NTNU
Kunnskap for en bedre verden

Introduksjon til databaser

Tore Mallaug, NTNU

1. Introduksjon til databaser

Resymé: Denne leksjonen gir en kort introduksjon til noen kjente emner innen databaser, og nevner noen ulike databaseløsninger på markedet i dag. Leksjonen bør leses sammen med kapittel 1 i boka Databaser.

Innhold

1.1. Kommentarer til læreboka

Merk deg! Leksjonene i faget er kun ment som et tillegg til læreboka. Følgelig bør leksjonen leses sammen med kapittel 1 i læreboka. Leksjonen alene gir ingen full oversikt over faget, dette er overlatt til læreboka.

Les hele kapittel 1. Hele kapittelet gir nyttig kunnskap om basisbegreper i databaseteori og forståelsen av hva en database og et databasesystem er. Alt er like viktig her.

I leksjonen skriver jeg litt om hva som kjennetegner databaser og hvilke krav en bør sette til dataene som lagres. Til slutt nevner jeg noen kjente databaseløsninger i dag. Noe av stoffet er hentet fra boka "Database Systems" (Beynon-Davies 2000).

Er du interessert i noen av databasesystemene nevnt i leksjon finner du mye informasjon på de ulike produktenes hjemmesider på Web (leksjonen inneholder ingen lenker, men det er enkelt å søke på de ulike produktnavnene).

1.2. Hva kjennetegner databaser

De fleste moderne foretak i dag, f.eks. skoler, banker, organisasjoner og bedrifter, har behov for å lagre relevante data relatert til den daglige virksomheten. I tillegg kommer behov for å arkivere eller loggføre data (f.eks. ut fra juridiske eller skattemessig krav), og behov for å gjøre noe data tilgjengelig for ulike målgrupper, f.eks. kunder på Web. Et totalt informasjonssystem vil i praksis inkluderer lagring av data i en eller form, og et databasesystem vil i de aller fleste tilfeller være det beste valget til formålet.

1.2.1. Lagring og representasjon av data

Vi kan si at en database er en modell, eller en avspeiling, av deler av virkeligheten til en organisasjon. På engelsk kan dette kalles ”Universe of Discourse”. Databasen kan lagre og representere en forenkling av virkeligheten. Hvor god og hvor nære virkeligheten denne forenklingen er avhenger av hvor mye semantisk informasjon en har klart og lagt inn i databases design (struktur). D.v.s. hvilke attributter som lagres og hvilke datatype og domene / begrensninger (eng. constraints) hver av disse har (mer om attributter i neste leksjon). Databasen er en **faktabase**, hvor faktaene bør si mest mulig om den virkeligheten de representerer.

Dataene i basen bør være mest mulig **persistent**. Med persistent menes her at dataene har en viss varighet, eller holdbarhet. Denne varigheten trenger riktignok ikke å være lang, siden enkelte databasen er svært dynamiske med tanke på endring av data. Men vi kan skille mellom mer permanente data og mer ”fluktige” data. Permanente data er for eksempel persondata som kunde-, student- og pasientdata, produktdata eller transaksjoner i et regnskap. Disse dataene er persistente i en kortere eller lengre periode, noen helt statiske (f.eks. fødselsdatoen til en person). Eksempel på mer fluktige data er beregningsdata som brukes i en rapport/utskrift. Ofte er det ikke noe poeng å lagre et resultat av en beregning i databasen, hvis resultatet er avhengig av data som endres hyppig. Da er det bedre å utføre beregningen på nytt ved å søke etter (rå-)data i databasen hver gang en applikasjon (program) etterspør resultatet. Da vil resultatet alltid bli mest mulig oppdatert. En kan tenke seg at en ikke lagrer saldoen på en bankkonto, men kun transaksjonene (beløpene som har gått inn og ut). Så lenge alle transaksjonene er trygt lagret i databasen vil nåværende saldo alltid kunne beregnes.

Et annet viktig begrep er **dataintegritet**. Generelt betyr integritet her at dataene skal være presise i forhold til virkeligheten, eller kvaliteten på dataene om du vil. Ofte går dette ut på å holde dataene, attributtene, i databasen oppdatert til enhver tid. Dette høres åpenbart ut, men ikke alltid like enkelt i praksis. Tenk for eksempel hvis du bytter postadresse. Det kan ta flere år før alle databasene som har registrert din gamle adresse omsider oppdateres slik at du får all post til ditt nye bosted.

