Laboration I - CSN
Databaser och webbgränssnitt

I denna laboration ska ni arbeta med en databas som innehåller data för ett svar från CSN. Svaret innehåller i sin tur uppgifter om studiemedel för en person. För att förbereda er inför detta arbete så har ni skapat en konceptuell modell över en XML-fil som representerar detta svar från CSN. Anledningen till varför ni har skapat denna modell är för att det är betydligt enklare att se hur databasen är strukturerad samt hur de olika entiteterna är sammankopplade i jämförelse mot att inspektera XML-filen eller de olika tabellerna i databasen. Er uppgift är sedan att hämta relevant data från databasen och presentera det i ett webbaserat gränssnitt.

Processen för vår applikation kan tänkas se ut enligt följande:

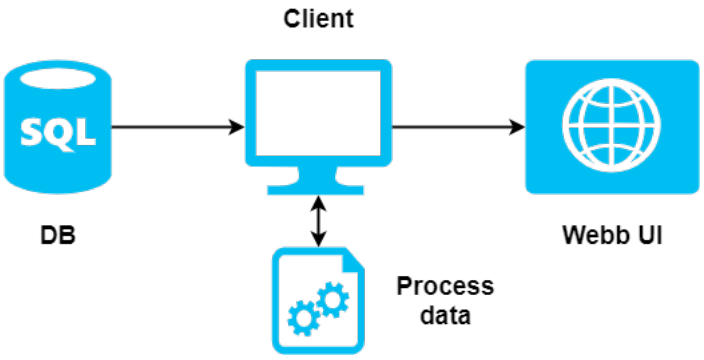

Fig. 1 – Processflödet för applikationen.

Där vi (1) hämtar information från databasen i klienten, (2) behandlar informationen och strukturerar den på så vis att den dels (2a) enbart innehåller den information vi vill presentera och dels (2b) att den följer XML-strukturen som vi vill nyttja vid presentation och sedan så (3) presenteras detta innehåll i ett webbaserat gränssnitt.

Denna laboration ska genomföras i projektgrupperna.

Innehållsförteckning

1 Introduktion

Vi har nu en bild av hur processen ser ut för vår applikation, men vi vet fortfarande inte hur man ska gå till väga för att "sätta ihop" alla delar.

1.1 Ladda ned skalprojektet

Det första steget blir att ladda ned skalprojektet från Studentportalen vilket du hittar där du hittade dessa instruktioner. Se till att packa upp applikationen klokt och placera den förslagsvis på hårddisk U:, under förutsättningen att du arbetar i laborationssalarna eller via fjärranslutning.

1.2 MVC applikation

När vi sedan öppnar projektet i Visual Studio så bemöts vi av strukturen för en s.k. MVC-applikation (Model View Controller). Vi kommer dock inte att använda denna struktur så som det är tänkt utan vi kommer enbart att arbeta med View- och Controller-delarna av strukturen. Det är i vår Controller-klass som vi kommer att skriva koden för att hämta informationen från vår databas samt strukturera informationen i ett XML format, m.h.a. LINQ-to-XML och functional construction.

Controllers Ni hittar den Controller-klass som vi kommer att arbeta med, `CSNController`, under mappen "Controllers". När ni öppnar denna klass så bemöts ni av en konstruktor som innehåller koden för att etablera en koppling till vår databas, metoden `Test` vilket är ett exempel som ni kan nyttja för att se hur dessa typer av applikationer fungerar samt hur den övergripande strukturen för de metoder ni ska implementera senare kommer att se ut och metodsignaturerna för `Uppgift1`, `Uppgift2` samt `Uppgift3` vilka ni ska implementera i denna laboration.

