



Odisee
DE CO-HOGESCHOOL

Entity Framework



Jens Baetens



Databank

Databanken

- ▣ Verzameling van gegevens die in een bepaalde structuur staan
- ▣ Voorbeelden:
 - SQL Server, MariaDB, MySQL, MongoDB (NoSQL)
- ▣ In deze cursus maken we gebruik van een RDBMS
 - Relational Database Management System
 - Tabellen, Kolommen, Rijen
 - SQL

SQL - Select

- ▣ SELECT columnName FROM tableName WHERE condition GROUP BY columnName HAVING condition ORDER BY columnName ;



SQL - Insert

▣ INSERT INTO tableName (columnName) VALUES (value);





SQL - Delete

- ▣ DELETE FROM tableName WHERE condition;





SQL - Update

▣ UPDATE tableName SET columnName = value WHERE condition;





Databanken in WPF/C#/.NET Framework

Databanken in Visual Studio

- Ga naar Tools > SQL Server > New Query
 - Indien dit niet lukt: Open de visual studio installer en controleer of het volgende geïnstalleerd is



Databank in VS

- ▣ Kies een server => MSSQLLOCALDB
- ▣ Maak een databank
 - CREATE DATABASE demo;
- ▣ Controleer in SQL Server Object Browser of de databank bestaat

Databanken aanspreken in C#

▣ ADO.NET

- ActiveX Data Objects
- Framework om queries aan te maken en te sturen naar RDBMS
- Zelf queries schrijven



ADO.NET

0 references

```
private List<string> searchUser(string name)
{
    List<string> names = new List<string>();
    SqlConnection connection = new SqlConnection("data source=(localdb)\\MSSQLLOCALDB; initial catalog=demo");
    SqlCommand command = new SqlCommand($"Select * from dbo.authors where first_name = '{name}';", connection);

    connection.Open();
    SqlDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
        names.Add($"{reader[1]}");
    }
    reader.Close();
    connection.Close();

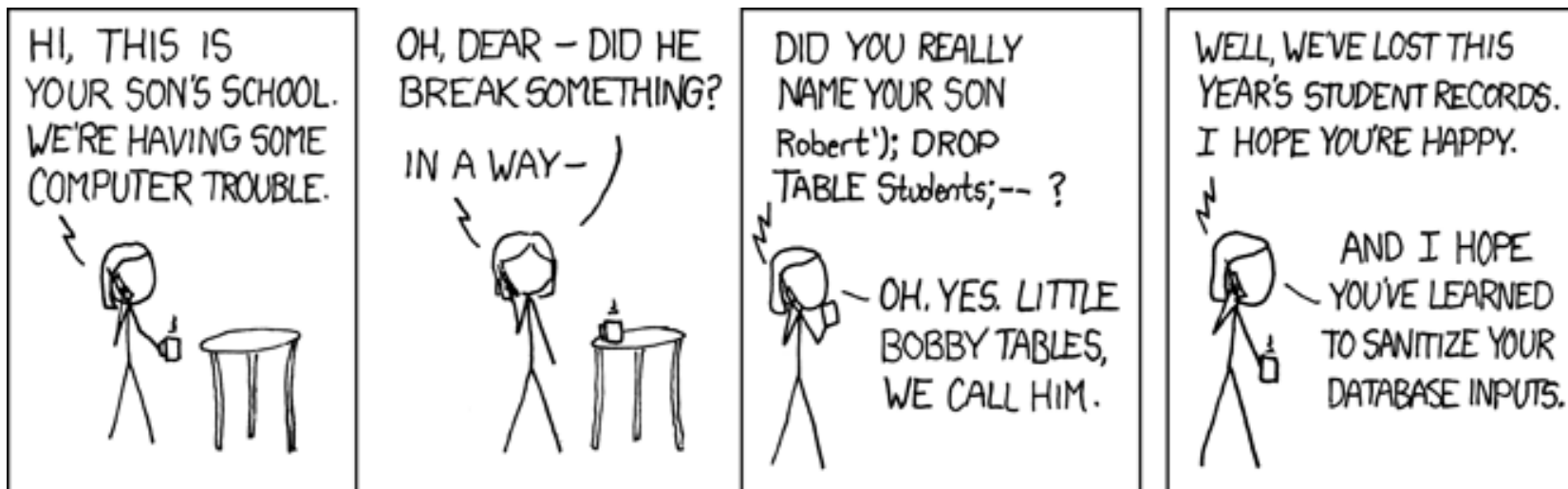
    return names;
}
```

Welk probleem kan hier voorvallen?