Hvor viktig det er å ha oppdaterte og korrekte data avhenger av den virkeligheten vi snakker om. For eksempel hvis et banksystem etterspør saldoen på en konto, bør databasesystemet returnere saldoen slik den er nå, ikke det den var i går eller i forrige uke. D.v.s. alle transaksjonene som angår kontoen må lagres kontinuerlig og raskest mulig i databasen uten feil eller mangler.

1.2.2. Funksjoner til en database

- Oppdateringsfunksjoner (skrive data)

Essensielt for en database er at det er mulig å skrive, oppdatere og slette lagrede data. Dette er de samme basisfunksjonene du også finner i et filsystem (knyttet til operativsystemet), men et

databasesystem tilbyr et høyere abstraksjonsnivå. Hvis du lagrer data ved bruk av databasesystem trenger du ikke tenke på filer, men heller hvilke attributter du synes er viktig å lagre, og egenskapene til disse.

- Funksjoner og begrensninger (constraints)

Knyttet til lagringen er det mulig, som nevnt tidligere, å sette begrensninger (regler) for dataintegritet og datatyper/domene til attributtene. Dette er en viktig funksjon for en database som du ikke finner i et tradisjonelt filsystem.

- Spørringsfunksjoner (lese data)

Mye av styrken til en database utover lagringen er at brukere kan aksessere, eller lese, data via et spørrespråk, typisk SQL. En organisasjon ønsker vanligvis ikke bare å lagre data på en forsvarlig måte, like viktig er det at dataene er tilgjengelig når (og hvor) du måtte ønske dem. En database gir store muligheter for datatilgjengelighet ved å tilby spørringer som returnerer raske svar. Vi kommer tilbake til spørringer utover i kurset.

- Flerbrukeraksess

Større databasesystem tilbyr at flere ulike brukere kan aksessere (skrive til / lese fra) en felles database samtidig. Dette er en svært nyttig funksjon for større organisasjoner. I relasjonsdatabaser vil det være mulig å opprette såkalte VIEW's i SQL som gir ulike brukergrupper tilgang til en begrenset del av databasen. På denne måten kan en kontrollere bruken av databasen. Dette vil også bedre datasikkerheten.

1.3. Valg av databaseløsning

Under følger noen punkter som kan vurderes ved valg av databaseløsning. Med databaseløsning her menes den totale databaseløsningen til en organisasjon eller bedrift, enten det er snakk om et sentralt databasesystem for hele organisasjonen eller flere mindre databasesystem som skal fungere sammen i et (internt) datanett. For å forenkle bildet litt snakker vi under om en database. Viktigheten for hvert punkt avhenger selvsagt av lokale behov. Vi forutsetter her at vi velger relasjonsdatabasesystemer.

- Lagringsstruktur, indeksering og "tuning"

God funksjonalitet for databaseadministrasjon er viktig hvis en skal lagre store mengder med data over tid og/eller man har strenge krav til oppetid og responstid fra databasen. Hvis det siste er viktig må man typisk velge en lagringsstruktur som dobbeltlagrer alle dataene i databasen, såkalt "speiling" av dataene. F.eks. banker og kortselskaper har slike krav til oppetid for kontoinformasjon om kundene for at minibanker og kortbetaling skal fungere optimalt hele døgnet. Dette står nevnt i læreboka kap. 8.3.

- Flerbrukerstøtte og datasikkerhet

Ved flere sluttbrukere av samme databasen er det selvsagt viktig at databasesystemet håndterer at flere aksesserer de samme dataene samtidig. Funksjonalitet som kreves her er at en unngår feil i data ved oppdatering (f.eks. hvis to brukere endrer de samme dataene samtidig), at responstida er god selv om mange brukere etterspør de samme dataene samtidig, og at datasikkerheten er i varetatt (bl.a. registrering og logging av brukere i databasen). Dette står nevnt i læreboka kap. 8.4 og 8.5.

- Pris og driftskostnader

Vi tar ikke stilling til økonomiske spørsmål i dette faget, men ved vurdering av ny databaseløsning bør det utføres et forprosjekt med en kost-nytte analyse av flere alternative løsninger.

