Laborationsrapport

Moment 3.2 – Entity Framework Core
DT102G, ASP.NET med C#

Författare: Rebecca Jönsson Seiron, rese1800@student.miun.se
Termin, år: VT, 2020

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

Sammanfattning

Uppgiften går ut på att visa att man förstår grunderna i Entity Framework Core. En databasdriven webbapplikation ska skapas som innehåller en fiktiv CD-samling. Uppgiften är uppdelad i 2 delar där den första delen enbart ger godkänt och den senare delen ger möjlighet till ett högre betyg.

Uppgift 1:

- Att ni använder models, controller och vyer.
- Att ni använder Entity Framework Core
- Att ni använder er av code first när ni skapar databasen
- Du ska kunna skapa, redigera, uppdatera och ta bort information om CD-skivor.
- Ni ska dels kunna få upp alla CD, samt klicka för att få fram information om just den CD:n.
- Förstå hur hur ni ska navigera och modifiera i inbyggda SQL server
- Förstå hur klassen DbContext används och hur klasser ärver från denna klass

Uppgift 2:

- Ni ska kunna låna ut CD till användare och er databas ska hålla reda på vem som har skivan och datum när detta gjordes
- Ni ska skapa minst en till tabell för att spara data om den som lånar CD.
- Skapa en sökfunktion för CD.

Både uppgift 1 och 2 är lösta.

Innehållsförteckning

1 Konstruktion

Jag började med att kolla på de videos som fanns under ”Teroi och läsanvisningar”[1]. Därefter funderade jag över hur mina tabeller i databasen skulle se ut och hur webbplatsen skulle fungera.

Det jag ville ha med var följande tabeller:

Cd:

- CdId – PK, int
- Namn - string
- ReleaseDate - DateTime
- Avalable – bool
- ArtistId – FK, int

Artist:

- ArtistId – PK, int
- Name - string

Track:

- TrackId – PK, int
- Name – string
- cdId – FK, int

Rent:

- RentId – PK, int
- Name – string
- RentDate – DateTime
- cdId – FK, int

Ett nytt projekt skapades med MVC mönstret. En modell med namn ”Cd” skapades med klasser som senare kommer finnas representerade i databasen, jag följde en bra guide som gav mig lite förståelse för hur man sätter upp det med foreignkeys[2]. Nödvändiga NuGet paket installerades:

- Microsoft.VisualStudio.Web.CodeGeneration.Design
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityframeworkCore.Design

Controllers skapades genom ”scaffolding” där modell klassen kopplades mot ”rätt” klass och data context klassen kopplades mot ”rätt” klass.

I startup.cs lades kod till gör att kunna koppla upp sig mot databasen:

```
services.AddDbContext<CdContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultDbString")
));
```

samt:

```
using moment3._2.Data;
```

I appsettings.json la jag till ”connection strängen”:

```
 "ConnectionStrings": {
"DefaultDbString":"Server=(localdb)\\
mssqllocaldb;Database=Cd;Trusted_Connection=True;MultipleActiveResultSets
=true"
  }
```

För generera databasen gick jag in i tools→NuGet Packagemanager → Package Manager Console och skrev:

Add-Migration InitialCreate
Update-Database

En migrationsfil med tillhörande mapp skapades samt dabasen.
Jag gjorde faktiskt om min cd modell ett par gånger innan jag förstod precis hur man skulle göra för att få till den tabellstruktur jag ville ha. Jag fick lite problem när jag försökte få in min ”track” i min cd-klass eftersom jag redan hade en ICollection för track där. Jag ville ha båda för att kunna läsa ut låtar i min ”cd-details”. Här var Visual studio till stor hjälp (genom ”lampan”) och meddelade att jag var tvungen att sätta en [NotMapped].

När databasen var på plats testades funktionerna på webbplatsen, för att snabbt kunna navigera ändrades menyn i _Layouts och routing i startup.cs (som nu pekar mot controllern Atrist).

