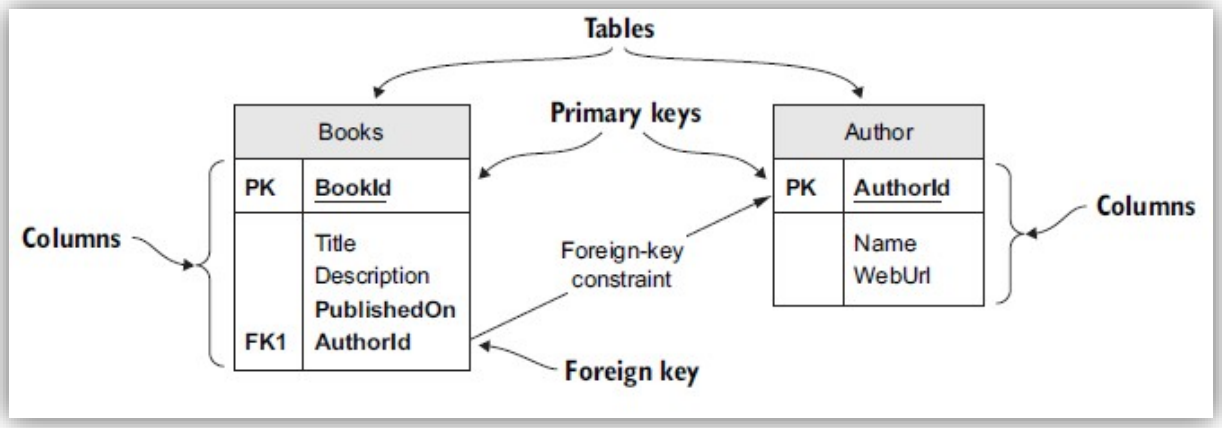
Entity Framework Core – Hoe werkt het ?

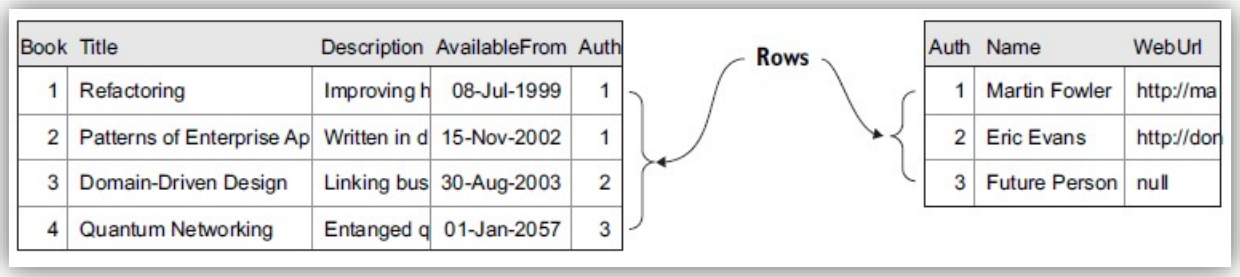
Entity Framework (EF) is een Object Relational Mapper (ORM) die programmeurs toelaat om op basis van .NET objecten te werken met een databank.

Modelleren

Aan de hand van een voorbeeld zullen we de werking van EF verduidelijken. Laten we starten met een eenvoudig relationeel databank model dat bestaat uit twee tabellen, een tabel voor boeken en een tabel voor auteurs. Een boek wordt uniek geïdentificeerd door middel van een Id (BookId) en dat is in de databank gemodelleerd door middel van een Primary Key (PK). Ook auteurs worden uniek geïdentificeerd door middel van een sleutel (PK), namelijk de AuthorId. Het verband tussen beiden is gelegd via een Foreign-key constraint die de AuthorId uit de Books tabel koppelt aan de AuthorId uit de Author tabel. Verder bevatten beide tabellen nog een aantal kolommen voor de attributen.

