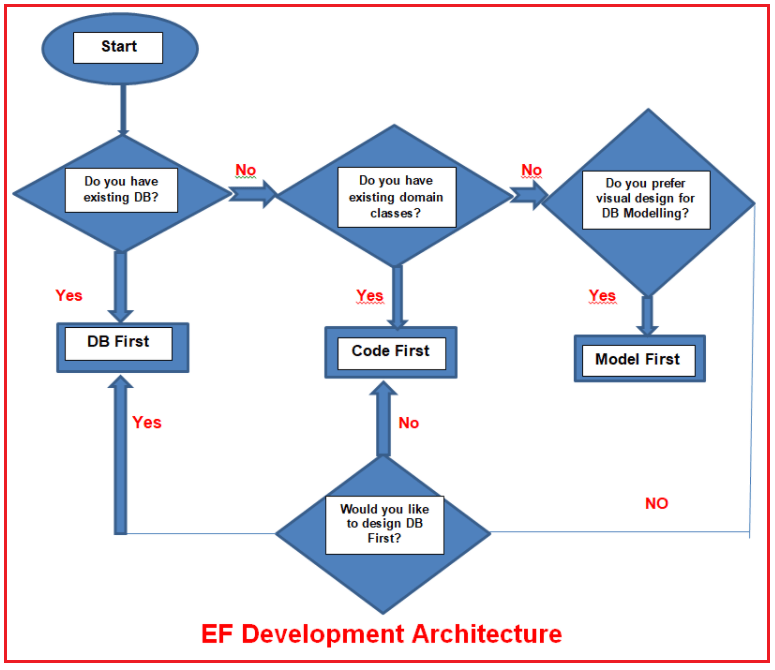
کنترل تمام در کد است و مدل auto generated در context نداریم

زمانی که db به صورت دستی تغییر نمی کند

**Model-First Approach**

بسیار شبیه به مدل code first است با این تفاوت که از visual edmx استفاده می شود و در یک visual designer طراحی انجام می شود و سپس entity ها ایجاد می شوند.

این حالت دستورات پایگاه داده را به ما می دهد.



**Introduction to Entity Framework Database First Approach**

در این روش ef به صورت اتوماتیک poco class ها یا در واقع classes , properties , dbContext و ... را به صورت خودکار از روی DataBase می سازد. در این روش باید اطلاعات لازم در رابطه با دیتابیس را به ef بدهیم و بر اساس table ,columns , relation کلاس های ما را می سازد.

**When to use**

زمانی که Db را از قبل داریم و از قبل طراحی شده است

زمانی که می خواهیم db به صورت manually تغییر کند و می خواهید model را بر اساس database تغییر دهید.

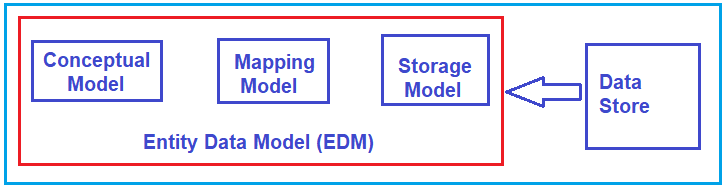
**نکته: در نسخه های قدیمی connection string در فایلی به نام app.config یا web.config قرار میگرفت**

**Entity Data Model (EDM) of Entity Framework Database First Approach**

با استفاده از Edm یا entity data model چهارچوب ef تمامی عملیات های مربوط به Data base را انجام می دهد. یک مدل است که توضیح می دهد entity ها و روابط بین آن ها

**3 بخش اصلی edm**

1. **The Storage Schema Definition Model (SSDL)**
2. **The Conceptual Schema Definition Model (CSDL)**
3. **The Mapping Model (Conceptual – Storage Mapping (MSL))**



**The Storage Schema Definition Model (SSDL)**

مدل طرحواره ذخیره سازی داده ها است. نمایش شماتیک ذخیره سازی داده های بکند است. به عبارت دیگر نمای xml پایگاه داده مربوطه را نشان می دهد.

به این معنی که جداول دیتابیس، ویو ها، استورپروسیجر ها، کلید اصلی، کلید خارجی و روابط بین جدول ها تمام این اطلاعات ارائه می شوند توسط storage schema definition layer یا ssdl و تماما در رابطه با backend data base است.

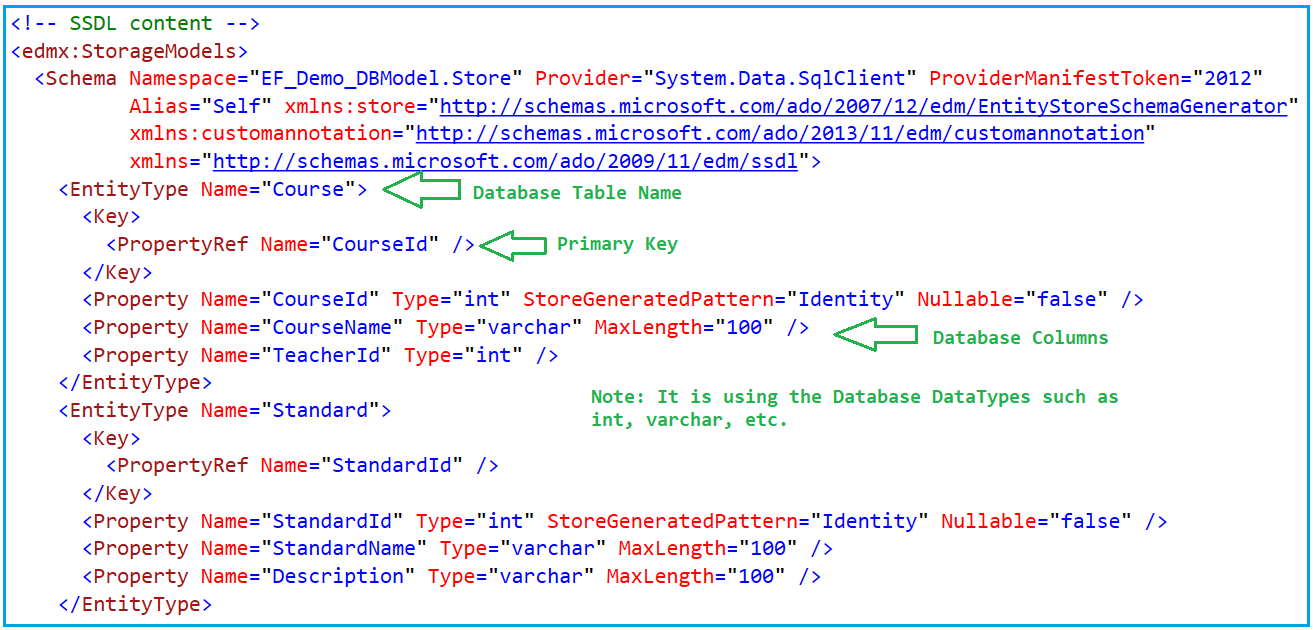
**The Conceptual Schema Definition Model (CSDL)**