1.1 Artist

Varken controller eller views för ”Artist” ändrades (bortsett från att jag bytt från engelska till svenska i viewn).

1.2 Cd

I CdsController och views ändrades destå mer. Det var endel från andra klasser jag ville hämta in till controller och views. I denna finns även sökfunktionen för Cd.

```
// GET: Cds
3 references
public async Task<IActionResult> Index(string searchString, int id)
{

    var cdContext = _context.Cds.Include(c => c.Artist);

    var cds = from m in _context.Cds
                select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        cds = cds.Where(s => s.Name.Contains(searchString));
       foreach(var t in cdContext)
        {
            var x = t.ArtistId;
            if(x == id)
            {
                ViewData["test"] = t.Name;

            }
        }


        return View(await cds.ToListAsync());
    }

    cds = _context.Cds.Include(c => c.Artist);



    var newdate = _context.Cds.Include(x => x.ReleaseDate);

    ViewData["Dates"] = newdate;
    return View(await cdContext.ToListAsync());
}
/*Search */
```

Jag skapade sökfunktionen i viewn med hjälp av en guide[3] och jag tog både kod till viewn och controllern rakt av. Det blev lite bekymmer när jag behövde hämta in ”Artist namn” eftersom man inte kan returnera flera properties. Jag valde därför att skicka med artistnamn i en ViewData. För att få fram det jag ville ha behövde jag loopa igenom cdContext och enbart plocka ut namn där ArtistId stämde överens med cdid, för att plocka idt ”utifrån” passades detta med som en ”int id” i Index(string searchString).

```
0 references
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var cd = await _context.Cds
        .Include(c => c.Artist).Include(c => c.Tracks)
        .FirstOrDefaultAsync(m => m.CdId == id);
    string date = cd.ReleaseDate.ToString("yyyy-MM-dd");
    ViewData["Date"] =date;

    if (cd == null)
    {
        return NotFound();
    }

    return View(cd);
}
```

I details viewn villa jag ha med låtar, för att lösa detta la jag bara till en .Include(c => c.Tracks) efter den redan förskrivna .Include(c => c.Artist). Jag ville enbart visa datum och inte tid så jag passade på att konvertera bort det och skicka vidare det nya datumet i en ViewData[”Date”] som lästes ut i viewn.

Resten av funktionerna i CdsController är i stort sett original bortsett från att jag bytt namn i de selectedlists som finns (så man inte bara får ut id utan även namn).

1.3 Track

Även TracksController är orörd.

1.4 Rent

I RentsController är lite tillagt, dels vill jag kunna ändra en cdskivas status (Avalable) till false när den är utlånad, ändra till true när den lämnas tillbaka och filtrera bort hydra skivor från ”lånelistan”.

```
public IActionResult Create()
{

    var idAndName = _context.Cds
        .Where(x => x.Avalable == true)
        .Select(x => new { x.CdId, x.Name });

    ViewBag.cdId = new SelectList(idAndName, "CdId", "Name");

    return View();
}
```

I viewn Create ville jag kunna läsa ut både och namn men enbart på de skivor som är ”Avalable”. Jag plockade då fram det jag behövde genom att filtrera från databasen och plocka namn och id enbart från rader där ”Avalable == true”. För att lösa query-delen hittade jag ett bra inlägg på Stackoverflow[4]

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create([Bind("RentId,Name,RentDate,cdId")] Rent rent)
{


    if (ModelState.IsValid)
    {
      Cd result = (from p in _context.Cds
                   where p.CdId == rent.cdId
                   select p).SingleOrDefault();
       result.Avalable = false;

       _context.SaveChanges();

       _context.Add(rent);
       await _context.SaveChangesAsync();
       return RedirectToAction(nameof(Index));
    }
    ViewBag.cdId = new SelectList(_context.Cds, "CdId", "Name");

    return View(rent);
}
```

För att kunna ändra status på den lånade cdskivan (”Avalable” ligger ju i tabellen CD och inte i tabellen rent) så görs ungefär samma typ av querie som vid create(), därefter görs en _context.SaveChanges(); den sista delen hittade jag också på stackoberflow[5].

