EF INTRO AND RELATIONS

AGENDA

- Entity Framework Core
- Creating your first project
- Entity Framework core
  - Keys
  - Properties
  - Relationships

ENTITY FRAMEWORK (CORE)

- O/RM - Object relation mapper

Relational database	Object Oriented langauge
Table	Class
Column	Property
Unique Row	Object
Rows	Collection of Objects
Foreign key	Reference
SQL - e.q. WHERE	.NET LINQ - e.g. WHERE(...)

OBJECT RELATION MAPPER - WHY?

- Avoid writing all database queries by hand. This work is tedious and error prone
- Help generate a database scheme from you OOP model
- or generate OOP model from an existing database
- Security - we will look into this later
- Avoid SQL inside code (e.g. C#)

DOWNSIDES

- Different paradigms in OOP and Relational database
- Pollution of OOP classes with annotations etc.
- 'Forget' that data is saved in database, meaning you write code that works in test, but not in production

ENTITY FRAMEWORK CORE

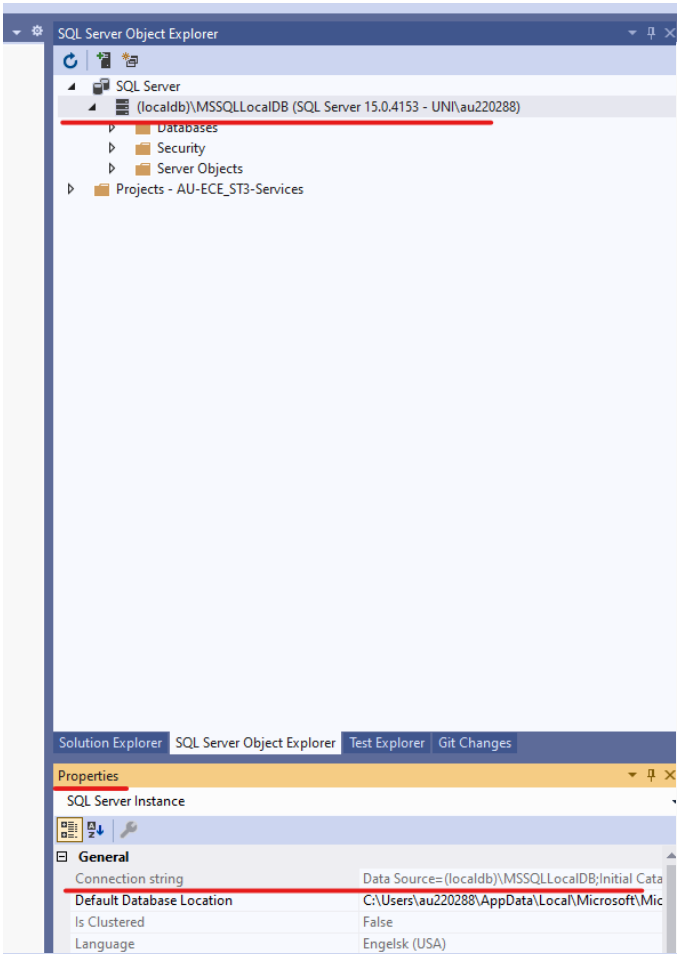
- Build in C#
- Is an O/RM
- Open source
- Cross platform
- Build to support NoSQL

Speaker notes

EF6/7 is not build to support NoSQL but is extended to do so

CONNECTION STRING LOCALDB

- 'Server Explorer' window in Visual Studio (Menu -> View -> Server Explorer)
  - Connect to database
  - From properties menu


SQL Server Object Explorer
SQL Server
(localdb)\MSSQLLocalDB (SQL Server 15.0.4153 - UNI\au220288)
Databases
Security
Server Objects
Projects - AU-ECE_ST3-Services
Solution Explorer
SQL Server Object Explorer
Test Explorer
Git Changes
Properties
SQL Server Instance
General
Connection string
Data Source=(localdb)\MSSQLLocalDB;Initial Cata
Default Database Location
C:\Users\au220288\AppData\Local\Microsoft\Mic
Is Clustered
False
Language
Engelsk (USA)