SQL injection

```
Program.searchUser("Tom'; Drop Table dbo.authors; --");
```

- Aanmaken queries is een kwetsbaarheid
- Technieken om je hiertegen te wapenen zijn
 - SQLParameters
 - Placeholders
- Gevaar is hiermee niet weg dus moet je er steeds opletten



▣ Nadelen

- Foutgevoelig
 - Geen compilatiefouten maar runtime fouten
- Gevoelig voor SQL injection
- Veel repetitief werk
 - Steeds aanmaken van de connectie, queries, ...
 - Veel overhead



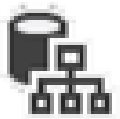
Entity Framework

Entity Framework

- ▣ ORM framework: Object Relational Mapping
 - SQL tabellen en relaties matchen met objecten in de code
- ▣ Geen SQL injection mogelijk
 - Wordt voor ons gecontroleerd
- ▣ Minder repetitief werk
 - De volledige connectie met de database wordt voor ons gedaan
- ▣ Zorg ervoor dat het NugetPackage voor EntityFramework geïnstalleerd is

Entity Framework

- ▣ 4 manieren om met dit framework te werken in .Net Framework



EF Designer
from
database



Empty EF
Designer
model



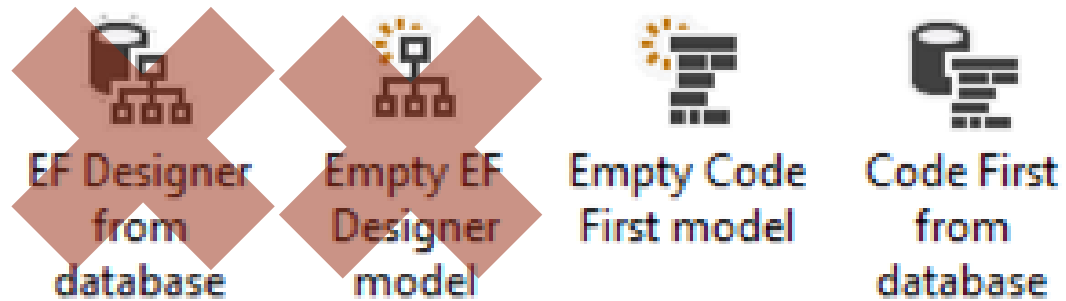
Empty Code
First model



Code First
from
database

Entity Framework

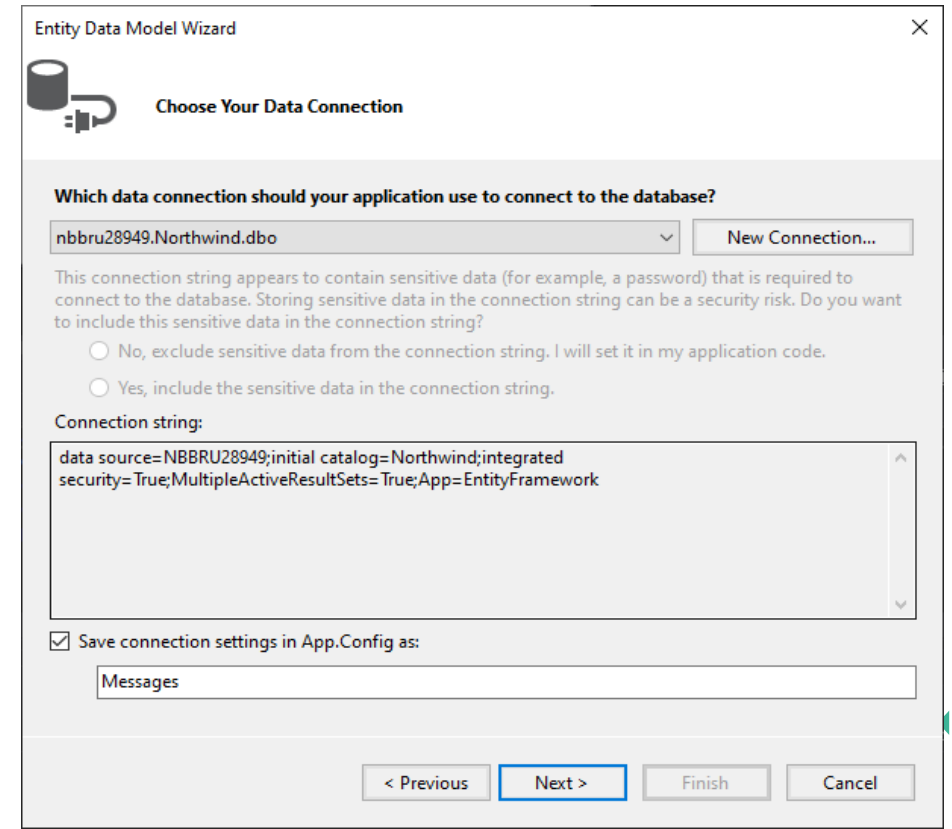
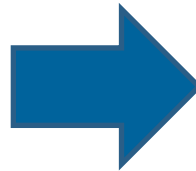
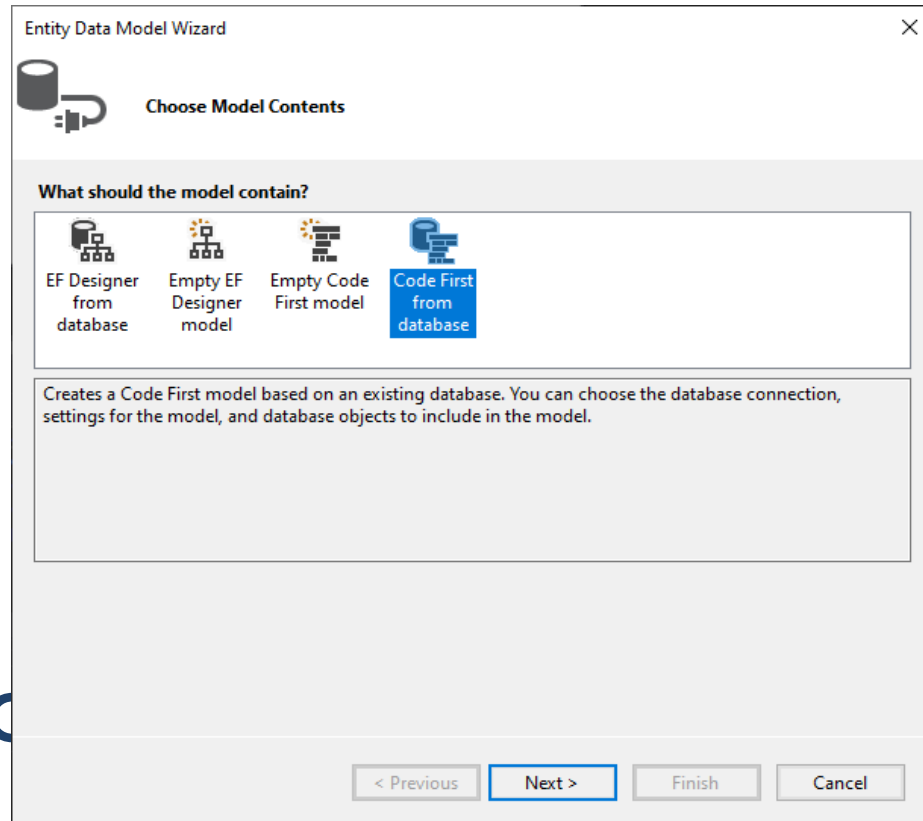
- ▣ 4 manieren om met dit framework te werken in .Net Framework



- ▣ In .Net core zijn er echter twee minder

Code first from database

- Eerst database aanmaken
- Daarna code genereren op basis van de database



Code first from database

■ Maak indien nodig een nieuwe connectie aan

Kies de servernaam
Op je lokale machine normaal gezien
(localdb)\MSSQLLOCALDB.
Deze wordt niet altijd automatisch herkend

Kies de database die je wil gebruiken.

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient) Change...

Server name:
(localdb)\MSSQLLOCALDB Refresh

Log on to the server

Authentication: Windows Authentication

User name:

Password:

☐ Save my password

Connect to a database

☒ Select or enter a database name:
Messages

☐ Attach a database file:
 Browse...

Logical name:


Advanced...