Varje metod i Controller-klassen returnerar en s.k. `ActionResult`. Vi kommer inte att gå in på vad exakt detta innefattar eller hur en MVC-applikation är strukturerad, men det vi behöver veta är att varje Controller-metod returnerar ett resultat till en View och i vårt fall så kommer detta att vara en XML (som i C# representeras av ett `XElement`). Om du är intresserad av att läsa mer om `ActionResult` och MVC så kan du nyttja följande länkar:

MVC
ActionResult

Om vi nu kollar på metoden `Test` så ser vi att vi har en övergripande struktur som består av fyra delar:

```
public ActionResult Test()
{
    string query = @"SELECT a.Arendenummer, s.Beskrivning, SUM(((Sluttid-startti
                    FROM Arende a, Belopp b, BeviljadTid bt, BeviljadTid_Belopp
                    WHERE a.Arendenummer = bt.Arendenummer AND s.Stodformskod =
                    AND btb.BeloppID = b.BeloppID AND btb.BeviljadTidID = bt.Bev
                    Group by a.Arendenummer
                    Order by a.Arendenummer ASC";
    XElement test = SQLResult(query, "BeviljadeTider2009", "BeviljadTid");
    XElement summa = new XElement("Total",
        (from b in test.Descendants("Summa")
         select (int)b).Sum());
    test.Add(summa);

    // skicka presentations xml:n till vyn /Views/Csn/Test,
    // i vyn kommer vi åt den genom variabeln "Model"
    return View(test);
}
```

Fig. 2 – Test-metoden.

- 1: En SQL-query i sträng-format
- 2: Vi skickar sedan denna SQL-query som en förfrågan till vår databas och lagrar resultatet i en XML.
- 3: Vi nyttjar sedan Functional Construction för att bygga ihop vår egna XML-struktur.
- 4: Vi skickar sedan denna XML till vår View för att presentera informationen i vårt gränssnitt.

Noterbart är att vi i steg 2 konverterar det svar vi får från databasen till en XML och samtidigt namnger vår rotnod samt vår omslutande nod. Den omslutande noden representerar varje "objekt" i vår XML, t.ex. som att vi i det förberedande materialet har ett flertal filmer under `<Movie>`-noder. Anledningen bakom att konstruera vår egna XML i steg 3 är för att vi t.ex. vill lägga till en totalsumma i vår XML som vi inte får ut direkt genom vår SQL-query.

Views När informationen är färdigstrukturerad så skickar vi sedan informationen till vår View som en XML. I resp. View-klass så skriver vi sedan en blandning av HTML och C# för att säga hur vi vill att informationen ska presenteras, t.ex. med hjälp av `<table>`-taggar för att visa innehåll i en HTML-tabell.

Varje metod i Controller-klassen har alltså en motsvarande View-klass. Ni hittar dessa View-filer i mappen "CSN" under mappen "Views" (Fig. 2):

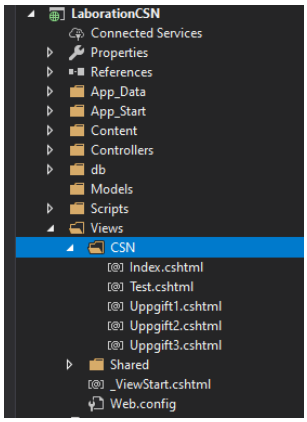

Fig. 3 – Views.

Om vi exempelvis öppnar filen Test.cshtml så bemöts vi av följande kod (Fig. 3):

```
@model System.Xml.Linq.XElement
@{
    ViewBag.Title = "Exempel: Beviljade tider 2009";
}
<h2>Exempel: Beviljade tider 2009</h2>
<table table class="table-bordered table_50">
    <thead>
        <tr>
            <th>
                Ärendenummer
            </th>
            <th>
                Stödform
            </th>
            <th>
                Summa
            </th>
        </tr>
    </thead>
    <tbody>
        @*..detta är en kommentar..*@
        @*Iterera igenom alla <BeviljadTid> noder i Model som håller vår xml*@
        @foreach (var beviljad in Model.Elements("BeviljadTid"))
        {
        <tr>
            <td>
                @((string)beviljad.Element("Arendenummer"))
            </td>
            <td>
                @((string)beviljad.Element("Beskrivning"))
            </td>
            <td>
                @((string)beviljad.Element("Summa"))
            </td>
        </tr>
        }
        <tr>
            <td></td>
            <td></td>
            <td>
                Total: @(Model.Element("Total").Value)
            </td>
        </tr>
    </tbody>
</table>
```

Fig. 4 – Test.cshtml

Vi ser då högst upp att det objekt som vi har skickat till vyn är en XML (`XElement`) följt av en del HTML där vi har en rubrik (`<h2>`) och en tabell (`<table>`). Vi ser även att vi kan nyttja C# genom att placera ett @ framför `foreach`-loopen vilket möjliggör att vi kan iterera över vår XML och fylla i tabellens celler för varje, i det här fallet, `<BeviljadTid>`-nod som finns i vår XML.

