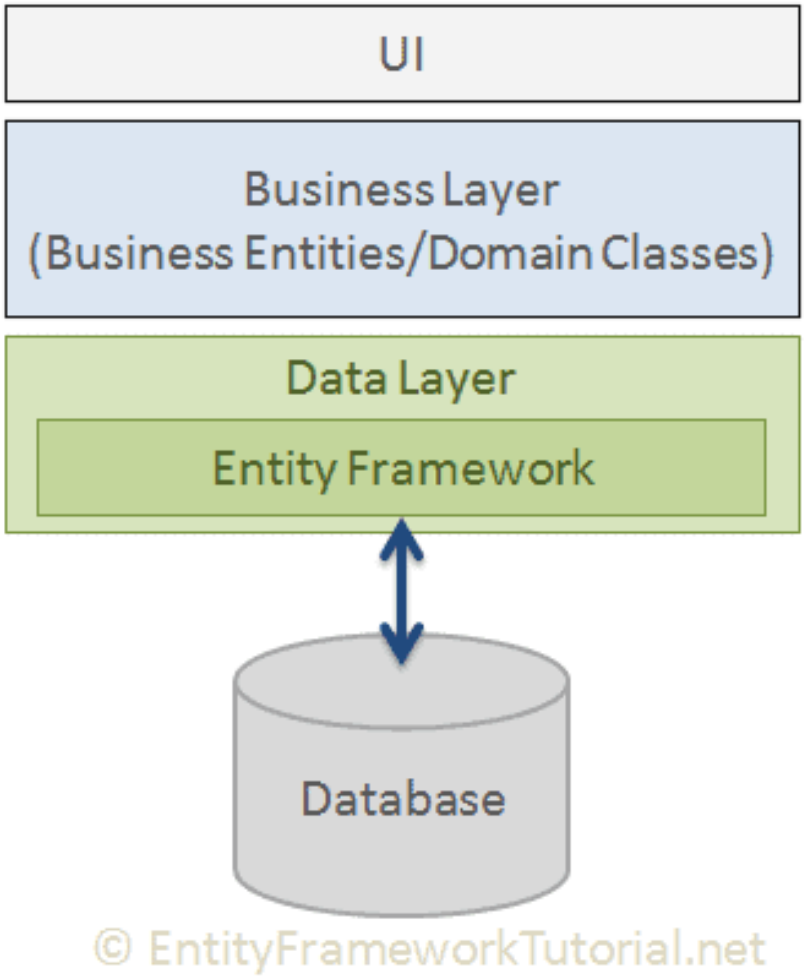
Dell IT Academy

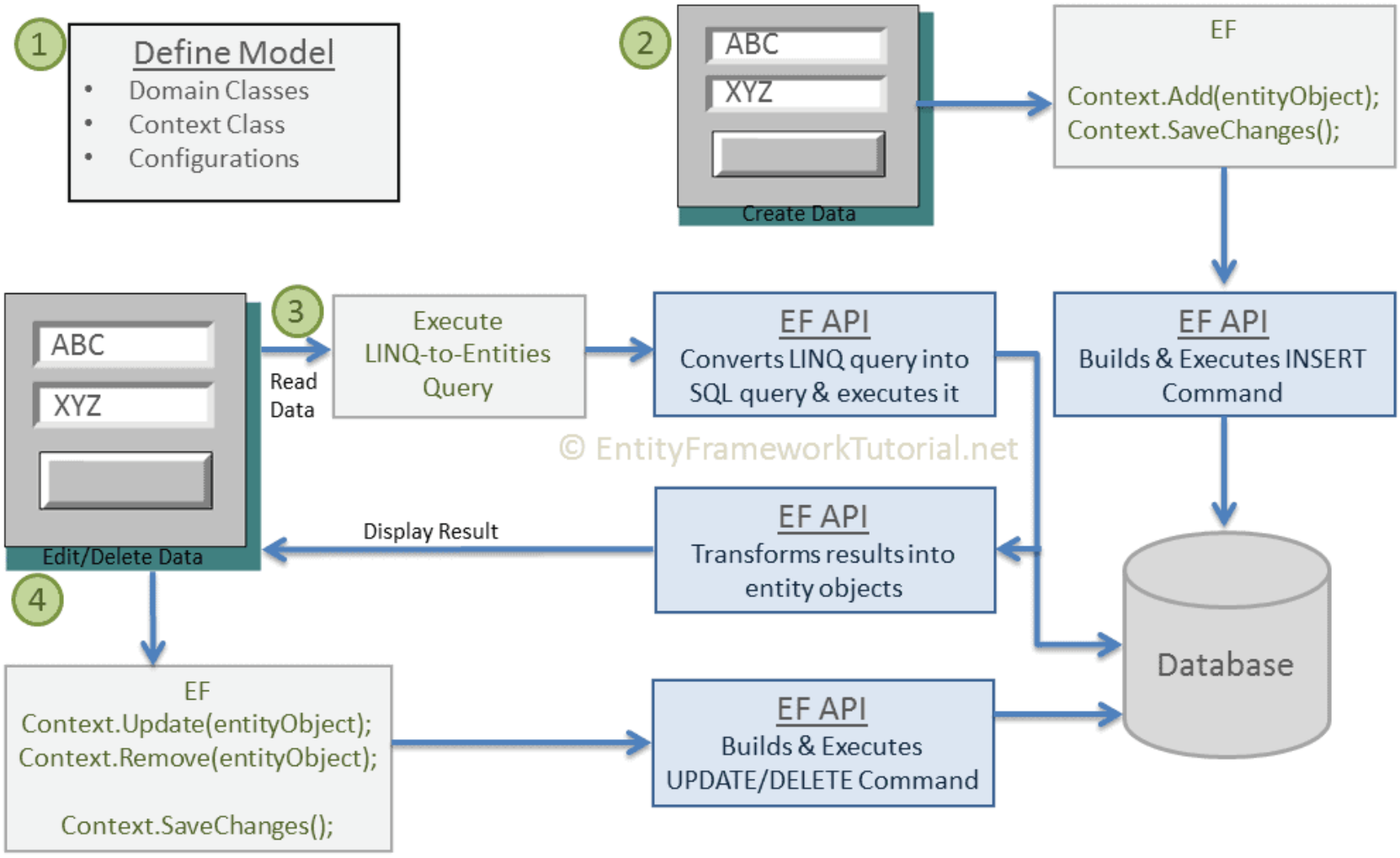
ENTITY FRAMEWORK CORE

Entity Framework Core

- É um framework da família .NET de persistência baseado no padrão *Data Mapper* multiplataforma
  - Windows, Linux e Mac
