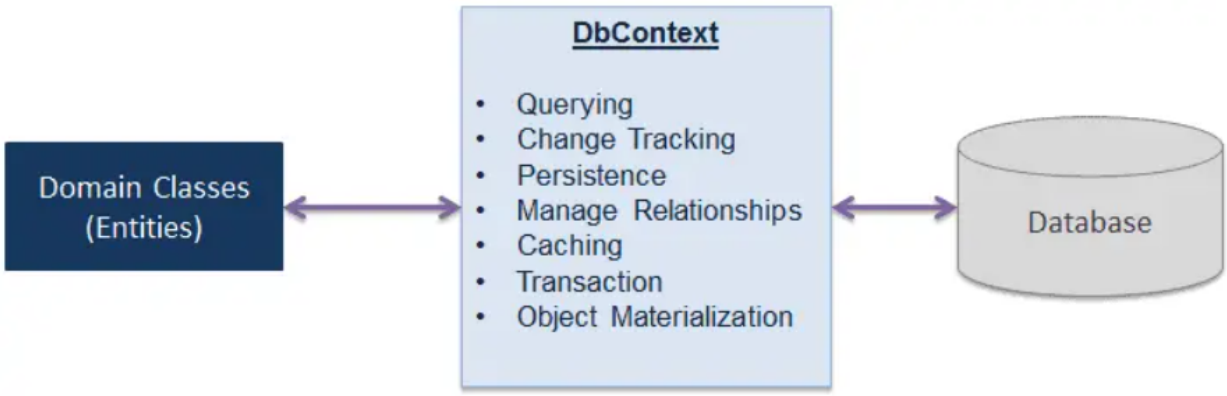
4.1.1 Архитектура Entity Framework алата

4.1.1.1 DbContext и DbSet

DbContext представља основу класу одговорну за интеракцију са базом података. Ова класа служи као интерфејс између доменских класа и табела у бази података. Одговорности DbContexta у раду са базом података су:

- **Управљање упитима**: Превођење LINQ-to-Entities упита у SQL упит који се шаље ка бази података
- **Праћење промена ентиета**: DbContext представља имплементацију Unit of Work шаблона, стога ова класа прати све промене које се дешавају над ентитетима и осигурава да база података буде у конзистентном стању
- **Перзистенција података**: На основу испраћеног стања ентитета, DbContext извршава одговарајуће DML наредбе над базом података
- **Кеширање:** Пружа могућност кеширање на првом нивоу и за трајање свог животног циклуса чува ентитете који су били добављени неким од претходних упита
- **Материјализација:** Процес током којег се подаци добављени из базе конвертују у објекте ентитета



DbSet је класа која представља интерфејс помоћу које се врши комуникација ка једној табели у бази података. Пружа могућност вршења Add, Remove, Find, Attach операција помоћу којих се генеришу неопходне DML операције над базом података. Такође, помоћу метода Local, SqlQuery и DbQuery пружа могућност, приступу локалним подацима у кеш меморији DbContexta, директно писање SQL упита, као и писање LINQ упита над ентитетима. DbSet се најчешће декларише као поље класе која наслеђује DbContext, која затим координише свим ентитетима кроз инстанце DbSet класе.

```
public partial class BenchmarkDbContext : DbContext
{
    public BenchmarkDbContext()
    {
    }

    public virtual DbSet<Post> Posts { get; set; }

    public virtual DbSet<Profile> Profiles { get; set; }

    public virtual DbSet<Tag> Tags { get; set; }

    public virtual DbSet<User> Users { get; set; }
}
```