Vi rekommenderar att ni placerar breakpoints i metoden `Test` samt i filen Test.cshtml och på så vis följer med i vad som händer varje steg innan ni går vidare till de uppgifter ni faktiskt ska implementera. Ett tips i förhållande till detta är att nyttja DataTips-verktyget i Visual Studio vilket tillåter oss att bl.a. kontrollera innehållet i det svar vi får från databasen. För att nyttja verktyget så placerar vi en breakpoint för metodanropet som vi vill testa. Vi kan därefter kontrollera innehållet i vår variabel och se om den innehåller den förväntade datan enligt Fig. 4:

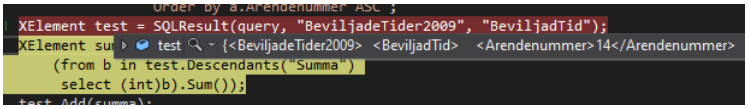

Fig. 5 – DataTips-verktyget.

Det kan däremot vara ganska svårt att tyda bara utifrån den långa raden vi kan se i Fig. 5. Ett sätt att lösa detta på är genom att nyttja ett annat verktyg, Data Visualizer. Man får tillgång till detta verktyg genom att klicka på förstoringsglaset som vi kan se i Fig. 5. Vi kan därefter välja om vi vill se det i XML-format, JSON-format, HTML eller text. Som vi kan se i Fig. 6 är detta väldigt smidigt för just XML.

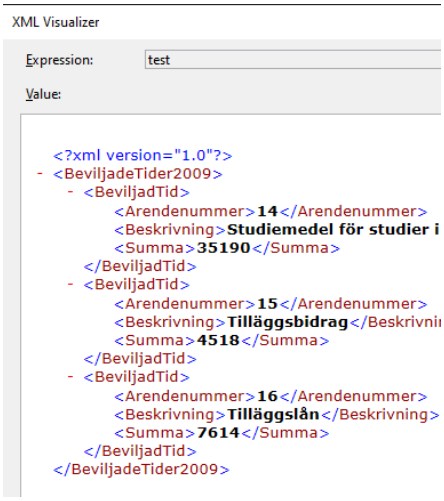

Fig. 6 – DataVisualizer-verktyget.

1.3 SQLite-databas

Utöver nyttjandet av de inbyggda verktygen i Visual Studio samt den konceptuella modell som ni har producerat över CSN-XML:en så finns det även en poäng i att kunna arbeta direkt mot databasen samt överblicka de olika tabellerna. Börja därför med att ladda ned filerna "SQLiteDBBrowser.exe" samt "csn.sqlite" som du hittar på Studentportalen.

Installera sedan programmet på hårddisk U: (om du arbetar i laborationssalarna eller via fjärranslutning) och välj förslagsvis en mapp som du enkelt kan navigera till. Kör därefter programmet.

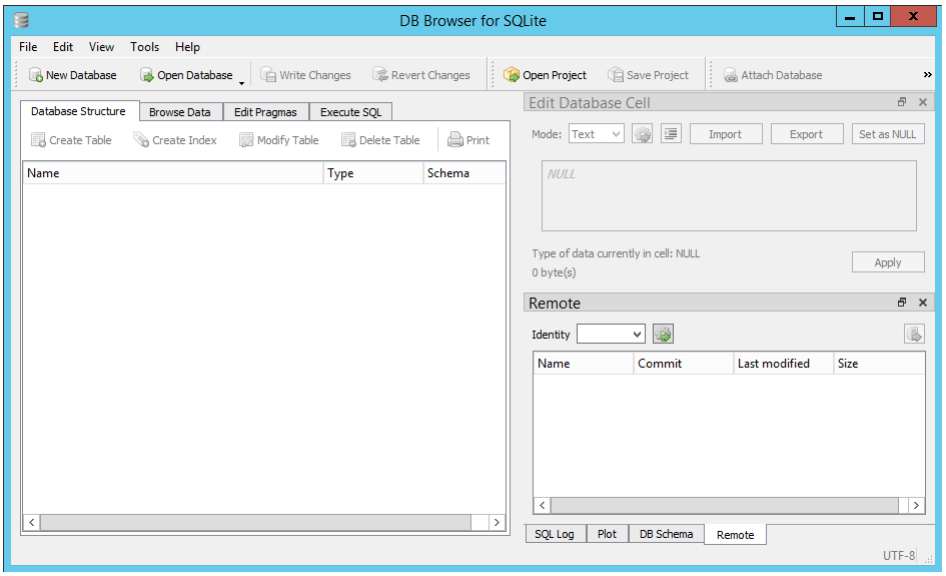

Fig. 7 – DB Browser.

