Kai A. Olsen

Praktisk programmering

CAPPELEN DAMM AKADEMISK

ISBN e-bok: 978-82-02-63000-3

ISBN trykt bok: 978-82-02-60647-3
1. utgave, 1. opplag 2018

Boken har fått støtte av Høgskolen i Molde, avdeling for logistikk.

Omslagsdesign: Cappelen Damm AS
Sats: Kai A. Olsen
Trykk og innbinding: Livonia Print SIA, Latvia

www.cda.no
akademisk@cappelendamm.no

Forord

De fleste programmeringsbøker handler om nettopp det – programmering. Problemet er at vi ofte ønsker å bruke kunnskapen om programmering til å lage nyttige systemer. Da må vi håndtere import av data, lagring og strukturering av data, et brukergrensesnitt (slik at brukerne våre kan styre systemet) og rapportering – i tillegg til selve programmeringsdelen.

I denne boken skal vi konsentrere oss om helheten, slik at en får en overordnet forståelse for både systemutvikling og programmering. Samtidig skal vi lage systemer som kan brukes til noe. For å få til dette skal vi bruke Microsoft Access. Dette systemet har alt vi trenger: Programmeringsspråk, database, brukergrensesnitt og et rapportverktøy. Det vi skal se, er at veien til målet, til et ferdig uttestet system, vil være langt enklere med Access enn med de fleste andre systemer. Det er viktig. Programmering er vanskelig nok om vi ikke skal slite med tungvinte verktøy.

Min erfaring fra en lang rekke systemutviklingsprosjekter for næringslivet, er at en kan komme i mål langt hurtigere og med lavere kostnad med Access enn med andre systemer. Det betyr ikke at Access er det beste valget i alle situasjoner. Er det strenge krav til sikkerhet, eller om det er tusenvis av brukere, vil jeg ikke valgt Access. Da finnes det tyngre verktøy. Ulempen med disse er akkurat det, når alle aspekter skal dekkes opp blir utviklingen mer tidkrevende og mer kostbar.

For å lære programmering er det tre ting som er viktig – arbeid, arbeid og arbeid! Det er dessverre sant. Problemet er at programmering har mange nivåer. For å kunne arbeide på toppnivået, der vi kanskje forsøker å se hvordan en maskin kan ta seg av en oppgave som tidligere er gjort av mennesker, må vi også beherske alle nivåene under. For å mestre det vanskelige, må vi beherske det enkle. Eneste måten å få dette til på er å opparbeide en rutine. Når de enkle delene går glatt, kan vi da ta fatt på å forstå neste nivå.

Vår løsning på disse utfordringene er gjennom rutine. Vi skal starte med enkle oppgaver, programmere disse, og gjenta dette med nye oppgaver. Etter hvert vil du beherske de enkle delene. Da skal vi opp et nytt nivå og takle noe vanskeligere oppgaver. Til slutt skal vi også prøve oss på virkelige systemer: fakturerings- og regnskapssystemer, bestillingssyste-

mer for hotell, restaurant og fly, systemer for å beregne gjennomsnittshastighet, m.m.

Istedenfor å bruke mye tid på å lære basisbegrepene skal vi gå rett på programmeringen. Det er som moderne språkkurs. Der snakker en det fremmede språket fra dag 1. Vi skal altså programmere systemer fra dag 1. Så er ideen at forståelsen av basiskomponentene skal komme til etter hvert. Etter å ha introdusert begrepene i praktiske eksempler, vil jeg summere opp og si mer om både programmering og Access.

I boken har vi brukt fargekodene fra alpinbakken. Grønt for det som er greit, blått for det som er mer krevende, rødt for det vanskelige og **sort** for det avanserte. Om en skulle ta eksamen i Praktisk programmering, vil oppgavene her fordeles fra grønt til rødt, ikke sort. Når vi tar med det sorte, er det for at du som programmerer skal vite litt om de mer avanserte løsningene.

Kommer du gjennom dette, mestrer du det å programmere. Du vil da ha et verktøy som kan brukes for å gjøre arbeidsdagen din mer effektiv, og ikke minst – til å hjelpe andre, slik at de også kan jobbe mer effektivt. Med kunnskap om programmering kan vi bidra til å digitalisere arbeidsoppgaver, som oftest de kjedelige og rutinepregede oppgavene. Det er helt nødvendig om vi skal klare å opprettholde en høy levestandard og et høyt lønnsnivå i Norge. Med kunnskap fra denne boken kan du bidra i dette viktige arbeidet.

Innhold

1 Introduksjon

Tenk deg at du skal gå på et forfatterkurs og lære å skrive romaner. Det vil være krevende nok, men nå har du meldt deg på et forfatterkurs i kinesisk. Her får du nye tegn, ny grammatikk og så skal du samtidig lære å skrive romaner. Heldigvis er det ikke fullt så vanskelig å lære programmering. Du vil kjenne igjen deler fra skolematematikken. Tegnene er heldigvis våre vanlige latinske, men du vil måtte lære en ny grammatikk. Fordelen er at dette er en mye mer stringent grammatikk enn den vi er vant med fra vanlige (naturlige) språk. Dernest må du forstå på hvilket nivå maskinen arbeider – hva som er dens styrke og begrensning. Det er nødvendig når du skal ta oppgaver fra den virkelige verden og beskrive disse som programmer.