- Слика 4.1 - Препоручен начин коришћења DbContext и DbSet класа

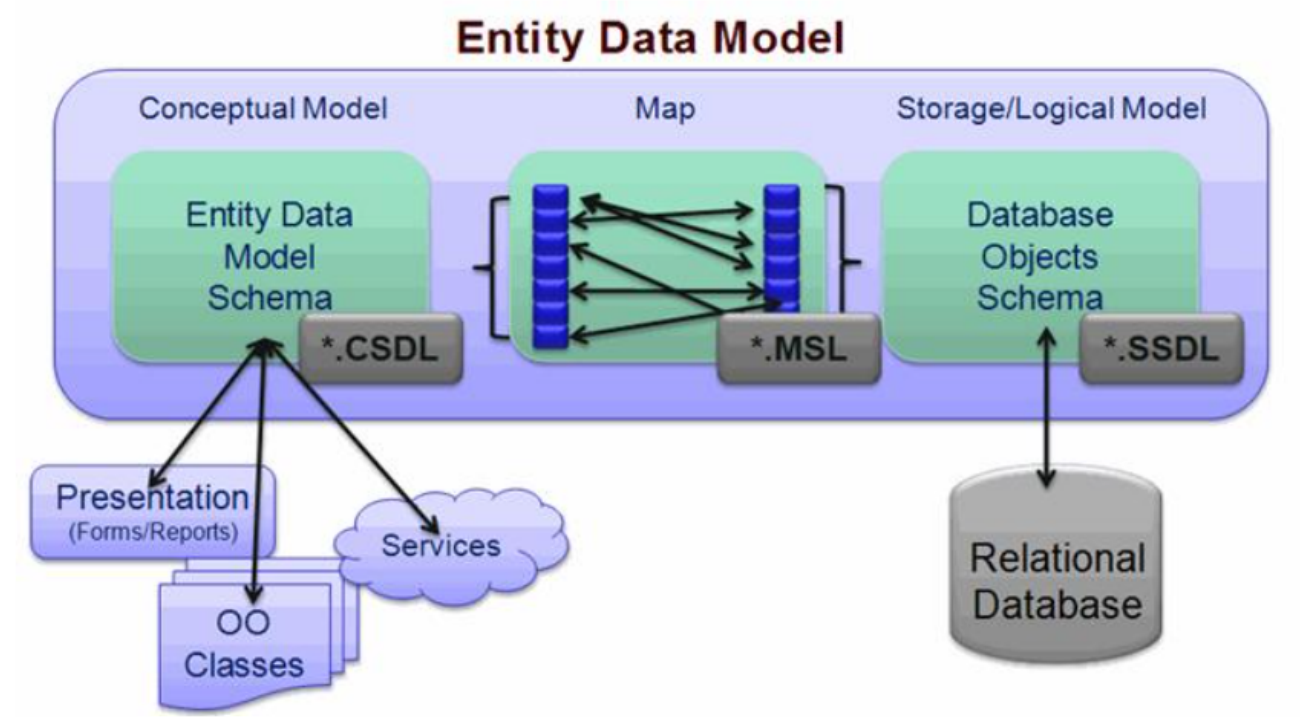
4.1.1.2 Entity Data Model

Entity Data Model (EDM) је скуп концепата који описују структуру података независно од начина на који се они чувају. ЕДМ је, заправо, *in-memory* репрезентација свих мета података неопходних за објектно-релационо мапирање и састоји се од: концептуалног модела, модела складишта (*storage model*) и мапирањима између њих. Овакав концепт омогућава логици апликације да се фокусира на домен, изолојући је од детаља приступа бази.

Концептуални модел чине доменске класе апликације, DbContext класа и DbSet колекције. Складишни модел заснован је на шеми базе података са којом је апликација повезана. Информације о мапирањима су неопходне како би се могло ефикасно врши пребацивање из концептуалног модела у складишни и обрнуто.

Како би ово мапирање било ефикасно, Ентити Фрејмворк користи три језика заснована на XML језику: Conceptual Schema Definition Language (CSDL), Store Schema Definition Language (SSDL) и Mapping Specification Language (MSL).

CSDL се фокусира на концептуални део апликације, он описује ентитете и њихове односе, без узимања у обзир специфичних детаља или структуре конкретне базе података.
SSDL описује логичку структуру података базе података повезане са апликацијом, укључујући табеле, колоне, везе и остале релационе атрибуте.
Да би се осигурало да оба света могу да комуницирају без ометања, MSL постоји као мост између ова два језика. Он дефинише правила и начине на које се концептуални модели из CSDL-а мапирају на енитете у бази података описаних путем SSDL-а.



4.1.1.3 Руковање конекцијама

У раду са базом података, веома је битно разумети како се управља конекцијама, почевши од њиховог успостављања до затварања, како не би дошло до ситуација у којима су угрожене перформансе или безбедност система.
Ентити Фрејмворк пружа ефикасан и оптимизован систем за руковање конекцијама.

Успостављање конекције

Када DbContext инстанца буде креирана, Ентити Фрејмворк не отвара одмах конекцију са базом података. Уместо тога, конекција ће бити отворена само када је то потребно, приликом извршења упита или када је потребно сачувати податке.

Животни циклус конекције

У својој основној поставци, Ентити Фрејмворк користи краткотрајне конекције, што значи да се конекција отвара само када је потребно и брзо се затвара после коришћења чиме се минимизују ресурси које конекција заузима. Међутим, у неким случајевима, као што су трансакције, конекција може да буде отворена дужи временски период.