STARTING WITH EFCORE 1

1. Create a .Net 5.0 console project for SqlServer
   - Create new .NET - Console App project in UI
2. Install Entity Framework Core
   - From the Visual Studio menu, select Project -> Manage NuGet Packages
   - Install 'Microsoft.EntityFrameworkCore.SqlServer' **and** 'Microsoft.EntityFrameworkCore.Design' **and** 'Microsoft.EntityFrameworkCore.Tools'* packages

* 'Microsoft.EntityFrameworkCore.Tools' can be installed globally

STARTING WITH EFCORE 2

3. Create a new class `MyDbContext` which inherits from `DbContext`
4. Adding a connection string (in .cs file)
   - In class `MyDbContext` add the following code

```
protected override void OnConfiguring(
    DbContextOptionsBuilder optionsBuilder) {
    optionsBuilder.UseSqlServer("<REPLACE WITH CONN STRING>");
}
```

Speaker notes

```
$ dotnet tool install --global dotnet-ef --version 5.0.3
$ mkdir MyFirstEFCoreProject
$ cd MyFirstEFCoreProject
$ dotnet new console
$ dotnet add package Microsoft.EntityFrameworkCore.SqlServer
$ dotnet add package Microsoft.EntityFrameworkCore.Design
```

Create the MyDbContext.cs as on slide and add code

DATABASE CONTEXT

- A class that inherits from EF Cores DbContext
- Contain information that EF Core needs to configure database mappings
- The class you use to access data in database
- Connection to database is created through:
  - Overriding method `OnConfiguring` and supply connection string
  - E.g. optionsBuilder.UseSqlServer(ConnectionString);
- Can also be `UseSqlite`, `UseMySql` etc.

CREATING MODEL CLASSES

- Create class Door
- Add properties with public getter and setter
- Primary key is by convention named '`Id`' or '`<class name>Id`' (case insensitive)

```
// in Door.cs
public class Door {
  public int DoorId {get;set;}
  public Location Location {get;set;}
  public string Type {get;set;}
}

// In MyDbContext.cs add
public DbSet<Door> doors { get; set; }
```

CREATE DATABASE 1

1. Doing code first - you let Entity Framework create your database. In VS2019 -

```
1. Open PowerShell (Tools -> Manage Nuget -> Package Manager C
2. > Install-Package Microsoft.EntityFrameworkCore.Tools (firs
3. > Add-Migration InitialCreate
4. > Update-Database
```

2. If you want to change database (**only in this lecture**)

```
1. Open PowerShell
2. > Update-Database 0
3. > Remove-Migration (in Package manager console)
4. Make changes in code
5. GOTO 1.1
```

Speaker notes

In CLI

```
$ dotnet ef migrations add InitialMigration
$ dotnet ef database update
// Change database
$ dotnet ef database update 0
$ dotnet ef migrations remove
Make changes in code
Goto top
```

CREATING MODEL IN DETAILS

- Looks at all `DbSet` properties
- Looks at properties in these classes
- Looks at linked classes
- Runs `OnModelCreating`
- -> results in database schema (which can be found in Migrations/AppDbContextModelSnapshot.cs)

Speaker notes

`OnModelCreate` is a method from `DbContext` which we can override.

CREATING DATA

- Write data with EF Core
- In C#

```
var door = new Door() {
  Location = location,
  Type = "Wood"
}
context.Doors.Add(door);
context.SaveChanges();
```

READ DATA

- Read data from EF Core
- In C#

```
context.Doors.AsNoTracking().Include(a => a.Location)
```

Is translated into:

```
SELECT b.DoorId, ..., a.LocationId, ...
FROM Door AS B
INNER JOIN Location AS a
  ON b.LocationId = a.LocationId
```

WHAT EF CORE DOES

- LINQ is translated into SQL and cached
- Data is read in one command / roundtrip (or few)
- Data is turned into instances of the .NET class
- No tracking snapshot is created in this instance - so this is readonly - more on this next time.