- Suporta múltiplas fontes de dados
  - https://docs.microsoft.com/en-us/ef/core/providers/index
  - Distribuído via pacotes NuGet
- Suporta o uso de LINQ – Language Integrated Query do .NET



Entity Framework Core

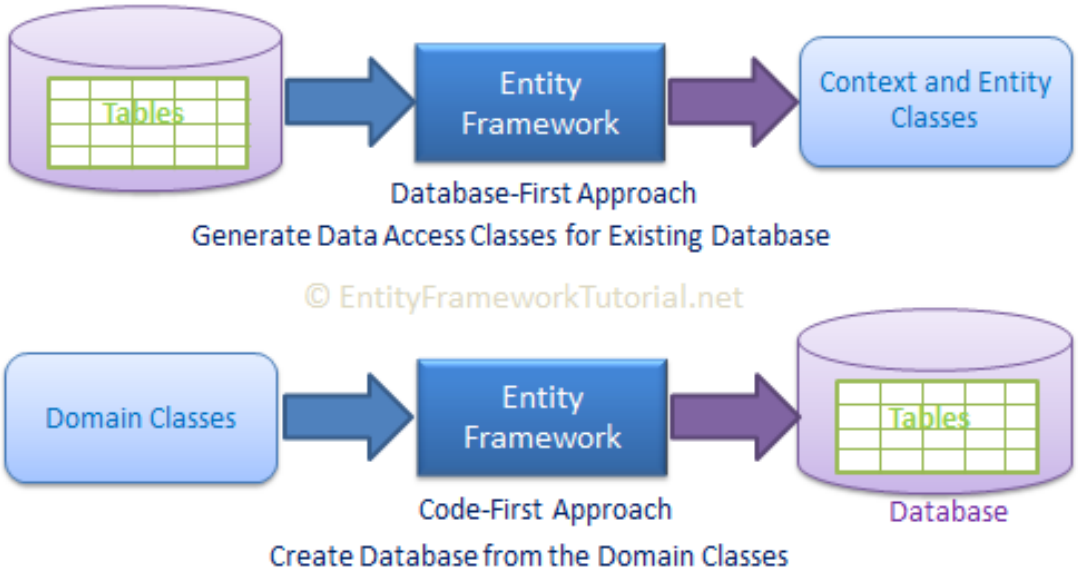


POCO - “*Plain Old C# Objects*”