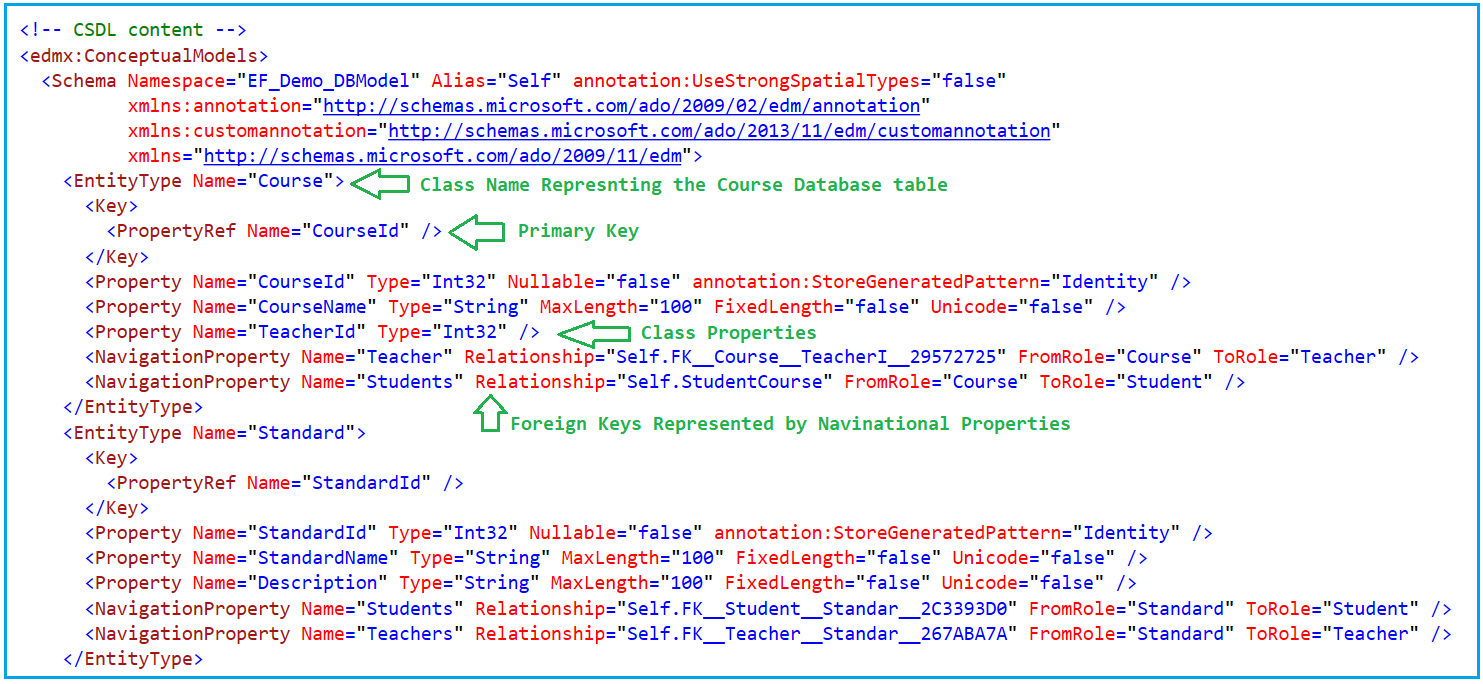
در واقع entity مدل ما است که کوئری های خود را بر روی آن می نویسیم و بر اساس table ها و sp و view و key ها و روابط بین table ها است این لایه وظیه ایجاد class ها و function ها را دارد یعنی تمامی sp ها و navigation prop ها و pk ها fk ها را می سازد و ما با آن ها در Ef کار می کنیم.

**Mapping Model (Conceptual – Storage Mapping (MSL))**

لایه ی است که وظیفه مپ کردن conceptual mode و storage model را دارد بنا بر این storage schema یا ssdl ارائه دهنده backend data base است و conceptual shcema definition layer یا csdl ارائه دهنده class ها و function است. اما کاری که این لایه می کند این است که مشخص می کند کدام class مربوط به کدام table و کدام prop مربوط به کدام table column است

نکته : می توان تمامی این لایه ها را در قالب xml مشاهده کرد.



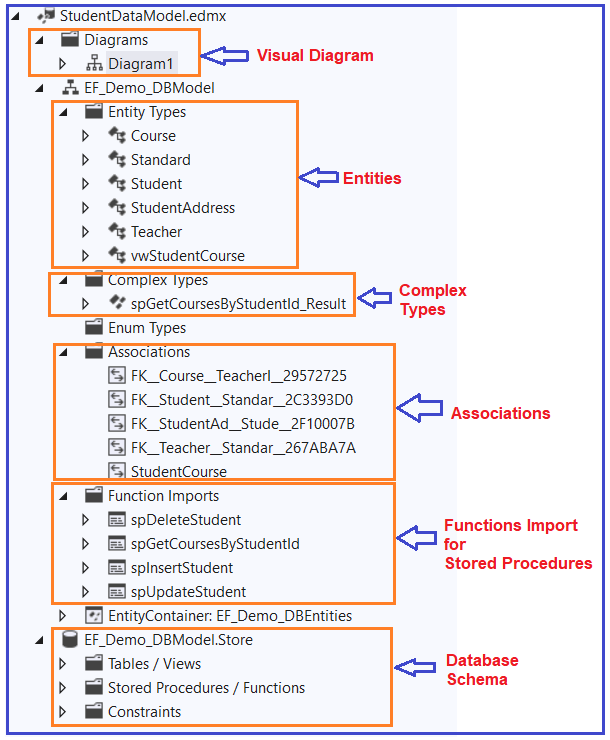




**Model Browser in Entity Framework**

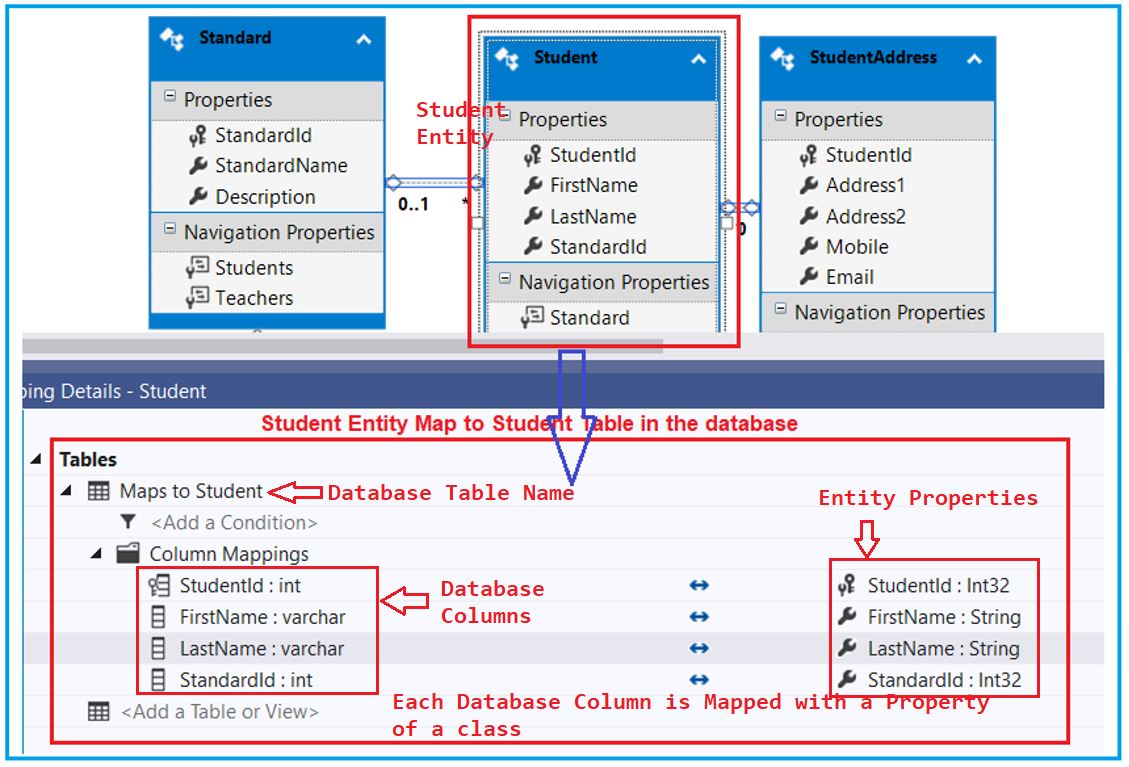
نکته : در حالت db first زمانی که edm می سازیم entity ها و view ها قابل مشاهده به صورت بصری هستند ولی sp خیر به دلیل این که به صورت function ذخیره می شوند.