UPDATE DATA

- Update data using EF Core
  - In C#

```
var door = context.Doors....;
door.Location.Address = 'new address';
db.SaveChanges()
```

Is translated into:

```
UPDATE Location SET Address = 'new address'
WHERE LocationId = '...'
```

WHAT EF CORE DOES

- LINQ is translated into SQL
- Tracking snapshopts are created - holding original values
- DetectChanges works out what has changed
- Transaction is started - all or nothing is saved
- SQL command is executed


Insanity: doing the same thing over and over again and expecting different results.
Unknown
BrainyQuote

EFCORE TATICS

Primary key

- **Conventions** - Class property with name '`<class-name>Id`' or '`Id`'
- **Data annotations** - Annotate property with `[Key]`
- **Fluent API** (always in `DbContext`) - In `DbContext.OnModelCreating`

```
public class Context : DbContext {
  public void protected override void
          OnModelCreating(ModelBuilder mb) {
    mb.Entity<Car>().HasKey(c => c.LicensePlate);
} }
```

CONSTRAINTS - FLUENT API

```csharp class MyContext : DbContext { ... DbSet clients {get; set;} protected override void OnModelCreating(ModelBuilder mb) { mb.Entity() .Property(b => b.LastName) .IsRequired() // Not null .HasMaxLength(64); }}
```

Speaker notes

You need to use one of these methods, not all :)

CONSTRAINTS - ANNOTATIONS

```
namespace MyApp.Models {
  public class Client {
       [Required]
       public int ID {get ; set;}
       [Required]
       [MaxLength(64)]
       public string FirstName {get; set;}
       [Required]
       [MaxLength(64)]
       public string LastName {get; set;}
       public string Email {get; set;}
       ...
       public Membership Membership {get; set;}
}}
```

KEYS

PRIMARY KEYS

- Convention

```
public int Id {get; set;}
public int <ClassName>Id { get; set;}
```

- or Annotation

```
[Key]
public int Identifier {get; set;}
```

- or Fluent API

```
protected override void OnModelCreating(ModelBuilder mb) {
  mb.Entity<Book>().HasKey(b => b.ID);
}
```

KEYS CONTINUED

- When using keys that are non-composite numeric and GUID you need to consider Value Generation
- Composite keys
  - Can only be configured by the Fluent API

```
protected override void OnModelCreating(ModelBuilder mb) {
  mb.Entity<Author>()
      .HasKey(a => new { a.FirstName, a.LastName});
}
```

KEY NAME

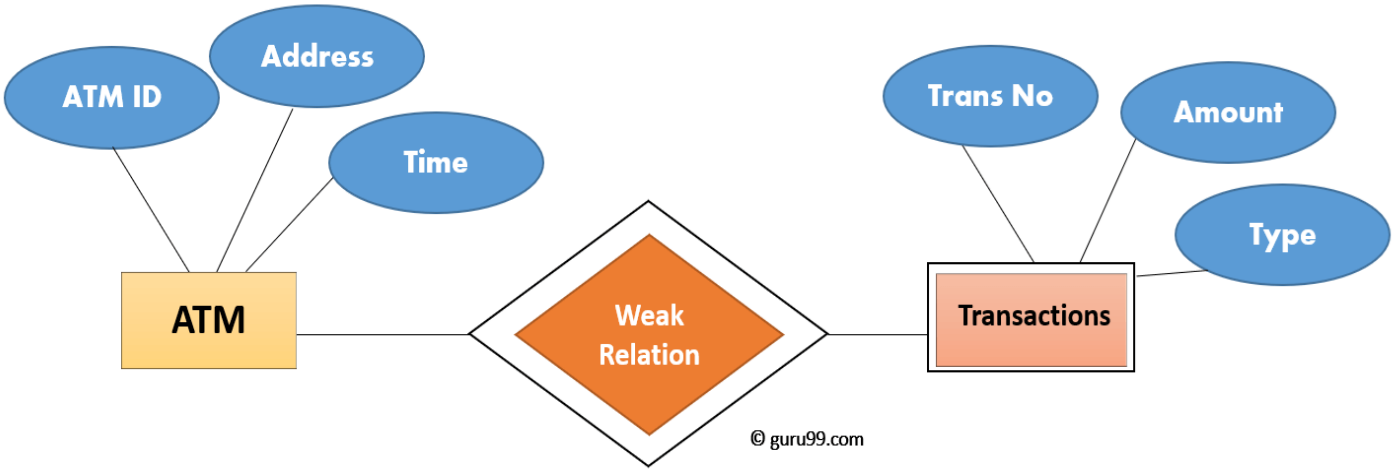
- EfCore naming of key is by convention PK_<type_name>
  - This can be changed by

```
protected override void OnModelCreating(ModelBuilder mb) {
    mb.Entity<Author>()
        .HasKey(a => a.Name)
        .HasName("PrimaryKey_Name");
}
```