Som vi ser, skal en ikke bare lære programmering, men også beherske et verktøy. Det er ikke ”rocket science”, men i starten vil du bruke mye tid på å finne fram. Nå vil denne boken gi en detaljert oppskrift, men også det kan feile – for eksempel om du har trykket en feil knapp. Da må du selv finne tilbake til der du var. Det kan ta tid. Og etter å ha rotet rundt i Access-grensesnittet en stund, har du kanskje glemt hva du egentlig skulle gjøre. Eneste svar på dette er å bruke verktøyet så mye at det blir rutine.

1.1 Utvikling av datasystemer

Hvorfor skal vi lære å utvikle datasystemer? Vi kan spørre oss om ikke alle de nyttige systemene allerede er utviklet. Dette er et godt spørsmål. I dag finner vi systemer og apper for det meste. Det ville ikke være særlig fornuftig å utvikle ditt eget tekstbehandlingssystem, når du kan få Microsoft Word for en billig penge, eller andre systemer gratis. Vi kan også diskutere om det er så mye å hente på å utvikle apper. Da konkurrerer vi med millioner av andre programmerere over hele verden. Alle utvikler apper og det finnes hundrevis, ofte tusenvis, av apper for hvert eneste formål. Leter du for eksempel etter en kalender-app, finner du flere tusen, både på Google Play og på App Store. Skal du få noen til å bruke din app, må du nok gi den bort gratis og isteden håpe på å tjene penger på fremtidige utgaver.

Derimot vil vi, i de aller fleste bedrifter, finne oppgaver som ikke er dekket av standard programvare. Vi skal gi flere eksempler i denne boken der det kan ha vært store besparelser på å lage egne systemer. Det kan også tenkes at hyllevaren, standard programpakker som vi kan kjøpe, kan være for kompleks eller kostbar, samtidig som den ikke dekker jobben fullt ut. Igjen en mulighet for en som kan programmere. I dette markedet

er det liten konkurranse. Min erfaring er at det i enhver bedrift finnes mange uløste programmeringsoppgaver.

Dersom du kan programmere, eller kode som mange kaller det, vil du stå sterkt i arbeidslivet. Kunnskap om informasjonsteknologi (IT) er blitt nødvendig i dag innen de fleste jobber. Her står kompetanse i programmering sentralt. Det gjelder spesielt om du også har utdanning eller erfaring fra et annet område. Da kan du bruke din programmeringskunnskap til å lage systemer innenfor dette området. For eksempel, om du vet mye om å lage møbler og programmering, er det ikke så sikkert at det er mange andre som har denne kombinasjonen. Da kan du utvikle systemer som er nyttige for dette anvendelsesområdet, uten å møte stor konkurranse fra ferdige systemer. Der de tilgjengelige systemene i markedet er laget for en generell bedrift, kan du lage noe som er tilpasset den bedriften der du arbeider, enten det er en møbelfabrikk, et salgsfirma, en eiendomsmegler, dagligvarebutikk, eller hva som helst annet.

Jeg har utviklet store programsystemer for flere bedrifter. Utgangspunktet har vært at det enten ikke var et tilbud av ferdige systemer, at disse systemene ikke dekket behovet, eller at de var for kostbare. En dramatisk fordel med å lage et eget system, er at det gir en konkurransefordel. Om en bedrift velger å bruke ferdige systemer, kommer en ikke bedre ut enn konkurrentene da disse bruker den samme programvaren, kanskje også de samme konsulentene. Da blir IT som elektrisitet, noe en må ha, men som ikke gir noe fortrinn overfor konkurrentene. Lager en noe eget, som ingen andre har, kan en få en strategisk konkurransefordel.

Disse eksemplene er dekket utførlig i boken. Ideen er å vise at overgangen fra de systemene vi programmerer i denne boken til komplette systemer for den virkelige verden ikke er så stor. Dvs. om du lærer deg systemutvikling, slik vi beskriver det her, kan du også utvikle nyttige og viktige systemer.

1.2 Hvorfor Microsoft Access?

Access er et anvendelig verktøy som kan brukes til både små og store oppgaver. Om du har 1, 10, 100 eller 1000 brukere kan Access gjøre jobben. Når det er sagt, ville jeg nok ikke bruke Access om det var snakk om å lage hovedsystemer for store banker eller forsikringsselskap. Derimot kan vi gjerne lage hovedsystemer for mindre bedrifter i Access. Som vi skal se, er Access spesielt egnet for mindre systemer. En sentral egenskap med verktøyet er at vi kan komme i mål med begrenset innsats, både i tid og penger.

For å ta et eksempel. Jeg fikk i oppdrag fra en av mine kunder, fra en bedrift der jeg har utviklet hovedsystemet, om å bygge inn mulighet for å oppgi priser i forskjellige valutaer. For å få til det trengte jeg en tabell med kurser, altså med valutakode (for eksempel USD, EUR) og kurs. I tillegg måtte jeg gi brukeren mulighet til å velge valuta for et prosjekt, og å konvertere til denne valutaen i alle utskrifter med priser. For dette fakturerte jeg to timer. Vanskeligere var det ikke. Samtidig, i et oppdrag for en annen kunde der jeg var rådgiver, og andre stod for programmeringen, fikk vi et estimat på 60 timer for akkurat samme jobben. I utgangspunktet synes vi dette var altfor mye, men etter å ha sett nærmere på estimatet fant vi ut at dette var realistisk. Med de verktøyene som ble brukt for dette prosjektet ville det ta 60 timer å gjøre jobben.