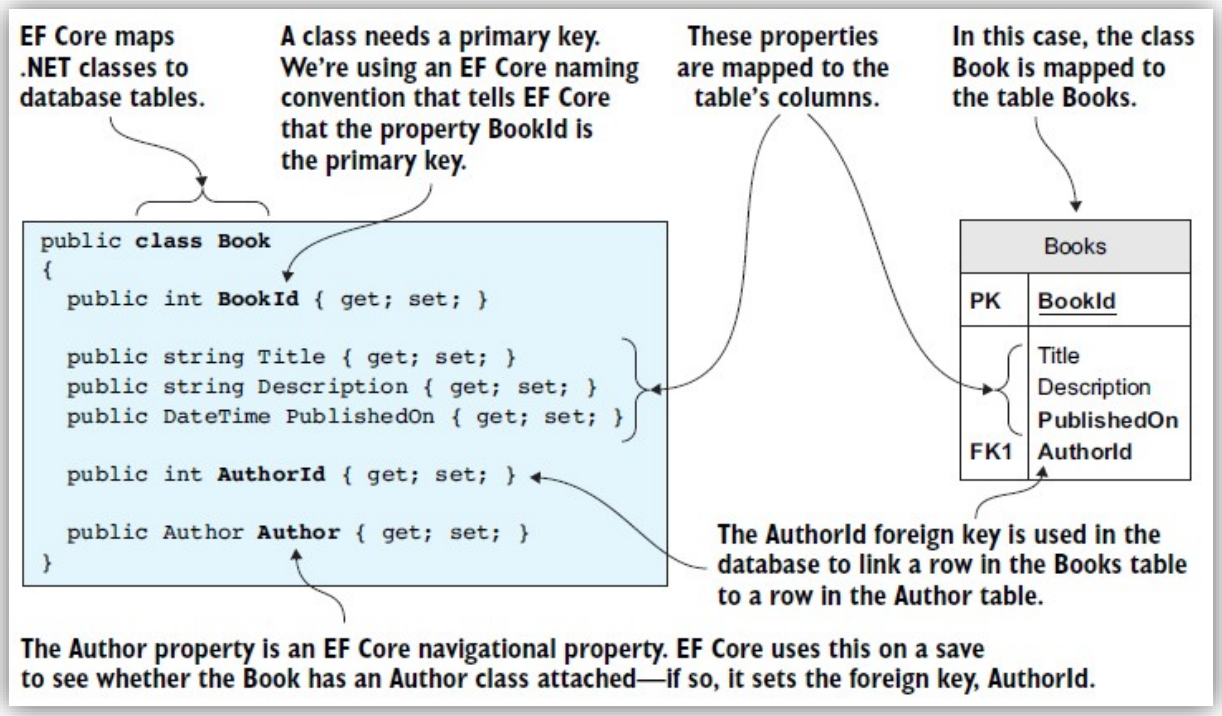


Een voorbeeld ziet er dan als volgt uit, waarbij de rijen uit de verschillende tabellen gekoppeld worden op basis van de AuthorId.

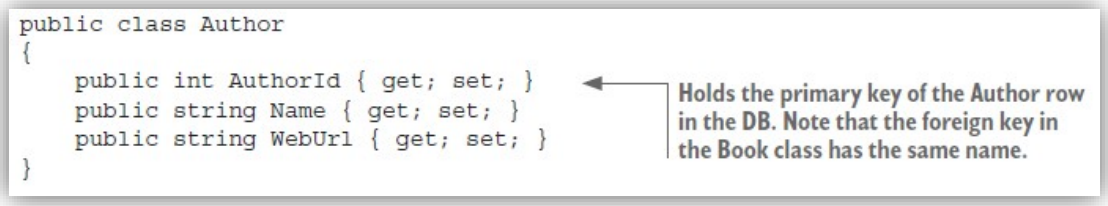


In C# werken we echter niet met tabellen, maar met klassen en objecten. In de volgende figuur zien we het verband tussen de klasse Book en de tabel Books in de databank. Bij het mappen van een klasse naar een tabel moeten we een primaire sleutel (PK) definiëren en dat kan op verschillende manieren. Hier in het voorbeeld maken we gebruik van EF Core Naming Convention die stelt dat een property met als naam de klasse, gevolgd door Id (BookId) automatisch als PK wordt gezien. De andere