با کلیک راست بر روی حالت بصری edm و انتخاب گزینه ی Model Browser می توان جزئیات بیشتری از مدل را مشاهده کرد شامل conceptual model , storage model , mapping info



**Entity-Table (C-S) Mapping in EDM Designer View**

برای مشاهده نحوه map شدن entity های موجود در edm به table های db می توان با کلیک راست برو روی هر یک از entity های موجود در حالت visual و انتخاب گزینه ی table-mapping نحوه map کردن را مشاهده کرد.

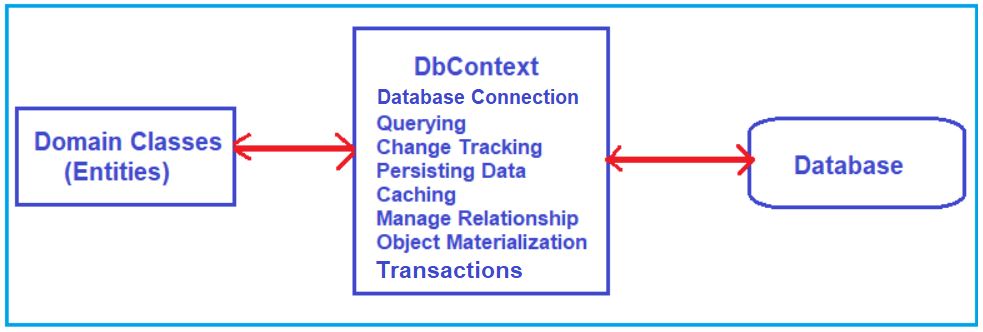


همانطورکه در تصویر مشاهده می کنید نحوه مپ شدن table موجود در بانک به موجودیت ef قابل مشاهده است و امکان تغییر دادن آن ها نیز فراهم است.

**DbContext Class in Entity Framework**

کلاس اصلی است که وظیفه تعامل با Db و انجام عملیات های crud را به عهده میگیرد. این کلاس ارائه دهنده ترکیبی از الگو های unit of work و repository است و به طور کلی می توان برای query نوشتن بر روی دیتابیس استفاده کرد.

همچنین برای اجرای view و sp نیز کاربرد دارد و در واقع یک پل میان domain و database است



نکته : ef core در واقع بر روی ado.net سوار است.

**وظایف DbContext**

Manage data base connection

**Querying**

ایجاد و تبدیل کوئری های linq ایجاد شده به زبان کوئری Db مقد

**chnageTracking**

وظیفه track کردن change های رخ داده بر روی entity ها بعد از این که از دیتابیس بازخوانی شدند را بر عهده دارد و بعد از این که متد SaveChanges() فراخوانی شده وظیفه دارد که دستورات لازم را برای ارسال به db اماده کند وارسال کند.

برای entity های unchaned and detached هیچ کوئری نمی سازد و بعد انجام عملیات track کردن آن entity را متوقف می کند.

**Persisting Data**

مسئول انجام عملیات های crud بر روی db است بر اساس entity state کوئری های لازم را می سازد مثلا اگر add state باشد کوئری insert می سازد یا اگر modified state باشد کوئری update می سازد.

**Caching**

یکی از ویژگی های مهم که بر روی عملکرد نرم افزار نیز تاثیر گذار است پیاده سازی first-level-cache توسط context است به این صورت که entity های خوانده شده از Db را در طور life time خود در حافظه نگه می دارد یعنی اگر کوئری مشابه مجددی باشد از cache میخواند نه دریافت داده از Db

**Manage Relationship**

با استفاده از csdl و ssdl در حالت db first و در حالت code first با استفاده از fluent api روابط بین entity ها را مدیریت می کند این که کدام entity با کدام table باید map شود و کدام prop باید با کدام column مپ شود و نوع رابطه one to one - one to many - many to many

**Object Materialization:**

تبدیل raw data دریافت شده از db به ef entity obejcts

نکته: هر موقع که یک class را در dbset<> مشاهده کردید به معنی یک entity بوده و می توانید با استفاده از context به آن دسترسی داشته باشید.

**نکته** : در کلاس context قابلیت مشاهده Sp ها به صورت function وجود دارد.

**نکته** : اگر متد Remove را بر روی یک entity که به صورت Added است اما هنوز در Db ذخیره نشده فراخوانی کنیم عملیات Add کردن آن entity متوقف می شود. اگر یک entity موجود در Db را remove کنیم در واقع state ان به حالت detached در می آید و دیگر توسط context ما Track نمی شود.

**DbSet in Entity Framework**

dbSet class در Ef به معنای یک entity است که می توان عملیات Crud را از طریق Db روی آن انجام داد. کلاس Context باید از کلاس پدر DbContext ارث برده باشد و entity ها را به عنوان prop معرفی کند. در واقع dbSet یک map به database table ها است.

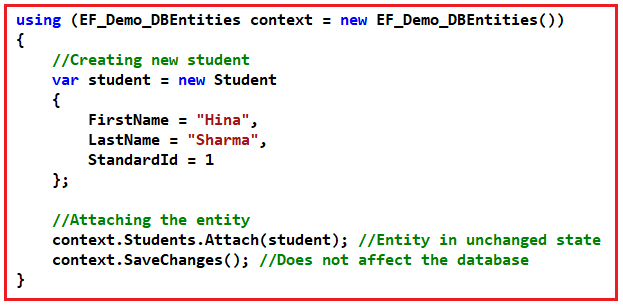
dbSet در واقع یک پیاده سازی از Repository pattern است.

**dbSet Methods**

به طور مثال متد Add که امکان اضافه کردن یک entity را به ما می دهد به این معنی که پس از فراخوانی Add توسط tracker دنبال می شود و بعد Save کردن در Db اجرا شده و state به حالت unchanged تغییر میکند.

**DbSet Attach Method in Entity Framework:**

برای اضافه کردن یک Entity به context است که در ابتدا با وضعیت unchanged قرار میگیرد.

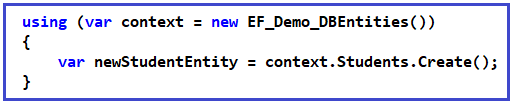


**DbSet AsNoTracking Method in Entity Framework:**

برای افزایش کارایی استفاده می شود در سناریو های که read only هستند داده های دریافت شده از Db

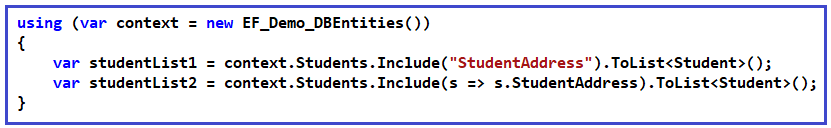
**DbSet Create Method in Entity Framework with Example:**

یک instance از entity داده شده برمیگرداند و این entity نه added است نه attached همچنین مقادیر prop ها به صورت default value هستند



**DbSet Include Method in Entity Framework:**

متد include برای مشخص کردن یک related object به نتیجه کوئری است مثلا لیست دانشجو ها به همراه address ان ها



استفاده از این روش بسیار مناسب تر است این که داده های مرتبط را با کوئری های متفاوت دریافت کنیم.

