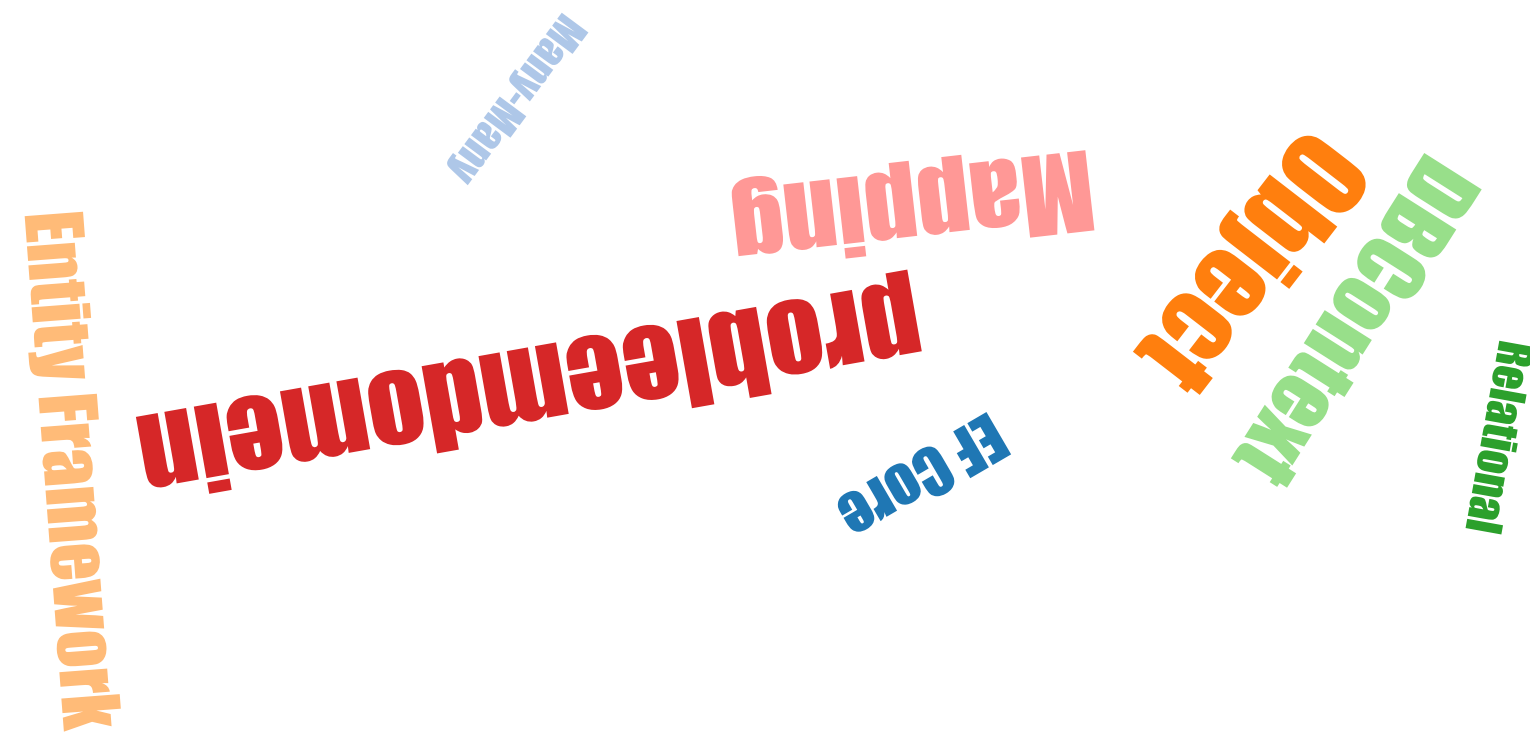


Klik [hier](#) voor pdf...



ORM

Les 6 Webdevelopment

WAAR STAAN WE?

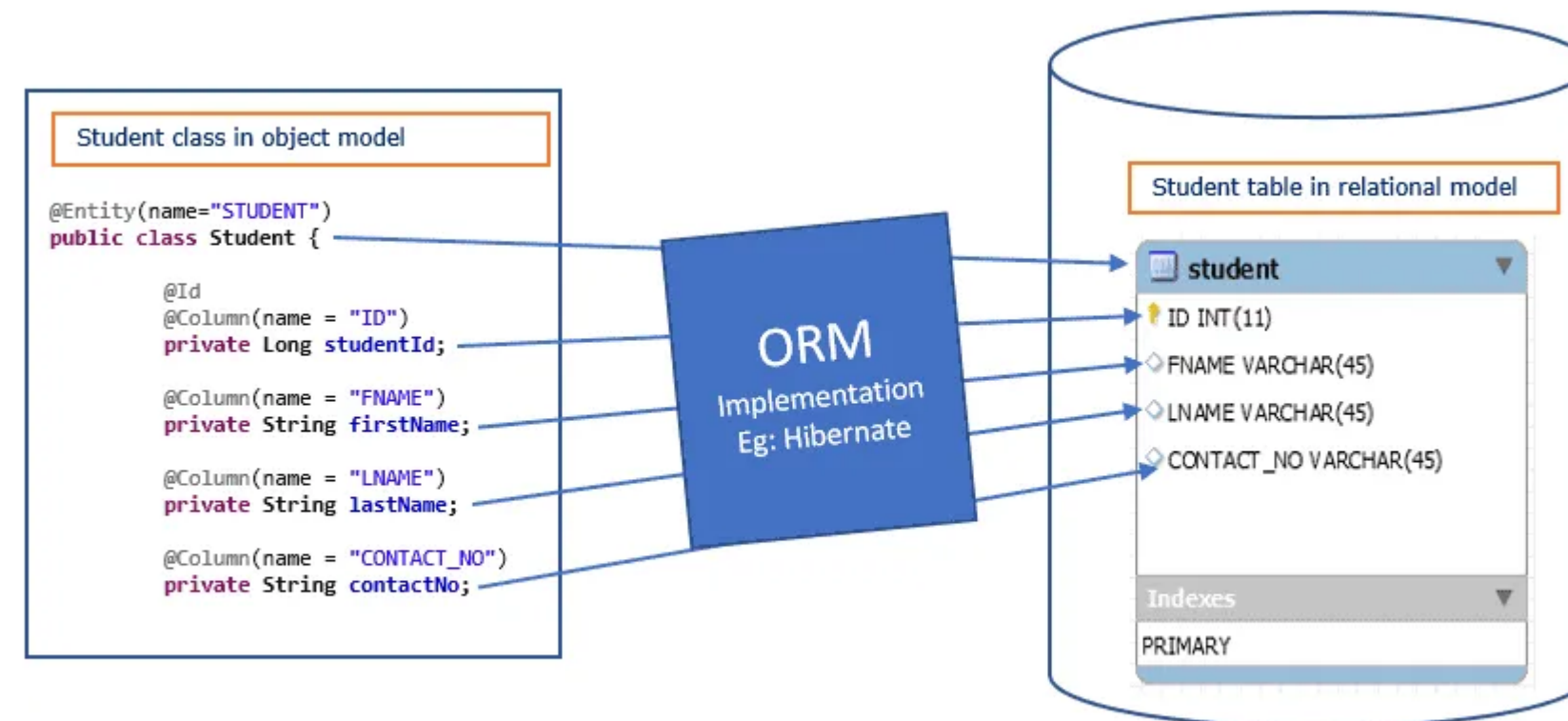
1. Frontend: HTML, CSS, Javascript
2. Frontend: Bootstrap
3. Backend: C#
4. MVC
5. Backend: C# LINQ
6. **ORM**
7. Testen
8. Layout
9. Zoeken/filteren, Sorteren, Pagineren
10. Web API, JSON, Ajax, Azure
11. Security
12. Architectuur

VANDAAG OP HET PROGRAMMA

- ORM
- Entity Framework Core
- Entity Framework Core: ingewikkelder
- De opdracht

ORM

Object Relational Mapping



ORM implements responsibility of mapping the Object to Relational Model.

 JavaByDeveloper

SOFTWAREONTWIKKELING

6. Maintenance

1. Analysis

2. Design

5. Release

**SOFTWARE
DEVELOPMENT
LIFE-CYCLE**



Bron: [Sumatosoft](#)

HERINNER...