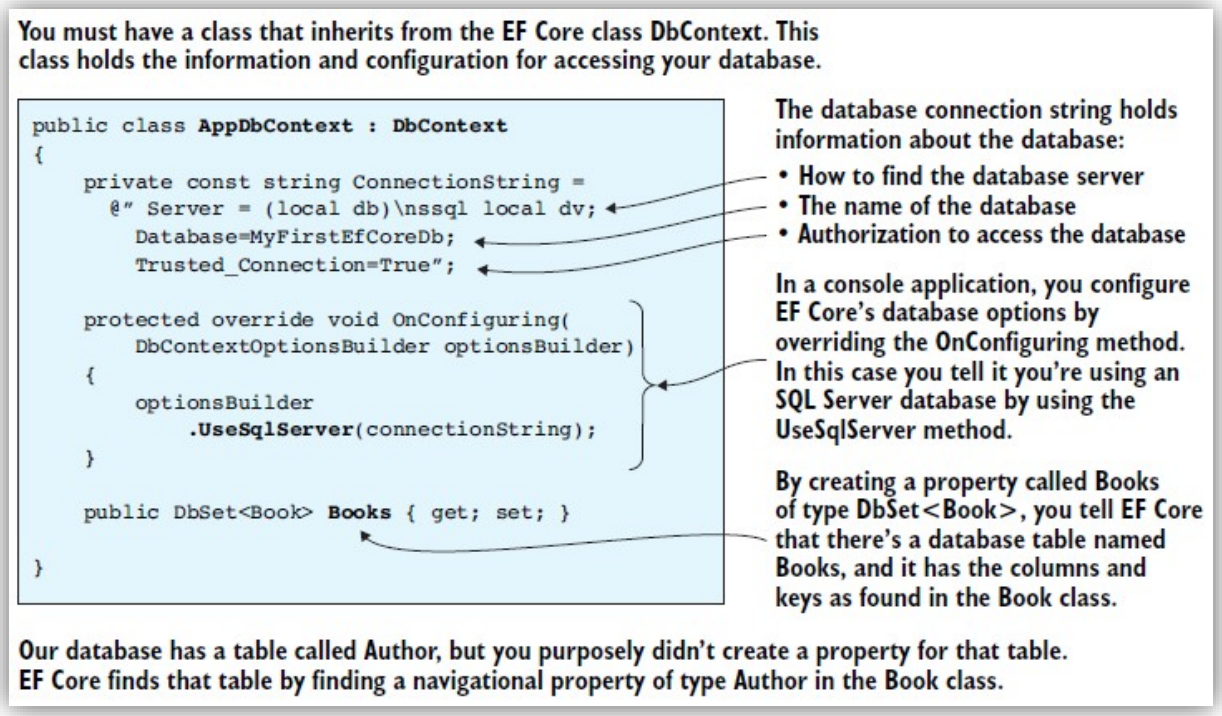
attributen worden gekoppeld aan de kolommen in de tabel. En voor de link met het object Author wordt er gebruik gemaakt van de Foreign Key (FK) constraint. Het object Author noemen we een navigational property (een property om dus naar een andere klasse/object te gaan).