```
// GET: Rents/Edit/5
0 references
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var rent = await _context.Rents.FindAsync(id);
    if (rent == null)
    {
        return NotFound();
    }
    var idAndName = _context.Cds
  .Where(x => x.Avalable == true)
  .Select(x => new { x.CdId, x.Name });



    ViewBag.cdId = new SelectList(idAndName, "CdId", "Name");
    return View(rent);
}
```

Edit är ganska lik create(), precis som i create() vill jag ha ut en lista i min view med både id och namn.

```
    // POST: Rents/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    0 references
    public async Task<IActionResult> DeleteConfirmed(int id)
    {

        var rent = await _context.Rents.FindAsync(id);

        _context.Rents.Remove(rent);

            Cd results = (from p in _context.Cds
                          where p.CdId == rent.cdId
                          select p).SingleOrDefault();
            results.Avalable = true;

            _context.SaveChanges();



        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    1 reference
    private bool RentExists(int id)
    {
        return _context.Rents.Any(e => e.RentId == id);
    }
}
```

Vid delete görs ungefär samma sak som i create(). När en ”rent” raderas (skivan återlämnas) vill jag att den åter igen ska vara ”Avalable” och ”Avalable” blir nu satt till ”true”.

1.5 Views

I views sker inga större förändringar. En del text byts från engelska till svenska, alla ställen som har ”fel” tidsformat har konverterats och de ViewData som skapats lagts in. I views för rent (create och edit) har input för datum (”RentDate”) gjorts till hidden och ett value har sats till ”DateTime.now”, jag kunde inte riktigt få till det där med timestamp i min modell så jag löste det såhär istället.

2 Slutsatser

Det var en rolig uppgift som gick relativt snabbt att göra, det mesta var ju redan ”förkodat”. Från föregående uppgift så undrade jag över hur de gjort när de hämtade in idn vid en update och delete, nu såg jag hur de gjort och det blev ju ganska självklart men det var kul att se att de använt sig av input type=”hidden” för att passa vidare idn, jag gjorde samma sak i föregående och tänkte att ”detta kan inte vara rätt egentligen, det är ju bara att inspektera i koden och ändra id manuellt om man vill förstöra något”.

Att använda code-first var helt nytt för mig, tänker framförallt på ”tänket”, vet inte hur många gånger jag svor för att jag inte fick till min databas som jag hade tänkt och tänkte att det hade varit mycket enklare att skapa den i typ phpmyadmin först istället. När polletten ramlade ner så förstod jag hur smidigt detta faktiskt är, om man kan med det dvs.

3 Källförteckning

[1] Moodle, ”Entity Framework Core – Teori och läsanvisningar”
https://elearn20.miun.se/moodle/mod/resource/view.php?id=593087
Publicerad Okänt. Hämtad 2020-03-01

[2] Entity Framework Tutorial, ”Data Annotations- ForeignKey Attribute in EF 6 & EF Core”,
https://www.entityframeworktutorial.net/code-first/foreignkey-dataannotations-attribute-in-code-first.aspx
Publicerad Okänt. Hämtad 2020-03-01

[3] Microsoft, ”Add search to an ASP.NET Core MVC app”
https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/search?view=aspnetcore-3.1
Publicerad 2018-12-13. Hämtad 2020-03-01

[4] Stackoverflow, ”C# LINQ How to get a data source from a db”
https://stackoverflow.com/questions/7570707/c-sharp-linq-how-to-get-a-data-source-from-a-db
Publicerad 2012. Hämtad 2020-03-01

[5] Stackoverflow, ”Fastest Way of Inserting in Entity Framework”
https://stackoverflow.com/questions/5940225/fastest-way-of-inserting-in-entity-framework
Publicerad 2012. Hämtad 2020-03-01