YRKESHÖGSKOLAN NOVIA

Import av data till Excel från en SQL databas

Patrik Hafström

Examensarbete för Tradenom (YH)-examen

Utbildningsprogrammet i Informationsbehandling

Raseborg 2019

EXAMENSARBETE

Författare: Patrik Hafström

Utbildning och ort: Informationsbehandling, Raseborg

Handledare: Kim Roos

Titel: Import av data till Excel från en SQL databas

Datum 29.01.2019 Sidantal 34 Bilagor -

Abstrakt

Inom den företagsekonomiska världen finns det ett ständigt behov att presentera data i olika former. Det kan vara periodredovisningar eller grafiska analyser som ska levereras åt en kund, eller kanske man använder en analytisk översikt för eget bruk.
Hur som helst, så finns siffror och data alltid sparade i databaser, och i de allra flesta ekonomiförvaltningar eller informationsbaserade organisationer använder man sig av dessa.

Med hjälp av Microsoft Excel kan man bygga de mest komplexa rapporterna och analyserna. Men det kanske inte alltid är så lätt att få ut stora mängder data från en databas till Excel om man inte vet hur man ska gå till väga. Det är där programmeringsspråket VBA, som finns inbyggt i Microsoft applikationerna, kommer in i bilden.

Jag kommer i avhandlingen berätta hur man med hjälp av VBA kod kan skapa ett program i Excel som kopplar upp sig mot en databas, och extraherar data från den. Avhandlingens huvudsyfte är att fungera som en slags handbok eller manual som man kan ha som hjälpmedel till hur man i praktiken kan använda sig av VBA för att importera stora mängder data till Excel, och sedan kunna presentera det och bygga upp snygga visuella modeller av den data.

Språk: Svenska Nyckelord: VBA, Excel, programmering, databas

BACHELOR'S THESIS

Author: Patrik Hafström
Degree Programme: Business Information Technology, Raseborg
Supervisor(s): Kim Roos

Title: Data Import to Excel from an SQL Database

Date 26.01.2019 Number of pages 34 Appendices -

Abstract

In the business world, there is a constant need to present data in various forms. It may be period reports or graphical analyses to be delivered to a customer, or you may use an analytical overview for your own use.
Numbers and data are always stored in databases, and they are used in most finance administrations or information-based organizations.

With the help of Microsoft Excel, you can build the most complex reports and analyses. But it may not always be so easy to get large amounts of data from one database to Excel if you do not know how to do it. This is where the programming language VBA, which is built into the Microsoft applications, comes into play.

In the thesis I will tell you how to use a VBA code to create a program in Excel that connects to a database, and extracts data from it.
The main purpose of the thesis is to act as a kind of guide or manual that one can have as a handbook to how use VBA in practice to import large amounts of data to Excel, and then be able to present it and create visually good-looking models of that data.

Language: Swedish Key words: VBA, Excel, programming, database

---

Innehållsförteckning

1 Inledning

Mitt examensarbete kommer att uppgöra grunden för en ingående analys av Visual Basic for Applications (VBA) programmeringsspråket och implementering av sådan kod i Microsoft Excel. Jag har studerat vilka möjligheter detta skapar och vad man kan bygga för program i ett företagsekonomiskt syfte, om man kombinerar framtagandet av data till en Excelmodell från en databas med VBA kod och implementerar detta i samband med till exempel en pivottabell i Excel för att framställa data från databasen på ett visuellt snyggt sätt.

Min avhandling kan delas upp i några olika områden, nämligen introduktionen, den teoretiska delen, den praktiska delen, och den avslutande delen. Först diskuterar jag syfte för avhandling och varför det kom sig att jag just fokuserade på detta ämne, som kan tyckas vara ett ganska specifikt område att studera. Jag behandlar även mina metoder för uppbyggnaden av arbetet och tillvägagångssätten jag använt för att få fram avhandlingen. Därefter kommer vi in i själva arbetets mer teoretiska delar. Jag börjar med att diskutera Visual Basic, det händelsestyrda programmeringsspråket av Microsoft som i själva verket VBA grundar sig på, så att läsaren ska få en bättre helhetsbild av programmeringssättet. Jag kommer förstås inte kunna lära en nybörjarprogrammerare ett helt programmeringsspråk från grunden, men jag försöker ge en så klar helhetsbild av språket som möjligt. Detta så att man kan greppa hur VBA fungerar bättre än om man inte ens hade hört talas om Visual Basic tidigare. Efter att Visual Basic är behandlat kommer avhandlingen att gå vidare till VBA kapitlet. Där diskuteras bakgrunden till vad VBA egentligen är och hur det är uppbyggt från Visual Basic. Sedan behandlas den syntax, dvs. sättet ett programmeringsspråk skrivs på (Beal u.å.), som krävs för att kunna föra över data med hjälp av kodstycken från databaser till applikationerna. Där beskrivs syntaxen grundligt med hjälp av bilder på koden i programmet. Dock kommer jag inte att behandla väldigt mycket övrig kodsyntax, utan endast funktionen av de centrala delarna i avhandlingens syfte. Till exempel kommer jag i avhandlingen att ta upp vad VBA programmet kräver för att kunna utföra de centralaste kommandon, som till exempel hur man tar kontakt med databasen via VBA, hur man med hjälp av SQL satser beskriver vilka specifika tabeller och parametrar man vill ta fram från databaserna, hur man sorterar stora mängder data och läser in dem på rätt ställen i stora Excelprogram, och även hur man stänger databasförbindelsen med kod. Därför kommer jag i samma veva att även nämna, ganska kort, vad SQL är och hur jag använt mig av det. Jag tar också här endast upp det

som är relevant för avhandlingen, som att man måste vara lite bekant med SQL när man hämtar data ifrån databasen med VBA. Jag kommer inte gå in närmare på SQL i denna avhandling än att nämna dess definition, och i samband med VBA kodandet kommer det finnas SQL satser i samma kodinstanser. Detta eftersom det är det sättet man kommunicerar med en databas på.

Efter att de teoretiska delarna har behandlats, går vi över till de mer praktiska delarna. Där tar jag fram hur man skapar kodmoduler i Excel, och skriver själva VBA koden i programmet för att få kontakt till en databas, och tar fram data därifrån. Detta är avhandlingens tyngdpunkt och i princip all teori kretsar kring detta kapitel. Jag kommer också, lite grundläggande, att behandla pivottabeller och dess uppbyggnad i Excel. Detta i och med att i kombination med att man tagit fram data från en databas, kan man framställa resultat på ett överskådliggörande sätt med pivottabeller. Detta speciellt om man har en större mängd data man vill visualisera i något syfte.
Den praktiska delen går jag igenom stegvis med att förklara de olika delarna och kodinstanserna medan en figur över koden i fråga visas. Alla figurer som används som exempel i detta arbete är mina egna som jag skrivit själv.

2 Syfte och bakomliggande idé

Valet bakom idén till min avhandling kom sig ganska naturligt efter ett tag in i mina studier. Databaser och programmering blev allt mer intressanta för mig och SQL blev en favorit att jobba med. Jag började jobba på IT-avdelningen på en bokföringsbyrå och ekonomiförvaltning efter en praktikperiod, och fick lära mig att i väldigt många fall är Microsoft Excel en av byggstenarna till budgetprogrammen och ekonomianalyserna som levereras till kunderna som kan vara allt från stora till små företag.

När man pratar om jobb som görs i Excel kan det vara många som rynkar på näsan och tänker att det nog skulle finnas bättre lösningar som levererar datauppställningar på bättre sätt, men då ska man också komma ihåg att ungefär upp till 50% som använder, eller bekantat sig med, Excel inte använder 80% av programmets alla funktioner och möjligheter. (Jelen & Alexander 2006, 1). Jag började arbeta med att programmera i Excel med VBA och SQL kod för att bygga olika analys- och budgeteringsprogram med data från väldigt stora databaser. Då växte syftet till min avhandling fram.

Man kan säga att det primära syftet med mitt arbete är att kunna framställa en så tydlig och lättförståelig manual eller ”handbok” som behandlar hur man kan använda VBA i ett program som Excel för att lätt kunna få stora mängder data från en databas. Man kan såklart använda sig av VBA till mycket mer än att kommunicera med databaser. Man kan till exempel automatisera en stor del av ett Excel program genom att skriva så kallade makron, som är små handlingar eller utföranden, i VBA om inte Excels färdigt inbyggda funktioner skulle vara tillräckliga. Men syftet av detta arbete kommer ligga på ett företagsekonomiskt plan, så fokuset kommer jag hålla på databaskommunikation och VBA kodens funktion i byggandet av olika analysprogram. Till exempel kan en bokföringsbyrå ha väldigt bra nytta av denna metod för att få fram stora mängder data för en kund och framställa en ekonomisk överblick för företagets del. Finns det stora mängder data för en kund lagrade i databaser och man vill framställa exempelvis budgeteringsprogram, pivotrapporter, eller andra analyser för en kunds del kan man använda sig av lätta kodsatser skrivna i VBA, för att därifrån extrahera data från databaserna till Excel där man sedan bygger vidare applikationer för att framställa siffrorna på ett snyggt och presentabelt sätt. Vilket de flesta kanske inte tror är möjligt, eller i alla fall inte i den grad som det går att göra. Men med VBA programmering i Excel kan man göra väldigt avancerade program och komplexa analyser.

3 Metoder för framställandet av avhandlingen

Det finns en hel del frågor som man kan behandla kring detta ämne. Till exempel hur man programmerar i Excel VBA, hur koden fungerar i samband med databaskommunikation, och vad man till slut ska göra med datamängderna som man importerat in till Excel programmet i fråga. Jag ska försöka svara och förklara dessa i detta arbete.

Den huvudsakliga metoden jag använt mig av för att bygga upp mina kunskaper om ämnet, är egentligen bara att sätta mig ner och börja programmera. Detta är oftast grundpelaren i att lära sig ett programmeringsspråk. Att försöka, och testa sig fram genom att själv skriva kod. Och när man stöter på problem försöker man lösa de, och fortsätter sedan i byggandet. Eftersom jag jobbar med detta i mitt vardagliga arbete har jag lagt ner väldigt många timmar på att lära mig Excel VBA programmering, och som med all programmering, lär man sig hela tiden mera.