- Objetos de domínio já existentes
- São objetos que não conhecem o mecanismo de persistência

Models

- Classes de modelo (POCO's) são mapeadas para tabelas, colunas e relacionamentos do banco de dados
- Formas de trabalho:
  - Code-first
  - Database-first



Models

- EFCore permite:
  - Utilizar convenções de mapeamento a fim de diminuir a quantidade de código necessário para mapear os objetos
  - Utilizar anotações sobre os objetos para configurar as regras de mapeamento
  - Utilizar uma API do tipo “fluente” para configurar as regras de mapeamento
    - Permite uma separação total entre um objeto de negócio e as regras de mapeamento
- Documentação:
  - https://docs.microsoft.com/en-us/ef/core/modeling/

Models

Tipo de dados C#	Tipo de dados SQL Server
int	int
string	nvarchar(Max)
decimal	decimal(18,2)
float	real
byte[]	varbinary(Max)
datetime	datetime
bool	bit
byte	tinyint
short	smallint
long	bigint
double	float
char	No mapping
sbyte	No mapping (throws exception)
object	No mapping

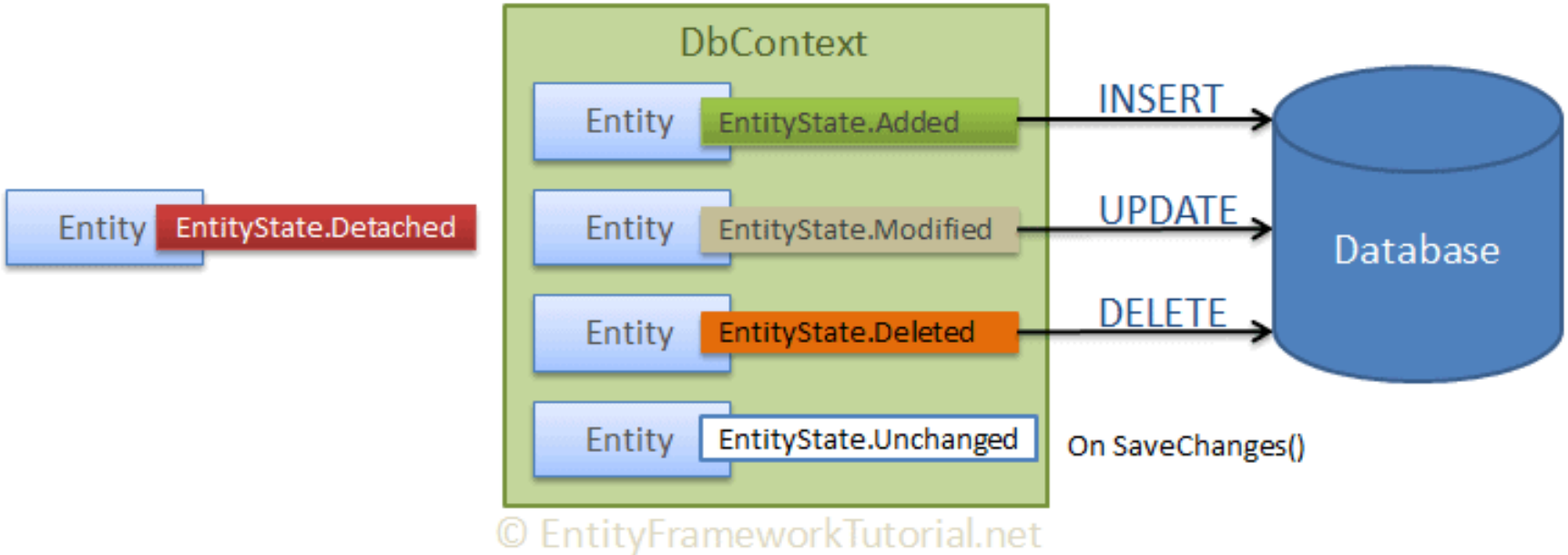
Contexto

- EFCore segue os padrões Data Mapper, Repository e Unit of Work
- Objeto **DbContext** é baseado nesses padrões e possui uma API para
  - Gerenciar objetos em memória (inclusive com cache)
  - Manter a ligação entre o banco de dados e as entidades mapeadas no modelo relacional
  - Gerenciar a conexão com a base de dados
  - Gerenciar o contexto transacional