Test Connection OK Cancel


Code First from database


- ▣ Kies de te overnemen tabellen
- ▣ Als dit goed gaat, worden een aantal klassen aangemaakt

Entity Data Model Wizard

 Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

☒  Tables

☐  Views

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

☐ Import selected stored procedures and functions into the entity model

< Previous Next > **Finish** Cancel

Code First without database

- ▣ Wanneer er nog geen database is
 - ▣ Database wordt gemaakt op basis van de klassen
 - ▣ Heeft in dit vak de voorkeur
-
- ▣ Er is ook een wizard hiervoor maar het is zo eenvoudig dat het niet nodig is

Code First without database

- ▣ Volgende stappen gaan we uitvoeren
 - NuGet Packages installeren
 - DbContext aanmaken
 - connectionString instellen
 - Aanmaken aantal dataklassen met
 - Default constructor
 - Public property met getter en setter voor elk field/member in je dataklasse



Code first without databases – NuGet Packages

- ▣ .Net Framework applicatie
 - EntityFramework
- ▣ .Net Core
 - EntityFrameworkCore
 - EntityFrameworkCore.SqlServer
 - EntityFrameworkCore.Tools



Code first without databases – DbContext

- ▣ Deze klasse spreekt de database aan via het entity framework
- ▣ Maak een klasse aan die ervan overerft

```
0 references  
class UserContext: DbContext  
{  
    ...  
}
```

Code first without databases – Connectionstring

- Geef in de DbContext – klasse aan met welke database er moet gecommuniceerd worden
 - ▬ Via connectionstring
 - Waar? (Ip-adres)
 - Hoe connecteren we (Username, pass, ...)
 - Welke taal gebruikt de database
 - Hoe noemt de database
- De structuur van deze string is afhankelijk van de data provider
 - ▬ SQL: "Data Source=TestServer;Initial Catalog=Pubs;User ID=Dan;Password=training"

Code first without databases – Connectionstring

- ▣ "Data Source=TestServer;Initial Catalog=Pubs;User ID=Dan;Password=training"
 - ▬ Datasource
 - Server waarop de databank draait
 - ▬ Initial catalog
 - Databank op de server die we gebruiken
 - ▬ User ID
 - Username om in te loggen (enkel nodig indien ze beveiligd is)
 - ▬ Password
 - Password om in te loggen (enkel nodig indien ze beveiligd is)

Code first without databases – Connectionstring

- ▣ Kan ingesteld worden in de `app.config` file
- ▣ Xml file om configuraties in te stellen

```
<configuration>
  <entityFramework>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer"/>
    </providers>
  </entityFramework>
  <connectionStrings>
    <add name="users" connectionString="data source = (localdb)\MSSQLLOCALDB; initial catalog=users;" providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```



Name om later de connectionstring terug te vinden



Connectionstring met alle info



Providername, deze geeft aan welk type db we gebruiken

Code first without databases – Connectionstring

- Geef de connectionstring mee in de constructor van de DbContext klasse

```
1 reference
class UserContext: DbContext
{
    0 references
    public UserContext(): base("users")
    {
        // users is de naam van de connectionstring, see app.config
    }
}
```

Code first without databases – User dataklasse

- ▣ DbContext ≈ Database
- ▣ Dataklassen ≈ Tabellen (Type DbSet<T>)
- ▣ Properties in deze klassen ≈ kolommen

```
1 reference
class UserContext: DbContext
{
    0 references
    public UserContext(): base("users")
    {
        // users is de naam van de connectionstring, see app.config
    }

    0 references
    public DbSet<User> users { get; set; }
}
```

```
1 reference
class User
{
    private string firstName;
    private string lastName;
    private int id;

    0 references
    public string FirstName { get { return firstName; } set { firstName = value; } }
    0 references
    public string LastName { get { return lastName; } set { lastName = value; } }
    0 references
    public int Id { get { return id; } set { id = value; } }
}
```

Code first without databases – User dataklasse

- ▣ DbSet<T> is een soort lijst
 - Elk item in de lijst stelt een rij of record voor
 - Elke DbSet wordt een tabel
 - Kolommen bepaalt door de properties in T
 - Elke rij komt overeen met een object

Code first without databases – User dataklasse

- ▣ In het vak data en informatieverwerking gezien dat een tabel een aantal **constrains** kan hebben
 - ▬ Primary key
 - ▬ Not Null
 - ▬ Foreign Key
- ▣ 2 manieren om dit toe te voegen
 - ▬ Data annotaties voor de eenvoudigere zaken
 - ▬ Fluent API voor complexere zaken

Data annotaties

- ▣ [Key]
 - ▬ Primary key (verplicht)
- ▣ [Column("blog_id")]
 - ▬ Andere naam dan property
- ▣ [Column(TypeName="varchar(200)")]
 - ▬ Bepaald type in de databank
- ▣ [MaxLength(500)]
 - ▬ Max lengte van string
- ▣ [Required]
 - ▬ Not null
- ▣ [NotMapped]
 - ▬ By default krijgen alle public properties een veld in de databank



Fluent API

- ▣ Dit wordt toegevoegd in de (te overriden) OnModelCreating methode
 - In DbContext
 - Meestal niet nodig maar hier kan alles specifiek ingesteld worden

Fluent API - examples

De tabel waarop we
wijzigingen willen doen.

Instellen van een
primary key bestaande
uit 2 velden

0 references