همچنین اگر از یک relation وارد بشویم هر دو entity دریافت می شوند.

var studentsList = context.Students

.Include**(**s =**>** s.Standard.Teachers**)**.ToList**()**;

**DbSet Find Method in Entity Framework:**

برای دریافت یک Entity مطابق با pk ارسال شده و اگر پیدا شود attached context می شوند.

**DbSet Remove Method in Entity Framework:**

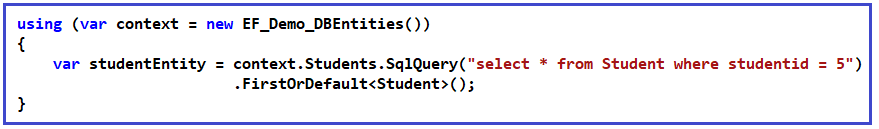
در ابتدا state را تغییر می دهد و سپس save را که انجام می دهیم بر روی Db تغییرات را اعمال می کند.

**DbSet RemoveRange Method in Entity Framework:**

برای حذف کردن یک collection از entity مورد نظر کاربرد دارد

**DbSet SqlQuery Method in Entity Framework:**

برای نوشتن raw sql کاربرد دارد و مقدار برگشتی Entity توسط context ما track می شود



**Relationships Between Entities**

روابط در دیتابیس های رابطه ای یک نیازمندی رایج است که ارتباط بین جداول را مدیریت می کند.

**Principal and Dependent entity**

در هر رابطه ای 2 جدول داریم که یکی Principal و دیگری Dependent است

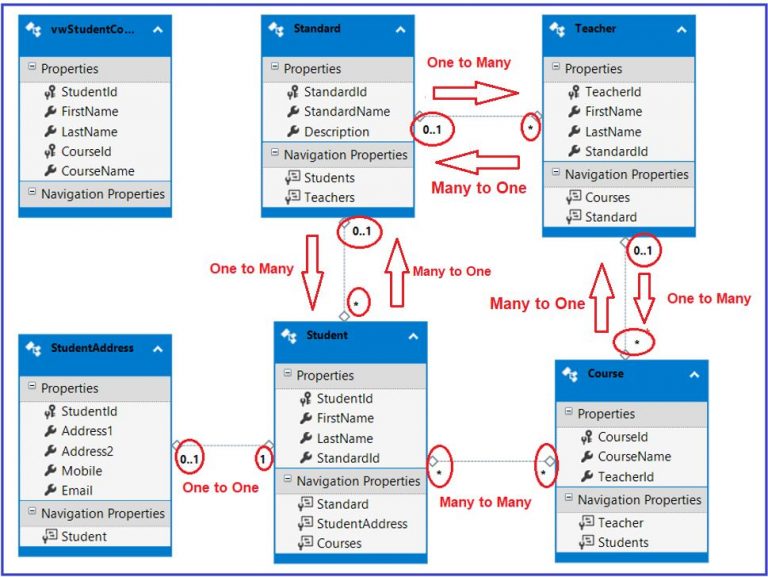
**Principal** جدولی است که شامل primary or unique key است

**Dependent** جدولی که شامل Foreign key است را dependent می گویند

همانطور که می دانید برای برقراری رابطه بین 2 جدول از foreign key استفاده میکنیم که نماینده کلید اصلی یک جدول است.

بنابر اصول نرمالسازی ما نباید تمامی informationرا در یک جدول ذخیره کنیم بلکه باید اطلاعات را به جداولی تقسیم کنیم و بین آن ها link برقرار باشد از طریق Relation

1. **Reference Navigation Property:** This is a Property that refers to a Single Related Entity, i.e., it is used to implement a One-to-One relationship between two entities.
2. **Collection Navigation Property:** This is a Property that refers to a Collection of Entities, i.e., it is used to implement One-to-Many Or Many to Many relationships between two entities.



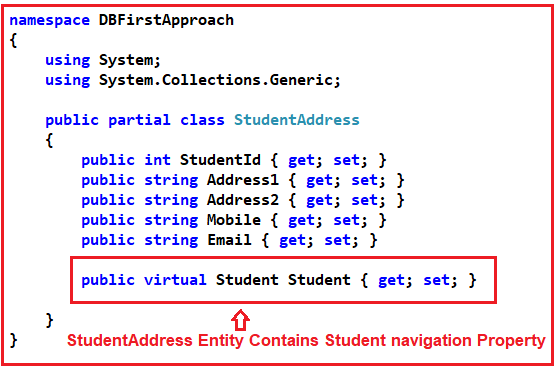
**One-To-One**

در این نوع رابطه یک ردیف داده از جدول A نمی تواند با بیش تر از یک ردیف داده از جدول b رابطه داشته باشد زمانی یک نوع رابطه برقرار می شود که هر دو ستون primary key باشند یا unique constraint داشته بشند و foreign key در این رابطه وجود ندارد.

در این حالت امکان برقراری one Or zero نیز وجود دارد یعنی یا با یک ردیف رابطه دارد یا هیچ رابطه ای وجود ندارد.

**در این حالت هر دور یک Reference navigation prop در یک دیگر دارند.**

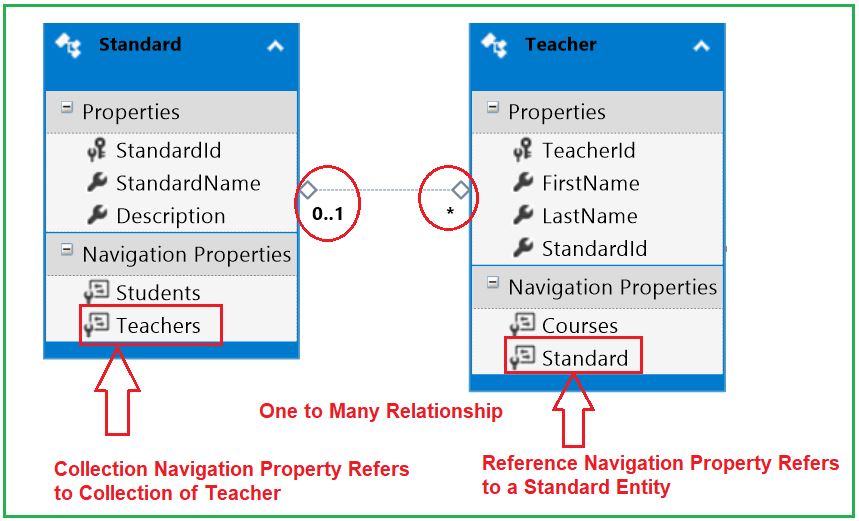
**در مثال زیر studentId در جدول StudentAddress از نوع Pk است.**

****

**One-to-Many**

رایج ترین نوع رابطه در بین جداول است. به این صورت که جدول a می تواند چندین ردیف داده که در جدول b با کلید جدول A match باشند می تواند داشته باشد و هر ردیف در جدول b می تواند با یک ردیف در جدول a match باشد.

**در جدول Teacher یک reference navigation prop است و در جدول standard یک collection navigation prop وجود دارد.**

****

در جدول Teacher یک ستون به نام StandardId داریم که از نوع ForeignKey است و به ستون StandardId مربوط به جدول standard اشاره دارد.