Dersom du ikke jobber for DNB, Gjensidige, Hydro eller en annen stor bedrift, bør Access vurderes som utviklingsverktøy. Og selv i store bedrifter kan en bruke Access til mindre funksjoner, eller til å lage en prototype. Siden det er så kjapt og effektivt å lage Access-applikasjoner kan det ofte lønne seg først å lage systemet i Access, bruke dette til å få erfaring med oppgaven, og så programmere i det verktøyet kunden krever at vi skal benytte. I min virksomhet over mange år har jeg flere ganger angret sterkt på at vi ikke har utviklet systemet i Access først. Det ville ha spart oss for unødige kostnader.

Programmeringsspråket i Access er VBA – Visual Basic for applications. Som navnet sier, er dette et Visual Basic-språk, med små avvik fra programmeringsspråket Visual Basic (VB). Kan du programmere i VBA, kan du også programmere i VB. Det kan være praktisk. Selv om Access kjører de fleste jobber fort og effektivt, kan det være situasjoner der vi trenger rask prosessering. I motsetning til VBA kan Visual Basic gi deg en exe-fil, altså en meget effektiv, kompilert, kjørbar versjon. Det at den er kompilert betyr at programsystemet har oversatt koden fra det nivået vi benytter når vi programmerer, til et langt mer effektivt maskinnært nivå. For at de aller fleste oppgaver er dette uten betydning. Men, som sagt, kan du VBA kan du også VB.

Veien fra VBA til andre programmeringsspråk, som C++, Java og lignende, er kort. En fordel med VBA er at det også inngår i alle andre Office-programmer, i Word, Excel og Outlook. Det kan være nyttig. Som vist senere i denne boken, kan vi bygge videre på disse programsystemene for å tilpasse disse akkurat til vårt formål. Dette gjelder også selve verktøyet, Microsoft Access. Vi må lære å bruke dette for å kunne programmere. Ingenting er vanskelig. Access er godt designet. Problemet

er at vi må beherske så mange deler. I boken viser vi hvordan alt skal gjøres – en gang. Neste gang du skal gjøre en operasjon, for eksempel opprette en tabell eller et skjema, må du klare det selv. I begynnelsen går det tregt. Ideen er å gjøre alt mange ganger. Da sitter det.

Access er et verktøy som lar oss komme fort til målet. Det budsjettet vi har for å lage hele systemet, kan være på samme størrelse som det andre, som anvender tyngre verktøy, kan bruke på en "kick off".

Nå har det sine klare fordeler at Access kan kjøre den koden du har skrevet inn direkte, uten oversetting til et mer maskinnært språk. Denne form for prosessering kaller vi *interpretering*. Mens det koster oss noe i kjøretid, har det meget store fordeler, ikke minst når vi skal debugge (feilsjekke) programmet. I Access kan vi styre kjøringen selv. Vi kan stoppe, klikke på de forskjellige delene av programmet for å se hvilke verdier disse har, vi kan til og med kjøre setninger om igjen. Har vi gjort en feil i en programsetning, kan vi rette feilen under kjøring, og så repetere kjøringen. For nybegynnere er dette ubetalelig. Ikke bare er det lett å feilsjekke systemet, men vi kaster ikke bort tid når vi skal rette feil. I tillegg, når vi i starten skal forsøke å forstå hvordan maskinen tolker det programmet vi har laget, er mulighetene til å kjøre en og en programlinje av stor pedagogisk betydning. Her kan du stoppe kjøringen på en linje, vurdere situasjonen og deretter be systemet kjøre en linje til.

En ulempe med Access er at den i utgangspunktet ikke er tilgjengelig på Apples maskiner. De som har valgt å melde seg opp som student (se under), vil få tilgang til Access via en nettside. Om du har en Apple-maskin, og ikke er student, må du installere en virtuell Windows-maskin og kjøre Access på denne.

Selv om vi har valgt Microsoft Access som verktøy, er ikke hensikten å gi en utførlig beskrivelse av dette verktøyet. Vår jobb er å bruke Access for å lære programmering og å lage nyttige systemer, ikke å utforske alle detaljene. Dersom en er interessert i det, finner en mye innføringsstoff på nett.

I boken har jeg valgt å bruke engelsk versjon av Access. Dette blir en smakssak, men fordelen med å bruke engelsk versjon er at setningene i programmene uansett skrives med engelske symboler (For, If Then, Exit, osv.). I tillegg, om du vil vite mer om skjema i Access og søker på nett får du langt flere treff om du søker på "Microsoft Access Form" enn med "Microsoft Access Skjema". Søker du på mer obskure begreper er det ikke sikkert at du finner noe treff på norsk. Videre, mye av læremate-

riellet, eksempler og ferdig kode på nett, er på engelsk. Som datafolk kan vi like godt innarbeide de internasjonale begrepene med en gang.