Jag har också själv skapat en testdatabas på en SQL server på mitt jobb som jag använder mig av i senare kapitel i denna avhandling, när jag beskriver kodandet i praktiken. Jag har valt att inte ägna denna avhandling till min arbetsplats, eftersom jag inte anser att det är relevant. Tanken är att detta går att anpassa i vilket syfte som helst, även för hemmabruk. Jag har endast valt att arbeta på en server på min arbetsplats, för att jag kunnat arbeta fram de praktiska delarna för denna avhandling på ett smidigt och bekvämt sätt för mig själv. Alla siffror och vad som syns innehållsmässigt i databasen jag använder för denna avhandling är trots allt fiktiva, och inga som helst namn på företag eller riktiga siffror kommer dyka upp i avhandlingen på grund av den självklara sekretessen. Allt jag beskriver i mina figurer i detta arbete är endast framtaget för denna avhandling.

Metoden för framtagandet av den relevanta VBA koden, har jag själv skrivit i ett eget skapat Excel program. Programmet ska fungera som en fiktiv pivotanalys eller översikt för ett fiktivt företag, som behandlas närmare i kapitel 7.

Den pivotmodell som jag använder i avhandlingen har jag också arbetat fram själv. Jag har valt att ställa upp data från databasen på ett separat blad i Excelmodellen och baserat pivoten på det. För detta har jag studerat pivotuppbyggnad och detta behandlas närmare i kapitel 8.

Koden som man behöver skriva, och som är centralt för stycket i fråga i texten, finns alltid på figurerna i samband med texten. Nämner jag syntax som man ska skriva ut i VBA modulerna har jag markerat dessa med *kursiv stil*. Tillexempel kan jag skriva att man alltid ska börja en VBA modul med syntaxen *Public Sub test()*, och då är alltså syntaxen man ska skriva i kursiverad stil.

4 Programmeringsspråket Visual Basic

För att få förstå hur Visual Basic for Applications (VBA) fungerar måste vi först bekanta oss med ursprunget till detta, vilket är ett händelsebaserat programmeringsspråk gjort av Microsoft med namnet Visual Basic.

Ett händelsebaserat programmeringsspråk är en form av programmering där en bakomliggande kod triggas och funktionerar på basen av en användarhändelse. Det kan till exempel vara ett musklick, eller ett tangenttryck, som då utgör att ett meddelande ska

visas, eller att programmet ska utföra någon prestation. (Micosoft Corporation 2012; Techopedia u.å.)

Visual Basic uppkom från början år 1991 som en faktor av BASIC, ett användarvänligt programmeringsspråk som var specifikt designat för nybörjare inom branschen i andra områden än matematik och vetenskap på 1960-talet. (Dartmouth 1964; Grigonis 2014). Visual Basic har därför några nyckelfaktorer som BASIC också bestod av. Den främsta kompabiliteten med dess föregångare var nyckelordet Dim, som man använder vid deklarationer. Också Return kommandon fungerar på samma sätt, och valmöjligheten att använda radnumrering för att lokalisera felmeddelanden är samma. Det uppkom dock en rad nya kännetecknande faktorer för Visual Basic. Nämligen ett grafiskt användargränssnitt (GUI) när man använder språket för att skapa applikationer och program, tillgång till databaser med hjälp av Data Access Objects (DAO) och Remote Data Objects (RDA), och även skapandet av ActiveX kontroller och objekt. Med hjälp av DAO och RDA kan man ta kontakt med databaser och hämta fram informationen därifrån direkt till programmen som man skapar. Detta används även i VBA, som vi senare kommer att bekanta oss närmare med. (Cooper 1996)

Hela konceptet som Visual Basic är uppbyggt på, är att språket använder sig av en kombination av att visuellt arrangera kontroller eller komponenter på en ”form” som är ett GUI fönster. Detta specificerar olika handlingar av dessa komponenter, samtidigt som man skriver rader med kod för att få in mer funktionalitet per automatik i handlingarna man skapar. Detta gör att man kan skapa lätta program utan att behöva skriva stora mängder kod. (Rouse 2007)

4.1 Historia

Visual Basic släpptes officiellt 1991, då det blev framjobbat från programmeringsspråket BASIC. Idén för en ”drag-och-släpp” metod för användargränssnittet jobbades fram från en form av en generatorprototyp som utvecklades av Alan Cooper och hans företag som kallades Tripod. (Cooper 1996; Grigonis 2014)

Visual Basic språket började tas fram under ett projekt kallat Basic Thunder år 1990, och i maj 1991 introducerades Visual Basic som ett programmeringsspråk. (Mack 2002)

I november 1992 släpptes Visual Basic 2.0. Programmeringsmiljön var lättare hanterlig och även snabbheten av programmeringen var förbättrad. Visual Basic 3.0 kom året därpå, sommaren 1993, både i Standard och Professional versioner. Det var också första gången

man kunde börja arbeta med Microsoft Jet Database Engine, vilket är den underliggande komponenten för att arbeta med databaser och deras innehåll i andra program. (Goodhew 1996)

I augusti 1995 släpptes nästa version, Visual Basic 4.0, som även var den första versionen som kunde skapa 32-bit och även 16-bit Windows program. Den kom i Standard, Professional, och Enterprise versioner, och nu introducerades även möjligheten att skriva icke-GUI klasser i Visual Basic, någonting som inte varit möjligt förut. (Code Project 2014)

I februari 1997 släppte Microsoft Visual Basic exklusivt för 32-bit versioner av Windows i och med Visual Basic 5.0. Och under sommaren 1998 kom Visual Basic 6.0 med en mängd olika förbättringar inom de flesta områden, och även förmågan att skapa webbaserade applikationer kom till. (Code Project 2014)

Visual Basic 6.0 stöddes ända till mars 2008 med uppdateringar men därefter slutade Microsoft uppdatera programmeringsspråket vidare. (Grigonis 2014)

4.2 Programmeringssyntax

Programmeringssyntaxen är samma i Visual Basic som VBA använder sig av. Eftersom vi senare kommer gå in mer i detalj på hur man skriver kod som gör en förbindelse med en databas i praktiken, kommer vi inte behandla övrig syntax något mer än i stora drag i denna text. Skulle man vilja, skulle man kunna skriva en hel egen separat avhandling bara baserat på programmeringssyntaxen i ett programmeringsspråk. Därför kommer jag endast ytligt behandla den syntaxen som är nödvändig att kunna då man vill ta fram data i VBA för avhandlingens syfte. Dock eftersom VBA är Visual Basic implementerat i Microsoft applikationer, som tillexempel Office produkter, tittar vi här överskådligt på de mest karaktäristiska dragen hos programmeringsspråket.

Visual Basics grunder härstammar från BASIC språket, som vi redan tidigare har nämnt. Dock finns det inga radnumreringar i Visual Basic, utan koden är grupperad i underrutiner (subroutines) eller metoder med kommandot *Sub* i början och kommandot *End Sub* i slutet.

Kodstycken har ingen kännetecknande symbol för att kodsträngen slutar, förutom radbyte. Man kan även förlänga stycken till följande rad genom att ange ett bottenstreck, _, på slutet av raden. Jag kommer, när jag skriver min kod senare i avhandlingen, använda mig av

bottenstrecken lite oftare än vanligt. Detta för att det vanligtvis inte så långa raderna kod inte kommer få plats inom marginalerna i detta arbete annars.

För att skriva kommentarer i koden markerar man de med en apostrof före texten (’). *’Till exempel skulle detta vara en kommentar.* (Microsoft 1993; Microsoft 2016)

Funktioner som är Loopar börjar och slutar med nyckelorden *Do – Loop*, *While – End*, *For – Next*. (Wenzel 2015)

När det kommer till variabler med flera tilldelningar, är detta inte möjligt i Visual Basic. *A = B = C* betyder inte att *A*, *B* och *C* är lika med varandra. Det booleska resultatet av ”Är B = C?” finns alltså lagrat i variabeln *A*, och därmed är resultatet som är lagrat i *A* antingen sant eller falskt.

Booleska konstanten *True* har det numeriska värdet av -1, för i de flesta programmeringsspråken är *True* inmatade som ett icke-noll värde, oftast 1 eller -1. (Barclay 2001)

Om en variabel inte har blivit tillkännagiven, är den variabeln av typ *Variant*. Detta kan ändras med Deftype påståenden som *DefInt, DefBool, DefVar, DefObj, DefStr*. Standardtypen kan skrivas över för en specifik hängivelse genom att använda nyckelfrasen *As (type)*. (Barclay 2001)

Som sagt var inte syftet med detta kapitel att ge heltäckande kunskaper i hur man programmerar med Visual Basic, men i alla fall ge en liten inblick i hur huvuddelarna och de mest kännetecknande delarna i språket fungerar.

I nästa kapitel kommer vi dyka in närmare i en av implementationerna av Visual Basic, nämligen avhandlingens huvudtyngdpunkt, VBA.

5 Excel VBA – Visual Basic for Applications

Som vi redan har konstaterat ett antal gånger är Visual Basic for Applications (VBA) en implementation av Microsofts händelsebaserade programmeringsspråk Visual Basic. 1993 introducerade Excel, i och med att sin nya version 5 släpptes, ett kraftfullt programmeringsspråk för att bygga makros och döpte det till Visual Basic for

Applications. Alla versioner sedan dess har stött det kraftfulla VBA språket gömmandes bakom Excelbladen. (Jelen & Alexander 2006, 201)

5.1 Introduktion

VBA gör det möjligt att bygga användardefinierade funktioner, automatisera processer, få tillgång till Windows API (som är Windows operativsystemets applikationsprogrammerings användargränssnitt), samt andra låg-nivå funktionaliteter genom så kallade DLL (dynamic-link library). VBA kan användas för att kontrollera många aspekter av en värdapplikation där VBA är implementerad. Därför fungerar den utmärkt i till exempel ett program som Excel för att ta fram data från en databas genom kod till programmen man skapar där. (Kaul 2013)

I praktiken finns det såklart många andra aspekter där VBA är väldigt behändigt att använda. Speciellt i företagsekonomiska-, eller affärssyften. Tillexempel kan VBA program användas för att simplifiera analyser av undersökningsdata, budgetering eller ekonomiska prognoser, skapa fakturor, utveckla grafer av importerade data, eller till och med skapa databaser i Excel för mindre företag. (Walkenbach 2013, 12)

VBA är implementerat i de flesta Microsoft Office Applikationerna, men också delvis i andra applikationer av andra företag som ArcGIS, AutoCAD, CoreIDraw, LibreOffice, Reflection, SolidWorks, och WordPerfect. (Dassault u.å.; Micro Focus u.å.)