- Tijdens *analyse* maken we een model van het *pobleemdomein*
- Tijdens *ontwerp* maken we
 - een datamodel van de *database* [herinner: DB Design Semester 3], en
 - een klassemamodel van de *code* [herinner: UML Semester 2]
- Er is een mismatch!
 - Een *string* wordt een *varchar*
- Onderhoudbaarheid/wijzigbaarheid: aanpassen database-schema -> aanpassen java-code & queries
aanpassen

HERINNER...

- Tijdens *analyse* maken we een model van het *pobleemdomein*
- Tijdens *ontwerp* maken we
 - een datamodel van de *database* [herinner: DB Design Semester 3], en
 - een klassemamodel van de *code* [herinner: UML Semester 2]
- Er is een mismatch!
 - Een *string* wordt een *varchar*
- Onderhoudbaarheid/wijzigbaarheid: aanpassen database-schema -> aanpassen java-code & queries
aanpassen

HERINNER...

- Tijdens *analyse* maken we een model van het *pobleemdomein*
- Tijdens *ontwerp* maken we
 - een datamodel van de *database* [herinner: DB Design Semester 3], en
 - een klassemamodel van de *code* [herinner: UML Semester 2]
- Er is een mismatch!
 - Een *string* wordt een *varchar*
- Onderhoudbaarheid/wijzigbaarheid: aanpassen database-schema -> aanpassen java-code & queries
aanpassen

HERINNER...

- Tijdens *analyse* maken we een model van het *pobleemdomein*
- Tijdens *ontwerp* maken we
 - een datamodel van de *database* [herinner: DB Design Semester 3], en
 - een klassemamodel van de *code* [herinner: UML Semester 2]
- Er is een mismatch!
 - Een *string* wordt een *varchar*
- Onderhoudbaarheid/wijzigbaarheid: aanpassen database-schema -> aanpassen java-code & queries
aanpassen

HERINNER...

- Tijdens *analyse* maken we een model van het *pobleemdomein*
- Tijdens *ontwerp* maken we
 - een datamodel van de *database* [herinner: DB Design Semester 3], en
 - een klassemodel van de *code* [herinner: UML Semester 2]
- Er is een mismatch!
 - Een `string` wordt een `varchar`
- Onderhoudbaarheid/wijzigbaarheid: aanpassen database-schema -> aanpassen java-code & queries
aanpassen

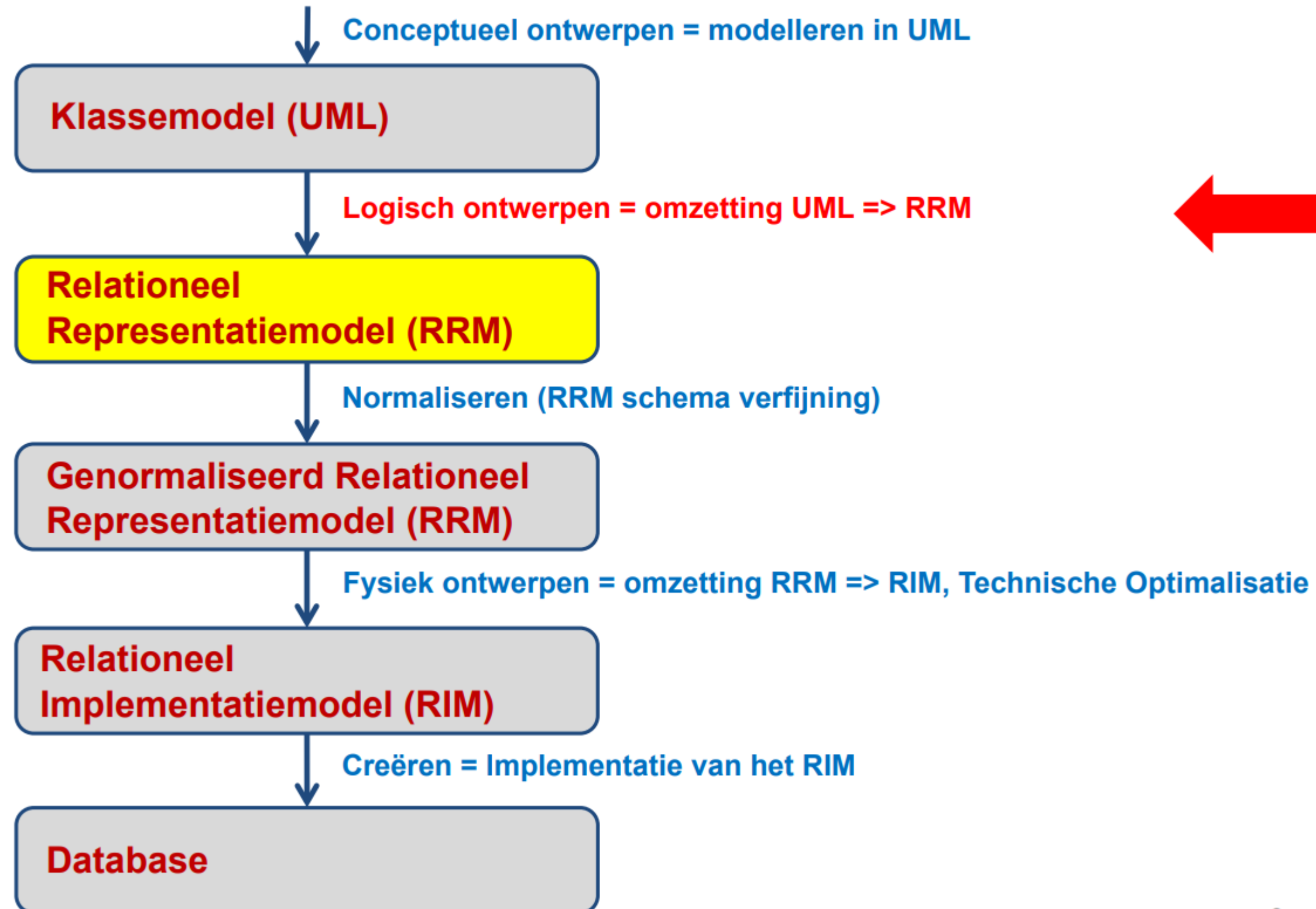
HERINNER...

- Tijdens *analyse* maken we een model van het *pobleemdomein*
- Tijdens *ontwerp* maken we
 - een datamodel van de *database* [herinner: DB Design Semester 3], en
 - een klassemodel van de *code* [herinner: UML Semester 2]
- Er is een mismatch!
 - Een **string** wordt een **varchar**
- Onderhoudbaarheid/wijzigbaarheid: aanpassen database-schema -> aanpassen java-code & queries
aanpassen

HERINNER...

- Tijdens *analyse* maken we een model van het *pobleemdomein*
- Tijdens *ontwerp* maken we
 - een datamodel van de *database* [herinner: DB Design Semester 3], en
 - een klassemodel van de *code* [herinner: UML Semester 2]
- Er is een mismatch!
 - Een `string` wordt een `varchar`
- Onderhoudbaarheid/wijzigbaarheid: aanpassen database-schema -> aanpassen java-code & queries
aanpassen

Database Design



DE ONHANDIGE OPLOSSING

C# code:

| |
|------------------|
| Docent |
| -code: int |
| -naam: string |
| -salaris: double |
| |

SQL code:

Beide oplossingen niet ideaal.