- Støtte for SQL

Alle relasjonsdatabasesystemer støtter spørrespråket SQL. Men det er (dessverre) ”dialekt”-forskjeller ute og går. Det beste er at et databasesystem er mest mulig ”tro” til en systemuavhengig SQL-versjon. (Dette kan f.eks. lette overgangen til et annet databasesystem senere).

- Støtte for referanseintegritet

Referanseintegritet mellom tabeller (relasjoner) i en relasjonsdatabase sikrer en mest mulig konsistens database. Alle relasjonsdatabasesystem støtter referanseintegritet, men tidlige versjoner av MySQL har ikke en fullgod løsning på dette punktet.

- Konvertering til/fra andre dataverktøy / databaser

Ved overgang til en ny databaseløsning er det ofte et stort behov for å legge inn gamle data i den nye databasen. Gamle data kan være lagret på ulike filformat, noe som kan by på problemer hvis ikke databasesystemet tilbyr funksjonalitet for konvertering fra ”kjente” filformat til tabeller i relasjonsdatabasen. Eksempel på slik konvertering kan være fra regneark (f.eks. MS Excel) til database.

- Konvertering til/fra UML

UML kan gi en dokumentasjon av databasestrukturen. UML kan brukes til å lage klassediagram som viser en figurativ fremstilling av databasestrukturen – mer om dette i datamodelleringsdelen av kurset senere. En mulighet til å vise databasestrukturen i UML er spesielt nyttig hvis en ønsker å programmere mot databasen, f.eks. i programmeringsspråket Java (kan f.eks. utnyttes av innleide datakonsulenter).

- Konvertering til/fra XML (evt. JSON)

XML er viktig ved datautveksling med eksterne samarbeidspartnere på Internett f.eks. i e-handel. Følgelig er det etter hvert et stort spørsmål hvordan data på XML-formatet skal lagres i databaser. De fleste kommersielle databasesystemene på markedet i dag, som Oracle og MS SQL Server, gir støtte for lagring av XML i relasjonsdatabase, enten ved omforming til ”vanlige” data i ”vanlige” tabeller, eller ved løsninger som utvider hva som kan lagres i databasen (objektrelasjonell løsning – kap.7 i læreboka (ikke pensum)). XML er ikke pensum i dette faget, men nevnes kort helt til slutt i læreboka (kap. 10.8).

Et alternativ til XML er JSON (JavaScript Object Notation). JSON har en noe enklere syntaks enn XML, og har seilt opp som et godt alternativ til lagring av i alle fall relativt enkle datastrukturer. F.eks. MySQL (fra versjon 5.7) kommer med støtte for lagring av JSON. Dette kan f.eks. brukes til å lagre koordinater i kartdata (GIS) ved bruk av GeoJSON. Heller ikke JSON er pensum i dette faget.

1.4. Noen kjente databaseløsninger

Under nevner vi fire ulike databaseløsninger. Vi ønsker ikke å ta stilling til hvilke kommersielle produkter som er best. Dette er en vurderingssak. Etter å ha gjennomført dette kurset vil du forhåpentligvis ha kunnskap som hjelper deg i valg av databaseløsning i f.eks. bedriften du jobber i.

1.4.1. Databasesystem for enkeltbruker

De enkleste databaseverktøyene i dag er de som er beregnet å kjøre på vanlig PC sammen med andre verktøy som tekstbehandling og regneark. MS Access er et godt eksempel på et slikt verktøy, og passer fint som ”treningsverktøy” i dette kurset siden du lett kan installere og kjøre verktøyet på egen PC.

Enkelte vil nok mene at MS Access egentlig ikke er et databasesystem, kun et verktøy i MS Office-pakken. Men MS Access inneholder faktisk mange av de funksjonene du finner i større databasesystem, for eksempel en nesten fullgod SQL-versjon for å lage spørringer mot databasen (SQL kommer vi tilbake til i leksjon 4 og 5).

Fordelen med en slik løsning er at den er enkel å kjøre på vanlig PC.

Ulempen er at den har dårlige funksjoner for flere brukere samtidig. En MS Access database vil nok fort gå ”varm” hvis mange brukere prøver å bruke samme databasen samtidig (det betyr at responstiden fra databasesystemet blir lang). Den passer best for en enkeltbruker eller for å tilby enkle databasetjenester på en Web-side, for eksempel enkle søk mot en ikke alt for stor prisliste.