5.2 Objekt-orienterad kod

När det kommer till själva koden är VBA väldigt långt relaterat till Visual Basic och Visual Basic Runtime Library med alla funktioner, därav den korta introduktionen till språket i förra kapitlet. Eftersom VBA egentligen är Visual Basic, och Visual Basic är en utbyggd variant av BASIC, kan man tro att det långt skulle vara samma programmerings principer. Men om man är bekant med BASICs koduppbyggnad lär man sig ganska snabbt att koden ser väldigt annorlunda ut i VBA.
Den största skillnaden är att BASIC är ett procedurspråk, medan Visual Basic och VBA är det som kallas för ett objekt-orienterat procedurprogrammeringsspråk. De flesta rader kod i VBA följer Substantiv/Verb syntaxmodellen, men i VBA kallas detta för Objekt/Metod. Objekten i detta fall är Excel arbetsböckerna, arbetsbladen, cellerna, eller cellräckvidderna (på engelska ”cell ranges”) som man arbetar med. Metoderna å andra sidan kan vara

typiska Excelkommandon som tillexempel *.Copy, .Paste, .PasteSpecial.* Med andra ord, i detta fall kopiera- och klistra-in-kommandon. En större kodfunktion man också stöter på i VBA är när man ska ange ett värde till ett adjektiv av ett objekt. I VBA kallas adjektiven för egenskaper, eller på engelska ”properties”. Tillexempel om du skriver *ActiveCel.Font.ColorIndex = 3*, har du angett typsnittet du skriver med för den aktiva cellen till röd färg. (Jelen & Alexander, 2006, 201)

Den största skillnaden mellan VBA och Visual Basic är att VBA kod normalt bara kan köras i en värdapplikation där VBA är implementerat, istället för att skapa ett helt fristående program som man kunde göra i Visual Basic. Dock är det möjligt att kontrollera en applikation från en annan om man använder sig av en automationsprocess i VBA. Då kan man till exempel skapa och öppna ett Microsoft Word dokument genom data i Excel genom en kod som är skriven där.

Koden som är skriven i VBA är kompilerad, vilket betyder att den ”översätts” från det språk koden är skriven i till Microsoft P-Code, vilket är en annan typ av programmeringsspråk som behövs föra att översätta koden till maskinkodsspråk via binära datasträngar. (Wayback Machine u.å.)

5.3 Hur fungerar VBA?

I VBA utför man olika kommandon i så kallade VBA moduler. Dessa moduler kan man skriva själv, eller så kan man använda Macro Recorder som finns färdigt i Excel. Macro Recorder fungerar som en inspelare av ett utförande som man vill spara som kod, så att man lätt kan utföra kommandot igen, och så slipper man fundera ut hur koden ska skrivas. Oftast behövs det dock lite kunskap i hur programmeringen fungerar om man vill använda Macro Recorder felfritt och utan problem.

Man kan editera eller visa VBA modulerna i det som kallas Visual Basic Editor. En VBA modul består sedan av olika så kallade Sub procedurer, där koden är skriven. Tillexempel kan en vanlig Sub procedur se ut som i Kodexempel 1, där resultatet av 1 plus 1 räknas ut i kodproceduren kallad addition. (Walkenbach 2013, 15-16)

Kodexempel 1. VBA Sub-procedur

```
Sub addition()
Sum = 1 + 1
MsgBox "Svaret är " & Sum
End Sub
```

En Sub procedur måste alltid börja med ordet *Sub* och sedan namnet på proceduren, och sluta med orden *End Sub.* En VBA modul består inte enbart utav Sub procedurer, utan kan även bestå av funktionsprocedurer. En funktionsprocedur återger alltid ett ensamt stående värde. En sådan här procedur går att använda i andra VBA procedurer, eller också återkalla den som en formel i ett arbetsblad. Kodexempel 2 visar ett exempel på en funktionsprocedur, kallad AdderaTva, där funktionen accepterar två nummer (argument) och returnerar summan av dessa två. (Walkenbach 2013, 16)

Kodexempel 2. VBA funktions-procedur

```
Function AdderaTva(arg1, arg2)
AdderaTva = arg1 + arg2
End Function
```

Excel ger oss ett väldigt stort antal olika objekt vi kan manipulera och styra i vårt arbete. Exempel på dessa är arbetsböcker, arbetsblad, celler, grafer eller former av figurer, och alla de här kan vi manipulera och styra med hjälp av VBA kod.

Som vi nämnde tidigare i detta kapitel är VBA ett objekt-orienterat programmeringsspråk. Det betyder att olika objekt fungerar som en slags ”hållare” eller ”container” för andra objekt. Till exempel är Excel själv ett objekt, kallat en applikation. Applikationsobjektet innehåller i sin tur andra objekt som tillexempel arbetsboksobjekt, eller Add-In-objekt. Arbetsboksobjekten kan innehålla arbetsbladsobjekt eller grafobjekt. Ett arbetsbladsobjekt kan i sin tur innehålla cellområdesobjekt, på engelska range object, eller ett pivottable-objekt. Så objekthierarkin kan vara väldigt komplex och gå in på väldigt många nivåer. Om man vill kan man hänvisa inne i VBA koden på vilket ställe i objekthierarkin man vill vara när ett utförande ska köra, som i Kodexempel 3, där vi refererar till arbetsboken ”Book1”, arbetsbladet i den boken, ”Sheet1”, och cellen A1 på det bladet. (Walkenbach 2013, 16-17)

Kodexempel 3. Cellreferering i VBA

```
Application.Workbooks("Book1.xlsx").Worksheets("Sheet1").Range ("A1")
```

Objekten har också vad man på engelska kallar ”properties”. Med detta menas egenskaper eller inställningar för objekten. Tillexempel har ett range objekt en sådan egenskap som heter ”value”. Vill du ge ett värde åt tillexempel cellen A1, hänvisar du till den cellen i fråga och skriver ut egenskapsnamnet efter, separerat med en punkt. Kodexempel 4 visar exempel på detta.

Kodexempel 4. Värdet på cellen A1

```
Worksheets("Sheet1").Range("A1").Value
```

VBA innehåller alla de vanliga aspekterna av ett programmeringsspråk, som variabler, arrays och loopar. Så lär man sig koda i VBA så kan man göra väldigt kreativa och komplexa program. (Walkenbach 2013, 17-18)

6 SQL

Jag har nämnt SQL några gånger i denna text i samband med när jag klargjort mitt syfte att använda VBA kod för att ta fram data från en databas till Excel för att kunna göra komplexa och stora affärsöversikter. Jag kommer i detta kapitel redogöra väldigt kort vad SQL är, och vad den SQL kod gör som jag kommer skriva i mina program. Främst kommer koderna i mina program att vara VBA kod, men efter att man deklarerat var man vill att all data kommer in i Excelprogrammet och tagit kontakt med databasen i fråga, behöver man skriva SQL satser för att VBA ska veta vad som ska hämtas från databasen.

SQL står för Structured Query Language och är ett domänspecifikt språk som är skapat för att hantera data i relationsdatabaser. (Gaur 2017)
SQL har tre huvuduppgifter: skapa en databas och definiera dess struktur, skapar frågor (på engelska är detta det som kallas för query) till databasen för att erhålla den nödvändiga data som man behöver i frågan (detta är det jag kommer använda mig av), och slutligen, kontrollera databasers säkerhet. (Wilton & Colby 2005, 11)

SQL som programmeringsspråk är väldigt olika från procedurspråk som tillexempel Visual Basic. Istället för att man programmerar kod som beskriver steg för steg hur man vill att programmet går till väga för att komma till önskat resultat, måste man i SQL, som är ett deklarerar programmeringsspråk, direkt berätta för programmet tydligt vad man vill ha för resultat. (Wilton & Colby 2005, 12-13)

6.1 SQL Queries

Data Manipulation Language (DML), ett underspråk till SQL, tar itu med frågorna och datamanipulationen som förekommer i en query.

En SQL query består av olika påståenden, klausuler och villkor. Ett påstående kan till exempel vara frågan efter en viss data från databasen. En klausul kommer då att fungera som det som specificerar gränserna på den data som frågan ställde, och villkoret anger de gränserna. (Wilton & Colby 2005, 12)

Säg tillexempel att det finns en databas för bilar. Du vill ta fram bilar endast med årsmodell mellan 2005 och 2006. Då kommer påståendet att vara din efterfrågan på att ta fram den data, och det gör du med påståendet *Select*. Att du endast vill att de bilarna med en viss årsmodell ska visas är din klausul och villkoret i frågan är att det specificeras mellan åren 2005 och 2006. I Kodexempel 5 är SQL koden skriven i helhet.