**Many-to-Many**

در این نوع رابطه یک ردیف داده از جدول a می تواند با چندین ردیف داده از جدول b مچ باشد و برعکس یک ردیف از جدول b می تواند با چندین ردیف از جدول A مچ باشد. در این نوع رابطه می توان با استفاده از 3 جدول کار را پیاده سازی کرد و 2 کلید خارجی یا foreign key داشته باشیم. یعنی کلید اصلی جدول سوم از نوع composite است.

در این نوع در entity های یک دیگر یک Collection navigation prop وجود دارد.

**CRUD Operations in Entity Framework**

2 سناریو برای انجام عملیات crud یا Create update delete Retrieve داریم

**Connect – Disconnect**

**سناریو اول Connected**

از همان dbContext برای بازیابی و ذخیره entity ها استفاده می شود. در این سناریو dbContext بر اساس entity که در چه state هست تصمیم به insert update delete میگیرد.

در این سناریو یک instance از dbContext تمام entity ها را ردیابی می کند و زمانی که یک entity ایجاد یا حذف می شود به صورت خودکار state را برای آن entity ست می کند.

امکان گرفتن لاگ از Db با کد زیر فراهم است.

context.Database.Log = Console.Write;

کد بالا لاگ گرفتن را از عملیات های زیر انجام می دهد

زمان برقراری اتصال به Db

زمان شروع شدن transaction

کوئری ایجاد شده توسط sql

زمان commit شدن transaction

زمان rollback شدن transaction

زمانی که اتصال بسته شود.

در سناریو بروزرسانی نیز به دلیل این که تمامی entity ها توسط context بازیابی شده اند track می شوند تغییرات را فهمیده و state را به modified تغییر داده و با saveChange تغییرات را در Db اعمال می کند.

**نکته: دستور update فقط برای prop های که تغییر داشته اند ایجاد می شود در این سناریو.**

**حذف**

برای حذف entity موجود و entity که به تازگی اضافه شده که توسط context دنبال می شوند کاربرد دارد یعنی مثلا یک entity که add شده اما در Db هنوز اعمال نشده اگر remove کنیم از فرآیند اضافه شدن به db حذف می شود و آن را detach می کند. در سناریو رایج باید اول entity را از Db خوانده سپس دستور remove را برای آن اجرا کرده و saveChange کنیم.

**Querying in Entity Framework**

3 روش برای دریافت داده از database

Linq to entities

Entity sql

Native sql

**LINQ-to-Entities Queries**

یکی از روش های دریافت داده از dbSet ها است مانند نمونه کد زیر به روش Method syntax

var student = context.Students.FirstOrDefault**(**x =**>** x.StudentId == 1**)**;

به روش **query syntax**

var student = **(from** std in context.Students

**where** std.StudentId == 1

**select** std**)**.FirstOrDefault**()**;

نکته: باید کلاس های که IDisposable را پیاده سازی کرده اند مثل Context با استفاده از Using() پیاده سازی شوند.

**Querying with Object Services and Entity SQL**

string sqlString = "SELECT VALUE st FROM EF\_Demo\_DBEntities.Students " +

"AS st WHERE st.StudentId == 1";

var objctx = (context as IObjectContextAdapter).ObjectContext;

Student student = objctx.CreateQuery<Student>(sqlString).FirstOrDefault();

**use EntityConnection and EntityCommand**

**using (var con = new EntityConnection("name=EF\_Demo\_DBEntities"))**

**{**

**con.Open();**

**EntityCommand cmd = con.CreateCommand();**

**cmd.CommandText = "SELECT VALUE st FROM EF\_Demo\_DBEntities.Students as st where st.StudentId=1";**

**Student student = new Student();**

**using (EntityDataReader rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess | CommandBehavior.CloseConnection))**

**{**

**while (rdr.Read())**

**{**

**student.StudentId = rdr.GetInt32(0);**

**student.FirstName = rdr.GetString(1);**

**student.LastName = rdr.GetString(2);**

**}**

**}**

**Console.WriteLine($"FirstName: {student?.FirstName}, LastName: {student?.LastName}");**

**Console.Read();**

**}**

**Native SQL Queries in Entity Framework:**

متد های که برای نوشتن مستقیم کوئری کاربرد دارند

1. **DbSet.SqlQuery()**
2. **DbContext.Database.SqlQuery()**
3. **DbContext.Database.ExecuteSqlCommand()**

**DbSet.SqlQuery()**

این متد امکان نوشتن کوئری مستقیم بر روی دیتابیس را می دهد که یک instance از entity مدنظر را می دهد.

var studentList =

context.Students.SqlQuery**(**"Select \* from Student"**)**.ToList**<**Student**>()**;

**نکته :**

در این روش به صورت خودکار مقادیر دریافتی از select که با نام prop های مدل داده شده match باشند به مدل map شده و لیست student ها دریافت می شود. **در غیر این صورت Exception می دهد**

در صورتی که کوئری نوشته شده ارتباطی به entity مه متد sqlQuery بر روی آن فراخوانی شده نداشته باشد دچار exception می شویم.

**ارسال Parameter در کوئری بالا**

**.SqlQuery(**"Select \* from Student where StudentId=@StudentId",

new SqlParameter("@StudentId", 1**));**

**SQL Query for Non-entity Types**

در این روش query بر اساس هیچ یک از entity های ما نوشته نشده است بلکه می تواند بر اساس دیتا تایپ مشخص شده توسط خود ما داده ی مرتبط را دریافت کنیم.

**string** studentName = context.Database.SqlQuery**<string>**

**(**"Select FirstName from Student where StudentId=1"**)**.FirstOrDefault**()**;

خروجی کد بالا یک string که شامل firstName است می باشد.

**Database.SqlQuery()**

برای دریافت هر نوع type کاربرد دارد

var student = context.Database.SqlQuery**<**Student**>(**"Select \* from Student where StudentId = 1"**)**.FirstOrDefault**()**;

**Database.ExecuteSqlCommand()**

این متد مناسب اجرا کردن کوئری های Dml یا data manipulation language کاربرد دارد.

کوئری insert – update – delete

**int** noOfRowUpdated = context.Database.ExecuteSqlCommand**(**"Update Student set FirstName = 'FirstName Changed' where StudentId = 1"**)**;

**int** noOfRowInserted = context.Database.ExecuteSqlCommand**(**"Insert into Student(FirstName, LastName, StandardId) values('F1', 'L1', 1)"**)**;

**int** noOfRowDeleted = context.Database.ExecuteSqlCommand**(**"Delete from Student where Studentid = 7"**)**;

**LINQ to Entities Queries in Entity Framework**

نکته : کلاس DbSet<> در Ef رابط IQueryable<> را پیاده سازی کرده اند به همین دلیل است که بر روی DbSet ها می توانیم query بنویسیم و در نهایت کوئری نوشته شده تبدیل می شود به کوئری قابل فهم برای sql و دیتابیس و نتیجه دریافتی به object مشخص شده map می شود و دریافت می شود.

Linq مخفف language integrated query است و در فضای نام system.Linq قرار دارد.

Linq این امکان را می دهد که بر روی داده های مختلفی مصل xms – ado.net – database – collection و ... کوئری نوشته.

