Databaser

# Kapittel 1 – introduksjon til databaser og databasesystemer

## **Læringsmål**

## - Hva som karakteriserer filbaserte system

## - Hva en database er

## - Hva som kjennetegner et databasesystem

## - Hva som kjennetegner ANSI/SPARC-arkitekturen

## - Hva en databasemodell er

## - Hva som kjennetegner en flerbrukerdatabase

|  |  |
| --- | --- |
| ***Desentraliserte system*** | Hvert applikasjonsprogram definerer og behandler sine egne data. |
| ***Data*** | Det vi gjerne vil lagre, altså bokstaver, tall eller tegn |
| ***Felt*** | Inneholder et sett med tegn som gir en viss logisk mening, for eksempel; et navn, tall dato, klokkeslett eller kode. Sier noe om tings egenskaper. |
| ***Post*** | Et sett med felt som hører sammen, f.eks. alle medlemmene i en sportsklubb, ansatte i en bedrift eller alle varene på et lager |
| Ei fil er altså en samling poster som inneholder et sett med logiske data. | |

Problemene med filbaserte system

* Dobbelt lagring av data (Redundans)
* Dataen er avhengig av programmet, og dermed er det gjerne faste spørringer, og for hver ny spørring må et nytt applikasjonsprogram utvikles
* Ingen kontroll over tilgang og behandling av data utover det som er implementert i applikasjonsprogrammet
* Data kan være lagret på ikke-kompatible filformat.

Databaser

* En database er en samling av logiske data som hører sammen, og en beskrivelse (datatype, lovlige data etc.) av disse dataene designet for å møte informasjons kravene i en organisasjon
* Databasen er et enhetlig lager av data som blir definert en gang.
* En database modellerer en del av verden, for eksempel en virksomhet, og dataene i databasen er kjente fakta som kan lagres og sier noe om virksomheten

Databasesystem

* Et databasesystem er en samling programmer og verktøy som gjør det mulig å opprette og bruke en database, og som sørger for en kontrollert tilgang til databasen
  + MySQL er et eksempel på et databasesystem
* Databasesystemet inneholder et datadefineringsspråk (Data Definition Language - DDL). Dette brukes til å opprette databaser og tabeller.
* Det inneholder også datamanipulerendespråk (Data Manipulation Language - DML) som inneholder et sett av operasjoner for grunnleggende datamanipulerede operasjoner på dataene i databasen.
  + F.eks. legg inn data, endre data og hente ut data
* Fordeler med databasesystemer
  + SQL
    - Gir muligheten for å aksessere og endre på data i databasen på en relativt enkel måte.
  + Stor datakraft
    - Databaser er effektiv og rask å gjøre oppslag i
  + Fleksibelt
    - Dette betyr at lett å foreta endringer.
    - Logisk datauavhengighet – endre den logiske strukturen på data uten å måtte skrive om en masse program
    - Fysisk uavhengighet - endre den fysiske lagringsstrukturen på data uten å måtte skrive om en masse program
  + Bedre sikkerhet
    - Man kan gi folk forskjellige rettigheter i databasen
  + Flerbrukerkontroll
    - Et databasesystem sørger for at det ikke oppstår problemer når flere vil endre data samtidig
  + Backup
    - Du slipper å bekymre deg for å miste data om strømmen skulle gå.
  + Individuell tilpasning
    - Man kan f.eks. opprette views slik at brukeren kun får den dataen den er interessert i
  + Datauavheninghet
    - Databasen tar seg av lagringen slik at flere frittstående applikasjoner har tilgang til de samme dataene
    - En database kan være et såkalt flerbrukersystem, med mange applikasjoner som bruker mye av de samme dataene
  + Prøve å unngå dataduplisering

Databasearkitektur

* ANSI/SPARC-arkitetkturen (American National Standards Institute, Standards Planning And Requirements Committee) er den viktigste generelle databasearkitekturen idag.
* ANSI/SPARC-arkitekturen er delt i 3 nivåer:
  + Det eksterne nivå
    - Er databasen slik den ser ut for den enkelte sluttbrukeren. Den totale beskrivelsen som ligger i det eksterne nivå, er databasens eksterne skjema.
  + Det konseptuelle (logiske) nivå
    - Representerer databasen på modellnivå
    - Man ser tabeller, kolonner, datatyper og integritetsregler
  + Internt
    - Representerer det nærmeste vi kommer av fysisk lagring av data.
    - Dette nivået sier noe om hvordan dataene får sitt interne lagringsformat.
* Samlet blir de 3 nivåene kalt arkitekturens skjema