DE ONHANDIGE OPLOSSING

C# code:

SQL code:

Beide oplossingen niet ideaal.

| |
|---|
| Docent |
| -code: int -naam: string -salaris: double |
| |
| <u>Bob : Docent</u> |
| -code: int = 1256 -naam: string = "Bob" -salaris: double = 20.5 |

DE ONHANDIGE OPLOSSING

C# code:

SQL code:

Beide oplossingen niet ideaal.

| |
|---|
| Docent |
| -code: int -naam: string -salaris: double |
| |
| <u>Bob : Docent</u> |
| -code: int = 1256 -naam: string = "Bob" -salaris: double = 20.5 |
| <u>Eva : Docent</u> |
| -code: int = 1257 -naam: string = "Eva" -salaris: double = 30.6 |

DE ONHANDIGE OPLOSSING

C# code:

SQL code:

Beide oplossingen niet ideaal.

| |
|---|
| Docent |
| -code: int -naam: string -salaris: double |
| |
| <u>Bob : Docent</u> |
| -code: int = 1256 -naam: string = "Bob" -salaris: double = 20.5 |
| <u>Eva : Docent</u> |
| -code: int = 1257 -naam: string = "Eva" -salaris: double = 30.6 |

| code | naam | salaris |
|------|------|---------|
| 1256 | Bob | 20.5 |
| 1257 | Eva | 30.6 |

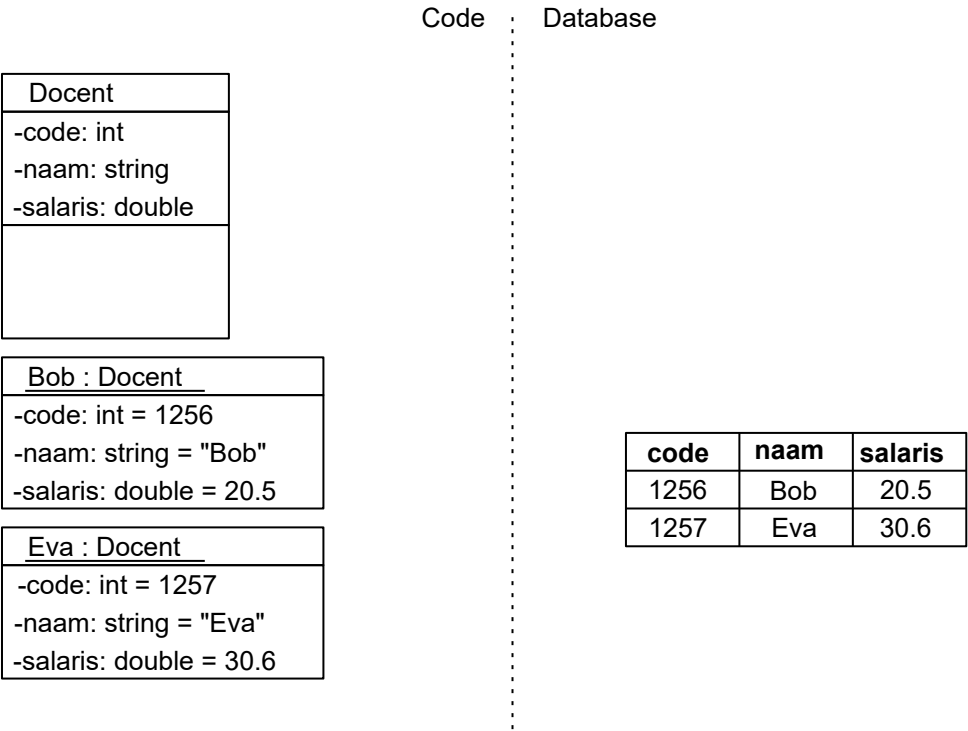


DE ONHANDIGE OPLOSSING

C# code:

SQL code:

Beide oplossingen niet ideaal.

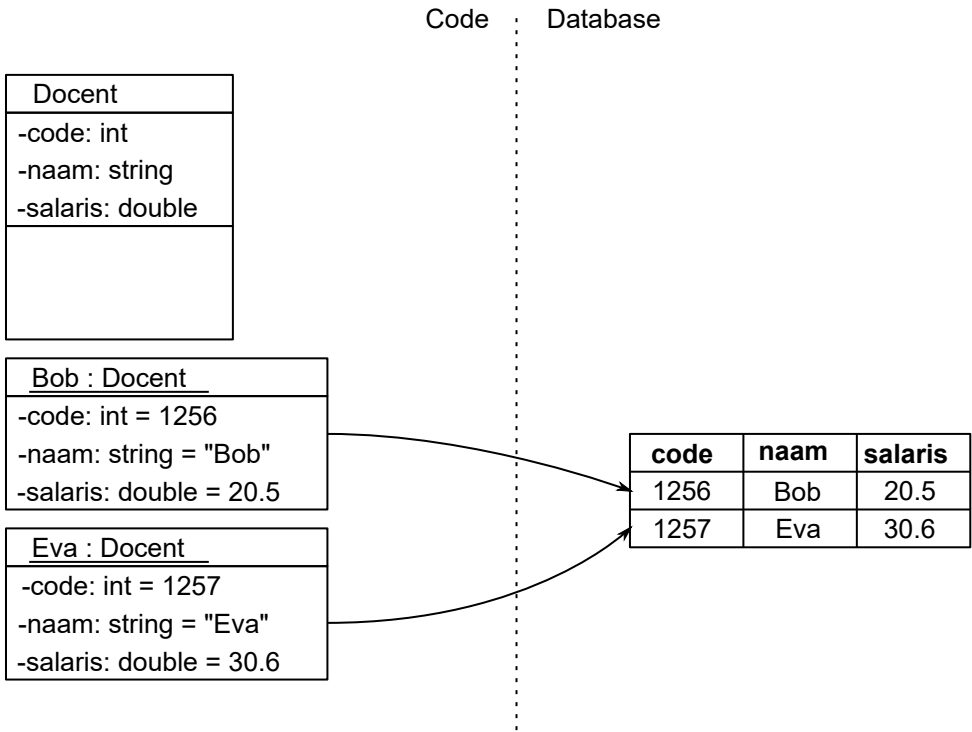


DE ONHANDIGE OPLOSSING

C# code:

SQL code:

Beide oplossingen niet ideaal.



DE ONHANDIGE OPLOSSING

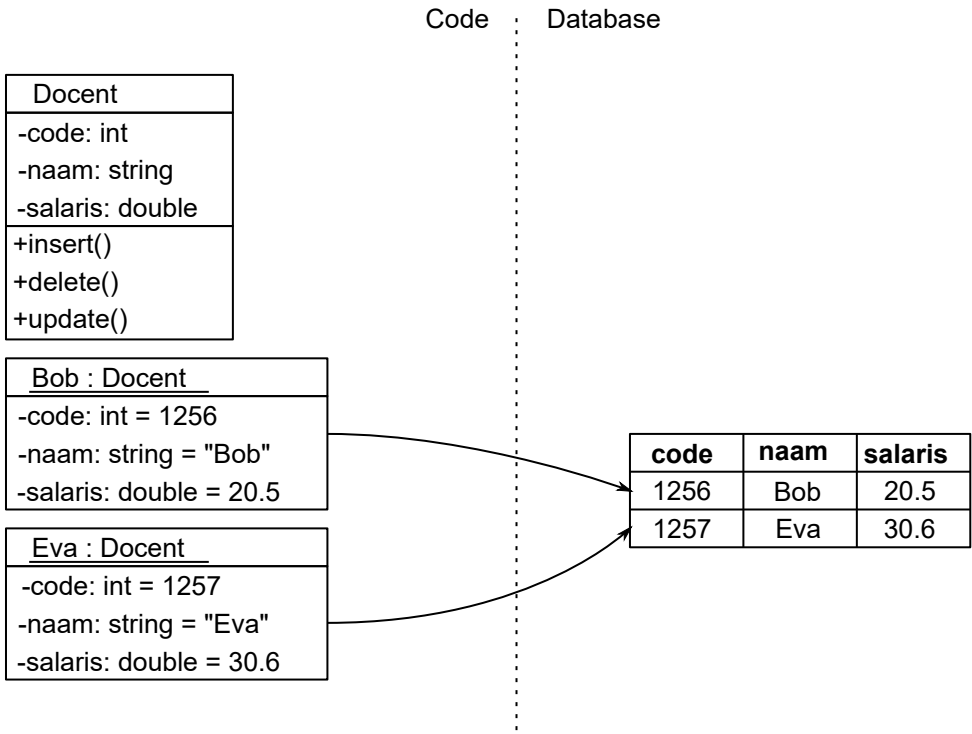
C# code:

```
public class Docent
{
    private int code;
    private string naam;
    private double salaris;
    public void delete()
    {
        string query = "delete from docent where code =" + code;
        Database.executeDeleteQuery(query);
    }
}
```

SQL code:

```
CREATE TABLE docent
(
    code int(10) unsigned NOT NULL auto_increment,
    naam varchar(45) NOT NULL,
    salaris double,
    PRIMARY KEY (code)
)
```

Beide oplossingen niet ideaal.