```
protected void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>()
        .HasKey(user => new {user.FirstName, user.LastName});

    modelBuilder.Entity<User>()
        .HasMany<Car>(user => user.Cars)
        .WithMany(car => car.Users);

    modelBuilder.Entity<User>()
        .HasOptional<Car>(user => user.FavoriteCar);
}
```

Instellen van een
veel op veel
relatie

Instellen van een
optionele waarde
FavoriteCar

Fluent API - examples

```
modelBuilder.Entity<Car>().Property(car => car.Model)
    .HasMaxLength(50)
    .IsOptional()
    .HasColumnName("reeks");
```

Instellen van een maximale
lengte

Instellen dat kolom model
optioneel is

Instellen dat we de naam
reeks gebruiken in de
database

DbContext - Recap

- ▣ EntityFramework packages nodig
- ▣ Connection String
 - Welke server, database, credentials
- ▣ DbContext \approx Database
- ▣ DbSet \approx Tabel
- ▣ DbSetItems \approx Rijen
- ▣ Properties \approx kolommen
- ▣ Constraints
 - Data annotations
 - Fluent API

Database aanspreken - Read

- ▣ Via DbContext en DbSet kan de databank aangesproken worden

```
UserContext context = new UserContext();  
return context.Users.ToList();
```

- ▣ De DbSet is een IEnumerable dus alle LINQ queries kunnen gebruikt worden

Database aanspreken - Create

- ▣ DbContext.<DbSet>.Add()
- ▣ Alle niet-nullable velden moeten een waarde hebben
 - ▬ Anders -> Exception
 - ▬ Vergeet niet de aanpassingen te bewaren met SaveChanges

```
UserContext userContext = new UserContext();
```

```
User user = new User();  
user.FirstName = "Jan";  
user.LastName = "Jansens";
```

```
userContext.Users.Add(user);  
userContext.SaveChanges();
```


Database aanspreken - Delete

- ▣ DbContext.<DbSet>.Remove()
- ▣ Let op cascading delete behaviour (via foreign keys)
 - ▬ Error wordt gethrowed indien het niet lukt
 - ▬ Kan ingesteld worden via de fluent api
 - ▬ Vergeet ook hier niet SaveChanges uit te voeren!

```
UserContext context = new UserContext();  
  
User user = context.Users.First(u => u.FirstName == "Jan");  
context.Users.Remove(user);  
  
context.SaveChanges();
```

Database aanspreken - Update

- ▣ Gebeurt eenvoudig via DbContext.SaveChanges()
- ▣ Let op dat dit enkel werkt voor data opgehaald via de context

```
UserContext context = new UserContext();  
UserContext context2 = new UserContext();  
  
User user = context.Users.First(u => u.FirstName == "Jan");  
User user2 = context.Users.First(u => u.FirstName == "John");  
  
user.FirstName = "Harry";  
user2.FirstName = "Ron";  
  
context.SaveChanges();  
  
//which changes are added?
```

Database aanspreken - Update

- Wanneer je toch objecten van een andere oorsprong wilt aanpassen
 - ▬ Zet de state op modified

```
UserContext context = new UserContext();
UserContext context2 = new UserContext();

User user = context.Users.First(u => u.FirstName == "Jan");
User user2 = context2.Users.First(u => u.FirstName == "John");

user.FirstName = "Harry";
user2.FirstName = "Ron";
context.Entry(user2).State = EntityState.Modified;

context.SaveChanges();

//which changes are added?
```

Database aanspreken – Best practices

- Geen goede gewoonte om in veel klassen DbContext aan te spreken
- Verstop dit een Helper Class
 - Repository of DataAccess
 - Deze bestaat meestal uit CRUD-operaties
 - Create, Read, Update, Delete