Contexto

- Objeto **DbSet**
  - Representa uma coleção de entidades em um contexto de persistência
  - É obtida a partir do *DbContext*
  - Provê métodos para operações CRUD sobre um determinado tipo de entidade

Contexto


DbContext
Entity
EntityState.Detached
Entity
EntityState.Added
Entity
EntityState.Modified
Entity
EntityState.Deleted
Entity
EntityState.Unchanged
INSERT
UPDATE
DELETE
Database
On SaveChanges()
© EntityFrameworkTutorial.net

Database-First

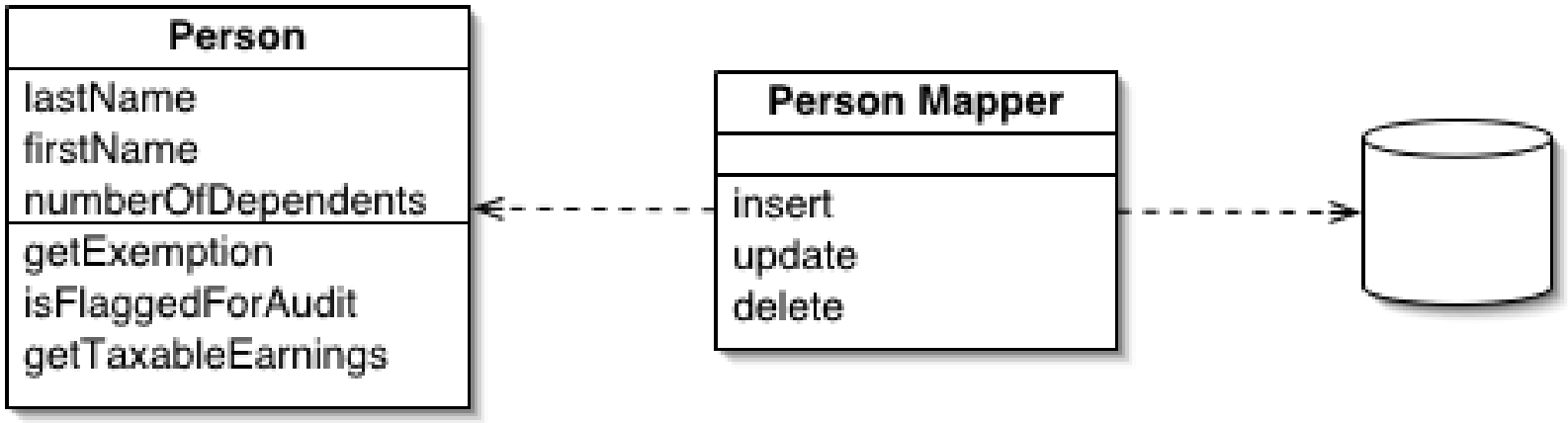
- Banco de dados está previamente criado e deve ser acessado via EFCore
- Pode-se utilizar ferramentas de engenharia-reversa via *scaffolding* para automatizar a criação da classes e *DbContext*
- Documentação:
  - https://docs.microsoft.com/en-us/ef/core/managing-schemas/scaffolding

Code-First

- Banco de dados é gerado a partir do código e anotações das classes em projetos
- Um framework de migração (Migrations) atualiza o banco a partir de alterações no modelo
- Documentação:
  - https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/

PADRÕES DE PROJETO

Data Mapper Pattern



The Data Mapper is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and also to isolate them from each other. With Data Mapper the in-memory objects needn't know even that there's a database present; they need no SQL interface code, and certainly no knowledge of the database schema.