INDEX & UNIQUENESS

```
public class MyDbContext: DbContext {
  protected override void OnModelCreating(ModelBuilder mb) {
    // Alternative key - unique
    mb.Entity<Book>().HasAlternateKey(b => b.Isbn)
        .HasName("UniqueIsbn");
    // Index - not nessesaryly unique
    mb.Entity<Book>().HasIndex(b => b.Isbn)
        // Remember isUnique with Index
        .HasName("Isbn index").IsUnique();
    // Composite key - also available with HasIndex
    mb.Entity<Author>()
        .HasKey(a => new { a.FirstName, a.LastName});
}}
```

PROPERTIES


ATM ID
Address
Time
ATM
Weak Relation
Trans No
Amount
Type
Transactions
© guru99.com

EXCLUDING PROPERTIES

- Annotations

```
public class Person {
...
[NotMapped]
public string FullName {
  get => $"{FirstName} {MiddleName} {LastName}";
}}
```

- Alternatively in Fluent Api

```
mb.Entity<Book>().Ignore(b => b.FullTitle);
mb.Ignore<BookMetadata>(); // For types
```

- `[NotMapped]` not needed when no public setter

DATABASE GENERATED VALUES

```
public class Books {
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public DateTime Created {get;set;}
}

public class MyDbContext: DbContext {
  protected override void OnModelCreating(ModelBuilder mb) {
    mb.Entity<Book>().Property(b => b.Created)
        .HasDefaultValue(DateTime.Now)
  }
}
```

- Above is migrations time decided. If value should dynamicly from SQL Server

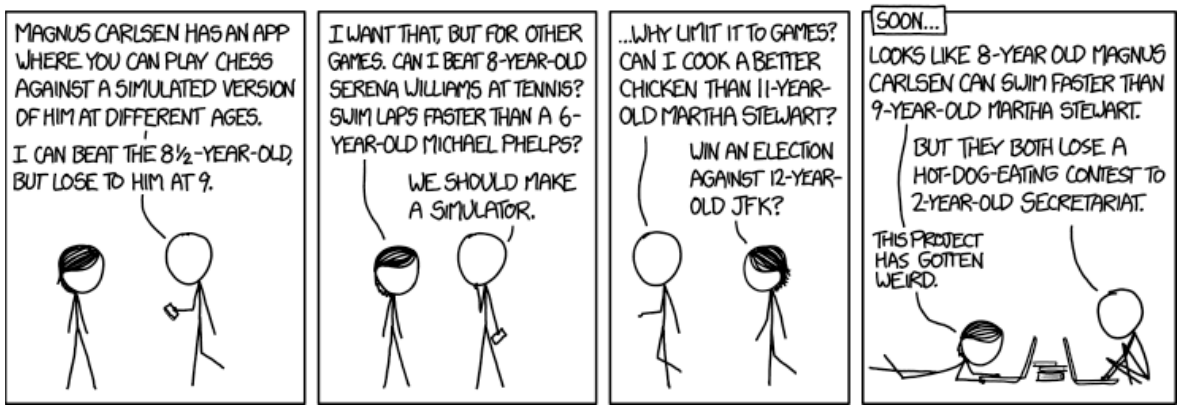
```
mb.Entity<Book>().Property(b => b.Created)
    .HasDefaultValueSql("getdate()");
```