```
2 references
class UserRepository
{
    private DbContext context = new DbContext();

    //create
    0 references
    public void AddUser(User user)
    {
        context.Users.Add(user);
        context.SaveChanges();
    }

    //read
    1 reference
    public List<User> GetAllUsers()
    {
        DbContext context = new DbContext();
        return context.Users.ToList();
    }

    //update
    0 references
    public void UpdateUser(User user)
    {
        // hier is niets nodig omdat het object reeds gelinkt is met de data
        context.SaveChanges();
    }

    //delete
    0 references
    public void DeleteUser(User user)
    {
        context.Users.Remove(user);
        context.SaveChanges();
    }
}
```

Migraties

- ▣ De eerste keer dat DbContext aangesproken wordt, wordt de database automatisch aangemaakt
- ▣ Wat als je datamodel wijzigt?
 - Dan moet er een database migratie gebeuren
 - Dit gebeurt niet automatisch
- ▣ Database migraties kunnen aangezet worden door in de package manager console een commando uit te voeren
 - Tools > NuGet Package Manager > Package Manager Console
 - Enable-Migrations



Lokale Migraties

- ▣ Voer het volgende uit in de package manager console:
 - ▢ Add-Migration
 - Geeft aan dat er een migratie is
 - ▢ Update-Database
 - Voert de migratie uit
- ▣ Dit werkt enkel lokaal. Met een remote database of een database op de pc van elke gebruiker kunnen deze commando's niet altijd uitgevoerd worden

Migraties in code

■ Add-Migration genereert een migratie-file

- ─ Migraties maak je op basis van je database en voorgaande migraties

```
2 references
public partial class Les : DbMigration
{
    0 references
    public override void Up()
    {
        AddColumn("dbo.Users", "BirthDate", c => c.DateTime());
    }

    0 references
    public override void Down()
    {
        DropColumn("dbo.Users", "BirthDate");
    }
}
```

■ Migratie in code uitvoeren

- ─ Voeg een lijn toe aan de constructor van DbContext
- ─ Geeft aan dat er gecontroleerd wordt op nieuwe migraties

```
9 references
public UserContext(): base("Users")
{
    Database.SetInitializer(new MigrateDatabaseToLatestVersion<UserContext, Configuration>());
}
```

Automatische migraties

- ▣ Migraties kunnen ook automatisch uitgevoerd en opgesteld worden
 - Enable-Migrations-EnableAutomaticMigrations
 - Code in constructor van DbContext nog steeds nodig
- ▣ Hiermee moet je niet elke keer add-migration doen
- ▣ Dit is niet altijd gewenst omdat er minder controle is over wat er exact gebeurt

Lambda's en Databases

- Reeds aangehaald dat de DbSet objecten IEnumerable implementeren.
 - Dus kan er gebruik gemaakt worden van lambda expressies en LINQ
 - Ze kunnen ook geschreven worden in een query vorm

```
0 references
public List<User> GetUsersWithFirstname(string firstname)
{
    return context.Users.Where(user=>user.FirstName == firstname).ToList();
}
```

```
0 references
public List<User> GetUsersWithFirstname(string firstname)
{
    var users = from user in context.Users
                where user.FirstName == firstname select user;
    return users.ToList();
}
```

Include

Foreign keys zijn een referentie naar een andere klasse

- Default worden deze niet ingevuld bij opvragen van de DbSet
- Deze moeten bewust geincluded worden

- Nieuwe versie is het een string met de property-naam

```
2 references
class UserRepository
{
    private UserContext context = new UserContext();

    //create
    0 references
    public void AddUser(User user)
    {
        context.Users.Add(user);
        context.SaveChanges();
    }

    //read
    //list of cars are not filled in with this function
    1 reference
    public List<User> GetAllUsers()
    {
        return context.Users.ToList();
    }

    public List<User> GetAllUsersWithCars()
    {
        return context.Users.Include(user => user.Cars).ToList();
    }
}
```



Meer info

- ▣ In deze les zijn de basis en best practices van te werken met entity framework behandeld.
- ▣ Echter is het onmogelijk om elke uitzondering te behandelen
- ▣ Meer informatie vind je op de volgende links
 - <https://docs.microsoft.com/en-us/ef/ef6/>
 - <https://www.entityframeworktutorial.net/>