**What is Projection?**

مکانیسمی است برای انتخاب کرده داده ها از یک Data source

**متد های Linq**

**Find()**

برای پیدا کردن یک entity بر اساس pk کاربرد دارد در صورتی که در context شی مورد نظر ما وجود داشته باشد بدون این که درخواستی به Db بفرستد آن را برمیگرداند اگر پیدا شود آن را Attach کرده و برمیگرداند و در غیر این صورت null برمیگرداند.

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

**First() – FirstOrDefault()**

برای زمانی کاربرد دارد که بخواهیم single entity را از db دریافت کنید اگر چند تا ایتم که مچ باشند داشته باشیم اولی را به ما می دهد

var student1 = context.Students

.FirstOrDefault**(**s =**>** s.FirstName == "Rohit"**)**;

تفاوت این 2 : در صورتی که Find مقدار متانسب را پیدا نکند Excetpion برمیگرداند و برای FirstOrDefautl با توجه به نوع داده مقدار Default را برمگیرداند.

**ToList()**

برای دریافت لیستی از یک نوع داده کاربرد دارد.

var studentList = context.Students.Where**(**x =**>** x.StandardId == 1**)**.ToList**()**;

**GroupBy**

دقیقا مشابه عملیاتی است که در sql با استفاده از group by انجام می شود. به این صورت که مجموعه ای از داده ها را بر اساس یک کلید گروه بندی می کند و برای هر گروه نتیجه را می دهد.

var studentsQS = **from** s in context.Students

**group** s **by** s.StandardId **into** studentsByStandard

**select** studentsByStandard;

**OrderBy**

برای دریافت لیست entity ها به صورت sort شده کاربرد دارد. تنها کاری که میکند دریافت داده ها به یک ترتیب خاص است.

**Anonymous Object Result:**

به جای تبدیل داده ها به هنگام دریافت به یک type مشخص از قبل مثل student می توانیم با استفاده از نوع anonymous بدون تبدیل به نوع مشخص داده ها را دریافت کنیم

**(from** std in context.Students

**select** new **{**

firstName = std.FirstName,

lastName = std.LastName **});**

**Join()**

برای دریافت داده های که match هستند از 2 منبع داده کاربرد دارد و داده های که match نیستند حذف شده و در خروجی نمی آیند یعنی تنها رکورد های که در 2 جدول موجود هستند دریافت می شوند.

var JoinUsingMS = context.Students //Outer Data Source

.Join**(**

context.StudentAddresses, //Inner Data Source

student =**>** student.StudentId, //Inner Key Selector

studentaddress =**>** studentaddress.StudentId, //Outer Key selector

**(**student, studentaddress**)** =**>** new //Projecting the data into an anonymous type

**{**

StudentName = student.FirstName +" "+ student.LastName,

AddressLine1 = studentaddress.Address1,

AddressLine2 = studentaddress.Address2,

**})**.ToList**()**;

**نکته :**

**چه حالت method syntax یا حالت query syntax استفاده شود فرقی در کوئری دیتابیس ساخته شده ایجاد نمی شود.**

**Eager Loading in Entity Framework**

**Eager Loading in Entity Framework**

**How many ways we can load the related entities in Entity Framework?**

3 راه برای لود کردن داده در Ef وجود دارد.

Eager – Lazy – Explicit

**نکته :** Navigation prop ها چیزی جز Fk های ما نیستند که تبدیل به np شده اند در واقع related entity های ما هستند.

**Eager Loading in Entity Framework**

فرآیندی است که در آن Ef کار لود کردن related entity ها را به همراه main entity انجام می دهد در این حالت related ها با separated query دریافت نمی شوند بلکه با یک single query دریافت شده و زمانی که cpu time , bandwidth داریم **ef با استفاده از join داده های مرتبط را دریافت می کند.**

**How to Implement Eager Loading in Entity Framework?**

به صورت پیشفرض ef لود کردن related ها را به صورت lazy انجام می دهد اما اگر بخواهیم eagerly ان ها را دریافت کنیم باید از متد **Include** استفاده کنیم.

2 تا overload دارد اولی دریافت string و جدا کردن entity های مرتبط با .

Include(“**Standard.Teachers**“)

var students = context.Students.Include**(**"Standard"**)**.ToList**()**;

به دلیل این که با یک کوئری داده های retrive می شوند از نظر عملکردی هم مناسب است.

context.Students.Include**(**x =**>** x.Standard**)**.ToList**()**;

var students = context.Students

.Include**(**x =**>** x.Standard**)**

.Include**(**x =**>** x.StudentAddress**)**

.Include**(**x =**>** x.Courses**)**

.ToList**()**;

روش فوق برای دریافت چندین entity relatedبه صورت هم زمان است.

**How to Load Multiple Levels of Related Entities**

به معنی حالتی است که رابطه ی ما به صورت one to many بوده یعنی navigation prop ما از نوع collection باشد.

var student = context.Students

.Include**(**"Standard"**)**

.Include**(**x =**>** x.StudentAddress**)**

.FirstOrDefault**(**s =**>** s.StandardId == 1**)**;

context.Students.Include**(**"Standard.Teachers"**)**.ToList**()**;

**Lazy Loading in Entity Framework**

فرآیندی است که در آن Ef موجودیت های مرتبط را بر اساس تقاضا دریافت می کند که به صورت رفتار پیش فرض Ef است. یعنی فرآیند دریافت related ها child ها را با تاخیر انجام می دهد تا زمانی که درخواستی باشد.

**Student student = context.Students.FirstOrDefault(std => std.StudentId == 1);**

زمانی که کد بالا اجرا می شود فقط student دریافت می شود نه navigation prop ها.

در صورتی که بعدا کد زیر را بنویسید context درخواست Select مرتبط را ایجاد می کند

**StudentAddress studentAddress = student.StudentAddress;**

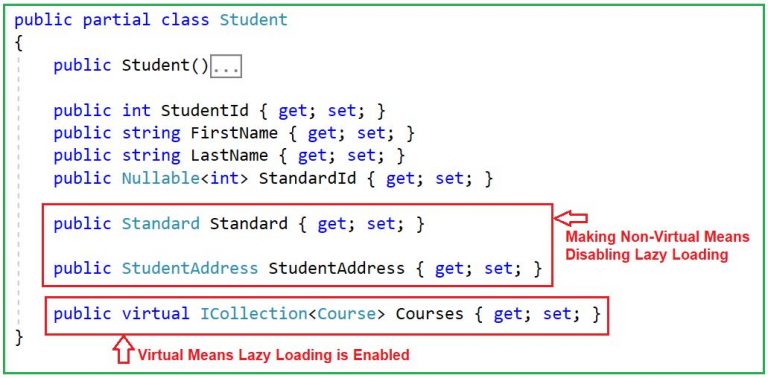
و همچنین برای کد زیر نیز این روند تکرار می شود.

**Standard standard = student.Standard;**

**How to Disable Lazy Loading in Entity Framework?**

امکان غیر فعال کردن برای تمام entity ها و یک entity خاص وجود دارد.

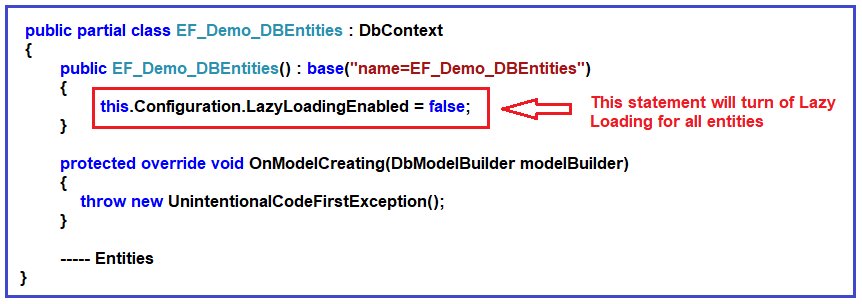
با حذف کردن کلمه کلیدی Virtual امکان پذیر است.