http://martinfowler.com/eaaCatalog/dataMapper.html

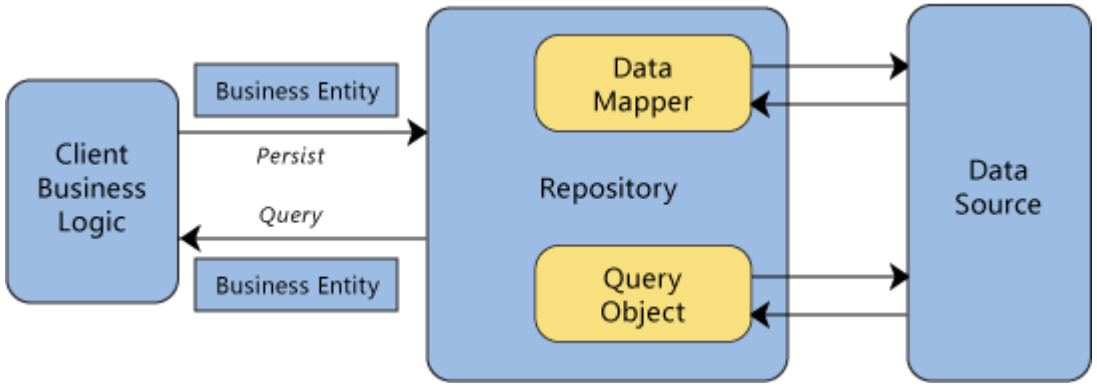
Repository Pattern

A system with a complex domain model often benefits from a layer, such as the one provided by Data Mapper (165), that isolates domain objects from details of the database access code. In such systems it can be worthwhile to build another layer of abstraction over the mapping layer where query construction code is concentrated. This becomes more important when there are a large number of domain classes or heavy querying. In these cases particularly, adding this layer helps minimize duplicate query logic.

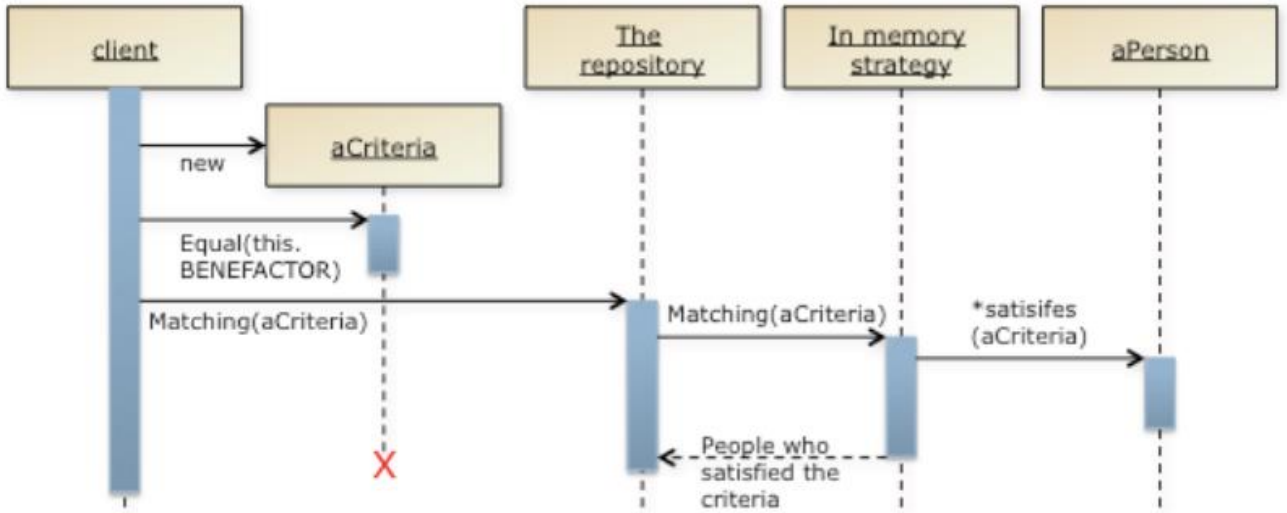
Data Mapper is more appropriate for systems with complex domain logic where the shape of the domain model will diverge considerably from the database model. Data Mapper also decouples your domain model classes from the persistence store. That might be important for cases where you need to reuse the domain model with different database engines, schemas, or even different storage mechanisms altogether

http://martinfowler.com/eaaCatalog/repository.html

Repository Pattern



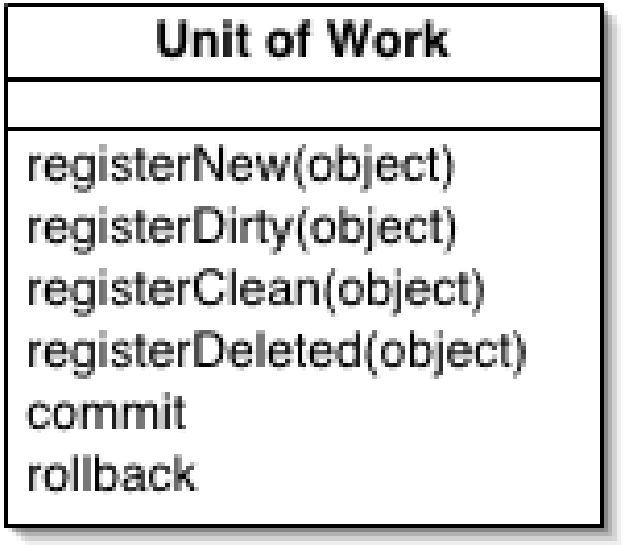
http://msdn.microsoft.com/en-us/library/ff649690.aspx