1.4.2. Flerbrukerdatabasesystem – kommersiell

Typisk vil en større organisasjon ha behov for:

- Lagre større mengder data av ymse slag
- La flere ulike brukere (eller brukergruppe) aksessere ulike delmengder av disse dataene

Det vil da være lurt å velge et kraftigere databaseverktøy som er skalerbar både med tanke på stor og økende datamengde og til ulike sluttbrukere som ønsker å bruke databasen samtidig i et datanett. Tradisjonelt vil en da velge en databasetjener fra en kjent leverandør. I dag er det Microsoft (SQL Server) og Oracle som er størst (Oracle er et av USA’s største programvareleverandør etter Microsoft, så det sier litt om hvor enormt markedet for databaseløsninger er, siden Oracle stort sett kun selger databaseløsninger). Det finnes også andre aktører på markedet som IBM (databasesystemet DB2).

Ulempen med en ”skikkelig” databaseløsninger fra disse leverandørene er at systemene er relativt store og dyre i innkjøp, og krever drift- og vedlikeholdsressurser (databaseadministrasjon, læreboka kapittel 8). Innkjøp og valg av databasetjener bør være en overveid beslutning ut fra en behovsanalyse og langsiktig tenkning.

Fordelen med slike løsninger er at de lett lar seg utvide til for eksempel å takle økt datamengde og nye behov og bruk. Databasesystemene har for eksempel innbygd interne rutiner som både optimaliserer lagringen og søketiden ved uthenting av data fra databasen. I tillegg har systemene funksjoner som letter administrasjonen av databasen, som adgangskontroll og sikkerhetskopi, så løsningen kan gi økt datasikkerhet. Enda et argument for (som selvsagt leverandører poengterer) er at programvare du har betalt for bør yte det du

ble forespeilt ved innkjøp. Hvis ikke har du jo noen du kan klage til! Et alternativ til ”dyre” databaseløsninger som er inn i tiden er åpne løsninger (se under).

1.4.3. Flerbrukerdatabasesystem – ”open source”

”Open Source” betyr her at all programvaren er åpent tilgjengelig på Internett for gratis nedlasting. Slike løsninger knyttes tradisjonelt opp mot operativsystemet Linux, men i prinsippet er det ikke noe problem å ”blande” valg av operativsystem og databasetjener. F.eks. at du kjører en Oracle-server på Linux, eller MySQL (se under) på MS Windows. Åpne løsningene er i praksis et reelt alternativ til Microsoft og Oracle, som begge innrømmet at de tar konkurransen fra Linux-miljøet alvorlig. Et svar fra Oracle er å tilby en begrenset versjon av databasesystemet for gratis nedlasting. Denne versjonen kan bl.a. brukes av studenter i en opplæringssituasjon eller programutviklere som ønsker å utvikle programvare mot Oracle. Vil du derimot bruke Oracle til aktiv lagring av data i en organisasjon må du fortsatt betale.

I dagens marked er de to mest populære åpne tilgjengelige databasesystemene MySQL og PostgreSQL. MySQL er i utgangspunktet enklere enn PostgreSQL, d.v.s. den har færre funksjoner i SQL, men utviklingen av MySQL går fort. Begge systemene har så langt færre funksjoner for flerbrukerproblematikk enn større kommersielle databasesystem, og er (foreløpig) dårligere på en del spesielle funksjoner som støtte for XML. MySQL er laget i Sverige, men ble vinteren 2008 kjøpt opp av amerikanske Sun Microsystems. Sun tilbyr også gratisprogramvare for databaseløsninger knyttet til programmeringsspråket Java (Java brukes ikke i dette kurset). En kan kanskje spørre seg hvordan Sun har tenkt å tjene penger på databaseløsninger som alle kan bruke gratis. Her er det nok "ettermarkedet" Sun tenker på. For selv om selve MySQL er gratis, vil mange kunder sikkert trenge datakonsulenthjelp (som installasjon, drift (databaseadministrasjon) og brukeropplæring), samt også ha behov for spesialtilpassete tilleggsmoduler utover "standard" MySQL. Dette kan Sun og andre ta seg betalt for på samme måte som Microsoft og Oracle tilbyr bruker- og kundesupport til sine kunder. Så selv om databasesystemet er gratis trenger det ikke å bety at driftskostnadene er lik null.

Fordelen med å velge MySQL eller PostgreSQL er at de er gratis.