**نکته : در صورتی که vritual را حذف کنید زمانی که np را فراخوانی کنید دیگر Data ندارد null است و درخواست به db ارسال نمی کند**

**How to disable it for all entites**

**از طریق کلاس Context و ctor امکان غیر فعال کردن وجود دارد برای تمامی En ها.**

****

**Lazy Loading vs Eager Loading**

**Eager**

برای دریافت entity های مرتبط از یک query به db استفاده می کند.

**Lazy**

برای دریافت entity های مرتبط با تاخیر کاربرد دارد و رفتار پیش فرض ef است

تفاوت :

یکی همه related ها را درجا دریافت می کند و دیگری با اولین درخواست و با تاخیر

Lazy برای بهبود performance کاربرد دارد

Lazy برای related entity ها درخواست select جدا میزند اما eager درخواست join با همان select اول میزند.

در صورتی که داده های related را در همان ابتدا بخواهید بهتر است از eager استفاده کنید به دلیل تعداد درخواست کمتر به db

**Explicit Loading in Entity Framework**

برای این که بتوانیم این نوع load کردن داده ها را تست کنیم لازم است که ابتدا lazy را از تنظیمات dbContext غیرفعال کرده.

Reference & Collection navigation property

برای لود کردن relatedEntity های که از نوع reference هستند باید باید از متد Reference() و Load() استفاده کرد.

context.Entry**(**student**)**.Reference**(**s =**>** s.StudentAddress**)**.Load**()**;

همچنین برای load کردن نوع های Collection باید از روش زیر استفاده کرد

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

در این حالت به صورت lazy موجودیت های related را می توانیم load کنیم.

**Disconnected Entities in Entity Framework**

در سناریو های امکان دارد که یک entity که توسط یک کلاس context از Db دریافت شده است توسط کلاس context دیگری استفاده شود در این حالت باید context بداند که entity در چه State است.

**نکته : تمامی entity های که توسط context ما track نمی شوند از نوع disconnected هستند.**

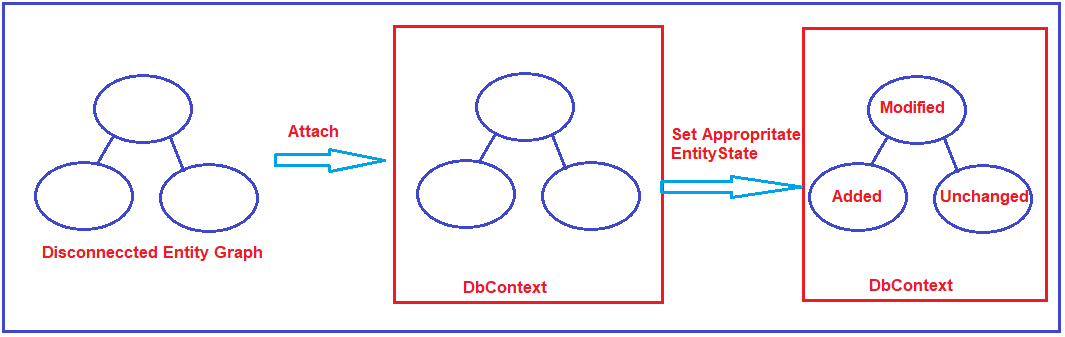
**Methods to Attach Disconnected Entities**

برای انجام این کار باید 2 قدم کار انجام دهیم.

یک : باید entity مورد نظر را به کلاس کانتس خود attach کنیم

دو : باید state را به صورت دستی برای entity ست کنیم.

**سناریو برای یک entity و child های آن**

****

**متد های که برای attach کردن entity کاربرد دارند :**

1. **DbContext.Entry()**
2. **DbSet.Add()**
3. **DbSet.Attach()**

**DbContext.Entry()**

از این متد برای set کردن state مربوط به entity می توان استفاده کرد.

همچنین کار Attach کردن entity به context را نیز انجام می دهد.

**نکته : در حالت Added برای تمامی child ها نیز state را یکسان با parent می گذارد.**

اگر parent را به حالت modified ست کنیم فرزند ها به صورت unchanged خواهند بود.

اگر parent به صورت deleted باشد فرزند ها نیز به صورت modified یا unchanged خواهد بود.



**context.Entry(entity).state = EntityState.Added/Modified/Deleted**

**using (var context = new EF\_Demo\_DBEntities())**

**{**

**//Attaching the Disconneect Student Entity Graph to the Context Object with Added State**

**context.Entry(student).State = EntityState.Added;**

**//Checking the Entity State of Each Entity of student Entity Graph**

**foreach (var entity in context.ChangeTracker.Entries())**

**{**

**Console.WriteLine($"Entity Name: {entity.Entity.GetType().Name} Entity State: {entity.State}");**

**}**

**DbSet.Add()**

موجودیت پدر و تمامی فرزند ها را به وضعیت Added اضافه می کند به context

**DbSet.Attach()**

تمامی فرزند ها به همراه پدر را به context با وضعت unchanged اضافه می کند

**Saving Disconnected Entity in Entity Framework**

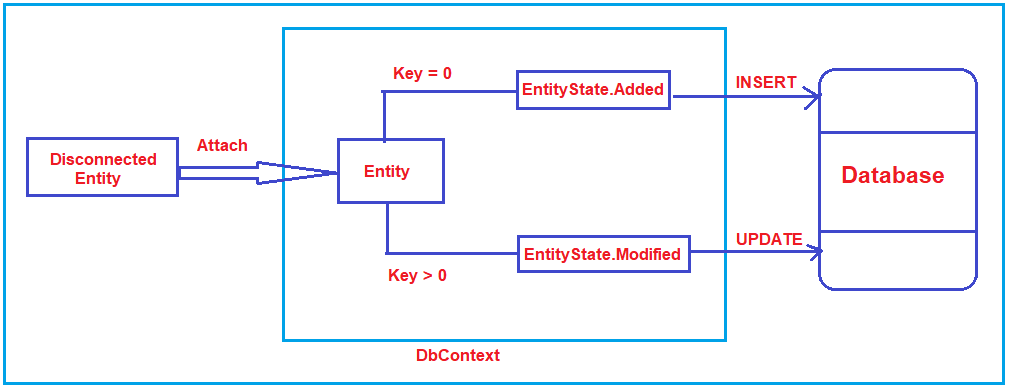
به هر موجودیتی که توسط dbContext ما track نمی شود یک موجودیت disconnected می گوییم.

به صورت پیش فرض DbContext تمامی entity های که از db ما return می شوند را track می کند و هر تغییر بر روی آن ها رهگیری شده و در نهایت با saveChaneg بر روی db اعمال می شوند.

**How to Save a Disconnected Entity**

روش ذخیره موجودیت ها در سناریو های disconnected متفاوت است با حالت connect به این دلیل که context در ابتدا نمی داند که وضعیت entity چه چیزی است یعنی add , modified و ... بنا بر این لازم است در ابتدا موجودیت attach شود و سپس state آن را مشخص کنیم

روش فهمیدن جدید بودن entity این است آیا Pk دارد یا نه.



context.Entry**(**newDisconnectedStudent**)**.State = newDisconnectedStudent.StudentId == 0 ? EntityState.Added : EntityState.Modified;

اگر برابر 0 باشد یعنی added و غیر از آن modified است.