Datamodeller

* En databasemodell er en integrert samling av begrep som beskriver data, sammenhengen mellom data og regler som er knyttet til data.
* En databasemodell er ment til å gi en oversikt over hvilken type info vi vil ha med i databasen.
* Postbaserte datamodeller
  + Relasjonsmodellen
    - En relasjonsdatabase er bygd opp av relasjoner, og relasjonene knyttes sammen ved hjelp av kombinasjonen primær- og fremmednøkkel
    - Om man skal lage en relasjonsdatabase, bør den først beskrives i en ER-modell og deretter oversettes til relasjonsmodell.
* Objektbaserte datamodeller
  + ER-modellen
    - Konseptuell modell som brukes til å beskrive hvordan virkeligheten fungerer
    - Sier noe om entiteter (objekt vi lagrer info om). En entitet vil ha en rekke egenskaper som er representert som en mengde attributter. En entititet vil ha en eller flere sammenhenger (relationships) til andre entiteter i databasen.
    - Det er viktig å skille mellom entiteter og entitetstyper
      * En entitetstype sier noe om hvilke data vi kan lagre i attributtene, hvilke sammenhenger vi tillater, og hvordan dette skal identifiseres i databasen
      * Vi har f.eks. entitetstypen ansatt, mens vi kan ha entiteten Ole Olsen.

Flerbruker databasearkitektur

* Klient/tjener-modellen splitter programsystemet i en klientapplikasjon og en tjenerkomponent. Klientapplikasjonen samler spørsmål og data fra brukeren, forbereder den til tjeneren og gjør deretter forespørselen til tjeneren. Tjenerkomponenten venter på sin side på forespørsler fra sine klienter. Når den får en forespørsel prosesserer det den og returnerer svaret til klienten. Klienten representerer tilslutt dette til brukeren gjennom et brukergrensesnitt.
* Med andre ord kan flere maskiner knyttes sammen i et kommunikasjonsnettverk slik at databearbeidingen kan foretas over flere maskiner.

# Kapittel 2 – Relasjonsmodellen

## **Læringsmål**

## - Relasjonsmodellens opphav og begrepene som brukes i relasjonsmodellen

## - Hvordan en relasjon blir brukt til å representere data

## - Sammenhengen mellom en matematisk relasjon og en relasjon i relasjonsmodellen

## - Relasjonenes egenskaper

## - Hvordan vi kan indentifisere super-, kandidat-, primær-, alternative og fremmednøkler

## - Hva som ligger i begrepene entitets- og referanseintegritet

## - grunnleggende relasjonsalgebra

* Relasjonsmodellen er en metode for å representere data på, og hvordan vi kan manipulere denne representasjonen. Modellen bygger på matematiske prinsipp hentet fra mengdeteori og predikatlogikk
* Delt inn i tre deler
  + Datastruktur

|  |  |  |
| --- | --- | --- |
| Fagkode | Fagnavn | Studiepoeng |
| TDAT2004 | Datakommunikasjon | 7.5 |
| TDAT2002 | Matematikk 2 | 15 |
| TDAT2003 | Systemutvikling 2 | 12 |