En ulempe med å bruke engelsk utgave av Access er at språket i denne boken blir litt både og. Siden vi bruker engelske begreper så ofte, form, record, query, m.m. har jeg valgt å bruke disse direkte, uten å sette på anførselstegn. Sluttproduktet blir praktisk, om enn ikke så veldig elegant.

1.3 Programmering

En liten tue kan velte et stort lass. Ingen steder er dette mer riktig enn i programmering. Har du glemt et komma, brukt komma istedenfor punktum, enkel apostrof (') istedenfor dobbel ("), eller tusen andre mulige feil, vil ikke programmet fungere. De enkleste feilene, der vi har feil syntaks (grammatikk), har brukt begreper vi ikke har beskrevet, osv. tar VBA seg av. Vi får da en feilmelding med angivelse av hvor feilen er. Noen feil oppdages først når programmet kjøres, for eksempel om vi dividerer med 0 eller skriver galt navn når vi skal lese fra en tabell i databasen.

En slik feil, en programmerer hadde glemt et element i en programsetning, førte til at deler av telefonsystemet i USA kollapset. Det var i 1990. Siden den gang har vi sett utallige spektakulære programmeringsfeil, som har hatt enorme konsekvenser, også tap av menneskeliv. Heldigvis skal ikke vi utvikle slike livskritiske oppgaver – i hvertfall ikke i starten.

Andre feil kan det tenkes at en ikke oppdager før programmet har vært i bruk i lang tid. Jeg fikk melding om en slik feil i dag – i en meget stor ordre var antallet av en komponent blitt så stort at Access ikke fikk plass i det feltet som var avsatt for dette i rapporten. Siden det å finne feil er en viktig oppgave for en programmerer skal vi legge vekt på denne oppgaven. Nå er vi så heldige at det er få språk som er så velegnet for feilfinning (debugging) som Access.

Det som er virkelig spennende, er at når vi har lært å programmere og behersker verktøyet, ja da har vi noe vi kan erobre verden med. Kanskje ikke verden, men vi vil ha et verktøy som vi kan bruke til å forenkle vår egen hverdag, jobben vår og jobben til andre. Mange har beskrevet programmering som en kunstform. Det gjør jeg også i et sluttkapittel i boken.

1.4 Hvordan bruke boken

Ideen med denne boken er at vi skal lære programmering ved å programmere. Vi starter med enkle case (eksempler) og går videre med mer

avanserte. Boken må leses sekvensielt. Etter at vi har innført grunnleggende begreper i tidlige kapitler, skal vi bruke disse i senere kapitler. For hvert nytt case anbefaler jeg at du først leser gjennom hele kapittelet, for deretter å programmere løsningen selv.

I tillegg til boken finnes det materiell åpent tilgjengelig på nett. Søk på Canvas og IBE151. Det siste er emnekoden for Praktisk Programmering ved Høgskolen i Molde. På nett finner en Access-filer av alle programeksemplene i boken. Men ikke vær redd for å taste inn programkoden på nytt. Det er mye å lære av dette. Ikke minst får du da øye for detaljene.

På nett finner du også videoer for hvert kapittel i boken. Videoene er lagt inn med screencast, altså slik at jeg programmerer mens castingen kopierer det som skjer på skjermen, samt det jeg sier. Videoene følger denne læreboken kapittel for kapittel, case for case. Fordelen med læreboken er at en kan tilegne seg stoffet i sitt eget tempo, gå tilbake til tidligere kapitler (noe som er viktig her) og bruke boken for oppslag (bak i boken finner du en komplett indeks). Mens en kan se en video en gang, kan en *arbeide med* boken.

Læreboken er på papir. En kunne kanskje tenke seg å ha denne på skjermen. Problemet er at du skal programmere de eksemplene som boken omtaler. Da ville det bli lite plass på skjermen om du både skulle ha programmene og boken der. Vær derfor glad for den ekstra plassen du får når boken er på papir. Dersom du kun har en laptop, vil jeg anbefale at du vurderer å kjøpe en større skjerm. Når programmene etter hvert blir store, vil det være viktig å få full oversikt over koden. Det gjelder spesielt når den dag kommer at du skal bruke det du har lært til å lage nyttige systemer. Selv har jeg to 32” skjermer på kontoret med stor oppløsning. Det er praktisk når systemene kommer opp i titusenvis av linjer med kode.

Bok, videoer og nettside er bare hjelpemidler. Skal du lære å programmere så må du gjøre det selv. Derfor har jeg oppgaver i slutten av hvert kapittel. Det anbefales å jobbe med disse. Helt bevisst gir jeg ingen løsninger på disse oppgavene. I tillegg til disse mindre oppgavene finner du et sett større oppgaver – det jeg har kalt øvinger. Dersom du velger å melde deg opp i emnet IBE151 og ta eksamen, må du levere inn løsninger på øvingene. En løsning skal være et tekstdokument med en analysedel, programkoden og en forklarende del, altså akkurat slik jeg presenterer case i denne boken.

1.5 Eksamen og studiepoeng

Boken er lagt opp for selvstudium. I tillegg vil du ha støtte i videoene på nett. Det viktige er at du lærer å programmere og å lage nyttige systemer. Men skal du bruke din kompetanse til å få jobb, kan det være greit med dokumentasjon på at du kan programmere. For å få det kan du melde deg opp som emnestudent i faget IBE151 ved Høgskolen i Molde (søk på IBE151 og Canvas).