Du möts då av gränssnittet i Fig. 7. Välj sedan att öppna en databas och navigera till filen "csn.sqlite" som du laddade ned tidigare. När du öppnar databasen så ser du samtliga tabeller och dess resp. kolumner.

I detta verktyg så kan du även navigera till Browse Data, där du kan se den faktiska datan som finns i resp. tabell. Det finns även möjlighet till att köra SQL queries genom att navigera till Execute SQL.

Vi rekommenderar starkt att ni skriver era SQL-queries i detta verktyg och sedan kopierar in frågan i Visual Studio när ni har fått ut korrekt resultat. Verktyget tillhandahåller exempelvis felmeddelanden och auto-complete för t.ex. tabellnamn och kolumner (Fig. 8).

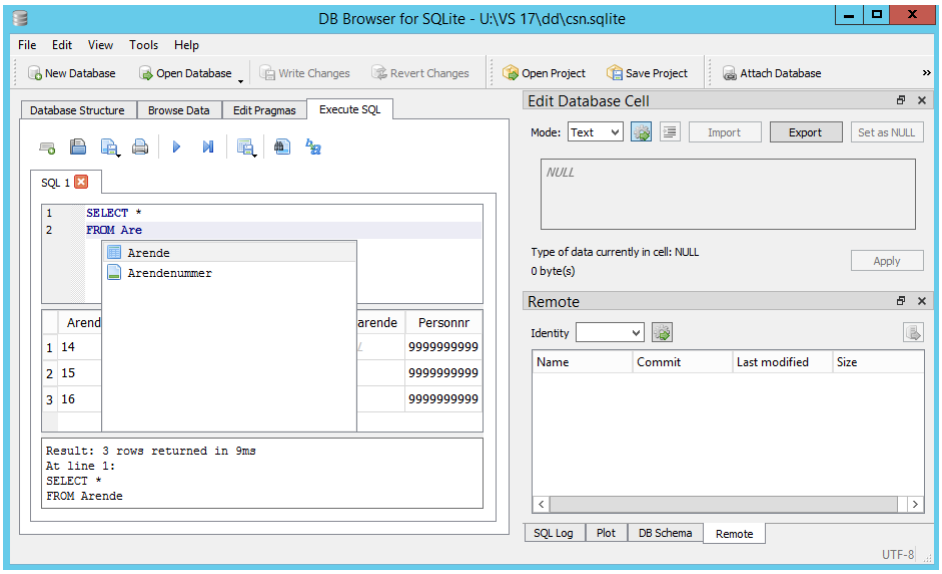

Fig. 8 – Execute SQL.

2 Uppgifter

Nedan finner ni de tre uppgifter som ni förväntas implementera för denna laboration. Tänk på att utgå från den konceptuella modell ni har producerat i samband med seminariet den 25/2 när ni skriver era SQL-queries. För att säkerställa att ni har fått ut korrekt information samt för att få en bild över hur vyn ska struktureras så ska ni tillgå följande lösningsförslag:

Lösningsförslag

Uppgift 1 Utbetalningar per ärende

Visa upp alla utbetalningar för varje ärende, för varje utbetalning ska utbetalningsdatum, utbetalningsstatus och summan visas. För varje ärende ska även den totala summan, den redan utbetalda summan och den kvarvarande summan presenteras. För varje ärende ska även dess löpnummer vara presenterat.

TIPS! GroupBy (SQO) eller group clause (Query Expression) är bra att använda för att lösa uppgiften. Förklaring av vad det är samt exempel på hur det nyttjas hittar du via följande länk: Group Clause. I förhållande till SQL-queryn så är det relevant att veta att man kan gruppera på fler än en kolumn. Passa även på att analysera den XML-fil som vi tidigare har arbetat med (som du hittar under App_Data mappen i projektet). Vi saknar exempelvis data för noden `<totbelopp>` i vår databas, så hur räknas detta värde ut?