DE ONHANDIGE OPLOSSING

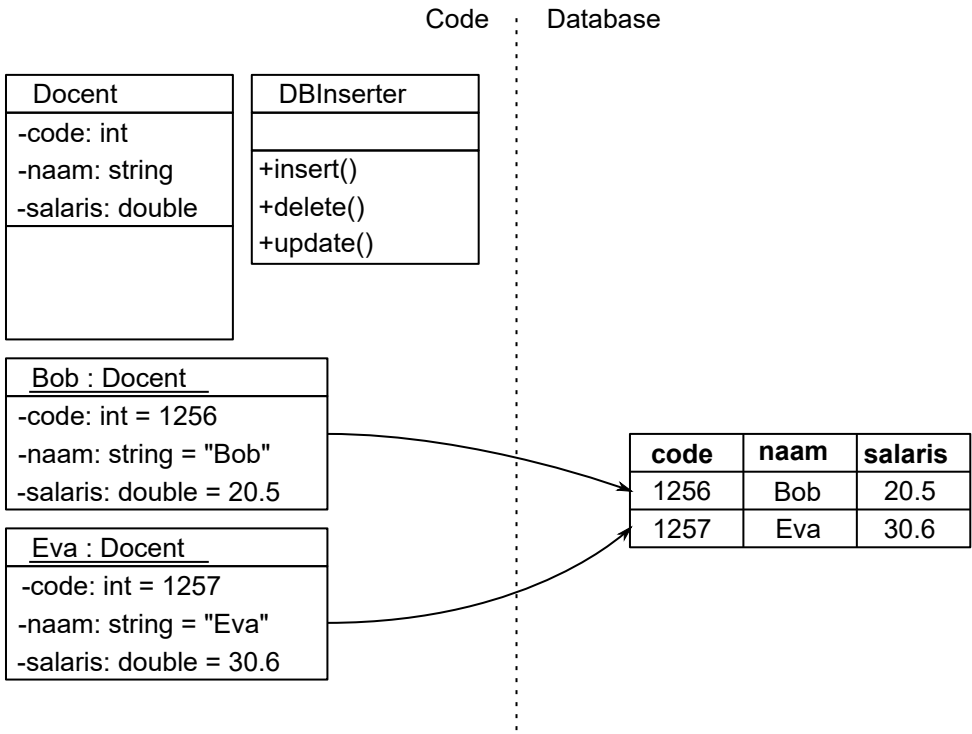
C# code:

```
public class Docent
{
    private int code;
    private string naam;
    private double salaris;
    public void delete()
    {
        string query = "delete from docent where code =" + code;
        Database.executeDeleteQuery(query);
    }
}
```

SQL code:

```
CREATE TABLE docent
(
    code int(10) unsigned NOT NULL auto_increment,
    naam varchar(45) NOT NULL,
    salaris double,
    PRIMARY KEY (code)
)
```

Beide oplossingen niet ideaal.



WE WILLEN ALLES AUTOMATISCH!

C# code:

```
public class MyContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder b) =>
        b.UseSqlite("Data Source=database.db");
    // b.UseSqlServer(...)
    public DbSet<Student> Studenten { get; set; }
}
public class Grade
{
    public int Id { get; set; }
    public int Value { get; set; }
}
public class Student
{
    public int Id { get; set; }
    public string Naam { get; set; }
    public List<Grade> Grades { get; set; }
}
```

CREATE code:

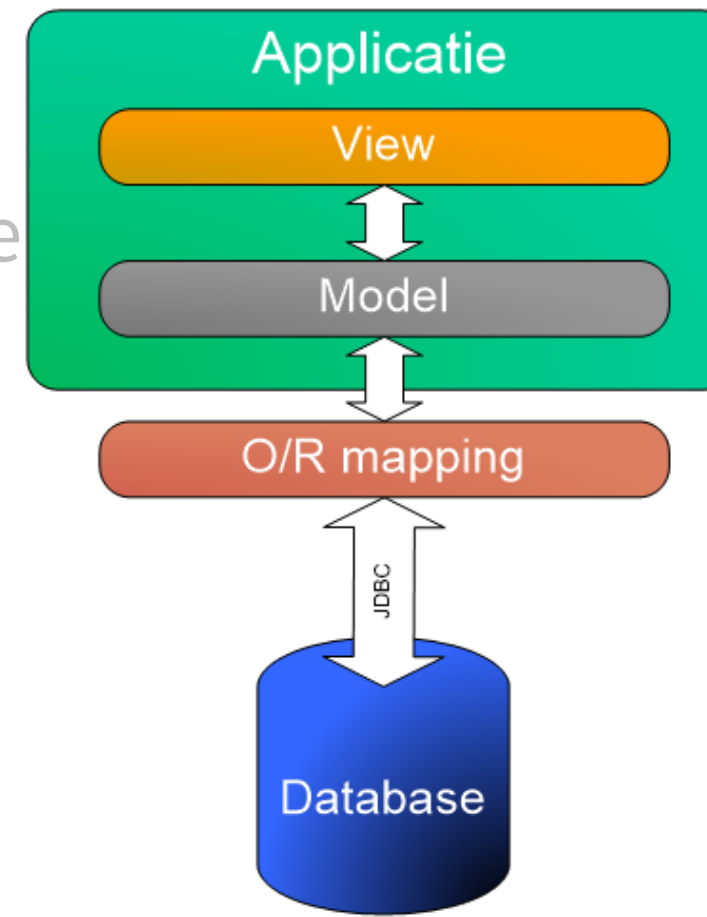
```
CREATE TABLE "Studenten" (
    "Id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    "Naam" TEXT NULL
)
CREATE TABLE "Grade" (
    "Id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    "Value" INTEGER NOT NULL,
    "StudentId" INTEGER NULL,
    FOREIGN KEY ("StudentId") REFERENCES
        "Studenten" ("Id") ON DELETE RESTRICT
)
```

Event handlers (achter de schermen):

```
public class Grade_db : Grade
{
    public int Id { get; set; }
    public int Value
    {
        get { return connectie.sql("Select ..."); }
        set { connectie.sql("Insert Into ...") }
    }
}
```

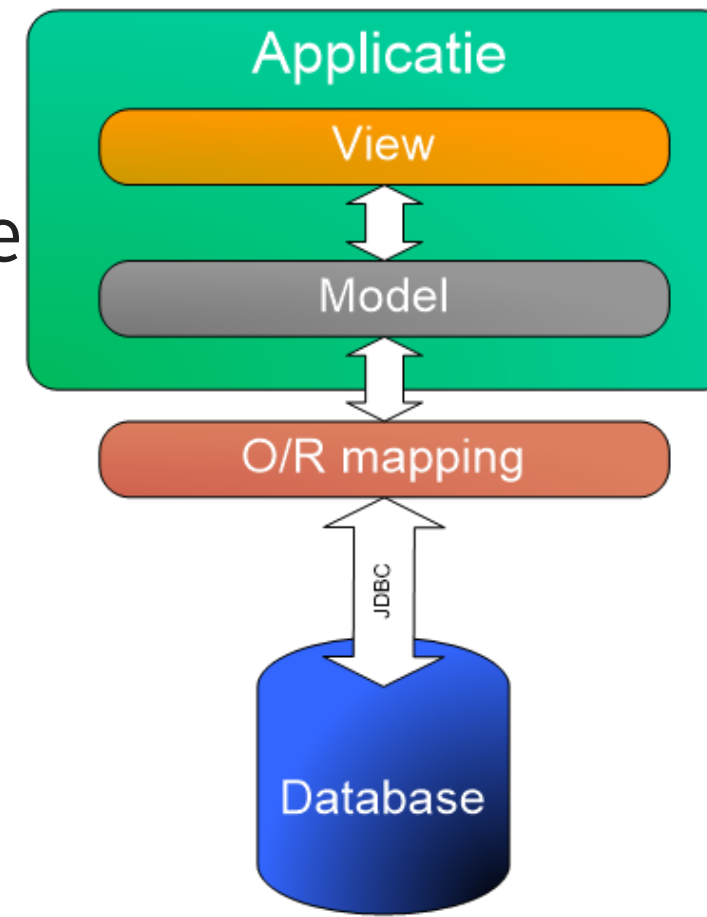
OBJECT RELATIONAL MAPPER

- O/R mapping zit tussen de applicatie en de database.
- O/R mapping vertaalt opdrachten van een applicatie naar opdrachten voor DBMS (maakt van het model een correcte SQL-opdracht die één van de CRUDS-operaties implementeert).
- O/R mapping stopt de data niet meer in een ResultSet, maar vertaalt de data uit de database direct naar objecten in het Model.
- Code bevat geen SQL meer (security).
- Minder code nodig, want de O/R mapping vervangt o.a. SELECT-, INSERT-, UPDATE- en DELETE-opdrachten.
- "Onafhankelijk" van DBMS (MS SQL Server, MySQL, PostgreSQL, etc.)