SHADOW PROPERTIES

- Hidden from Model
- To make OOP model clean
- Steps:
  1. Remove the `Created` property from `Books.cs`
  2. In `MyDbContext` add in `OnModelCreating`

```
mb.Entity<Book>().Property<DateTime>("Created")
.HasDefaultValueSql("getdate()");
```

RELATIONSHIPS



1-1 RELATIONSHIP (1/2)

```
1  public class Membership {
2    [Required] public int ID {get; set;}
3    [Required] public Genre Genre {get; set;}
4    public int ClientId {get;set;}
5    public Client Client {get;set;} // Navigational Property
6  }
7  public class Client {
8    [Required] public int ID {get ; set;}
9    [Required] public string FirstName {get; set;}
10   [Required] public string LastName {get; set;}
11   public string Email {get; set;}
12   ...
13   public Membership Membership {get; set;}
14 }
```

1-1 RELATIONSHIP (1/2)

```
1  public class Membership {
2    [Required] public int ID {get; set;}
3    [Required] public Genre Genre {get; set;}
4    public int ClientId {get;set;}
5    public Client Client {get;set;} // Navigational Property
6  }
7  public class Client {
8    [Required] public int ID {get ; set;}
9    [Required] public string FirstName {get; set;}
10   [Required] public string LastName {get; set;}
11   public string Email {get; set;}
12   ...
13   public Membership Membership {get; set;}
14 }
```

Speaker notes

`ClientId` determines in which table the foreign key is placed

- Navigational properties and foreign keys should be on the form
  - `public <ClassType> <ClassType> {get;set;}`
    `public <IdType> <ClassName>Id {get;set;}`

1-1 RELATIONSHIP (2/2)

- The same in Fluent API

```
1 public class MyDbContext: DbContext {
2   protected override void OnModelCreating(ModelBuilder mb) {
3     mb.Entity<Client>()
4           .HasOne(s => s.Membership)
5           .WithOne(l => l.Client)
6           .HasForeignKey<Membership>();
7 }}
```

1-1 RELATIONSHIP (2/2)

- The same in Fluent API

```
1 public class MyDbContext: DbContext {
2   protected override void OnModelCreating(ModelBuilder mb) {
3     mb.Entity<Client>()
4           .HasOne(s => s.Membership)
5           .WithOne(l => l.Client)
6           .HasForeignKey<Membership>();
7 }}
```

1-1 RELATIONSHIP (2/2)

- The same in Fluent API

```
1 public class MyDbContext: DbContext {
2   protected override void OnModelCreating(ModelBuilder mb) {
3     mb.Entity<Client>()
4           .HasOne(s => s.Membership)
5           .WithOne(l => l.Client)
6           .HasForeignKey<Membership>();
7 }}
```

1-N RELATIONSHIP (1/2)

```
1  public class Book {
2    public int ID { get; set;}
3    [MaxLength(32)] public string Title {get; set;}
4    public Author Author {get; set;}
5    public int AuthorId {get; set;}
6  }
7  public class Author {
8      [Key] public int ID {get; set;}
9      public string FirstName {get; set;}
10     public DateTime DoB {get; set;}
11     public string Nationality {get;set; }
12     ...
13     public List<Book> Books {get; set;}
14 }
```