Tänk på att jämföra ditt resultat mot lösningsförslaget innan du går vidare till nästa uppgift för att veta om du har implementerat uppgiften korrekt.

Uppgift 2 Utbetalningar och bidragstyper

För varje utbetalningsdatum visa den totala summan för alla utbetalningar för det datumet, visa även uppdelningen mellan de olika bidragstyperna för den totala summan. **TIPS!** GroupBy (SQO) eller group clause (Query Expression) är bra att använda för att lösa uppgiften.

Tänk på att jämföra ditt resultat mot lösningsförslaget innan du går vidare till nästa uppgift för att veta om du har implementerat uppgiften korrekt.

Uppgift 3 Beviljade Tider

Visa upp alla perioder som personen är beviljad någon typ av bidrag. För varje period ska ett tidsintervall, summa och typ av bidrag presenteras. Notera att typ av bidrag som efterfrågas i denna uppgift syftar till en nod som ligger direkt under ”Arende” noden.

Tänk på att jämföra ditt resultat mot lösningsförslaget för att veta om du har implementerat uppgiften korrekt.

3 Vanliga fel

Nedan beskrivs vanligt förekommande fel som kan uppstå i denna laboration samt förslag på hur man kan lösa dessa fel.

Applikationen startar inte när jag klickar på start
Om du märker att applikationen stängs ned av sig självt vid körning och du ser ett felmeddelande i stil av: "'iisexpress.exe' has exited without a trace" så löses detta problem enklast genom att byta port för hela projektet. Du gör detta genom att högerklicka på projektet i Solution Explorer och navigera till Properties (Fig. 9)

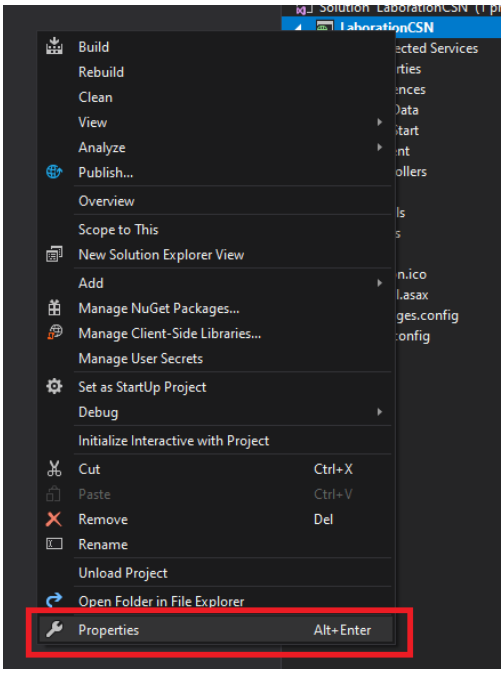

Fig. 9 – Egenskaper för ett projekt.

Väl inuti Properties så navigerar du till "Web" och ändrar sedan portnummer i Project URL-länken (Fig. 10).

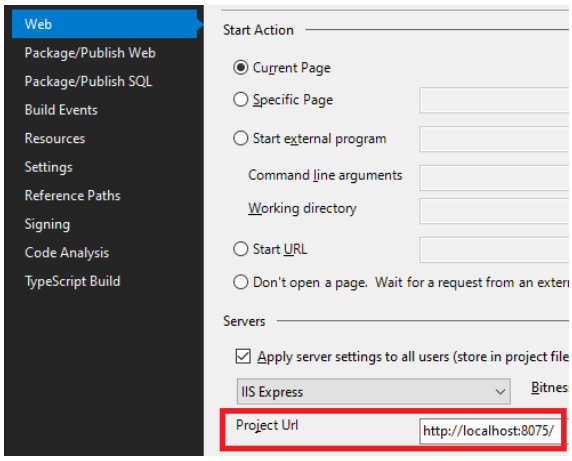

Fig. 10 – Project-URL länken.

4 Examination

Säkerställ att allting är korrekt implementerat genom att jämföra din implementation mot lösningsförslaget, att all funktionalitet är uppfylld och att applikationen inte kraschar. Laborationen ska genomföras och lämnas in i projektgrupperna.