Som student vil en kunne laste ned Access fra Høgskolen, eventuelt bruke Access via en virtuell maskin som er tilgjengelig på nett. Har du planer om å gå opp til eksamen må du levere inn øvingene til de frister som er oppgitt. Du vil da også få hjelp av faglærere og øvingslærer. Eksamen må tas i Molde.

IBE151 inngår i årsstudiet i Informatikk og digitalisering. Mens emnet gir 15 studiepoeng, gir hele årsstudiet 60 studiepoeng. En kan imidlertid ta fagene i den rekkefølgen en vil, gjerne over flere år. Se www.himolde.no

2 Access – Komme i gang

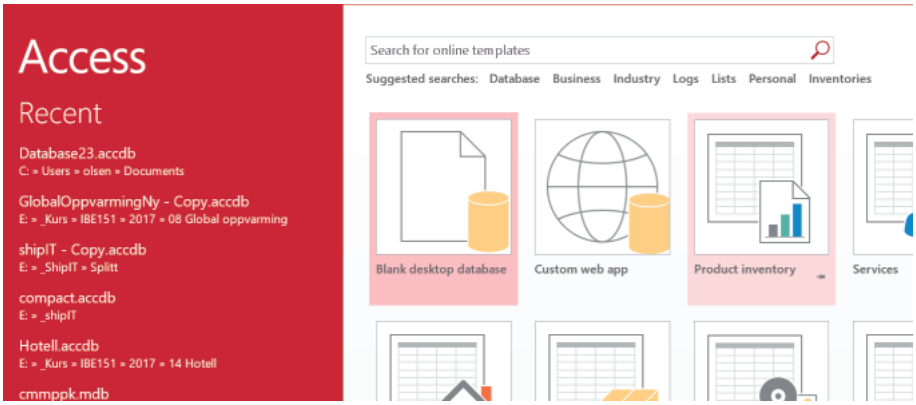
Microsoft Access er en del av Office Pro-pakken. Har en denne vil en finne Access her. Alternativt kan en leie tilgang til Access. Det får en for en billig penge. Studenter vil ofte kunne få gratis tilgang gjennom sin institusjon. Merk at du må ha utviklingsversjonen av Access.

Access finnes i mange versjoner. I praksis er det likegyldig hvilken utgave du bruker, men en av de nyere kan være en fordel. Selv bruker jeg Access 2013. Det finnes nyere versjoner, men det kan være en fordel å ligge litt på etterskudd her. Da er vi mer sikre på at feil er rettet før vi tar versjonen i bruk.

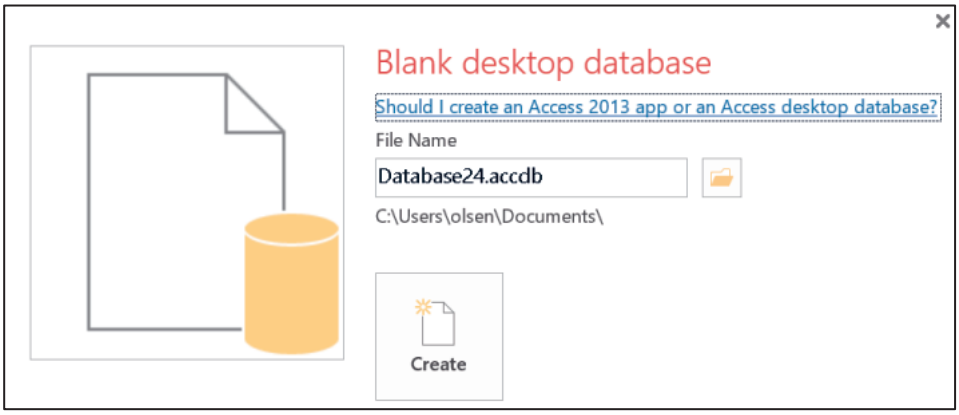
Som ved andre Office-produkter kan vi velge visnings-språk (display-språk). For min del har jeg valgt engelsk. Eksemplene i boken er derfor med engelsk versjon av Access. Selv om vi bruker engelsk som display-språk, kan vi selvfølgelig sette opp norsk tastatur og, om vi vil, norske hjelpetekster. Jeg vil imidlertid advare mot det siste. Mesteparten av hjelpetekstene er maskinoversatt. Det blir dårlig norsk, så for de fleste er det nok lettere å lese originalteksten på engelsk.

2.1 Starte Access

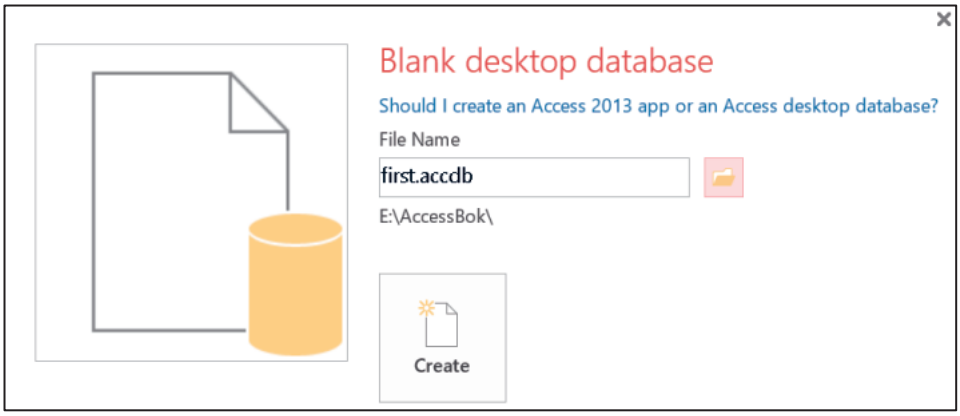
Etter å ha installert Access og lagt ikonet på skrivebordet starter du systemet med et dobbeltklikk.