**Disconnected Entity Graph**

Graph به معنی یک Entity است که دارای related entities است.

در مرحله ی اول با توجه به مقدار pk می توانیم تشخیص دهیم که آیا در وضعیت add هستیم یا modified

**قطعه کد زیر فرآیند کامل set graph state را انجام می دهد.**

**var student = new Student()**

**{**

**//Root Entity with Empty key**

**FirstName = "Steven",**

**LastName = "Smith",**

**Standard = new Standard() //Child Entity with key value**

**{**

**StandardId = 1,**

**Description = "STD1 Description"**

**},**

**Courses = new List<Course>() {**

**new Course(){ CourseName = "AI" }, //Child Entity Empty key**

**new Course(){ CourseId = 1 } //Child Entity with key value**

**}**

**};**

**student.StudentAddress = new StudentAddress() //Child Entity with StudentId as 0**

**{**

**StudentId = student.StudentId,**

**Address1 = "Address Line1",**

**Address2 = "Address Line2"**

**};**

**using (var context = new EF\_Demo\_DBEntities())**

**{**

**//To See the Generated SQL**

**context.Database.Log = Console.Write;**

**//Set the Student entity state based on StudentId**

**context.Entry(student).State = student.StudentId == 0 ? EntityState.Added : EntityState.Modified;**

**//Set the Standard entity state based on StandardId**

**context.Entry(student.Standard).State = student.Standard.StandardId == 0 ? EntityState.Added : EntityState.Modified;**

**//Set the Course Entity state based on CourseId**

**foreach (var course in student.Courses)**

**{**

**context.Entry(course).State = course.CourseId == 0 ? EntityState.Added : EntityState.Modified;**

**}**

**Asynchronous Programming with Entity Framework**

برنامه نویسی ناهمزمان **برای جلوگیری از thread blocking در هنگام اجرای query در Db کاربرد دارد**.

**کار اصلی آن اجرا در background است و main thread به انجام کار های خود می پردازد.**

مثلا main thread باعث پاسخ گو ماندن ui شده و background کار خود را انجام می دهد.

به طور کلی اجازه ی می دهد که مدریت پردازش thread ها در یک روش کارآمد انجام شود.

**ایده اصلی عدم block شدن thread به دلیل منتظر نتیجه بودن است.**

**مثال در مواقعی که با webservice , db , file management کار داریم مناسب است.**

**در مواقع ارسال درخواست http می توان تعداد درخواست بیشتری فرستاد به دلیل این که هر درخواست را یک thread اجرا می کند.**

زمانی که متد های Async اجرا می شوند یک thread از pool اجرای کار را بر عهده می گیرد.

نکته : زمانی که از .Result استفاده می کنیم صبر می کند تا نتیجه بیایید و thread block رخ می دهد.

**نکته : در درجه اول برای آزاد کردن current thread کاربرد دارد برای انجام وظایف دیگر**

**نکته : در برنامه های desktop , console , wpf برای پاسخ گو نگه داشتن Ui کاربرد دارد و در web app ها برای پردازش سایر درخواست ها کاربرد دارد هر چه thread بیشتر تعداد Req که پاسخ می دهد بیشتر است.**

**Bulk Insert, Update and Delete**

روش اول: AddRange()

برای افزودن یک array یا list از Entity ها به context با وضعیت Added کاربرد دارد و با یک دستور مجموعه ای از entity ها را اضافه می کند.

BulkUpdate

برای پیاده سازی آپدیت گروهی لازم است ابتدا تمامی entity های که لازم است آن ها را بروزرسانی کنیم از database دریافت کرده و سپس بر روی property که لازمه تغییرات را اعمال کنیم و بعد از آن state به حالت modified تغییر می کند و پس از این که عملیات بروزرسانی را برای تمامی entity ها اعمال کردیم باید یک بار saveChange را فراخوانی کنیم تا ذخیره شوند.

**3 تا درخواست update برای 3 تا entity تغییر کرده ارسال می کند.**

**به طور کلی تعداد رفت و برگشت برابر تعداد موجودیت ها است چه update – insert delete –**

**Bulk Delete**

لیست entity های که می خواهیم delete شوند را دریافت می کنیم و سپس همه گی را در متد **RemoveRange**() قرار می دهیم. ضمنا نیازی نیست داده ها دریافت شوند صرفا کوئری کافی است.

Var studentsList = context.Students.Where**(**std =**>** std.LastName.Equals**(**LastName**))**;

context.Students.RemoveRange**(**studentsList**)**;

context.SaveChanges**()**;

**مشکلی در روش های بالا وجود دارد این است که بر اساس state هر Entity دستورات delete update insert را ایجاد می کند.**

**Bulk Insert With Z Entity**

**مزیت اصلی کاهش تعداد درخواست ها به Db است رفت و برگشت کمتر.**

**BulkInsert Extension Method**

برای درج تعداد زیادی entity با یک رفت آمد در db کاربرد دارد

**context.BulkInsert(listStudents);**  
**context.BulkInsertAsync(listStudents, cancellationToken);**

عملیات درج را با استفاده از دستور sql merge پیاده سازی می کند که سبب کاهش رفت و برگشت به Db و بهبود چشمگیر عملکرد برنامه می شود.

در تستی برای 1000 عملیات درج سرعت z entity حدود 20ms است و ef حدود 490 ms است.

**علت اصلی که سرعت بیشتری داریم در این حالت تعداد round trip های کمتر است**

**Options of bulk insert**

**BatchSize** به ما این اجازه را می ده که مثلا 50 تا Entity را به عنوان یک batch درج کند و 100 موجودیت را به 2 batch یا دسته تبدیل کند.

context.BulkInsert**(**studentList, options =**>** **{**

options.BatchSize = 50;

**})**;

**InsertKeepIdentity** به ما این امکان را میدهد که Id را خودمان سمت بکند مقدار دهیم در مدل و می ماند.

**Context.BulkInsert(studentList, options => options.InsertKeepIdentity = true);**

**Insert only if the Entity Does not Already Exist**

**فقط در صورتی اجازه درج می دهد که قبلا در Db وجود نداشته باشد به این صورت که ابتدا فعال شود وسپس کلیدی که باید بررسی شود را مشخص می کنیم**

**context.BulkInsert(studentList, options => {**

**options.InsertIfNotExists = true;**

**options.ColumnPrimaryKeyExpression = c => c.FirstName;**

**});**

**Insert with Related or Child Entities**

افزودن Entity ها به همراه related childs

1. **IncludeGraph**: This option allows us to automatically insert all entities as part of the graph.
2. **IncludeGraphBuilder**: This option allows us to customize how to insert entities for a specific type.

context.BulkInsert**(**studentList, options =**>** options.IncludeGraph = **true)**;

بعد باید تمامی child های موجودیت مقدار داشته باشند تا مشکلی پیش نیاید.

**Stored Procedure in Entity Framework**

یک سری کوئری کامپایل شده است که از قبل نوشته شده در sql و با هر بار فراخوانی کاری را انجام می دهد.

مواقعی است که نمی خواهیم از دستورات linq و Ef برای کوئری زدن به دیتابیس استفاده کنیم. به طور مثال می خواهیم با استفاده از sp کاری مشابه عملیات های add یا delete یا insert و یا update را انجام دهیم.

زمانی که در حال ساختن data model هستیم خود ef به صورت خودکار sp ها را در قالب function به ما می دهد و میتوانیم با استفاده از context به آن ها دسترسی داشته باشیم