Kodexempel 5. SQL-kod

```
SELECT arsModell
FROM bilar
WHERE ar BETWEEN '2005' AND '2006';
```

En SQL fråga, eller en query som vi kallar det, slutar alltid med ett semikolon. Även om många databashanteringssystem där man kan skriva SQL kod tillåter användare att glömma ett semikolon, är det bra övning att alltid inkluderade det. (Wilton & Colby 2005, 16)

6.2 SQL Syntax

Den kanske viktigaste delen i en SQL kod för oss, med tanke på detta arbete, är SELECT påståendet. Med SELECT påståendet kommer vi att ta fram de elementen från en databas som vi vill ha implementerat i vårt Excel program. I Kodexempel 6 ser vi hur vi väljer att aktivera column1 och column2 från en databastabell som heter table_name. (Wilton & Colby 2005, 53)

Kodexempel 6. SELECT påstående

```
Select column1, column2 FROM table_name;
```

Den andra viktiga punkten i SQL syntaxen för vår del är påståendet WHERE. Med detta påstående kan vi filtrera den information vi vill hämta från en tabell i databasen. Tillexempel som vi gör i Kodexempel 7 då vi väljer att ta fram elementet ”LastName” från tabellen ”MembersDetails”, men endast där elementet ”City” är lika med ”Big City”. Alltså med andra ord vill man se alla efternamn på medlemmar som bor i Big City. (Wilton & Colby 2005, 56-57)

Kodexempel 7. WHERE påstående

```
SELECT LastName _
FROM MembersDetails _
WHERE City = 'Big City'
```

Ifall man aldrig hört om eller bekantat sig med SQL tidigare, är jag medveten om att detta kapitel inte kommer att ge någon större inblick i hur det fungerar eller hur man programmerar i det. Men eftersom det kommer dyka upp SQL kod i programmeringssatserna jag skriver

senare i arbetet, har jag i alla fall gett en kort introduktion i språket och de centralaste delarna vad gäller syntaxen vi kommer att använda.

7 Uppkoppling mellan databas och Excel med VBA kod

Nu när vi har behandlat de teoretiska aspekterna av detta arbetets syfte, samt bakgrundsfakta till vad det är vi jobbar med, är det dags att börja behandla själva dataförbindelsen mellan en databas och Excel i VBA.

Detta kapitel är uppdelat i olika segment beroende på vad som händer i kodandet. Detta för att göra en så överskådlig bild som möjligt av vad som måste göras och hurdan syntax som gör det vi önskar göra. Först kommer jag behandla tillvägagångssättet för att skapa en ny kodmodul i Excel VBA där vi skriver koden som öppnar förbindelsen till databasen. Sedan visar jag hur vi använder oss av SQL uttryck för att hämta data från databasen, och få in den på rätt plats i vår Excelmodell. Slutligen går jag igenom hur förbindelsen stängs till databasen.

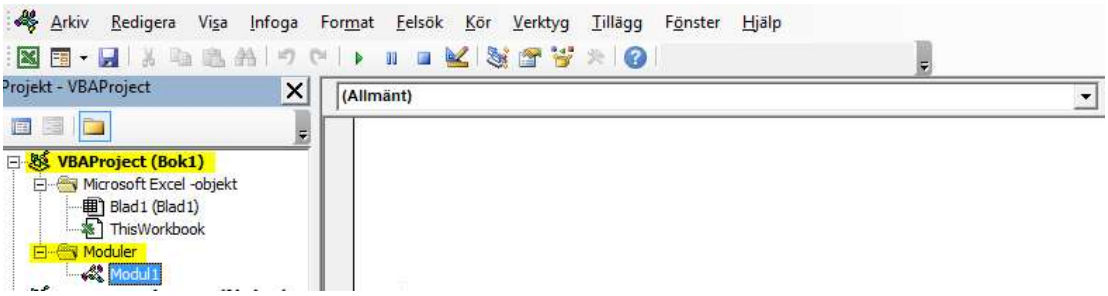
Det finns otaliga sätt att skriva kod som gör samma saker, och att programmera i Excel VBA är inget undantag för detta. Det betyder att det tillvägagångssätt jag använder mig av i detta arbete inte är det enda som är rätt, utan jag vill bara visa hur det kan bli gjort på det sätt jag tycker att det är simplast och effektivast.

För att visa hur koden skrivs steg för steg, har jag valt att visa samma kodmodul i alla respektive figurer i detta kapitel, med gul markering på vad som är aktuellt just för det steget vi på är då.

7.1 Börja skriva koden i VBA-moduler

Det första man gör innan man kan börja programmera i Excel, är att man måste aktivera Utvecklarläget, så att man kan skriva kod i VBA. Det gör man genom att man högerklickar uppe i den översta balken i Excel, och väljer ”anpassa menyfliken”. Sedan letar man upp Utvecklare i den högra spalten och väljer att lägga till den, så den dyker upp i den vänstra. På detta vis har man nu fått Utvecklarläget aktiverat och man kan börja skriva kod i VBA. (Walkenbach 2013, 20)

När man ska börja skriva VBA kod i Excel måste man först skapa en ny modul i Visual Basic for Applications programmet. Detta gör man genom att öppna VBA i Excelprogrammet. Tangenterna Alt + F11 öppnar Visual Basic for Applications kodfönstret. Därefter lägger man till en ny modul genom att högerklicka i den högra spalten och välja Insert och sedan Module, då får man upp ett tomt kodfönster som i Figur 1. (Zhang 2018)



Figur 1. En tom ny VBA Modul

Innan vi börjar skriva kod i modulerna, måste vi se till att vår version av Excel stöder att vi kör våra program i den, det vill säga våra makros. Eftersom det i dagens värld går väldigt lätt att skapa virus gömt i kod, åtminstone för människor som är insatta i mer avancerad programmering, har Microsoft kommit med en lösning mot detta i och med deras Säkerhetscenter som finns inbyggt i Excel, där makros kan vara avstängda helt. Man kan till exempel ha ett färdigskrivet program i Excel som man behöv på arbetsplatsen, eller dela med sig av till kunder som ska kunna köra samma program, men deras makros är avstängda och på så vis kommer aldrig programmet att fungera.
Säkerhetscentret hittas under fliken Arkiv → Alternativ → Säkerhetscenter → Inställningar för Säkerhetscenter. I inställningarna hittar man sedan Makroinställningar, och här måste man se till att det går att aktivera dem om man vill. Det rekommenderas dock inte att helt stänger av makrosäkerhetskollen, eftersom det är precis då som skadlig kod kan köras från okända Excel filer. (Walkenbach 2013, 5)

Efter att en ny tom Modul har blivit skapad måste man börja kodblocket med ordet *Sub*, och sedan ge kodstycket ett namn. Efter namnet kommer en tom parentes. Med hjälp av namnet på ”subben” man har skapat, kan man till exempel senare hänvisa till koden i en annan Sub som är skriven i en annan modul, om man skulle behöva det. För att markera

slutet på kodstycket i fråga skriver man *End Sub*. Mellan de här båda orden skriver man sedan sin kod. (Frolov 2018)

I Kodexempel 8 ser vi en tom Sub som är döpt till ”exempel”, som fortsättningsvis är tom på kod.
Man kan döpa en Sub till egentligen vad som helst, men det finns ett par regler man nog måste följa. Tillexempel kan man använda sig av både bokstäver och siffror, men det första tecknet måste vara en bokstav och namnet får inte vara längre än 255 tecken långt. Man får inte heller ha ett mellanslag eller specialtecken i namnet, men VBA skiljer inte på stora och små bokstäver (detta gäller även i koden). Idealet är att ha ett namn som beskriver proceduren som händer, så ifall man har ett flertal subs och moduler, kan man redan med hjälp av namnet veta vad kodstycket i fråga utför för uppgift. (Walkenbach 2013, 69)

Kodexempel 8. En tom Sub med namnet exempel

```
Sub exempel()

End Sub
```

Det finns också olika typer av Subs man kan använda sig av. En Private Sub kan endast användas i den modulen som ”subben” är skapad i, medan en Public Sub kan användas i andra moduler i samma VBA projekt.

7.2 Öppna en anslutning mellan databasen och VBA

Efter att man har skapat en sub finns det såklart hur många olika tillvägagångssätt att gå för att fortsätta skriva sitt program. I Kodexempel 9 visas de rader kod som är relevanta för oss i vårt program, i och med att det är just dessa kodsträngar som uppger anslutningen mellan SQL-databasen och vårt VBA program.

När man skriver kod i VBA märker man snabbt att det är ett väldigt ”lätt” sätt att skriva kod på, och påminner lite om att skriva ett script i Windows Notepad där de samma principerna gäller, som till exempel att radbrytning sker via Enter och de familjära textredigeringsteknikerna fungerar via tangenterna som förväntat. (Walkenbach 2013, 27)

Kodexempel 9. Öppna en anslutning till databasen i VBA

```
Sub exempel()

Set conn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.recordset")
conn.Open "Provider=sqloledb;Data Source=(databas instansen);User Id=(användarnamn);Password=(lösenord)"


End Sub
```

Anslutningsvariablerna har jag valt att kalla ”conn” och ”rs” som förkortning för connection och recordset, men dessa kan döpas till vad man själv tycker att passar. I de gulmarkerade områdena på tredje raden i Kodexempel 9 ska man minnas att lägga in sina egna uppgifter om databasen man ansluter till, och ta bort parenteserna.

Det första som ögat kanske lägger märke till i kodstycket är bokstäverna ADODB på den första kod raden. ADODB är ett populärt databaslager egentligen skapat för PHP, och tillåter också användaren att använda samma kod för att upprätta anslutning till en väldigt stor mängd databastyper. Vi måste alltså göra det klart för vårt program att vi skapar ett objekt i VBA som öppnar upp en ADO-databasanslutning, där bokstäverna DB i slutet står för database och ordet Connection efter ADODB visar att det är en anslutning vi pratar om. (Regad 2018)

Den andra raden handlar om att öppna ett så kallat recordset efter att anslutningen till ADO-databasen är öppnad. När man använder sig av ADO gör man det möjligt att manipulera data som kommer igenom med hjälp av olika recordsetobjekt, så detta behövs alltså för att vi senare ska kunna välja vilka element vi vill ha ut från databasen in till vårt program. (Milener 2017a)

Vi har nu alltså öppnat en anslutning mellan en hittills ospecificerad databas och öppnat ett recordset som gör det möjligt att börja ta ut och manipulera data på önskvärt sätt.

Den tredje raden som innehåller mest information är den rad som specificerar vilken databasinstans och varifrån data skall komma via den nyöppnade anslutningen. Provider specificerar den korrekta OLEDB leverantören för SQL Servern man vill ha anslutning till. En OLEDB är ett applikationsverktyg som är byggt för att få tillgång till diverse datalagringar genom andra små applikationer som är specificerade för det syftet. Data Source betyder att det specificerar namnet på servern, eller databasinstansen man försöker få kontakt till. User ID och Password är användaruppgifterna till servern så att man har

rättigheten att ta kontakt till serven där databasen ligger och att man kan få ut data därifrån. (Milener 2017b)

7.3 Hämta tabeller från databasen till programmet i VBA

Efter att vi har öppnat anslutningen till databasen och servern är det dags att börja hämta data från databasen till Excel, och detta gör man på ett väldigt lätt och snyggt uppställt sätt i VBA.