Du får nå mange valg, mellom en rekke ferdige systemer. Vi velger imidlertid Blank desktop database, og får så anledning til å gi navn på systemet og velge hvor det skal lagres:

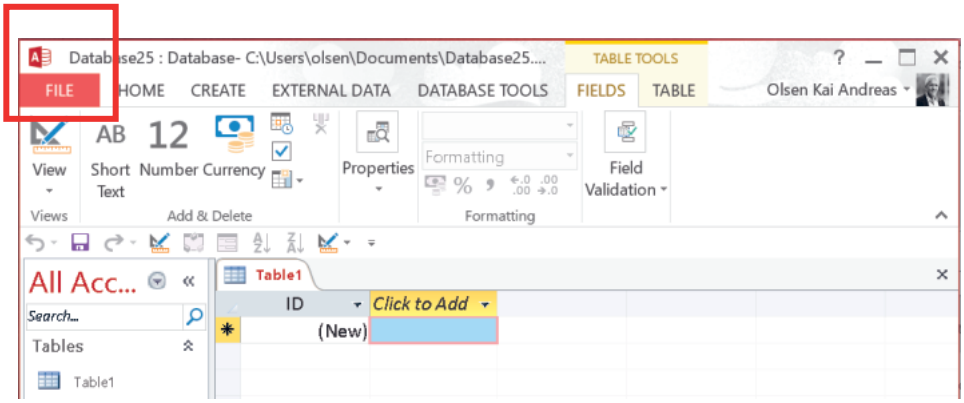


Vi skriver inn navn på Access-filen, her first.accdb (se figuren under) og trykker Create. Extension accdb gis til alle Access-filer (akkurat som .docx i Word og .xlsx i Excel). Med Access kan vi også lage en .accde versjon. Her har brukeren ikke tilgang til koden, men det har liten hensikt for oss.



2.2 Oppsett (Options)

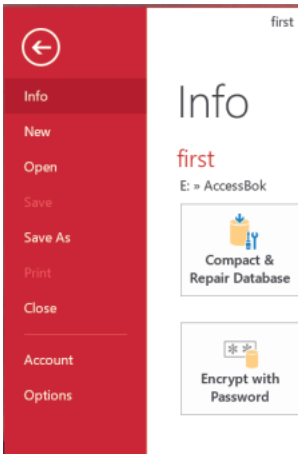
Access viser nå:



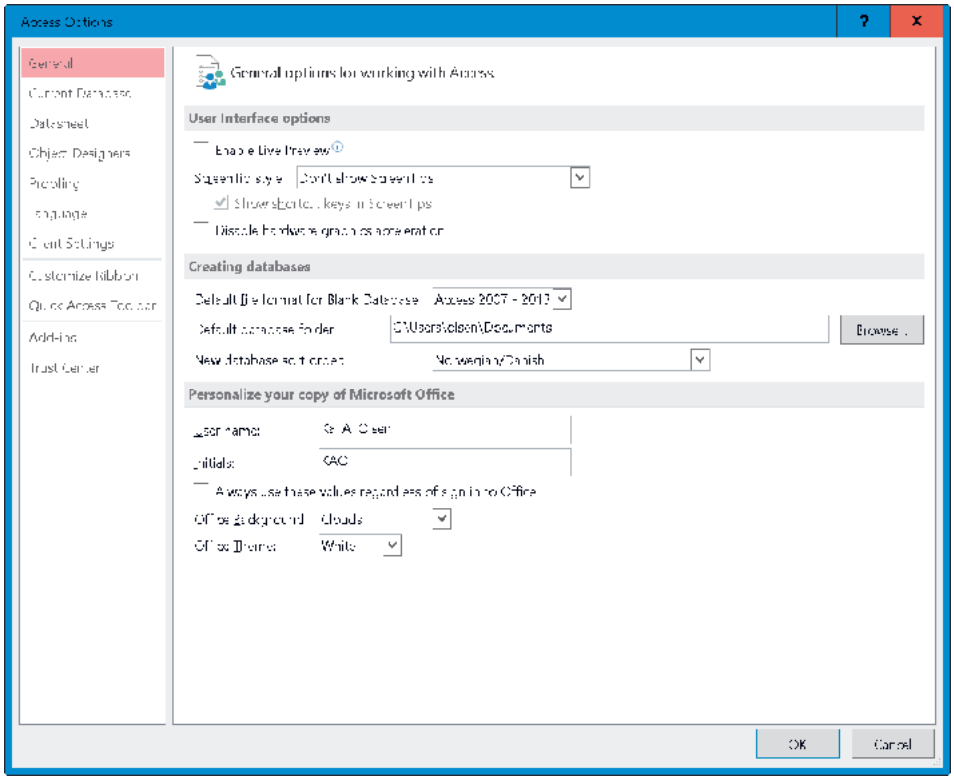
Dersom du ikke ser kolonnen til venstre, her med All Access Objects trykker du F11.