Ulempen er at du i utgangspunktet mister tryggheten ved å ha en kjent leverandør i ryggen, selv om denne ulempen nok er i ferd med å forsvinne ettersom også de som tilbyr åpne løsninger i dag er kjente og store leverandører (f.eks. Sun). For de som ikke trenger konsulenthjelp finnes det også mange råd og vink på Web.

1.4.4. Programsystem med ”innebygd” database

Et alternativ til et databasesystem er å bruke ferdige programsystem som lagrer data ”internt” i programmet. Et eksempel er elektroniske pasientjournaler (EPJ). Hvis et sykehus ønsker å lagre pasientdata elektronisk, kjøper de neppe et databasesystem, men heller et EPJ-system. EPJ kan ta seg av lagringen uten at en trenger å tenke på noe databasesystem. Et annet eksempel kan være regnskapssystemer som lagrer regnskapspostene i programmet. Eller et elektronisk administrasjonssystem.

Fordelen med det vi kan kalle en ”innebygd” database er at en ikke trenger å kjøre et frittstående databasesystem. Datalagringen kan også spesialtilpasses kravene til det konkrete systemet.

Ulempen er at en mister friheten et frittstående databasesystem gir. Ofte vil en organisasjon ha et totalt lagringsbehov som inkluderer ulike typer data som et enkelt programsystem neppe

dekker fullt ut. Ut fra et databasesynspunkt vil det da være en fordel å samle alle dataene i en felles database i stede for å ha dem spredd rundt på ulike systemer som kanskje har problemer med å utveksle data seg imellom (dette omtales i kapittel 9 i læreboka).

1.5. NoSQL og store datamengder

Det har alltid eksistert alternativer til relasjonsdatabaser. I de senere år har populariteten til NoSQL[1]-løsninger vært økende. En NoSQL-database bruker ikke SQL som spørrespråk mot dataene lagret i systemet og representerer ikke data som tabeller (relasjoner) på et logisk nivå.

Argumentene for og imot NoSQL-løsninger bunner ut i hvor stor grad en har behov for funksjonaliteten til tradisjonelle relasjonsdatabasesystemer. Noen argumenter for NoSQL er at for enkelte typer data er det viktigst å kunne lagre unna data raskt og effektiv, gjerne som en kontinuerlig strøm av data, og at man ikke alltid har behov for flerbrukerstøtten (transaksjonshåndteringen) relasjonsdatabasesystemer tilbyr. Videre kan det argumenteres for at det gir stor fleksibilitet å slippe å forholde seg til et fast skjema i databasen. I dette faget lærer vi to typer skjema; ER-modellen (ER-diagram) på et konseptuelt nivå, og relasjonsmodellen (tabeller) på et logisk nivå. Begge setter noen strukturelle begrensninger på hvordan f.eks. kundedata skal representeres i databasen. Dette er et av poengene med å bruke skjema. Et annet poeng er at en må tenke igjennom hvilke data som er viktige (viktigst) å lagre når en designer databasen. Dropper man skjemaet (eng. schemaless) kan en i prinsippet lagre kundedata akkurat slik det måtte passe, og det gir stor fleksibilitet hvis f.eks. hvilke kundedata en ønsker å lagre endrer seg hyppig. Samtidig er nok begge disse punktene; mer eller mindre mangel på transaksjonshåndtering og skjema, eksempler på punkter hvor det er uenighet mellom tilhengere av NoSQL og tilhengere av mere tradisjonelle databaseløsninger.

Det ingen trenger å være uenige om er imidlertid at mengden data (informasjon) som en lagrer i dag er veldig mye større enn før datakommunikasjon og Web tok av. I tillegg er fysiske lagringskapasitet blitt veldig billig med åra, så en trenger ikke å tenke veldig mye på hvor mye data en lagrer lenger (relativt sett). Behovet for å skufle unna data i såkalte skalerbare løsninger er stort. En skalerbar løsning klarer å tilpasse seg stadig større mengder lagrede data ved å skalere opp både antall maskiner (noder) og antall lagringsenheter (som harddisker) i det totale systemet. Dette takler NoSQL-løsninger bra. Et godt eksempel er lagring av "datastrømmer" fra sensorer, f.eks. lagring av værobservasjoner. Dette kan også kobles til "Tingenes internett" (eng. "Internet of Things"), dvs. at i fremtiden vil alle dingser være koblet til Internett, og ha behov for å lagre unna data i større eller mindre grad.