Kodexempel 10 visar hur vi först öppnar vårt recordset som vi har skapat anslutning till tidigare, så vi får tillgång till databasen i fråga. Detta gör vi alltså genom att skriva *rs.Open ””*. Sedan kommer SQL syntaxen in i bilden, som vi överskådligt gick igenom i kapitel 6. Här använder jag mig nu av SQL påståendet *Select* för att ta fram de tabeller man vill ha från databasen. Som figuren visar tar vi i detta exempel fram tabellerna ”Namn”, ”Ålder”, ”Land” och ”Epost” från databasen ”Person”, och sedan sorterar vi dem enligt tabellen ”Namn” i alfabetisk ordning.

Kodexempel 10. Hämta data från databasen till Excel i VBA

```
Sub exempel()

Set conn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.recordset")
conn.Open "Provider=sqloledb;Data Source=(databas instansen);User Id=(användarnamn);Password=(lösenord)"

    rs.Open "" _
    & "Select Namn, Ålder, Land, Epost" _
    & "From Person " _
    & "GROUP BY Namn, conn"


End Sub
```

Efteråt används det VBA kod igen för att visa för modulen i vilket Excelblad man vill att data man hämtar ska visas i. Det gör man genom att man skapar ett objekt som man hänvisar att är ett Excel arbetsblad och sedan vilket blad det är frågan om. I Kodexempel 11 har vi använt oss av att referera till arbetsbladet i fråga genom variabeln ”dbSheet” (men här duger vilket namn som helst), och även att det är frågan om Blad1 i detta Excel program (vårt enda arbetsblad i detta program). Vi har även gett variabeln x värdet 1, för detta kommer lite senare att hjälpa oss rada upp data i Excelbladet.

Kodexempel 11. Referera vilket blad data ska komma till

```
Sub exempel()

Set conn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.recordset")
conn.Open "Provider=sqloledb;Data Source=(databas instansen);User Id=(användarnamn);Password=(lösenord)"

    rs.Open "" _
    & "Select Namn, Ålder, Land, Epost" _
    & "From Person " _
    & "GROUP BY Namn, conn"

Dim dbSheet As Worksheet
Set dbSheet = Sheets("Blad1")
x = 1


End Sub
```

När man öppnar ett recordset är den nuvarande posten det första av alla poster, och då är *BOF* och *EOF* inställningarna konfigurerade som *False*. Om det inte finns några poster konfigureras de om till *True*. *BOF* betyder att den nuvarande postens position är före den första posten i recordsetobjektet. *EOF* indikerar på att nuvarande posten är efter den sista posten i recordsetobjektet. Därför kan man använda sig av en *Do-While-Not* sats före man berättar var man vill positionera ut elementen från databasen i Excel filen. Som Kodexempel 12 visar använder jag mig av *Do-While-Not* satsen och lägger ut elementen från databasen så länge *EOF* inte är sant och det fortfarande finns poster att hämta från recordsetet. Efter det bestämmer vi vilken kolumn i Excelbladet som ska innehålla vilket av elementen vi hämtar från tabellen. Här är det viktigt att komma ihåg att elementen från en tabell alltid räknas från 0, alltså har det första elementet 0 som ordningsnummer. Som exempel ser vi då att det första elementet (0) som är Namn kommer att dyka upp i kolumn 1, alltså i Excel blir detta kolumn A, och på rad X som är den raden som just då är aktuell, med start från rad nummer 1 vilket vi definierade X som. Därefter definierar vi om X så att det ska gå neråt med en rad i gången, varav X då blir lika med X + 1. Jag har också lagt in en kommentar för vad varje kolumn innehåller för data, bara för dokumentationens skull. Jobbar man med stora databaser och tar ut en större mängd information från en databas är det bra för en själv, och andra som kanske ska jobba med koden efter dig, att veta vilken tabell som kommer in var i programmet.

Efter att alla element en gång är hämtade från tabellen i databasen går vi vidare i Excelbladet för att lägga till nästa post där, detta genom att berätta för recordsetet att det ska flyttas framåt genom *rs.MoveNext*. Därefter körs en *Loop* sats som håller på tills det inte finns fler poster att hämta ur den tabellen.

Kodexempel 12. Berätta hur data ska ställas upp i Excel bladet

```
Sub exempel()

Set conn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.recordset")
conn.Open "Provider=sqloledb;Data Source=(databas instansen);User Id=(användarnamn);Password=(lösenord)"

    rs.Open "" _
    & "Select Namn, Ålder, Land, Epost" _
    & "From Person " _
    & "GROUP BY Namn, conn"

Dim dbSheet As Worksheet
Set dbSheet = Sheets("Blad1")
x = 1

    Do While Not rs.EOF
        dbSheet.Cells(x, 1) = rs.Fields(0) 'Namn
        dbSheet.Cells(x, 2) = rs.Fields(1) 'Ålder
        dbSheet.Cells(x, 3) = rs.Fields(2) 'Land
        dbSheet.Cells(x, 4) = rs.Fields(3) 'Epost

        x = x + 1
        rs.MoveNext
    Loop

End Sub
```

Det sista vi måste göra är att stänga anslutningen till recordsetet och databasen, så att programmet vet att det har slutfört operationen. Det ser vi hur man gör i Kodexempel 13.

Kodexempel 13. Stäng anslutningen till databas

```
Sub exempel()

Set conn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.recordset")
conn.Open "Provider=sqloledb;Data Source=(databas instansen);User Id=(användarnamn);Password=(lösenord)"

    rs.Open "" _
    & "Select Namn, Ålder, Land, Epost" _
    & "From Person " _
    & "GROUP BY Namn, conn"

Dim dbSheet As Worksheet
Set dbSheet = Sheets("Blad1")
x = 1


    Do While Not rs.EOF
        dbSheet.Cells(x, 1) = rs.Fields(0) 'Namn
        dbSheet.Cells(x, 2) = rs.Fields(1) 'Ålder
        dbSheet.Cells(x, 3) = rs.Fields(2) 'Land
        dbSheet.Cells(x, 4) = rs.Fields(3) 'Epost

        x = x + 1
        rs.MoveNext
    Loop

rs.Close
conn.Close
Set rs.ActiveConnection = Nothing
Set conn = Nothing
Set rs = Nothing

End Sub
```

Det är även det sista vi behöv göra för att programmet ska fungera. För att sedan köra programmet trycker man på F5 tangenten, och då ska data ställas upp i Excelbladet som vi har skrivit det i koden. Det finns även andra möjligheter att köra ett VBA program. Tillexempel kan man skapa en rektangulär form som man gör till en knapp i Excel på arbetsbladet. Sedan tilldelar man makron ”Exempel” (vårt namn på suben) till knappen. Det betyder att nu när man trycker på knappen så körs koden, utan att man behöv gå in i VBA kodfönstret och köra den manuellt. (Walkenbach 2013, 70)

Som jag redan tidigare har nämnt är detta endast ett tillvägagångssätt av många att få detta gjort, men med ganska minimal kod kommer vi här fram till önskvärt resultat.

När man nu sedan ska spara sitt program, måste man minnas att det inte längre fungerar att spara filen som en vanlig Excelfil. På grund av att filen innehåller Makrokod behöver vi spara den som en ”macro-enabled file”, med ett XLSM-tillägg istället för XLSX-tillägg som vanlig Excelfiler har. (Walkenbach 2013, 27-28)

8 Uppbyggnad av pivottabeller i Excel

Eftersom detta arbete beskriver hur vi använder VBA kod för att ta fram data till Excel, och sedan använda den data för att bygga upp olika modeller/analyser/rapporter eller program för olika företagsekonomiska syften, kommer vi nu att bekanta oss med pivotfunktionen i Excel.

Om man använder Excel för att skriva kod och programmera, vet man förmodligen också redan vad en pivottabell är för något. En pivottabell fungerar som en sammanställning av data man har i ett Excel program. Mängden data man kan använda i en pivot varierar från bara några rader och kolumner data, till riktigt stora mängder i komplexa modeller. Många företag, stora som små, använder sig av pivottabeller för månads- eller årsredovisningar av deras siffror. Låt oss säga en datamängd på 65 000 rader, vilket skulle vara näst intill omöjligt att hantera i vanliga fall, blir inget problem för en pivottabell att på sekunder formatera om till önskad uppställning. Det kräver dessutom ingen större kunskap att hantera en pivottabell. Om man kan flytta en muspekare och dra olika objekt, då kan man också hantera en pivottabell. Men som vi redan har konstaterat tidigare i detta arbete, lämnar ungefär hälften av alla Excelanvändare 80% av programmets alla funktioner orörda, med bland annat VBA programkod och pivottabeller inkluderade. Det finns inget annat verktyg i

Excel, och även få andra program överlag, som kan mätas med en pivottabells flexibilitet och analytiska förmåga. (Jelen & Alexander 2006, 1)

Därför är en pivottabell ett väldigt viktigt verktyg i Excel inom företagsekonomin, och tillsammans med VBA kod kan man göra otroligt heltäckande och komplexa modeller och program.

8.1 Pivottabellens historia

Själva konceptet om pivottabeller uppkom egentligen år 1986 när Pito Salas jobbade fram ett kalkylarksprogram för Lotus Development Corporation som kallades Lotus Improv. Salas räknade fram att kalkylark alltid hade ett visst mönster av data, och att man då kunde bygga fram ett redskap som kände av dessa mönster och kunde på det viset bygga förbättrade datamodeller än vad man hade kunnat göra tidigare. Lotus började arbeta fram program med denna princip, och år 1991 implementerades denna teknik i Steve Jobss NeXT computer. 1993 var en version av detta introducerad för Windows.