1-N RELATIONSHIP (1/2)

```
1  public class Book {
2    public int ID { get; set;}
3    [MaxLength(32)] public string Title {get; set;}
4    public Author Author {get; set;}
5    public int AuthorId {get; set;}
6  }
7  public class Author {
8      [Key] public int ID {get; set;}
9      public string FirstName {get; set;}
10     public DateTime DoB {get; set;}
11     public string Nationality {get;set; }
12     ...
13     public List<Book> Books {get; set;}
14 }
```

1-N RELATIONSHIP (2/2)

- Or with Fluent API

```
public class MyDbContext: DbContext {
  protected override void OnModelCreating(ModelBuilder mb) {
    mb.Entity<Book>()
      .HasOne(b => b.Author)
      .WithMany(a => a.Books)
      .HasForeignKey(b => b.AuthorId);
}}
```

1-N RELATIONSHIP (2/2)

- Or with Fluent API

```
1 public class MyDbContext: DbContext {
2   protected override void OnModelCreating(ModelBuilder mb) {
3     mb.Entity<Book>()
4       .HasOne(b => b.Author)
5       .WithMany(a => a.Books)
6       .HasForeignKey(b => b.AuthorId);
7 }}
```

1-N RELATIONSHIP (2/2)

- Or with Fluent API

```
public class MyDbContext: DbContext {
  protected override void OnModelCreating(ModelBuilder mb) {
    mb.Entity<Book>()
      .HasOne(b => b.Author)
      .WithMany(a => a.Books)
      .HasForeignKey(b => b.AuthorId);
}}
```

N-M RELATIONSHIP (1/3)

```
1  public class Book {
2      public int BookId
3      ...
4      public ICollection<PersonalLibrary> PeronalLibraries
5                  {get;set;}
6  }
7  public class PersonalLibrary {
8      public int PersonalLibraryId {get;set}
9      ...
10     public ICollection<Book> Books {get; set;}
11 }
```

- This creates a shadow table in database
  - PersonalLibraryBook

N-M RELATIONSHIP (1/3)

```
1  public class Book {
2      public int BookId
3      ...
4      public ICollection<PersonalLibrary> PeronalLibraries
5                  {get;set;}
6  }
7  public class PersonalLibrary {
8      public int PersonalLibraryId {get;set}
9      ...
10     public ICollection<Book> Books {get; set;}
11 }
```

- This creates a shadow table in database
  - PersonalLibraryBook

N-M RELATIONSHIP (2/3)

- Create shadow class

```
1 public class PersonalLibraryBook {
2 public int BookId {get; set;}
3 public Book Book {get; set;}
4 public int PersonalLibraryId {get; set;}
5 public PersonalLibrary PersonalLibrary {get; set;}
6 }
```

- Add navigational properties in classes `Book` and `PersonalLibrary`

```
public List<PersonalLibraryBook> PersonalLibraryBooks
                                        {get; set;}
```

N-M RELATIONSHIP (2/3)

- Create shadow class

```
1 public class PersonalLibraryBook {
2 public int BookId {get; set;}
3 public Book Book {get; set;}
4 public int PersonalLibraryId {get; set;}
5 public PersonalLibrary PersonalLibrary {get; set;}
6 }
```

- Add navigational properties in classes `Book` and `PersonalLibrary`

```
public List<PersonalLibraryBook> PersonalLibraryBooks
                                        {get; set;}
```

N-M RELATIONSHIP (3/3)

- In `OnModelCreating`

```
public class MyDbContext: DbContext {
  protected override void OnModelCreating(ModelBuilder mb)
    // Book - PersonalLibrary (many to many relationship)
    mb.Entity<PersonalLibraryBook>()
        .HasKey(p => new {p.BookId, p.PersonalLibraryId});
    mb.Entity<PersonalLibraryBook>()
        .HasOne(       plb => plb.Book)
        .WithMany(     b   => b.PersonalLibraryBooks)
        .HasForeignKey(plb => plb.BookId);
    mb.Entity<PersonalLibraryBook>()
        .HasOne(       plb => plb.PersonalLibrary)
        .WithMany(     pl  => pl.PersonalLibraryBooks)
        .HasForeignKey(plb => plb.PersonalLibraryId);
}}
```