Dessverre har en eller annen programmerer hos Microsoft funnet ut at vi ofte ønsker å starte med å definere en tabell, derfor får vi denne (Table1) generert automatisk. Dessuten har denne programmereren funnet ut at tabellen skal ha et ID-felt, m.m. Trykk derfor X til høyre i Table1-linjen så kvitter vi oss med denne tabellen.

Det vi skal gjøre nå er å sette opp Access slik vi ønsker, slik at vi blant annet slipper denne automatikken. Trykk på File øverst til venstre (markert i figuren over), og du får fram denne menyen:



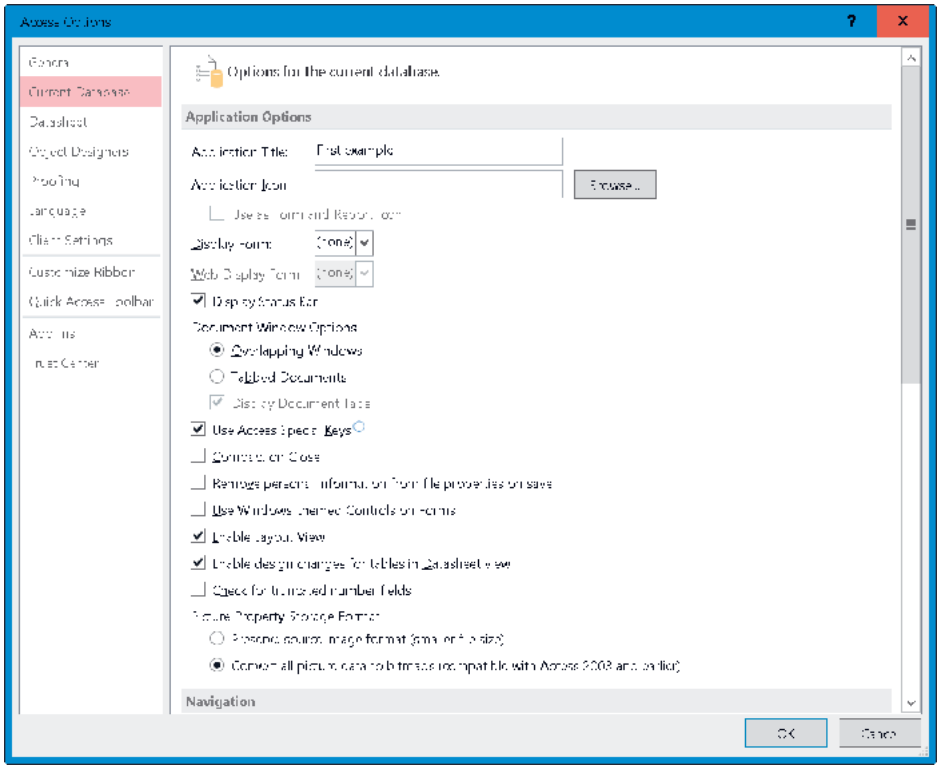
Velg Options (Alternativer i norsk versjon) og du får fram dette bildet:



Det første vi kan sette er General, slik som vist i figuren over. Her kan du legge inn eget navn og egne initialer, men ingenting her er av særlig betydning. Det blir annerledes for Current Database (se neste side).

Her kan en gi navn på applikasjonen (i vårt tilfelle First example) og velge ikon. Under Display form kan en fortelle Access hvilket skjema (form) en skal starte opp med. Vi skal komme tilbake til denne muligheten senere (kapittel 48). Av de andre settingene har jeg foretrukket å operere med Overlapping Windows og ikke Tabbed Documents. Klikk bort markeringen av ”Enable design changes for tables in Datasheet view”. Da blir det ikke lenger mulig å endre tabellen under visning av denne, og vi kan unngå at en bruker roter til databasen vår.

Du står selvfølgelig fritt til å velge dine egne preferanser her, men eksemplene i boken er satt opp ut fra dette.



Det som er litt kjedelig, er at dette står under Current database. Du må altså legge inn disse settingene for hver Access-applikasjon. Det kan imidlertid unngås ved å lagre det vi har gjort nå – trykk OK, deretter File > Close og vi får lagret first.accdb – en tom fil, men med de settings vi ønsker. Fra nå av kan vi bare kopiere denne når vi skal lage en ny Access-applikasjon, og får da med oss oppsettet.

Vi skal senere, i kapittel 5 vise hvordan vi setter oppsettet for programeditoren.

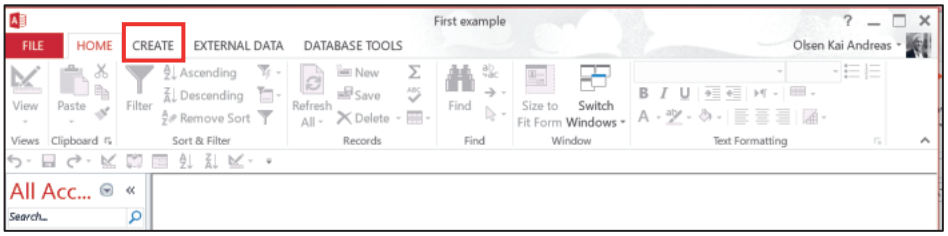
3 Access – Konstruere skjema (form)

Hva skal vi gjøre:

Her skal vi først lage en ny Access-fil, deretter opprette et skjema i denne. Så skal vi vise hvordan vi kan legge inn elementer, her en enkel knapp, i dette skjemaet. Dette er viktig basiskunnskap for å lage Access-applikasjoner.