* + - Relasjon
      * Fortelle sammenhengen mellom attributtene. For eksempel relasjonen som utgjør Fag over, består av fagkode, fagnavn og studiepoeng.
    - Attributt
      * Hodet til relasjonen består av et fast antall attributtnavn (Fagkode, fagnavn, studiepoeng). Her har hvert attributt en verdi som svarer nøyaktig til en underliggende datatype.
      * Attributtene inneholder kun atomiske verdier.
      * Datatypen er hvilken type verdien attributtene kan ha, f.eks. tall, tekst, dato osv.
      * Antallet attributter i en relasjon utgjør relasjonens grad. Her i eksempelet har vi 3 attributter, dermed grad 3
    - Domene
      * Verdiområdet, altså en mengde med mulige lovlige verdier som et attributt kan ha.
    - Tuppel
      * Relasjonens kropp består av en mengde med tupler, der antallet og innholdet varierer med tiden. (i eksempelet har vi 3 tupler)
      * Det finnes ingen like tupler i en relasjon.
        + Viktig med at hver tuppel har en entydig primærnøkkel
      * Antall tupler i relasjonen utgjør relasjonens kardinalat
    - Relasjonelle nøkler
      * Supernøkkel
        + Er en attributtkombinasjon som entydig identifiserer en tuppel i en relasjon
        + En tuppel kan ha mange supernøkler
        + Kombinasjonen av alle attributtene er alltid en supernøkkel
      * Kandidatnøkkel
        + Om vi har funnet en relasjons supernøkkel og fjernet de overflødige attributtene får vi relasjonens kandidtatnøkkel (minimal supernøkkel)
        + Er en hvilken som helst entydig dataverdi for et enkelt attributt eller en kombinasjon av attributter som det bare finnes en av i relasjonen
        + Alle relasjoner har en kandidatnøkkel, fordi relasjonen per definisjon ikke kan ha to tupler som er like
        + Fra relasjonenes kandidatnøkler velger vi primærnøkkelen.
      * Alternative nøkler
        + Kun en av kandidatnøklene velges som primærnøkkel. De øvrige kandidatnøklene kalles da alternative nøkler.
      * Primærnøkkel
        + Entydig Identifikasjon på en tuppel i en relasjonsmodell
        + Kan ikke ha null verdi dvs. ikke være oppgitt
      * Sammensatt nøkkel
        + Ofte er det ikke nok med ett attributt som primærnøkkel for å entydelig identifisere en tuppel. Dermed må vi bruke flere.
        + For sammensatte nøkler (komposittnøkler) kan ingen av attributtene i nøkkelen ha null verdi.
        + En sammensattnøkkel betraktes som en primærnøkkel
      * Surrogatnøkkel
        + Brukes når det ikke er en naturlig eller fornuftig primærnøkkel å hente for en tuppel
        + Opprettet entydig for å være en primærnøkkel for tuppelen, dette er gjerne et tellefelt i databasen.
      * Fremmednøkkel
        + Viktigste bindingen i en relasjonsdatabase
        + En fremmednøkkel er et attributt eller en kombinasjons av attributter som referer til verdien til en primærnøkkel i en annen relasjon.
  + Dataintegritet
    - Henspeiler datakvaliteten i databasen. Ved vedlikehold av databasen, sørger integritetsreglene for at dataene i databasen er riktig til enhver tid.
    - Entitetsintegritet
      * Ingen attributter som utgjør hele eller deler av en primærnøkkel kan være uten verdi
      * Altså ingen primærnøkkel kan være uten verdi
    - Referanseintegritet
      * Fremmednøkkelen skal alltid referere til en eksisterende primærnøkkel.
      * Fremmednøkkelen kan ha en null verdi, men dette indikerer at den ikke refererer til en annen relasjon.
  + Datamanipulering
    - Når vi snakker om manipulering av databaser snakker vi som regel om å lese og skrive data til og fra databasen.
    - Relasjonsalgebra
      * Seleksjon
        + Lager en ny relasjon ut fra bestemte tupler i en eksisterende relasjon
        + Horisontaldeling av relasjonen

Et bilde som inneholder tekst, hvit, tog

Automatisk generert beskrivelse

* + - * Projeksjon
        + Lager en ny relasjon ut fra bestemte attributter i en eksisterende relasjon
        + Vertikal deling av relasjonen

Et bilde som inneholder tekst, skjerm, flislagt, hvit

Automatisk generert beskrivelse

* + - * Produkt
        + Det kartetiske produktet eller kryssproduktet lager en ny relasjon med alle mulige sammensetninger av tupler fra to relasjoner A og B
        + Antall tupler i R er lik antall tupler i A multiplisert med antall tupler i B

Et bilde som inneholder foto, hvit, klokke, svart

Automatisk generert beskrivelse

* + - Forening
      * Lager en ny relasjon med sammensetninger av tupler fra to relasjoner på et felles attributt, slik at hver sammensetning tilfredsstiller en gitt betingelse
      * Indreforeninger
        + Likhetsforening (equijoin)

Forening basert på likhet. Her vil kun de tuplene som har sammenfallende verdier i de spesifiserte attributtene.

* + - * + Theta-forening (thetajoin)

Om vi har et vilkår k for attributtene der A.k < B.k for alle fellesattributter

* + - * + Naturlig forening (natural join)

En likhetforening hvor vi har fjernet duplikatattributtene.

* + - * Ytterforening
        + Venstre ytterforening

Returnerer alle tuplene i venstre relasjon selv om de ikke finnes i den høyre. Disse vil få verdien null i attributtene tilhørende høyre tabell

* + - * + Høyre ytterforening

Returnerer alle tuplene i høyrerelasjon selv om de ikke finnes i den venstre. Disse vil få verdien null i attributtene tilhørende venstre tabell

* + - * + Fullstendig ytterforening

Her vil alle tuplene fra begge tabellene dukke opp, selv om de ikke er koblet sammen.