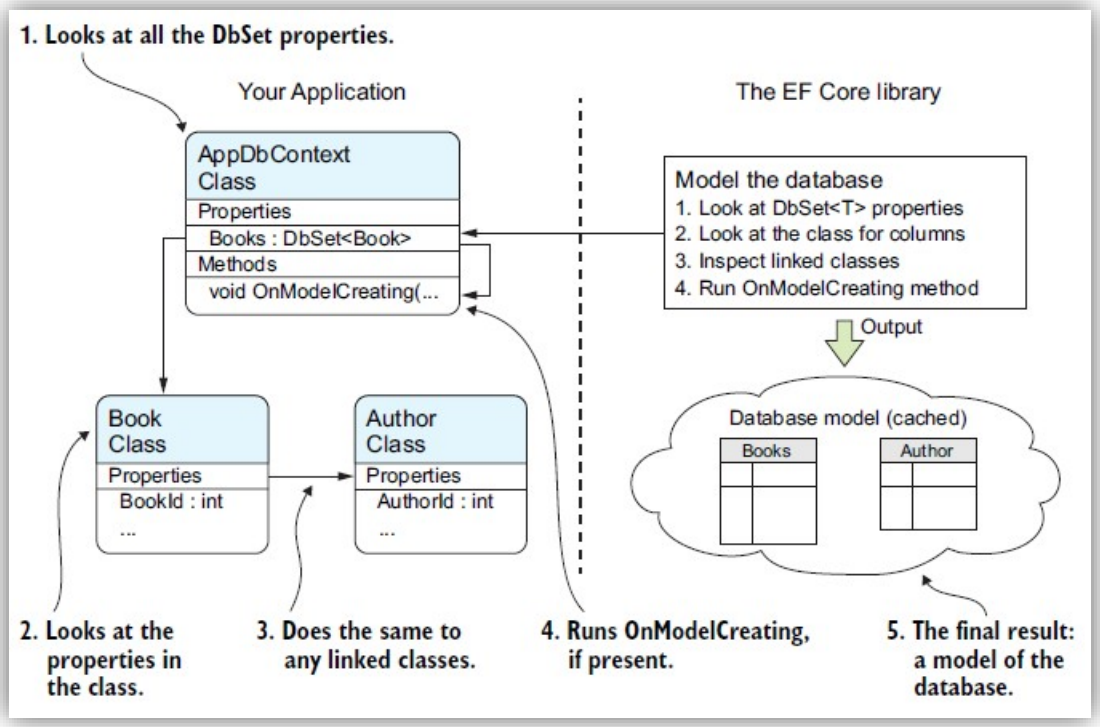
De klasse Author heeft als PK AuthorID die eveneens automatisch is toe gekent door EF.



De toegang tot de databank wordt geregeld door middel van de klasse AppDbContext die overerft van de klasse DbContext. Door middel van de property DbSet<Book> Books leggen we het verband met de tabel Books in de databank. Het type <Book> dat we meegeven aan DbSet definieert het verband tussen de attributen van de klasse en de kolommen uit de tabel (zoals eerder beschreven). Door middel van de methode OnConfiguration kunnen we configuratie info definiëren, in dit voorbeeld beperken we ons door aan te geven dat we met SQL Server gaan werken (optionBuilder.UseSqlServer) en geven we ook de connectiestring mee.