Затварање и ослобађање конекције

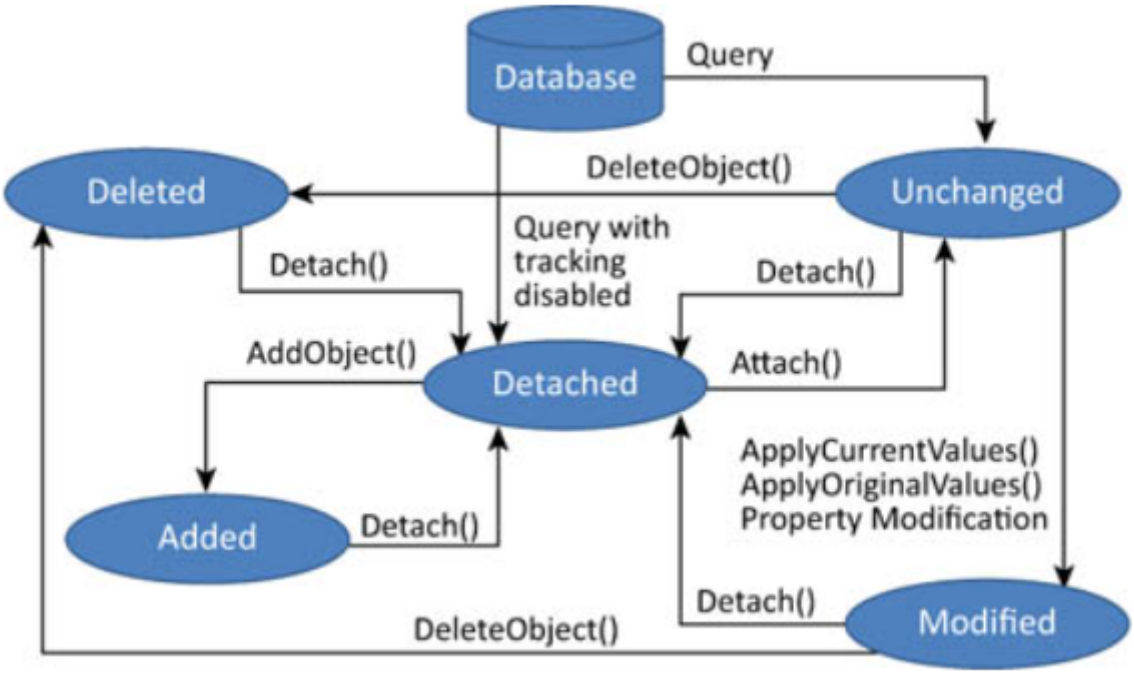
Када је рад са базом података завршен, Ентити Фрејмворк аутоматски затвара конекцију како би осигура да не дође до непотребног трошења ресурса. Такође, препоручено је да се користи *using* исказ приликом рада са инстанцом DbContext класе како би се осигурало правилно ослобађање ресурса.

4.1.1.4 Праћење промена ентитета

Entity Framework користи механизам пратења промена да би идентификовао и записао све измене на ентитетима од последњег учитавања из базе података. Ова способност је кључна за рад са ОРМ системима и осигурава ефикасност и усаглашеност при ажурирању базе података.

Ентитет може имати једно од пет стања која одређују како ће се на њега гледати при следећем позиву методе *SaveChanges* инстанце класе DbContext:

1. **Unchanged (Непромењен)** - Ентитет је учитан из базе података и није било измена на њему од тада
2. **Modified (Измењен)** - Након учитавања из базе података, вршене су промене на ентитету. Ова променаће бити сачувана у бази података при следећем позиву методе *SaveChanges*
3. **Deleted (Обрисан)** - Ентитет је обележен за брисање и биће уклоњен из базе података при следећем позиву методе *SaveChanges*
4. **Added (Додат)** - Ентитет је додат у контекст али још увек не постоји у бази података. Биће креиран нови запис у бази података при следећем позиву *SaveChanges* методе
5. **Detached (Одвојен)** - Ентитет је у контексту али не прати промене на њему. Ниједна промена овог ентитета неће бити рефлектована у бази података



Ентитети могу да пређу у одвојено стање уколико се уништи инстанца DbContext класе или уколико се то експлицитно затражи.

Генерално посматарано, DbContext је дизајниран да представља краткотрајну јединицу рада (Unit of Work). Што значи да је уништавање инстанце ове класе природан начин да се престану пратити стања објеката. Из овога се може претпоставити један типичан животни циклус инстанце DbContexta:

1. Креирање DbContext инстанце
2. Праћење ентитета
3. Вршење измене над ентитетима
4. Позив методе *SaveChanges* како би се измене примениле у бази података
5. Уништавање DbContext инстанце

Разматрањем најбитнијих делова архитектуре Ентити Фрејмворка, пружен је увид у основне компоненте и начин на који оне интерагују у процесу рада са базом података. У наредној секцији, детаљније ће се истражити процес извршавања упита, са главним акцентом на LINQ упите и начином на који се они преводе у SQL.