Bruksområder for store mengder data kan knyttes til stordata (Big Data), og inkluderer dataanalyser som bl.a. kan brukes til å finne ut hvor potensielle kunder bor geografisk. Eller hvilke hus/nabolag et politisk parti bør ringe på foran et valg; hvor det er størst mulighet til å finne potensielle velgere ut fra demografiske data e.a. Det å analysere historiske data er ikke noe nytt. I relasjonsdatabasesammenheng har det lenge eksistert såkalte datavarehus med muligheten til å lagre aggregerte data på grunnlag av rådata fra en eller flere databaser. Dataene samlet sammen i datavarehuset kan brukes til datautvinning (eng. data mining), f.eks. for å beregne salgsprognoser ut fra innsamlet salgsdata. Med stadig større mengder data tilgjengelig har dataanalyse fått en ny aktualitet. De etiske perspektivene, f.eks. knytt til

[1] NoSQL kan best tolkes bokstavelig til et system som ikke bruker SQL. En litt mere utflytende tolkning er å lese det som "Not only SQL", dvs. et system som ikke bare bruker SQL, men kombinerer SQL med andre løsninger.

personvern, ved bruk av store mengder lagret informasjon er en annen side ved dette som dette faget ikke tar stilling til.

Hvis vi sier at en NoSQL-database lagrer data som objekter, eller poster om du vil, kan en skille mellom 4 ulike typer løsninger:

1) Nøkkel-verdi -basert (eng. Key-Value), hvor hvert objekt er en verdi assosiert med en nøkkel.
2) Kolonnebasert, hvor hvert objekt består av flere verdier assosiert med en nøkkel. Et eksempel på et slikt system er Cassandra.
3) Dokumentbasert, hvor hvert objekt kan betraktes som et dokument med en intern struktur hvor det kan finnes (interne) relasjoner mellom verdier. Ofte vil slike systemer representere data i et XML/JSON -liknende format, dvs. en trestruktur som kan betraktes som et dokument. Et eksempel, blant flere, på et slikt system er MongoDB.
4) Grafbaserte, hvor objekter i databasen representeres i en grafstruktur (nettverk) med relasjoner til andre objekter. Et sosialt nettverk er bygd opp på denne måten, med potensielle lenker på kryss og tvers mellom deltakerne.

En kan også tenke seg hybridløsninger med en blanding av relasjonsdatabase og NoSQL. NoSQL trenger ikke erstatte en ren relasjonsdatabase i en bedrift eller organisasjon, men komme som et tillegg for lagring av data som en ikke trenger å ha tilgjengelig i relasjonsdatabasen. F.eks. en web-butikk kan fortsatt ha (strukturerte) data om kundene og bestillinger i en relasjonsdatabase, mens f.eks. data om kundenes "adferd", som hvilke varer kundene klikker mest på, i web-butikken evt. kan lagres i NoSQL.

NoSQL er ikke pensum i dette faget, men det er (heldigvis) ikke slik at det du lærer i faget er helt fånyttes hvis du skal jobbe med NoSQL-løsninger. Faget presenterer grunnleggende ideer om hvordan en kan strukturere data (inkl. datamodellering), hvordan en kan lagre og endre data på en strukturert måte og hvordan hente data ut fra databasen (SQL). Dette er nyttig lærdom å ta med seg uansett hvordan du velger å lagre dataene dine i fremtiden.

1.6. Litteraturliste

Her nevnes noen utenlandske databasebøker inkludert en del velkjente bøker som finnes i flere utgaver (se etter nyeste utgave hvis du ønsker å kjøpe noen av dem):

Beynon-Davis, P.: Database Systems, Macmillan.

Blaha, M.R.: A Manager's Guide to Database Technology, Building and Perchasing Better Applications, Prentice Hall.

Date, C. J. : Introduction to Database Systems, Addison-Wesley.

Elmasri, R., Navathe, S. : Fundamentals of Database Systems, Addison-Wesley.

Conolly, T., Begg, C., Strachan, A. : Database Systems, A Practical Approach to Design, Implementation, and Management, Addison-Wesley.

McFadden, F.R., Hoffer, J.A., Prescott, M.B. : Modern Database Management, Addison-Wesley.