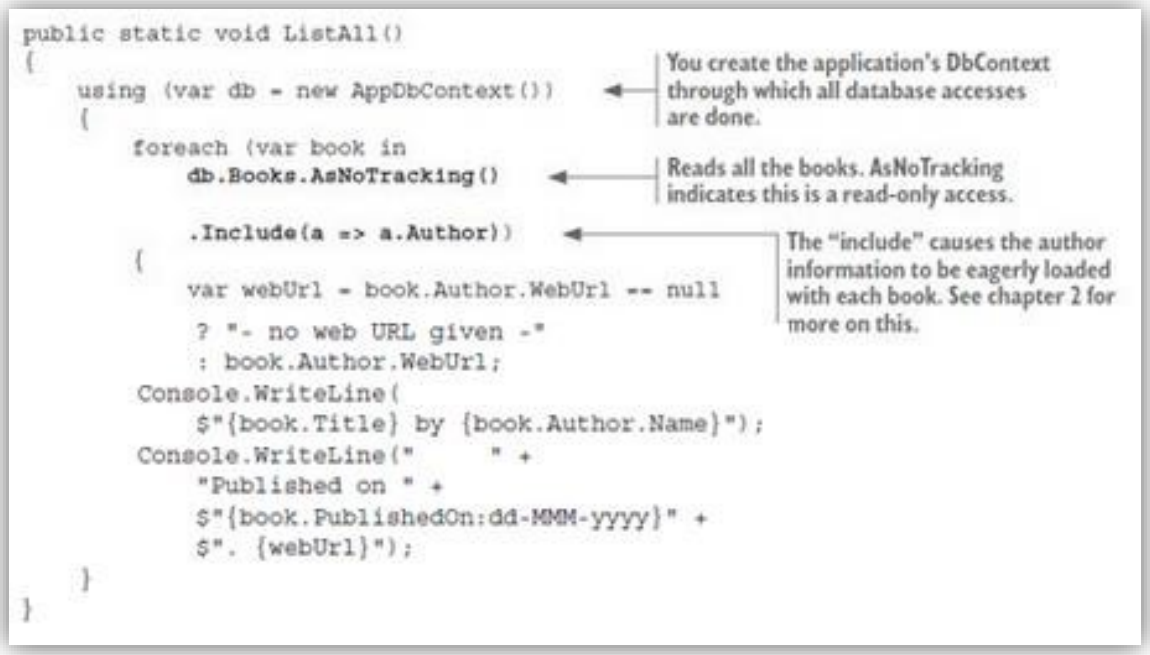


Van zodra de klasse AppDbContext wordt aangemaakt start er een modelleringsproces zoals beschreven in de volgende figuur. EF core bekijkt de DbSet<> properties en linkt deze aan tabellen in de databank, analyseert de klassen om de link te leggen met de kolommen, analyseert de verbanden tussen de klassen (navigational properties) en voert uiteindelijk de OnModelCreating methode uit wat uiteindelijk resulteert in een mlodel van de databank.

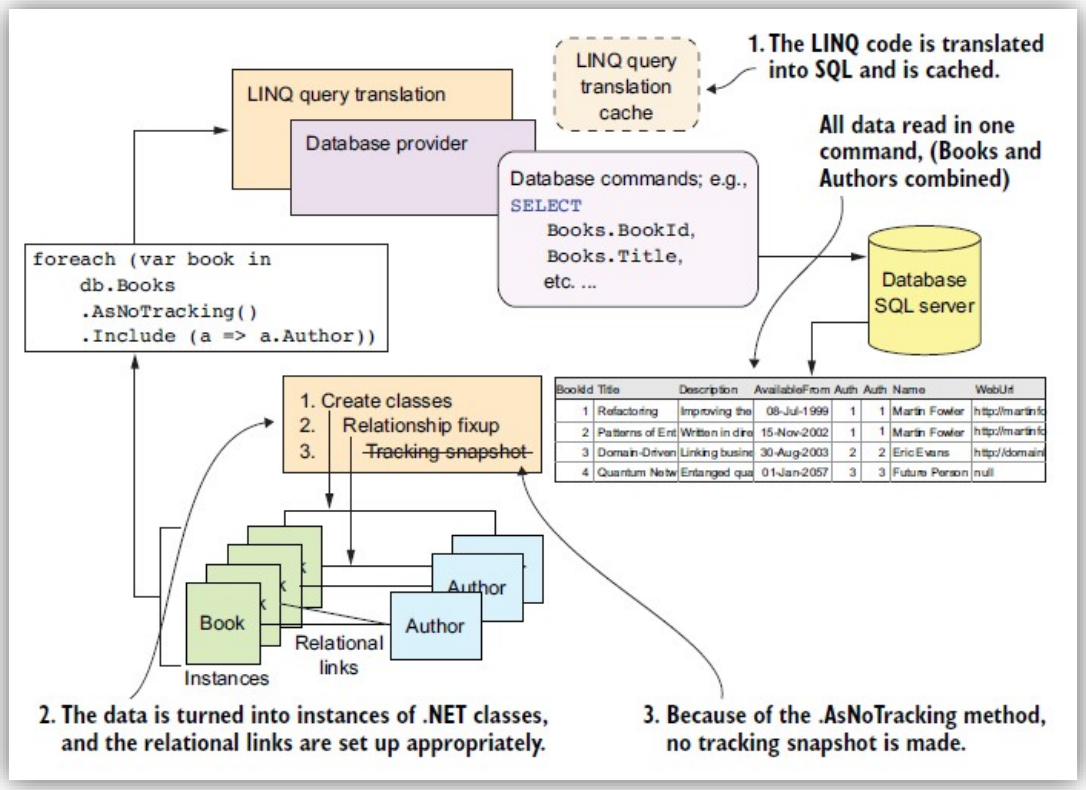


Gebruik

Eénmaal de link gelegd tussen de klassen en het databankmodel kunnen we aan de slag. Het volgende codefragment geeft weer hoe we gegevens uit de databank kunnen opvragen. We starten met het maken van een AppDbContext object waarmee we alle toegang tot de databank wordt geregeld. Het benaderen van een tabel gebeurt door middel van de DbSets, in dit voorbeeld Books. Indien we enkel gegevens wensen te lezen dan gebruiken we de methode AsNoTracking, die ervoor zorgt dat de objecten niet worden getraceerd (later hierover meer). Om ook de Author objecten op te vragen die bij de boeken horen moeten we dit expliciet meegeven en dat kan door de methode Include.