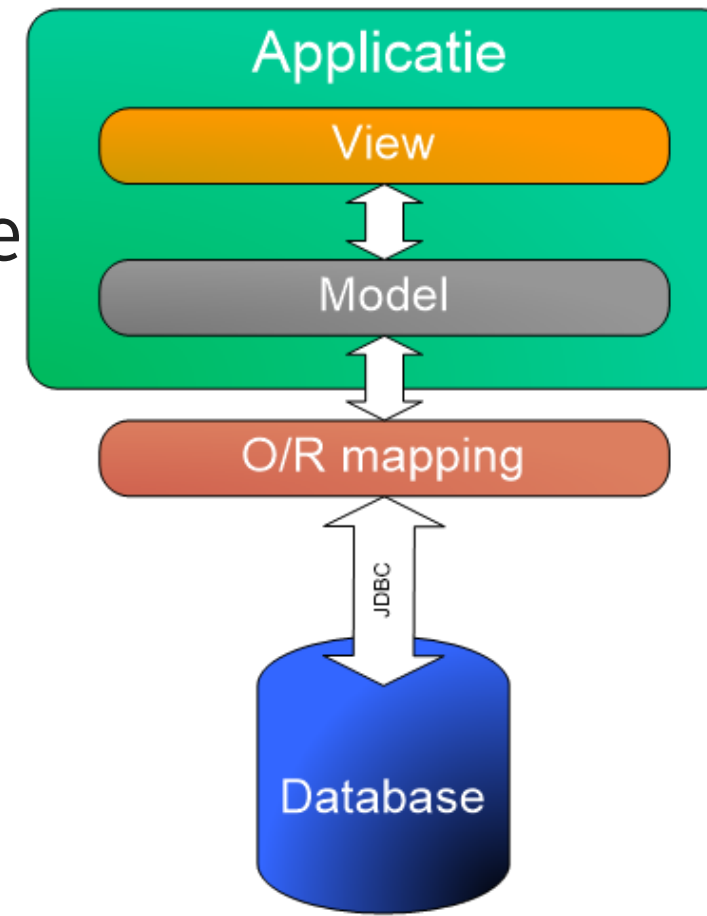
OBJECT RELATIONAL MAPPER

- O/R mapping zit tussen de applicatie en de database.
- O/R mapping vertaalt opdrachten van een applicatie naar opdrachten voor DBMS (maakt van het model een correcte SQL-opdracht die één van de CRUDS-operaties implementeert).
- O/R mapping stopt de data niet meer in een ResultSet, maar vertaalt de data uit de database direct naar objecten in het Model.
- Code bevat geen SQL meer (security).
- Minder code nodig, want de O/R mapping vervangt o.a. SELECT-, INSERT-, UPDATE- en DELETE-opdrachten.
- "Onafhankelijk" van DBMS (MS SQL Server, MySQL, PostgreSQL, etc.)



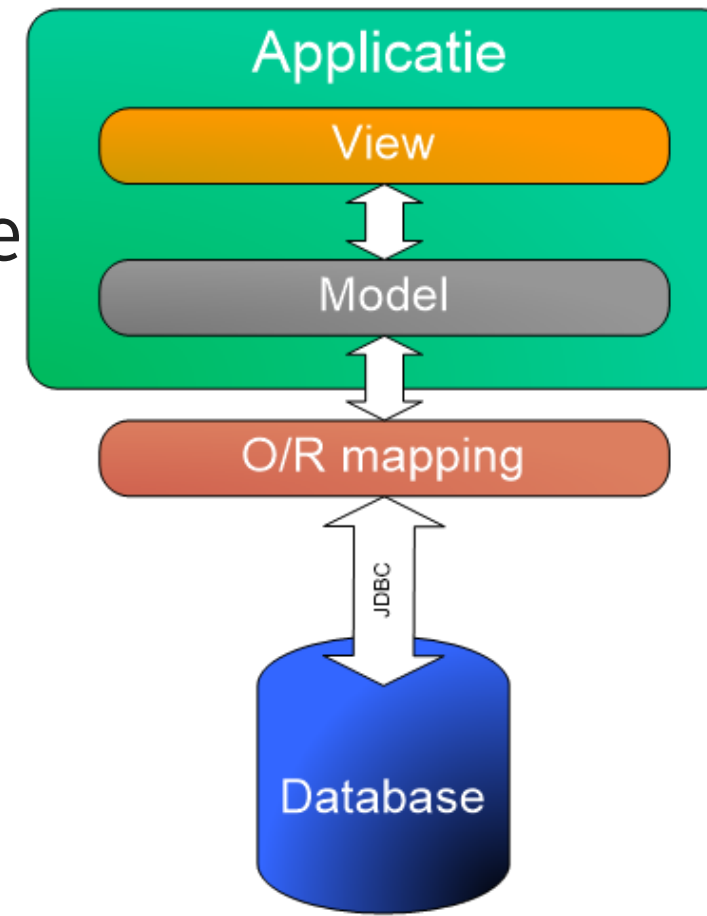
OBJECT RELATIONAL MAPPER

- O/R mapping zit tussen de applicatie en de database.
- O/R mapping vertaalt opdrachten van een applicatie naar opdrachten voor DBMS (maakt van het model een correcte SQL-opdracht die één van de CRUDS-operaties implementeert).
- O/R mapping stopt de data niet meer in een ResultSet, maar vertaalt de data uit de database direct naar objecten in het Model.
- Code bevat geen SQL meer (security).
- Minder code nodig, want de O/R mapping vervangt o.a. SELECT-, INSERT-, UPDATE- en DELETE-opdrachten.
- "Onafhankelijk" van DBMS (MS SQL Server, MySQL, PostgreSQL, etc.)



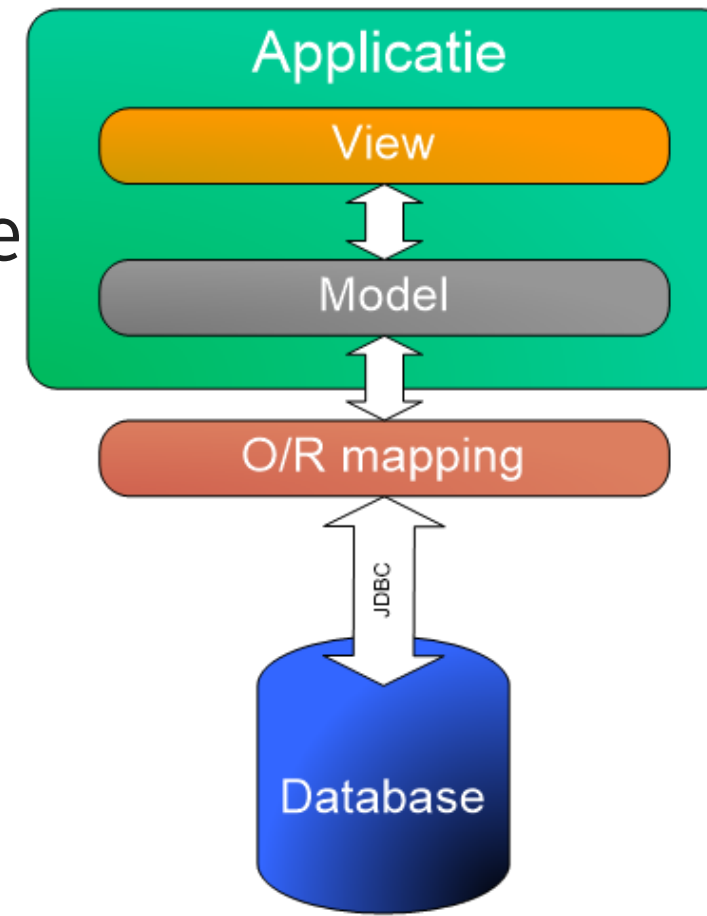
OBJECT RELATIONAL MAPPER

- O/R mapping zit tussen de applicatie en de database.
- O/R mapping vertaalt opdrachten van een applicatie naar opdrachten voor DBMS (maakt van het model een correcte SQL-opdracht die één van de CRUDS-operaties implementeert).
- O/R mapping stopt de data niet meer in een ResultSet, maar vertaalt de data uit de database direct naar objecten in het Model.
- Code bevat geen SQL meer (security).
- Minder code nodig, want de O/R mapping vervangt o.a. SELECT-, INSERT-, UPDATE- en DELETE-opdrachten.
- "Onafhankelijk" van DBMS (MS SQL Server, MySQL, PostgreSQL, etc.)