N-M RELATIONSHIP (3/3)

- In `OnModelCreating`

```
public class MyDbContext: DbContext {
  protected override void OnModelCreating(ModelBuilder mb)
    // Book - PersonalLibrary (many to many relationship)
    mb.Entity<PersonalLibraryBook>()
        .HasKey(p => new {p.BookId, p.PersonalLibraryId});
    mb.Entity<PersonalLibraryBook>()
        .HasOne(       plb => plb.Book)
        .WithMany(     b   => b.PersonalLibraryBooks)
        .HasForeignKey(plb => plb.BookId);
    mb.Entity<PersonalLibraryBook>()
        .HasOne(       plb => plb.PersonalLibrary)
        .WithMany(     pl  => pl.PersonalLibraryBooks)
        .HasForeignKey(plb => plb.PersonalLibraryId);
}}
```

N-M RELATIONSHIP (3/3)

- In `OnModelCreating`

```
public class MyDbContext: DbContext {
  protected override void OnModelCreating(ModelBuilder mb)
    // Book - PersonalLibrary (many to many relationship)
    mb.Entity<PersonalLibraryBook>()
        .HasKey(p => new {p.BookId, p.PersonalLibraryId});
    mb.Entity<PersonalLibraryBook>()
        .HasOne(       plb => plb.Book)
        .WithMany(     b   => b.PersonalLibraryBooks)
        .HasForeignKey(plb => plb.BookId);
    mb.Entity<PersonalLibraryBook>()
        .HasOne(       plb => plb.PersonalLibrary)
        .WithMany(     pl  => pl.PersonalLibraryBooks)
        .HasForeignKey(plb => plb.PersonalLibraryId);
}}
```

RELATIONSHIPS CONFIGURATIONS IN FLUENT API

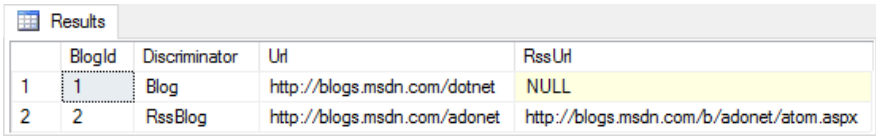
- Required and optional
  - `.IsRequired()` og `.IsRequired(false)`
- Deletion
  - `.OnDelete(DeleteBehavior.Cascade)`
  - Other behaviour (as in SQL) are available
- References non-primary key
  - `.HasPrincipalKey(c => c.BookISBN32)`
- Like primary key, constraint name can be changed
  - `.HasConstraintName("FKey_Book_Libra`

INHERITANCE (1/3)

- By convention derived class are managed in a TPH (table-per-hierarchy) pattern
- A discriminator column to identify type.
- Types should be **explicitly** added as `DbSet` to `MyDbContext` or in Fluent API

```
modelBuilder.Entity<RssBlog>().HasBaseType<Blog>();
```

INHERITANCE (2/3)



Results

	BlogId	Discriminator	Url	RssUrl
1	1	Blog	http://blogs.msdn.com/dotnet	NULL
2	2	RssBlog	http://blogs.msdn.com/adonet	http://blogs.msdn.com/b/adonet/atom.aspx

INHERITANCE (3/3)

- Discriminator is a database attribute and can be manipulated
  - Used to tell about type
  - Use Fluent API to change name values

```
modelBuilder.Entity<Blog>()
  .HasDiscriminator<string>("blog_type")
  .HasValue<Blog>("blog_base")
  .HasValue<RssBlog>("blog_rss");
```

EXERCISES

REFERENCES

- Insanity: https://www.brainyquote.com/quotes/unknown_133991
- ERD: https://ermodelexample.com/how-to-draw-erd-diagram/
- XKCD: https://imgs.xkcd.com/comics/magnus.png