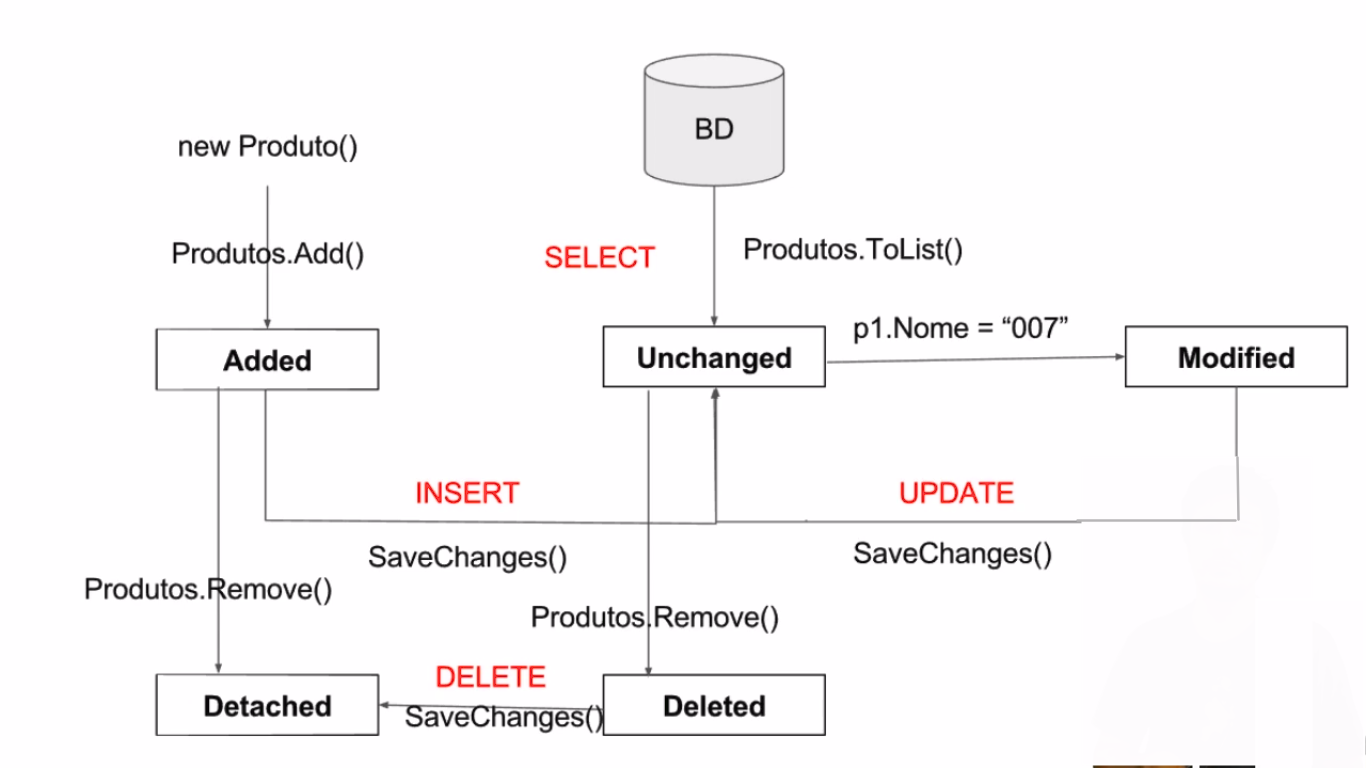
Entity Framework Core: Banco de dados de forma eficiente

* Introdução ao Entity Framework Core
  + Quais das opções abaixo são vantajosas quando estamos trabalhando com um Mapeamento Objeto Relacional (ORM)?
    - A maior produtividade do desenvolvedor, pois uma vez que ele fica fluente com o ORM que ele desejar usar, ele ganha muito mais tempo para focar em outras partes da aplicação e não precisa se dedicar tanto aos detalhes da persistência tradicional de dados.
    - A maior facilidade de quando refatoramos um código, pois evitamos de ter que fazer toda a parte de conversão de objetos, montar parâmetros e tudo relacionado a montar uma query SQL padrão na mão.
    - A possibilidade de não se preocupar em ter que escrever SQL, pois o ORM se encarrega na maior parte disto.
  + Database Provider
    - <https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>
  + Selecione todos os passos abaixo que são necessários para podermos utilizar o Entity Framework Core em nossos projetos:
    - Informar no evento de configuração do contexto o nome do banco e sua localização.
    - Criar as propriedades no contexto para dizer quais classes serão persistidas.
    - Criar um contexto próprio que herda da classe DbContext.
    - Instalar o Entity Framework Core através do gerenciador de pacotes Nugget (*Microsoft.EntityFrameworkCore*).
  + Entity suporta transações?
    - <https://docs.microsoft.com/pt-br/ef/ef6/saving/transactions?redirectedfrom=MSDN>
* Como o entity monta o SQL?
  + Então, como o ChangeTracker sabe que, quando uma propriedade foi alterada, ele deve fazer um UPDATE no banco?
    - O Entity guarda um snapshot dos valores dos objetos por padrão. Quando aquele objeto começa a ser monitorado pelo Entity, seja através de métodos que recuperam objetos do banco via SELECT (por exemplo ToList, First, Find, etc.), seja através do método Entry que cria uma entrada no ChangeTracker para o objeto passado como argumento do método.



* + Added
    - O objeto é novo, foi adicionado ao contexto, e o método SaveChanges ainda não foi executado. Depois que as mudanças são salvas, o estado do objeto muda para Unchanged. Objetos no estado Added não têm seus valores rastreados em sua instância de EntityEntry.
  + Deleted
    - O objeto foi excluído do contexto. Depois que as mudanças foram salvas, seu estado muda para Detached.
  + Detached
    - O objeto existe, mas não está sendo monitorado. Uma entidade fica nesse estado imediatamente após ter sido criada e antes de ser adicionada ao contexto. Ela também fica nesse estado depois que foi removida do contexto através do método Detach ou se é carregada por um método com opção NoTracking. Não existem instâncias de EntityEntry associadas a objetos com esse estado.
  + Modified
    - Uma das propriedades escalares do objeto foi modificada e o método SaveChanges ainda não foi executado. Quando o monitoramento automático de mudanças está desligado, o estado é alterado para Modified apenas quando o método DetectChanges é chamado. Quando as mudanças são salvas, o estado do objeto muda para Unchanged.
  + Unchanged
    - O objeto não foi modificado desde que foi anexado ao contexto ou desde a última vez que o método SaveChanges foi chamado.
* Sincronizando o banco de dados com o modelo de classes
  + Migrations
    - Drop-Database
      * Esse comando é utilizado para dropar o banco de dados apontado pelo contexto.
    - Remove-Migration
      * Esse comando é utilizado para remover a última migração não aplicada no banco de dados apontado pelo contexto.
    - Scaffold-DbContext
      * Esse comando é utilizado para criar uma classe que estende de **DbContext**, além de classes que representam as tabelas do banco.
  + Para descobrir quais comandos estão disponíveis no recurso de Migrations, use este:
    - **Get-Help EntityFramework**
* Relacionamento **Um** para **Um** no **Entity**
  + Shadow Properties