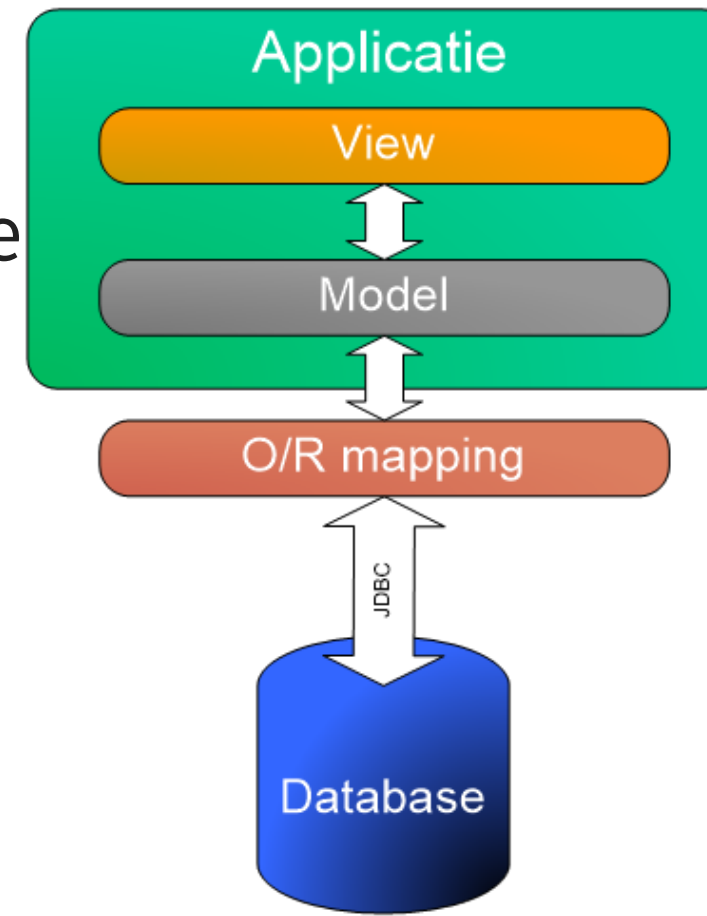
OBJECT RELATIONAL MAPPER

- O/R mapping zit tussen de applicatie en de database.
- O/R mapping vertaalt opdrachten van een applicatie naar opdrachten voor DBMS (maakt van het model een correcte SQL-opdracht die één van de CRUDS-operaties implementeert).
- O/R mapping stopt de data niet meer in een ResultSet, maar vertaalt de data uit de database direct naar objecten in het Model.
- Code bevat geen SQL meer (security).
- Minder code nodig, want de O/R mapping vervangt o.a. SELECT-, INSERT-, UPDATE- en DELETE-opdrachten.
- "Onafhankelijk" van DBMS (MS SQL Server, MySQL, PostgreSQL, etc.)



OBJECT RELATIONAL MAPPER

- O/R mapping zit tussen de applicatie en de database.
- O/R mapping vertaalt opdrachten van een applicatie naar opdrachten voor DBMS (maakt van het model een correcte SQL-opdracht die één van de CRUDS-operaties implementeert).
- O/R mapping stopt de data niet meer in een ResultSet, maar vertaalt de data uit de database direct naar objecten in het Model.
- Code bevat geen SQL meer (security).
- Minder code nodig, want de O/R mapping vervangt o.a. SELECT-, INSERT-, UPDATE- en DELETE-opdrachten.
- "Onafhankelijk" van DBMS (MS SQL Server, MySQL, PostgreSQL, etc.)



CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkom je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkom je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste reductie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste reductie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

CODE FIRST

Vergelijken

- Er is ook Database First, welke is beter?
- Bij Code First wordt voorkomen je dat de klassen "anemic" worden
- Bij Domain Driven Development (DDD) is het domein model rijk, dus Code First

Wat wordt er allemaal gedaan?

Wat is het verschil tussen data model en klassediagram?

- ID's
- Van datastructuur (lijst) naar foreign keys
- Tussentabellen voor many-to-many
- Overerving: Single Table Inheritance / Multiple Table Inheritance
- Gewenste redudatie in de code wordt genormaliseerd
- Performance: indexen in de database
- Overig: extra constraints/checks/triggers/...

ENTITY FRAMEWORK CORE

HOE?

Stappen:

- "Installeer" Entity Framework Core
- Maak C# klasse aan
- Maak C# DbContext aan (connectie-instellingen)
- Maak de database aan met 1 druk op de knop

HOE?

Stappen:

- "Installeer" Entity Framework Core
- Maak C# klasse aan
- Maak C# DbContext aan (connectie-instellingen)
- Maak de database aan met 1 druk op de knop

HOE?

Stappen:

- "Installeer" Entity Framework Core
- Maak C# klasse aan
- Maak C# `DbContext` aan (connectie-instellingen)
- Maak de database aan met 1 druk op de knop



HOE?

Stappen:

- "Installeer" Entity Framework Core
- Maak C# klasse aan
- Maak C# `DbContext` aan (connectie-instellingen)
- Maak de database aan met 1 druk op de knop

HOE? INSTALLEER EF CORE

Stappen:

- "Installeer" Entity Framework Core:
 - Nodig:
 - `Microsoft.EntityFrameworkCore.Design`
 - `Microsoft.EntityFrameworkCore.Sqlite`
 - `Microsoft.EntityFrameworkCore.Tools`
 - Voor  Visual Studio
 - Gebruik de Package Manager Console: `Install-Package ...`
 - Dependencies -> Manage NuGet Packages -> Browse -> ... -> Install
 - Voor  VS Code:

```
> dotnet add package ...
```

HOE? MAAK C# KLASSEN AAN

```
public class Grade
{
    public int Id { get; set; }
    public int Value { get; set; }
}
public class Student
{
    public int Id { get; set; }
    public string Naam { get; set; }
    public List<Grade> Grades { get; set; }
}
```


HOE? MAAK C# DBCONTEXT AAN

```
public class MyContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder b) =>
        b.UseSqlite("Data Source=database.db");
    public DbSet<Student> Studenten { get; set; }
}
```

MAAK DATABASE AAN: 1 DRUK OP DE KNOP





- In  **Visual Studio** gebruiken we de Package manager console:

```
Add-Migration MijnEersteMigration  
Update-Database
```

- In  **VS Code**:

```
> dotnet ef migrations add MijnEersteMigration  
> dotnet ef database update
```

BEKIJK DE DATABASE

- Voor SQL Server:
 - Voor  **Visual Studio**: SQL Server Object Explorer
 - Voor  **VS Code**: gebruik de extensie SQL Server (mssql)
- Voor SQLite:
 - Voor  **Visual Studio**: gebruik externe tool DB Browser for SQLite
 - Voor  **VS Code**: gebruik de extensie SQLite

SHOWTIME: VANUIT C# STUDENTEN AANMAKEN

```
DbContext mycontext = new MyContext();  
var Bob = new Student { Naam = "Bob" };  
Bob.Grades.Add(new Grade { Value = 5 });  
mycontext.Add(Bob);  
// verwijderen en aanpassen gaat even makkelijk  
mycontext.SaveChanges();  
Console.WriteLine(mycontext.Studenten.Where(s => s.Naam.StartsWith("B")).Count);
```

- ID's worden automatisch aangemaakt!
- Zodra je de C# klasse aanpast, krijg je een foutmelding.
 - Voeg een migration toe met een nieuwe naam
 - Update de database
- Er is veel magie:
 - Moet Id heten
 - Moeten properties gebruiken
 - ...

SHOWTIME: VANUIT C# STUDENTEN AANMAKEN

```
DbContext mycontext = new MyContext();  
var Bob = new Student { Naam = "Bob" };  
Bob.Grades.Add(new Grade { Value = 5 });  
mycontext.Add(Bob);  
// verwijderen en aanpassen gaat even makkelijk  
mycontext.SaveChanges();  
Console.WriteLine(mycontext.Studenten.Where(s => s.Naam.StartsWith("B")).Count);
```

- ID's worden automatisch aangemaakt!
- Zodra je de C# klasse aanpast, krijg je een foutmelding.
 - Voeg een migration toe met een nieuwe naam
 - Update de database
- Er is veel magie:
 - Moet Id heten
 - Moeten properties gebruiken
 - ...