Själva huvudprincipen Improv använde, som kom att bli dagens pivottabeller, var att data, data visningar, och formler skulle bli behandlade som separata enheter. Detta löste man genom att, för första gången någonsin, en datasats var givet ett namn som man sedan kunde använda för att gruppera datamängden i större kategorier. Detta medförde, genom att döpa olika datamängder och kategorisera dem, att man kunde ordna om datauppställningen genom att endast dra med muspekaren kategorinamnet på den data man ville flytta. Nu kunde man alltså börja bygga modeller, och bygga om de när man själv ville utan någon större svårighet.

Microsoft tog denna idé och skapade för första gången sin pivottabellfunktionalitet i Excel 5 år 1993. Några år senare i och med Excel 97, gjorde Microsoft några avancerade förändringar i sin pivottabellfunktionalitet, som tillexempel att lägga till funktionen att skapa kalkylerade fält från ursprungliga pivotfält. Excel 97 medförde också mer programfunktionalitet i pivottabellerna, i och med att pivotcacheminnet uppkom. Detta blev speciellt viktigt för programmerare som använde sig av pivottabeller i samband med VBA programkod. I Excel 2000 uppkom för första gången pivotgraferna, och nu kunde man börja skapa kosmetiska och väldigt visuellt uppbyggda grafer i samband med pivottabellerna, och än en gång ökade pivottabellers funktion i samband med analytiska uppgifter och framställningar. (Jelen & Alexander 2006, 4–5)

8.2 Användningen av pivottabeller

Man kan tycka att det är en ”onödig” tidsanvändning att lägga ner tid på att bygga en pivot, eftersom den data man vill kolla på redan finns i Excel programmet, och man lätt kan filtrera fram det man vill se på. Men i Excel finns inget snabbare och mer systematiskt sätt att kalkylera och formatera data, som med en pivottabell, trots att man kanske tycker det är onödigt. Tillexempel kan man ha data som visar transaktioner av en kund per dag i ett år, men de vill ha en rapport på transaktioner på månads nivå, då är det ungefär 10 musklick det tar att gruppera om datum på månads nivå och skapa själva pivottabellen. Till skillnad från att göra det genom att ändra om alla dagar till månader och sedan räkna ut summor per månader manuellt, så gör pivottabellen detta för dig automatiskt. Dessutom gör pivottabellen också datamängden klar att användas på ett relativt enkelt sätt i olika grafer, om man så önskar sig.

Det finns alltid några typiska situationer eller uppgifter där det lönar sig att använda sig av pivottabeller i Excel. Den första är om man har väldigt stor mängd data som har blivit för svår att kontrollera eller analysera på ett meningsfullt sätt manuellt.

Den andra är om man behöver hitta unika relationer mellan data, unika värden i ett datafält, eller unika trender i data under i olika perioder.

Den tredje situationen är om du vet att du kommer behöva ändra om din dataanalys vid upprepade tillfällen, eller från olika synvinklar.

Den fjärde situationen är om du vet att det kommer att läggas till data till programmet med jämna mellanrum, så siffrorna ändras. I detta fall är det väldigt lätt att bara lägga till data i datakällan, och uppdatera pivottabellen så sköter den resten av sig själv, och rapporten är ”up to date” på någon sekund igen.

Den femte och sista situationen där det lönar sig att ta till en pivottabell är det vi redan har nämnt, nämligen att ha data i ett sådant format där det är lätt att skapa en graf av informationen. (Jelen & Alexander 2006, 9-12)

8.3 Strukturen av en pivottabell

Eftersom vi snart kommer att gå igenom hur vi använder data vi tagit fram med VBA kod till Excel från en databas, måste man veta hur man strukturerar upp den data som man vill använda i en pivottabell. En pivottabell består av fyra olika områden.

Det första området, som också är obligatoriskt att fylla i med data för en pivottabell, är dataområdet. Här läggs det in som ska kalkyleras eller mätas. Det som alltså är själva rapportens siffror. Om man använder transaktionsexemplet från tidigare, skulle alltså själva summorna av transaktionerna på månads nivå gå in i detta fält. Det är även möjligt att ha flera uträkningar i detta fält i samma pivotrapport.

Det andra området är rad området. Detta område kan lämnas tomt i rapporten, men det är vanligt att det finns åtminstone ett värde här. I samma transaktionsexempel skulle till exempel kunderna vara upplistade i detta område.

Det tredje området är kolumnområdet. Detta område är till för att visa och analysera trender, för tillexempel kunder eller vad man har lagt in på rad nivå. I vårt transaktionsexempel skulle det vara månader som är listade i detta område.

Det sista området är sid området. Detta är ett frivilligt område som syns utanför rapporten och kan tillexempel innehålla kunderna, eller det man har på rad nivå. Då kommer de att dyka upp i en rullgardinsmeny och man själv på ett lätt sätt kan filtrera vad man vill att ska visas i rapporten. (Jelen & Alexander 2006, 12-14).

8.4 Samverkan mellan pivottabeller och VBA programmering

För att summera detta kapitel, har vi nu kunskapen hur vi gör en pivottabell och hur vi skriver ett kodstycke i VBA för att hämta data till Excel och ställa upp det där. Om vi nu skriver en ny kodsträng i vårt nuvarande program som skapades i kapitel 7, för att ta fram nya data ser min kod ut som Kodexempel 14 visar. Här har jag först och främst öppnat anslutningen till databasinstansen, och därefter tagit fram med en SQL sats tabellerna Rapportkod, Namnet som binds till rapportkoden, Perioden, Året, Beloppen, Orten och Företaget från en databas med namnet KoncernDB. Denna fiktiva databas visar försäljningen, kostnaderna och lönerna för fem olika företag inom samma koncern med verksamhetsställen på olika platser i Finland, visat på månads nivå. Därefter använder jag VBA kod för att ställa upp det i Excel på ett datablad, precis som jag gjorde i förra kapitlet med exempeldatabasen.

Kodexempel 14. Kod i VBA för pivotrapport

```
conn.Open "Data Source=(SQL-instans);User Id=(användarnamn);Password=(lösenord)"
rs.Open "" _
& "Select RapKod, RapKodNm, Per, Yr, Sum(Belopp), Ort, Företag" _
& "FROM KoncernDB " _
& "WHERE Per >= 01 AND Per <= 12 " _
& "AND ((Yr = 2018) OR (Yr = 2017)) " _
& "GROUP BY RapKod, RapKodNm, Per, Yr, Ort ", conn

Do While Not rs.EOF
    dbSheet.Cells(x, 1) = rs.Fields(0) & " " & rs.Fields(1) 'RapKod + Namn
    dbSheet.Cells(x, 2) = rs.Fields(3) 'År
    dbSheet.Cells(x, 3) = rs.Fields(2) 'Per
    dbSheet.Cells(x, 4) = rs.Fields(4) 'Belopp
    dbSheet.Cells(x, 5) = rs.Fields(5) 'Ort
    dbSheet.Cells(x, 6) = rs.Fields(6) 'Företag

    x = x + 1
    End If
    rs.MoveNext
Loop
rs.Close
```

Min Excelfil består av ett blad där jag ställer upp data från databasen, med namnet ”Data”, och ett blad som jag har gjort min pivotrapport på med namnet ”Årsredovisning”. När man kör koden i Kodexempel 14, så ställs data upp i sex olika kolumner på databladet, som Figur 2 visar. I koden berättar vi att data ska grupperas först och främst av rapportkoden, vilket i min databas är uppdelat i tre olika koder (1 = försäljning, 2 = kostnader, 3 = personalkostnader). Därav syns i Figur 2 enbart försäljningssiffror för de olika företagen.

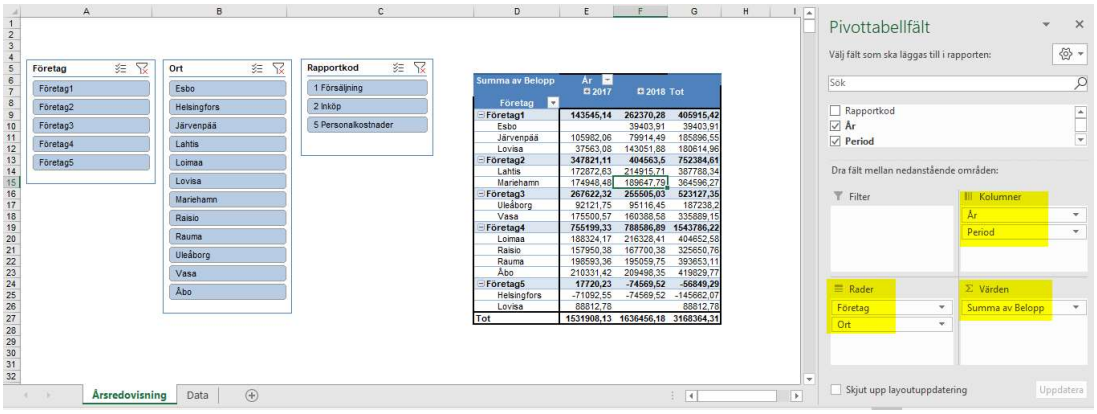
	A	B	C	D	E	F	G
1	Rapportkod	År	Period	Belopp	Ort	Företag	
2	1 Försäljning	2017	1	1870,62	Järvenpää	Företag1	
3	1 Försäljning	2018	1	18931,42	Lovisa	Företag1	
4	1 Försäljning	2018	2	18121,5	Lovisa	Företag1	
5	1 Försäljning	2018	3	18587,37	Lovisa	Företag1	
6	1 Försäljning	2017	4	7279,09	Järvenpää	Företag1	
7	1 Försäljning	2018	4	20569,39	Lovisa	Företag1	
8	1 Försäljning	2018	5	6915,2	Järvenpää	Företag1	
9	1 Försäljning	2018	1	9090,12	Järvenpää	Företag1	
10	1 Försäljning	2017	2	6291,08	Järvenpää	Företag1	
11	1 Försäljning	2018	2	10004,16	Järvenpää	Företag1	
12	1 Försäljning	2017	3	12682,24	Järvenpää	Företag1	
13	1 Försäljning	2018	3	9144,43	Järvenpää	Företag1	
14	1 Försäljning	2018	4	9870,09	Järvenpää	Företag1	
15	1 Försäljning	2017	5	7381,2	Järvenpää	Företag1	
16	1 Försäljning	2018	5	16304,31	Lovisa	Företag1	
17	1 Försäljning	2017	6	5796,88	Järvenpää	Företag1	
18	1 Försäljning	2018	6	5302,48	Järvenpää	Företag1	
19	1 Försäljning	2017	7	4882,1	Järvenpää	Företag1	
20	1 Försäljning	2018	7	5174,4	Järvenpää	Företag1	
21	1 Försäljning	2018	8	4231,42	Järvenpää	Företag1	
22	1 Försäljning	2017	9	4429,84	Järvenpää	Företag1	
23	1 Försäljning	2017	10	13139,56	Lovisa	Företag1	
24	1 Försäljning	2017	11	6567,72	Järvenpää	Företag1	
25	1 Försäljning	2018	11	10928,5	Lovisa	Företag1	
26	1 Försäljning	2017	12	21965,59	Lovisa	Företag1	
27	1 Försäljning	2018	12	11708,24	Lovisa	Företag1	
28	1 Försäljning	2018	6	13294,13	Lovisa	Företag1	
29	1 Försäljning	2018	7	13471,18	Lovisa	Företag1	
30	1 Försäljning	2017	8	4259,57	Järvenpää	Företag1	
31	1 Försäljning	2018	8	10346,05	Lovisa	Företag1	
32	1 Försäljning	2017	9	12046,03	Lovisa	Företag1	
33	1 Försäljning	2018	9	2804,91	Järvenpää	Företag1	
34	1 Försäljning	2018	9	9354,33	Lovisa	Företag1	
35	1 Försäljning	2017	10	5254,11	Järvenpää	Företag1	
36	1 Försäljning	2018	10	12902,15	Lovisa	Företag1	
37	1 Försäljning	2017	11	12097,18	Lovisa	Företag1	
38	1 Försäljning	2018	11	20930,18	Esbo	Företag1	