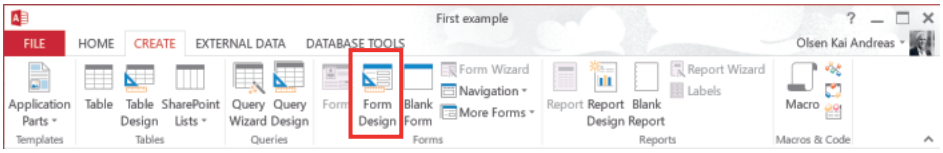
Finn fram til first.accdb, som vi laget i forrige kapittel, og kopier denne (trykk CTRL C, CTRL V). Gi kopien navn testSkjema.accdb (klikk på filnavnet og skriv inn det nye). Dobbeltklikk på testSkjema.accdb for å åpne denne.

3.1 Lage et nytt skjema

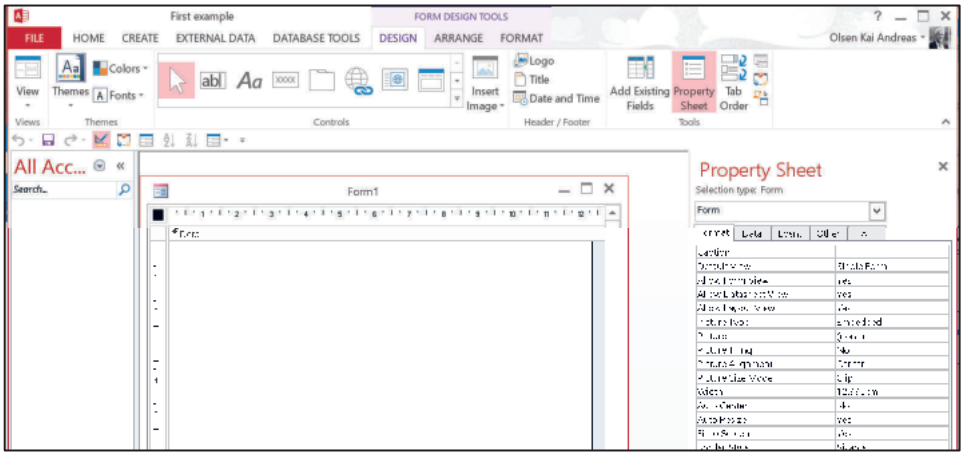


Det første vi skal gjøre er å opprette et skjema (form). Her skal vi legge inn brukergrensesnittet, knapper og felt m.m.

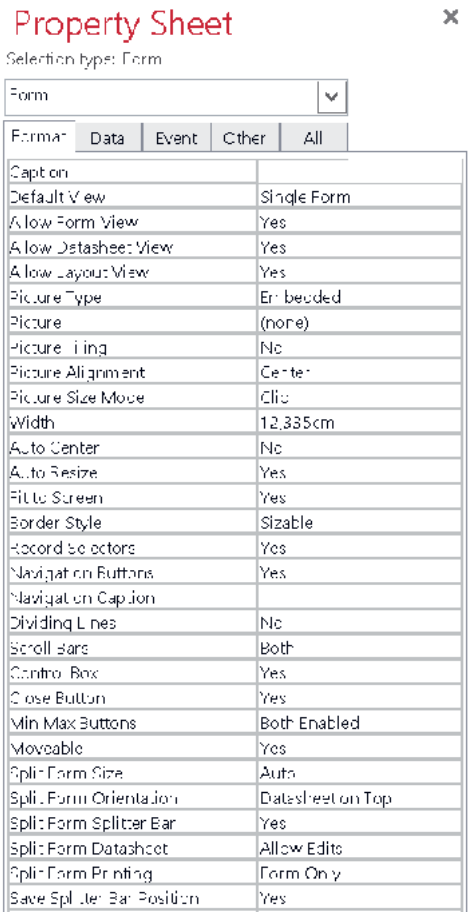
For å lage et nytt skjema trykker vi Create (se over) og velger deretter Form Design:



Vi får nå opp et blankt skjema (se neste side).



Egenskapene (Properties) til dette skjemaet står til høyre i et eget vindu. Om ikke egenskapene vises, kan du høyreklikke i skjemaet og velge Properties. Ser du andre egenskaper enn de over, må du klikke i det grå området utenfor skjemaet for å markere at det er egenskapene til hele skjemaet du vil se. I første omgang kan du skrive inn en tekst i feltet Caption øverst i fanen for Format, for eksempel ”testskjema”. Vi skal senere vise hvordan disse egenskapene kan tilpasses vårt formål.



Tips: Du kan også velge No for Record Selectors (en kjapp måte er å dobbeltklikke i feltet) og Navigating buttons, og får med det fjernet en kommandolinje som Access plasserer nederst i ethvert skjema, uvisst av hvilken grunn.