SHOWTIME: VANUIT C# STUDENTEN AANMAKEN

```
DbContext mycontext = new MyContext();  
var Bob = new Student { Naam = "Bob" };  
Bob.Grades.Add(new Grade { Value = 5 });  
mycontext.Add(Bob);  
// verwijderen en aanpassen gaat even makkelijk  
mycontext.SaveChanges();  
Console.WriteLine(mycontext.Studenten.Where(s => s.Naam.StartsWith("B")).Count);
```

- ID's worden automatisch aangemaakt!
- Zodra je de C# klasse aanpast, krijg je een foutmelding.
 - Voeg een migration toe met een nieuwe naam
 - Update de database
- Er is veel magie:
 - Moet Id heten
 - Moeten properties gebruiken
 - ...

SHOWTIME: VANUIT C# STUDENTEN AANMAKEN

```
DbContext mycontext = new MyContext();  
var Bob = new Student { Naam = "Bob" };  
Bob.Grades.Add(new Grade { Value = 5 });  
mycontext.Add(Bob);  
// verwijderen en aanpassen gaat even makkelijk  
mycontext.SaveChanges();  
Console.WriteLine(mycontext.Studenten.Where(s => s.Naam.StartsWith("B")).Count);
```

- ID's worden automatisch aangemaakt!
- Zodra je de C# klasse aanpast, krijg je een foutmelding.
 - Voeg een migration toe met een nieuwe naam
 - Update de database
- Er is veel magie:
 - Moet Id heten
 - Moeten properties gebruiken
 - ...

SHOWTIME: VANUIT C# STUDENTEN AANMAKEN

```
DbContext mycontext = new MyContext();  
var Bob = new Student { Naam = "Bob" };  
Bob.Grades.Add(new Grade { Value = 5 });  
mycontext.Add(Bob);  
// verwijderen en aanpassen gaat even makkelijk  
mycontext.SaveChanges();  
Console.WriteLine(mycontext.Studenten.Where(s => s.Naam.StartsWith("B")).Count);
```

- ID's worden automatisch aangemaakt!
- Zodra je de C# klasse aanpast, krijg je een foutmelding.
 - Voeg een migration toe met een nieuwe naam
 - Update de database
- Er is veel magie:
 - Moet Id heten
 - Moeten properties gebruiken
 - ...

SHOWTIME: VANUIT C# STUDENTEN AANMAKEN

```
DbContext mycontext = new MyContext();  
var Bob = new Student { Naam = "Bob" };  
Bob.Grades.Add(new Grade { Value = 5 });  
mycontext.Add(Bob);  
// verwijderen en aanpassen gaat even makkelijk  
mycontext.SaveChanges();  
Console.WriteLine(mycontext.Studenten.Where(s => s.Naam.StartsWith("B")).Count);
```

- ID's worden automatisch aangemaakt!
- Zodra je de C# klasse aanpast, krijg je een foutmelding.
 - Voeg een migration toe met een nieuwe naam
 - Update de database
- Er is veel magie:
 - Moet Id heten
 - Moeten properties gebruiken
 - ...

ENTITY FRAMEWORK CORE: INGEWIKKELDER

DATABASE CONFIGUREREN: DATA-ANNOTATIES

De volgende annotaties moeten jullie kennen:

- [Key]
- [Required]
- [StringLength]
- [Table]
- [Column]
- [NotMapped]

Voorbeeld:

```
[Table("Students")]
public class Student
{
    [Key] public int StudentNr { get; set; }
    [Required] [StringLength(20)] public string Naam { get; set; }
    public List<Grade> Grades { get; set; } = new List<Grade>();
}
```

DATABASE CONFIGUREREN: FLUENT API

```
public class MyContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite(@"Data Source=database.db");
    }
    public DbSet<Student> Studenten { get; set; }
    protected override void OnModelCreating(ModelBuilder b)
    {
        b.Entity<Student>()
            .Property(s => s.Naam)
            .IsRequired()
            .HasMaxLength(20);
    }
}
```

Met de Fluent API kan je meer dan data annotaties schrijven, maar ingewikkelder en gescheiden van de code schrijven.

DATABASE CONFIGUREREN: FLUENT API

```
public class MyContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite(@"Data Source=database.db");
    }
    public DbSet<Student> Studenten { get; set; }
    protected override void OnModelCreating(ModelBuilder b)
    {
        b.Entity<Student>()
            .Property(s => s.Naam)
            .IsRequired()
            .HasMaxLength(20);
    }
}
```

Met de Fluent API kan je meer dan data annotaties schrijven, maar ingewikkelder en gescheiden van de code schrijven.

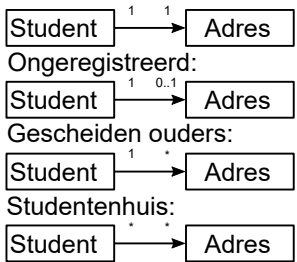
SEEDEN VAN DATABASE

Later hierover meer...

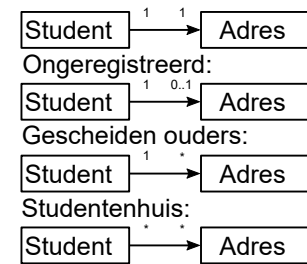
```
public static class DbInitializer
{
    public static void Initialize(MyContext context)
    {
        if (context.Studenten.Any()) return;
        context.Studenten.Add(new Student { Naam = "Bob" });
        context.Studenten.Add(new Student { Naam = "Alice" });
        context.SaveChanges();
    }
}
```

RELATIES

HOE ZAT DAT OOK AL WEER? VERGEET EF EVEN...

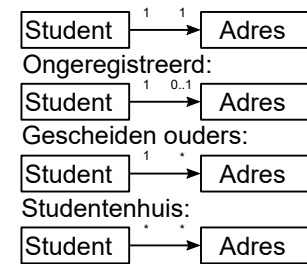


HOE ZAT DAT OOK AL WEER? VERGEET EF EVEN...



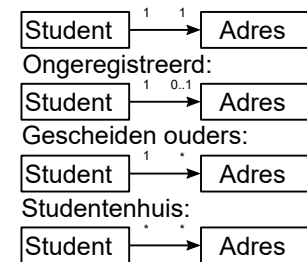
```
public class Student
{
    public Student ()
    {
        Adres = new Adres ();
    }
    public Adres Adres { get; set; }
}
```

HOE ZAT DAT OOK AL WEER? VERGEET EF EVEN...



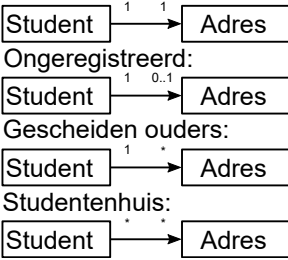
```
public class Student
{
    public Student () { }
    public Adres Adres { get; set; }
}
```

HOE ZAT DAT OOK AL WEER? VERGEET EF EVEN...

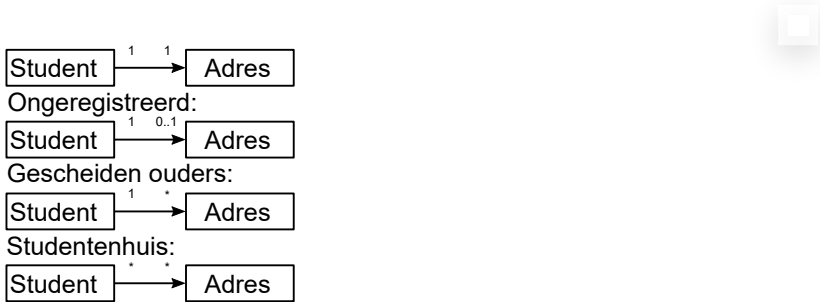


```
public class Student
{
    public Student ()
    {
        adressen = new List<Adres> ();
    }
    private List<Adres> adressen;
    public void voegAdresToe (Adres a)
    {
        if (!adressen.Contains (a))
            adressen.Add (a) ;
    }
}
```

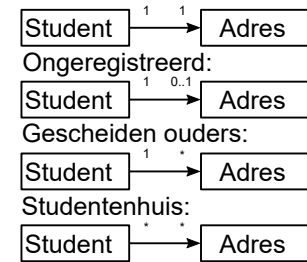
NU IN EF CORE



NU IN EF CORE



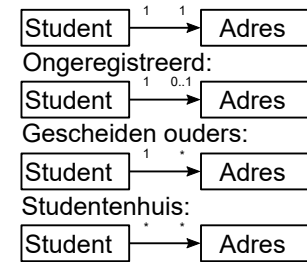
NU IN EF CORE



```
public class Adres
{
    public int Id { get; set; }
    public string Tekst { get; set; }
    public Student Student { get; set; }
}

public class Student
{
    public int Id { get; set; }
    public Adres Adres { get; set; }
    public int AdresId { get; set; }
}
```