Årsredovisning | Data

Figur 2. Datauppställning för pivotrapport

Som vi ser i Figur 3, har jag lagt fälten År och Period på kolumnnivå. Företag och Ort på har jag lagt på rad nivå. Och som värden i dataområdet har jag såklart lagt summorna av de olika grupperna. Tar man upp inställningarna för pivottabellen så ser man pivottabellfälten på höger sida i Excel, där man sedan kan dra fälten var man vill ha dem, så man lätt kan bygga upp och bygga om sin pivotrapport.

Jag har även satt in tre stycken utsnitt av pivoten i min Excelrapport. Ett för företag, ett för ort och ett för rapportkoden. Ett utsnitt är som en egen filtrering av pivotrapporten, och här kan man sedan på ett lätt sätt filtrera den och ta med endast de delarna man vill se i sin rapport.



Figur 3. Färdigt uppställd pivotrapport

På detta sätt kan man alltså skapa simpla, men snygga, rapporter för kunder eller för analyser av verksamheten hos en klient eller internt i ett företag. Ju mer data man har, och ju mer erfarenhet av att bygga dessa modeller, kan man skapa väldigt komplexa rapporter med hjälp av pivottabeller.

8.5 Uppdatera pivottabellen med VBA-kod

Det finns ännu ett relevant objekt att behandla när det kommer till pivottabeller. Eftersom det är lätt hänt att det kommer mer data till databasen man använder sig av, och då också mer data in till Excelmodellen, vill man ju att pivotrapporten ska vara uppdaterad och visa korrekta siffror. Detta går lätt att fixa genom att manuellt uppdatera den i

pivotinställningarna. Men eftersom vi kör in data i Excel med VBA kod, kan man i samma veva se till att rapporten uppdateras samtidigt som datamängden gör det.

I Kodexempel 15 ser vi kodstycket som gör detta. Först och främst skapar vi variabeln *pt1* som en pivottabell, och anger vilken pivot i Excelfilen vi menar. Här anger man vilket kalkylblad pivottabellen är skapad på (i mitt fall är det blad 2), och sedan namnet på pivottabellen. Jag har inte döpt om min tabell och därav är namnet fortfarande ”Pivottable1”. De två följande raderna berättar hur pivotcacheminnet ska reagera, vilket betyder att minnet inte ska spara element som inte längre finns i datakällan. Den sista raden kod berättar helt enkelt att vi vill att pivottabellen ska uppdateras enligt den nya källdata som precis har kommit in.

Kodexempel 15. Uppdatera pivottabeller med VBA-kod

```
'    Uppdaterar pivottabellerna
Dim pt1 As PivotTable
Set pt1 = Sheet2.PivotTables("Pivottable1")
pt1.PivotCache.MissingItemsLimit = xlMissingItemsNone
pt1.PivotCache.Refresh
pt1.RefreshTable
```

9 Kritisk granskning av VBA

När man jobbar med VBA och programmerar med språket, måste man minnas att det har funnits sedan början av 1990-talet, och att efter år 2010 har Microsoft slutat att släppa nya versioner av det. Det betyder att jämfört med många andra objekt-orienterade programmeringsspråk är inte VBA det mest smidigaste när det kommer till kodandet, och möjligheterna är ändå inte oändliga. Ur ett användarperspektiv kan man argumentera att VBA är ett klumpigt programmeringsspråk i jämförelse med andra, men man ska då också komma ihåg att VBA är ett händelsebaserat programmeringsspråk som är designat för att automatisera olika funktioner i Microsoft applikationer. Den största nackdelen med detta är att andra som ska ha nyttan av ditt VBA program måste ha sin egen kopia av Excel, eller den Office applikation du har använt. Det är helt omöjligt att konvertera VBA programmet till ett eget fristående program som inte behöver en Office applikation. Det kan också

förekomma fel mellan Excel versioner när Microsoft gör sina uppdateringar. Tillexempel kan viss kod som skulle göra en sak, inte fungera i en äldre version av Excel om någon annan öppnar programmet där. Dock är det positiva med ett händelsebaserat programmeringsspråk som VBA, att vem som helst kan använda det om programmet är tillräckligt bra utfört och skapat.

Om man ska diskutera kring den största säkerhetsrisken med att använda VBA, som i och för sig gäller med alla programmeringsspråk, är ifall någon skapar ett VBA program i skadligt syfte. Men till skillnad från många andra programmeringsspråk ligger inte säkerhetsåtgärderna i händerna på skaparen utan det ligger oftast i händerna på användaren. För alla användare som tar sig an en fil innehållande VBA kod kan enkelt stänga av makros i Officeinställningarna om man är osäker på utgivaren eller skaparen av filen eller programmet. Väljer man sedan att aktivera makros och ta risken så har man satt igång VBA processen själv. Detta utgör en större säkerhet än många andra program gjorda med andra programmeringsspråk där användaren inte har så stor chans att skydda sig om de tar emot och lägger igång ett program med skadlig kod. (Walkenbach 2013, 14–15)

10 VBA användartips

Det finns några punkter man kan försöka komma ihåg för att sin VBA erfarenhet ska bli så bra som möjligt, och dessutom att programmen man skapar ska bli så användarvänliga och smidiga som bara går.

För det första, för att undgå felmeddelanden när man skapar sina program, ska man minnas att alltid deklarera alla variabler man skapar i koden. Om man ser till att alla variabler är deklarerade, kommer kodandet bli mycket smidigare.

För det andra, är inte Excel det bästa programmet att hålla kod hemligt. Detta är viktigt att ha i åtanke när man lägger ett lösenord på sin programkod, för det är inte svårt för en avancerad användare att komma åt koden ändå.

För det tredje, är det alltid positivt att ha en städad kodmodul. Ingen överflödig kod, eller dåliga namn på variabler som man inte förstår vad de gör. Vill man efter en tid gå och editera koden lönar det sig att ha den så snyggt uppstädad som möjligt så man hittar snabbt i långa kodstycken.

För det fjärde ska man aldrig skriva långa kodprocedurer i samma kodmodul. Då blir redigeringar plötsligt väldigt mycket svårare att göra.

För det femte, som vi redan nämnde, har inte alla ett Excel program till förfogande. Och har de inte det så är VBA programmet oanvändbart. Så det gäller att veta att till exempel en kund har Excel i bruk innan man skickar ett program till denne. Det gäller även att komma ihåg att påminna personen i fråga att aktivera makros i sin egen Excelfil, annars kommer inte VBA programmet att kunna aktiveras. Sedan måste man också ha i åtanke att personen i fråga kan ha en äldre version av Excel, och då kan problem också uppstå.

Det sjätte tipset är att våga experimentera när man skriver koden. Detta har hjälpt mig personligen flera gånger när jag skapat program för kunder. Om man inte vet hur någon del av en kod kommer fungera, lönar det sig att skriva ett litet exempelprogram med endast den delen av koden för att se hur den egentligen fungerar, innan man implementerar den i själva programmet.

Det sjunde tipset är att alltid programmera och skapa programmen utifrån kundens synpunkt, ifall man skapar ett program för en sådan. Det gäller att minnas att programmet ska kunna utföras av någon som inte har någon som helst erfarenhet av Excel eller programmering över huvud taget.

Det sista tipset är ett väldigt vanligt tips inom programmering, som de flesta kan ha svårt med ibland, och det är att alltid ha backups på koden och programmen. Utan detta kan skapandet av ett nytt program bli väldigt jobbigt om någonting skulle gå fel. (Walkenbach 2013, 369–373)

Om man har dessa punkter och tips i tankarna när man jobbar med VBA fungerar det väldigt bra i de syften som jag i denna avhandling har behandlat. Och man ska minnas att Excel med VBA fortfarande är, efter alla dessa år, i toppen av att skapa företagsekonomiska program.

11 Slutlig analys

Efter att jag valt detta ämne som min avhandling, var jag lite osäker en tid på mitt val. Inte för att intresset för ämnet inte skulle finnas där, och även kunskapen var jag rätt så säker på när jag satt mig ner och började jobba fram denna text. Dock var jag osäker på hur jag

skulle bygga upp hela arbetet i sig. Men nu efter att avhandlingen är gjord känner jag mig nöjd.