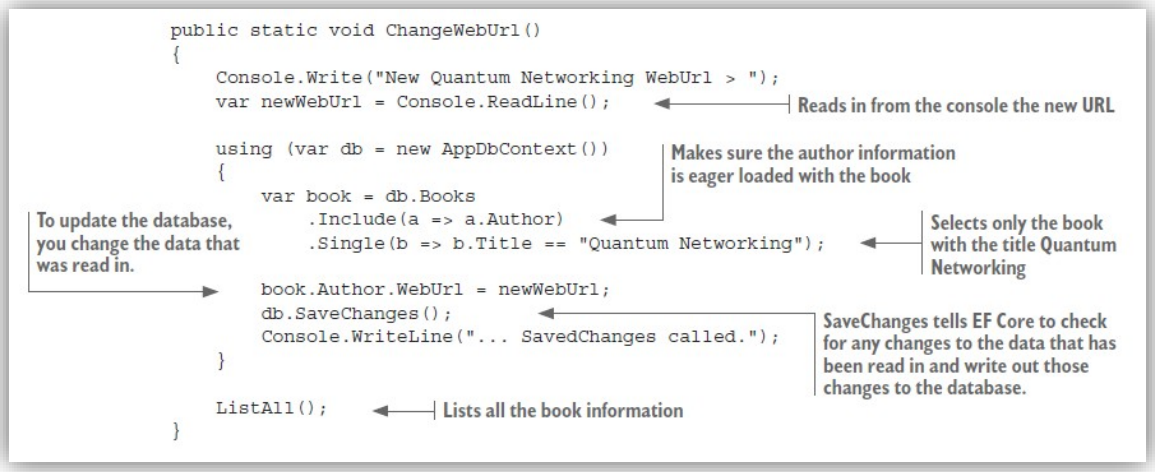


Het query proces wordt in de volgende figuur nog eens schematisch voorgesteld. We starten met een LINQ query die wordt omgezet in een SQL statement. Dit SQL statement wordt uitgevoerd in de databank en het resultaat wordt als een 'tabel' terug gegeven. Deze resultaten worden nu vertaald naar de .NET klassen, waarbij ook de relaties worden opgebouwd. Indien de AsNoTracking methode werd gebruikt zal er geen snapshot worden gemaakt van de opgevraagde objecten. Het resultaat is dan een collectie van objecten.