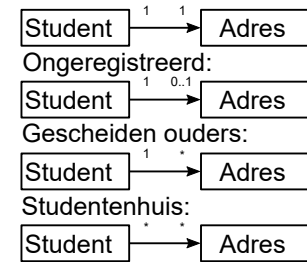
NU IN EF CORE



```
public class Adres
{
    public int Id { get; set; }
    public string Tekst { get; set; }
    public int StudentId { get; set; } // hoeft niet
    public Student Student { get; set; } // hoeft niet
}

public class Student
{
    public int Id { get; set; }
    public List<Adres> Adressen { get; set; }
}
```

NU IN EF CORE



```
public class Adres
{
    public int Id { get; set; }
    public string Tekst { get; set; }
    public int StudentId { get; set; }
    public List<StudentAdres> StudentAdressen { get; set; }
}

public class Student
{
    public int Id { get; set; }
    public List<StudentAdres> StudentAdressen { get; set; }
}

public class StudentAdres
{
    public int Id { get; set; }
    public int StudentId { get; set; }
    public Student Student { get; set; }
    public string AdresId { get; set; }
    public Adres Adres { get; set; }
}
```


CONFIGUREREN

Soms wil je een andere naam (bijvoorbeeld als er meerdere relaties zijn).

Met data-annotaties:

Met Fluent API:

CONFIGUREREN

Soms wil je een andere naam (bijvoorbeeld als er meerdere relaties zijn).

Met data-annotaties:

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int AuthorUserId { get; set; }
    public User Author { get; set; }

    public int ContributorUserId { get; set; }
    public User Contributor { get; set; }
}

public class User
{
    public string UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    [InverseProperty("Author")]
    public List<Post> AuthoredPosts { get; set; }

    [InverseProperty("Contributor")]
    public List<Post> ContributedToPosts { get; set; }
}
```

Met Fluent API:



CONFIGUREREN

Soms wil je een andere naam (bijvoorbeeld als er meerdere relaties zijn).

Met data-annotaties:

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int AuthorUserId { get; set; }
    public User Author { get; set; }

    public int ContributorUserId { get; set; }
    public User Contributor { get; set; }
}

public class User
{
    public string UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    [InverseProperty("Author")]
    public List<Post> AuthoredPosts { get; set; }

    [InverseProperty("Contributor")]
    public List<Post> ContributedToPosts { get; set; }
}
```

Met Fluent API:

```
class MyContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Author)
            .WithMany(u => u.AuthoredPosts);
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Contributor)
            .WithMany(u => u.ContributedToPosts);
    }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int AuthorUserId { get; set; }
    public User Author { get; set; }

    public int ContributorUserId { get; set; }
    public User Contributor { get; set; }
}

public class User
{
    public string UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public List<Post> AuthoredPosts { get; set; }
    public List<Post> ContributedToPosts { get; set; }
}
```

LOADING

Wordt meteen de hele database geladen als er overal relaties zijn? Nee.

```
class MyContext : DbContext
{
    public DbSet Posts { get; set; }
    public DbSet Users { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
        optionsBuilder
            .UseSqlite(@"Data Source=C:\Users\Pascal\source\repos\ConsoleApp3\database.db");
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity()
            .HasOne(p => p.Author)
                .WithMany(u => u.AuthoredPosts);

        modelBuilder.Entity()
            .HasOne(p => p.Contributor)
                .WithMany(u => u.ContributedToPosts);
    }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int AuthorUserId { get; set; }
    public virtual User Author { get; set; }

    public int ContributorUserId { get; set; }
    virtual public User Contributor { get; set; }
}

public class User
{
    public string UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    virtual public List AuthoredPosts { get; set; }
    virtual public List ContributedToPosts { get; set; }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        using (var mycontext = new MyContext())
        {
            /*
            User u = new User { UserId = "Bob1994", FirstName = "Bob" };
            Post p = new Post { Title = "Titel", Author = u };
            mycontext.Add(p);
            mycontext.SaveChanges();
            */
            Console.WriteLine(mycontext.Posts.ToList()[0].Author.UserId);
        }
    }
}
```

LOADING: VERSCHILLENDE SOORTEN

- Eager loading: gerelateerde data opgehaald bij initiële query

```
var posts = context.Posts
    .Include(post => post.Author)
    // .ThenInclude(...)
    .ToList();
```

- Explicit loading: gerelateerde data expliciet opgehaald op later tijdstip

```
var post = context.Posts
    .Single(p => p.PostId == 1);
context.Entry(post)
    .Reference(p => p.Author)
    // .Collection(...)
    .Load();
```

- Lazy loading: alles automatisch, kan traag zijn bij opvragen
 - Gebruik Microsoft.EntityFrameworkCore.Proxies
 - Roep UseLazyLoadingProxies() aan
 - Gebruik bij alle properties virtual

LOADING: VERSCHILLENDE SOORTEN

- Eager loading: gerelateerde data opgehaald bij initiële query

```
var posts = context.Posts
    .Include(post => post.Author)
    // .ThenInclude(...)
    .ToList();
```

- Explicit loading: gerelateerde data expliciet opgehaald op later tijdstip

```
var post = context.Posts
    .Single(p => p.PostId == 1);
context.Entry(post)
    .Reference(p => p.Author)
    // .Collection(...)
    .Load();
```

- Lazy loading: alles automatisch, kan traag zijn bij opvragen
 - Gebruik Microsoft.EntityFrameworkCore.Proxies
 - Roep `UseLazyLoadingProxies()` aan
 - Gebruik bij alle properties `virtual`

LOADING: VERSCHILLENDE SOORTEN

- Eager loading: gerelateerde data opgehaald bij initiële query

```
var posts = context.Posts
    .Include(post => post.Author)
    // .ThenInclude(...)
    .ToList();
```

- Explicit loading: gerelateerde data expliciet opgehaald op later tijdstip

```
var post = context.Posts
    .Single(p => p.PostId == 1);
context.Entry(post)
    .Reference(p => p.Author)
    // .Collection(...)
    .Load();
```

- Lazy loading: alles automatisch, kan traag zijn bij opvragen
 - Gebruik Microsoft.EntityFrameworkCore.Proxies
 - Roep UseLazyLoadingProxies() aan
 - Gebruik bij alle properties virtual

LOADING: VERSCHILLENDE SOORTEN

- Eager loading: gerelateerde data opgehaald bij initiële query

```
var posts = context.Posts
    .Include(post => post.Author)
    // .ThenInclude(...)
    .ToList();
```

- Explicit loading: gerelateerde data expliciet opgehaald op later tijdstip

```
var post = context.Posts
    .Single(p => p.PostId == 1);
context.Entry(post)
    .Reference(p => p.Author)
    // .Collection(...)
    .Load();
```

- Lazy loading: alles automatisch, kan traag zijn bij opvragen
 - Gebruik Microsoft.EntityFrameworkCore.Proxies
 - Roep UseLazyLoadingProxies() aan
 - Gebruik bij alle properties virtual

DE OPDRACHT

De opdracht:

Bij deze opdracht mag je creativiteit gebruiken om iets uit de werkelijkheid te modelleren. Maak eerst een klasse-diagram en vervolgens de bijbehorende C# code (met data-annotaties en de `DbContext`). Zorg dat `Add-Migration` en `Update-Database` werken.

Eisen:

- Maak tenminste één een-op-een relatie aan.
- Maak tenminste één een-op-veel relatie aan.
- Maak tenminste één veel-op-veel relatie aan.
- Gebruik tenminste één keer overerving. Zie de [docs](#).
- Maak voor tenminste één configuratie gebruik van de Fluent API, i.p.v. de data-annotaties.
- Maak tenminste één keer zinvol gebruik van een composite sleutel. Maak ook tenminste één foreign key naar deze sleutel. Zie de [docs](#).
- Maak tenminste een keer gebruik van `[Table]`
- Maak tenminste een keer gebruik van `[Column]`
- Maak tenminste een keer zinvol gebruik van `[StringLength]`
- Maak tenminste een keer zinvol gebruik van `[NotMapped]`

Resultaat:

  Lever het UML diagram en de C# code in een `.docx` in.