Det är ett ämne som jag känner till väl i och med mitt nuvarande jobb där detta är den vanligaste arbetsuppgiften jag har. Dels lärde jag mig grunden till detta under kurser under mina studier, och dels på egen hand då jag redan länge har lekt runt med Excel och på senare tid börjat programmera. Eftersom jag har lärt mig mer och mer från mitt jobb, kunde detta arbete knyta ihop hela min studiegång på ett bra sätt tycker jag. Allt gick på det viset hand i hand, då jag kunde lära mig mer till mitt jobb via att skriva denna avhandling och sätta mig mer in i det på ett mer undervisande plan som denna avhandling kom att få. Eftersom tanken med avhandlingen skulle vara en slags handbok i hur man sammanför Excel och databaser med VBA kod, och sedan hur man kan utnyttja detta i företagsekonomiska syften och uppgifter.

Det svåraste med min avhandling, innehållsmässigt, var nog att inte ta med för mycket information och delområden, men inte heller ha med för lite så det skulle vara osammanhängande. När man behandlar databaser, programmering och andra olika funktioner finns det en hel del att nämna och ta med. Jag valde att först och främst diskutera kring mitt val av ämnet och argumentera varför jag tyckte att det var intressant. Som jag redan nämnde i början av avhandlingen, kan det tyckas att ämnen och jobb kring Excel och VBA är ett föråldrat koncept, men eftersom det fortfarande används i så bred utsträckning som det gör tyckte jag det inte spelade så stor roll. Därför är de flesta källorna jag använt mig av i denna avhandling allt från fem till femton år gamla, men fortfarande relevanta. Efter att jag argumenterade för mitt val av avhandlingsämne, gick jag in på vad programmeringsspråket Visual Basic är för något. Detta valde jag att göra eftersom VBA egentligen helt och hållet är Visual Basic i form av ett händelsebaserat programmeringsspråk. Därefter behandlade jag VBA i ett skilt kapitel, eftersom detta ändå är tyngdpunkten i arbetet. Innan jag kunde gå mer in i det praktiska, kände jag ändå att jag behövde klargöra vad SQL är, när det ändå används både VBA kod och SQL kod i programmerandet, därav ett eget kapitel för det. Sedan gick jag vidare för att praktiskt beskriva hur kodandet utförs mellan databasen och Excel i VBA. Till slut visade jag vad man kunde göra med data man hade fått in med kod, i och med pivottabeller. Här gav jag först en teoretisk bakgrundsdel till vad pivottabeller är för något, och sedan en praktisk del där jag visade hur man byggde en simpel pivotrapport.

Från ett självkritiskt perspektiv kunde jag ha lagt ner mer tid på att diskutera kring databasuppbyggandet jag gjorde i bakgrunden för detta arbete. Jag arbetar alltid med

väldigt stora databaser i mitt jobb, men eftersom jag inte kunde använda mig av någon färdig databas för mina exempel gjorde jag egna från början. Där genererade jag in fiktiva namn och siffror som jag sedan kunde använda mig av. Dock visade jag aldrig databaserna i fråga, eftersom databashanteringsprogrammet jag använde mig av också förblir osagt, och bilder på en sådan databas kan förorsaka sekretessfel.

Det är även svårt att med några bilder presentera hur koden egentligen körs och fyller en hel Excelfil med data. Detta skulle jag ha kunnat lösa genom tillexempel gjort exempeldatabaserna på en egen server i till exempel Microsoft Access, och då även kunnat visa databasuppbyggandet på ett närmare sätt. Dock skulle detta ha tagit väldigt mycket längre tid och också blivit till en helt ny underdimension av detta arbetets originalsyfte.

Som jag redan tidigare har nämnt, kan jag omöjligt lära ut ett programmeringsspråk till en hel nybörjare inom programmering bara genom denna avhandling. En aspekt skulle ha varit att satsa mer på kodbeskrivningar och syntaxförklaringar. Jag behandlade både Visual Basic och SQL i denna text väldigt ytligt, och här kunde jag ha satsat mer på denna del. Det var speciellt svårt att veta vad jag skulle ta med här och hur mycket tid jag skulle lägga ner på att berätta om de olika områdena. Eftersom egentligen detta bara byggde grunden till arbetets huvudpunkt valde jag till slut att endast behandla centrala delar för avhandlingen, och då blev det en väldigt ytlig genomgång. Det skulle vid flera tillfällen ha varit möjligt att skriva en hel avhandling om bara ett delområde i detta arbete, och därför kan det vara besvärligt att inte gå in för mycket på djupet på vissa områden, men inte heller lämna det borta helt och hållet.

Också i kapitlet angående pivottabeller, finns det en hel del att lära ut och prata om. Här blev det också endast de centrala delarna i korta drag för denna avhandling som behandlades. Vilket också kunde ha utgjort en större del av arbetet och jag kunde ha visat mer olika typer av pivotuppbyggnader. Men också i det fallet skulle avhandlingen blivit mycket längre och mer tidskrävande.

Överlag har jag försökt hålla en systematisk uppbyggnad genom hela arbetet och argumenterat för ordningen på de olika kapitlen. Men en ständig kamp mellan vad som är relevant utifrån detta arbete, och vad som blir överkurs eller onödiga fakta, har det varit. Ändå är jag nöjd med slutresultatet och hoppas läsaren har lärt sig något nytt, eller åtminstone lagt grunden till några nya idéer och öppnat ögonen för hur man kan använda Excel och VBA i arbetslivet, och det var mitt största mål hela tiden.

Källförteckning

Barclay, D., 2001. *Classic Visual Basic*. [Online]
http://vb.mvps.org/tips/truth/ [hämtat: 13.03.2018].

Beal, V. (u.å.). *What is syntax? Webopedia Definition*. [Online]
https://www.webopedia.com/TERM/S/syntax.html [hämtat: 22.11.2018].

Code Project, 2014. *Visual Basic 6.0: A giant more powerful than ever*. [Online]
https://www.codeproject.com/Articles/710181/Visual-Basic-6-0-A-giant-more-powerful-than-ever [hämtat: 13.03.2018].

Cooper, A., 1996. *Why I am called "the Father of Visual Basic".* [Online]
https://www.cooper.com/alan/father_of_vb.html [hämtat: 13.03.2018].

Dartmouth, 1964. *Basic*. [Online]
http://bitsavers.trailing-edge.com/pdf/dartmouth/BASIC_Oct64.pdf [hämtat: 13.03.2018].

Dassault (u.å.). *VBA.* [Online]
http://help.solidworks.com/2016/English/SolidWorks/sldworks/c_vba.htm
[hämtat: 16.03.2018].

Frolov, A., 2018. *How to insert and run VBA code in Excel – tutorial for beginners*. [Online]
https://www.ablebits.com/office-addins-blog/2013/12/06/add-run-vba-makro-excel/
[hämtat 13.04.2018].

Gaur, A., 2017. *SQL*. [Online]
https://www.britannica.com/technology/SQL [hämtat: 09.04.2018].

Girgonis, R., 2014. *Where Is the Successor to Visual Basic?*. [Online]
https://www.newsmax.com/RichardGrigonis/Visual-Basic-Microsoft-programmers/2014/03/27/id/562161/ [hämtat: 13.03.2018].

Goodhew, T., 1996. *Jet Engine: History*. [Online]
http://www.avdf.com/nov96/acc_jet.html [hämtat: 13.03.2018].

Jelen, B. & Alexander, M., 2006. *Pivot Table Data Crunching*. Indianapolis: Que.

Kaul, A., 2013. *What is Excel VBA? : Excel VBA Basis 001*. [Online]
https://www.exceltrick.com/excel-vba-basics/what-is-excel-vba/ [hämtat: 16.03.2018]

Mack, G., 2002. *History of Visual Basic*. [Online]
http://www.ojodepez-fanzine.net/network/qbdl/history_of_visual_basic.html [hämtat: 13.03.2018].

Micro Focus (u.å.). *Reflection Desktop VBA Guide*. [Online]
http://docs.attachmate.com/reflection/16-1/vba-prog-guide/ [hämtat: 16.03.2018].

Microsoft, 1993. *Programmers Guide. Microsoft Visual Basic – Programming System for Windows*. Redmond: Microsoft Corporation.

Microsoft, 2012. *P: Safe Asynchronous Event-Driven Programming*. [Online]

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-8.pdf [hämtat: 13.03.2018].

Microsoft, 2016. *Visual Studio 2003 Visual Basic and Visual C#*. Redmond: Microsoft Corporation.

Milener, G., 2017a. *Recordset Object (ADO)*. [Online] https://docs.microsoft.com/en-us/sql/ado/reference/ado-api/recordset-object-ado?view=sql-server-2017 [hämtat: 09.05.2018].

Milener, G., 2017b. *Microsoft OLE DB Provider for SQL Server Overview*. [Online] https://docs.microsoft.com/en-us/sql/ado/guide/appendixes/microsoft-ole-db-provider-for-sql-server?view=sql-server-2017 [hämtat: 09.05.2018].

Regad, D., 2018. *ADOdb – Database Abstraction Layer for PHP*. [Online] http://adodb.org/dokuwiki/doku.php [hämtat: 09.05.2018].

Rouse, M., 2007. *Visual Basic (VB)*. [Online] https://searchwindevelopment.techtarget.com/definition/Visual-Basic [hämtat: 13.03.2018].

Techopedia (u.å.). *Visual Basic (VB)*. [Online] https://www.techopedia.com/definition/3962/visual-basic-vb [hämtat: 13.03.2018].

Walkenbach, J., 2013. *Microsoft Excel VBA Programming for Dummies*. New Jersey: John Wiley & Sons, Inc.

Wayback Machine (u.å.). *Visual/Access Basic Is Both a Compiler and an Interpeter*. [Online] https://web.archive.org/web/20121021000129/http:/support.microsoft.com/kb/109382 [hämtat: 16.03.2018].

Wilton, P. & Colby, J., 2005. *Beginning SQL*. Indianapolis: Wiley Publishing, Inc.

Wenzel, M., 2015. *Loop Structures (Visual Basic)*. [Online] https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/control-flow/loop-structures [hämtat: 13.03.2018].

Zhang, T., 2018. *Getting Started with VBA in Office*. [Online] https://msdn.microsoft.com/en-us/vba/office-shared-vba/articles/getting-started-with-vba-in-office [hämtat: 13.04.2018].

Figurförteckning

Kodexempelförteckning