http://martinfowler.com/eaaCatalog/repository.html

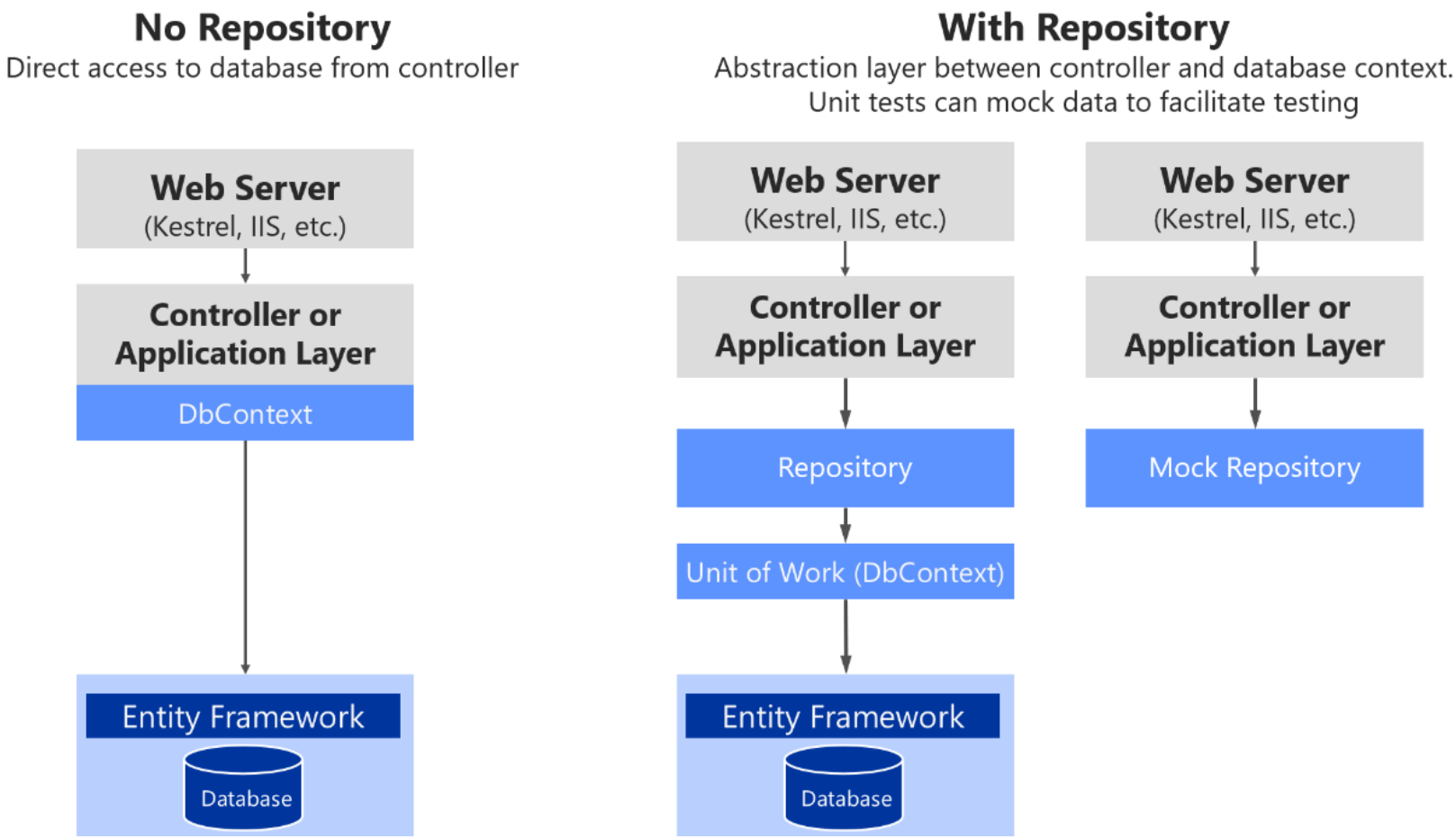
Unit of Work Pattern

- *Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems.*



https://martinfowler.com/eaaCatalog/unitOfWork.html

Unit of Work Pattern



https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implementation-entity-framework-core