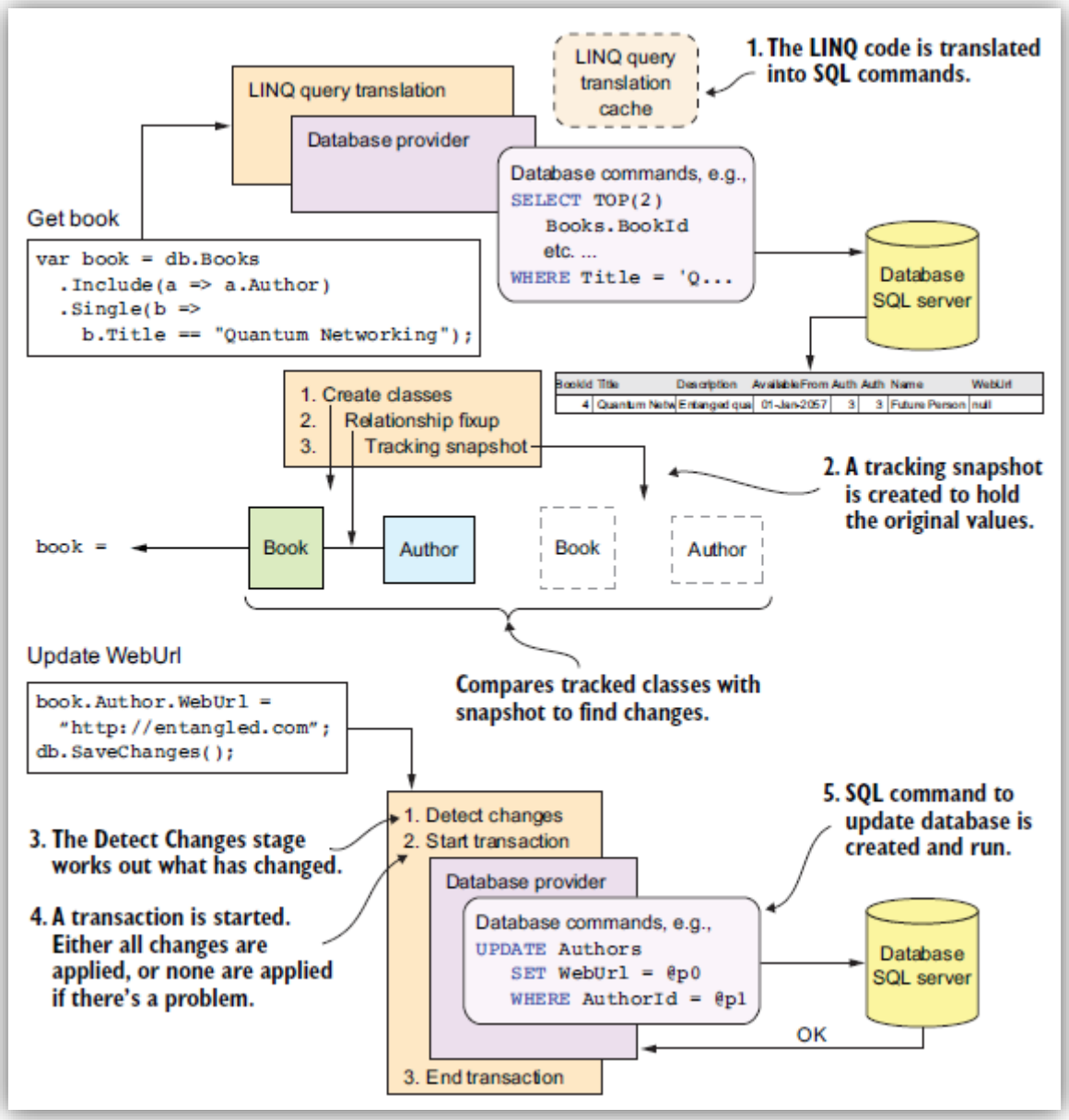


Wanneer we objecten wensen up te daten dan zullen we wel gebruik maken van het trackingsysteem van EF. Van elk opgevraagd object wordt de toestand bijgehouden in de ”context”, dit kan doordat er een snapshot wordt gemaakt van de objecten op het moment dat ze worden opgevraagd. Elke verandering die we nu doen op één van deze objecten wordt gedetecteerd door het trackingsysteem waardoor EF weet wat er is veranderd en de nodige update queries kan opbouwen en uitvoeren.

In het volgende voorbeeld vragen we één specifiek boek op, namelijk het boek met als titel ‘Quantum Networking’. Door middel van de Include-methode hebben we ook de info van de auteur. Daarna veranderen we het attribuut WebUrl van het object Author, deze verandering wordt door EF gedetecteerd en bijgehouden. Wanneer we nu de methode SaveChanges die bij onze context hoort (overgeerfd van DbContext) uitvoeren worden alle veranderingen doorgevoerd naar de databank.



In de volgende figuur wordt het proces nog eens duidelijk geschetst. We starten met een LINQ query om een specifiek boek op te vragen. Van zodra de gegevens uit de databank worden omgezet naar de .NET objecten wordt er ook een snapshot gemaakt die de originele waarden bijhoudt. Na het aanpassen van het object en het oproepen van de functie SaveChanges starten de volgende handelingen. Eerst worden de veranderingen gedetecteerd door te vergelijken met de waarden in het snapshot. Daarna wordt er een transactie gestart met daarin de update statements en uiteindelijk wordt de transactie beëindigd. Aangezien er met transacties wordt gewerkt, zullen ofwel alle aanpassingen door worden gevoerd of geen enkele.



Opmerking : alle figuren komen uit het boek 'Entity Framework Core In Action'.

Referenties

Entity Framework Core In Action, Jon P. Smith, Manning

https://blog.oneunicorn.com/2012/03/10/secrets-of-detectchanges-part-1-what-does-detectchanges-do/