* + - Settoperasjoner
      * Union
        + Union lager en ny relasjon med tupler som finnes i den ene eller begge av to relasjoner
        + Relasjonene må være unionkompatible, altså de må ha samme antall attributter, og at attributtene har samme datatype og domene, men ikke nødvendigvis samme navn.
        + Ser på det som å slå sammen alle radene i to relasjoner, der de tuplene som er lik kun dukker opp en gang.
      * Snitt
        + Lager en ny relasjon med tupler som finnes i begge av to relasjoner
        + Relasjonene må være unionkompatible
      * Differanse
        + Lager en relasjon med tuplene som finnes i den første men ikke i den andre av de to relasjonene
    - Grupperingsoperatorer
      * Danner grupper av tupler som har en eller annen egenskap felles. Dette betyr at tupler med samme verdi i ett eller flere gitte attributter, danner en gruppe.
      * Divisjon som er en av 8 grupperingsoperasjoner, og regnes som en grupperingsopreasjon fordi grupperingen som gjøres under utførelsen.
* Hvordan vet vi at vi har en relasjonsdatabase?
  + Det må ha relasjonsdatabser, det vil si databaser som brukeren kan oppfatte som tabeller og ingenting annet enn tabeller
  + Det må minst ha operasjonene seleksjon, projeksjon og forening uten å måtte forhånds definere fysiske aksessveier for å kunne brukedisse operasjonene.

# Kapittel 3 – Normalisering

## **Læringsmål**

## - Hvorfor vi må normalisere relasjonene våre

## - Problemene knyttet til dobbeltlagring av data

## - Begrepet funksjonell determinering (avhengighet)

## - Hvordan vi normaliserer

## - Hvordan vi kan bestemme normalformen til en relasjon når det gjelder de vanligste normalformene

## - Når de er vi ikke normaliserer

Hva er normalisering og hvorfor normaliserer vi?

* Normalisering går ut på å dele opp databasen i mange små tabeller for å unngå at informasjonen blir registrert flere steder (redundans) i databasen, noe som skaper unødvendig stor datalagring. Man gjør det også for å forhindre at man får data som ikke stemmer inn i databasen, såkalte inkonsistenser. Dette er uregelmessigheter som kan oppstå må man endrer data u databasen.
* Tabell = relasjon, Attributt = kolonne, tuppel = rad

Funksjonelle determinering (avhengigheter)

* Ideelt sett skal verdiene til alle kolonnene i en tabell være determinert av verdier i primærnøkkelen.Vi kaller kolonnen som er valgt til primærnøkkel for determinanten (en kandidatnøkkel er også en determinant).
* Eksempel
  + Om vi har en tabell over Sted som har kolonene postnr og poststed, vil de funksjonelle determineringene i tabellen vøre beskrevet slikt
    - sted.postnr 🡪 sted.poststed.
    - Altså dersom vi vet postnr vet vi poststedet, det finnes kun ett postnr pr poststed. Men om vi kun har poststedet kan den ha flere forskjellige postnr

Unormalisert form

* En tabell på unormalisert form (UNF) kan inneholde det vi kaller for repeterende grupper (ikke-atomiske verdier).
* For eksempel
  + Om vi har en tabell over studenter med en kolonne for fag de har, vil denne kolonnen plutselig inneholde flere verdier, dermed er den ikke-atomisk. For å løse dette kan vi normalisere tabellen

Første normalform

* Første normalform (1NF) krever at tabellen har en primærnøkkel og kun atomiske verdier.
* Eksempel

Vi har denne tabellen som vi vet ikke er på 1NF siden den ikke har kun atomiske verdier:

|  |  |  |  |
| --- | --- | --- | --- |
| Studnr | Etternavn | Fornavn | Fagkode |
| 101 | Hansen | Ole | LO323D, LO343D |
| 102 | Jensen | Karen | LV370D, LO323, LO248D |

Vi endrer tabellen slik at den kun har atomiske verdier altså at den ikke har noen repeterende grupper:

|  |  |  |  |
| --- | --- | --- | --- |
| Studnr | Etternavn | Fornavn | Fagkode |
| 101 | Hansen | Ole | LO323D |
| 101 | Hansen | Ole | LO343D |
| 102 | Jensen | Karen | LV370D |
| 102 | Jensen | Karen | LO323 |
| 102 | Jensen | Karen | LO248D |

Denne tabellen vil oppfylle 1NF, men ofte er ikke 1NF fullkommen

Andre normalform

* For at en tabell skal være på andre normalform (2NF), må den oppfylle alle kravene for 1NF, og i tillegg så må de kolonnene som ikke inngår i primærnøkkelen være avhengig av hele primærnøkkelen ikke bare deler av den.
* En tabell er på 2NF om hver kolonne som ikke er en del av primærnøkkelen determineres av hver kandidatnøkkel
* Tabeller som har ikke sammensatt primærnøkkel og ingen andre kandidatnøkkel automatisk er på 2NF, fordi det da aldri vil være partielle determineringer i tabellen
* Eksempel

Vi har tabellen:

|  |  |  |  |
| --- | --- | --- | --- |
| Studnr | Fagkode | Fagnavn | Semester |
| 102 | LO248D | Datakommunikasjon | V98 |
| 104 | LO248D | Datakommunikasjon | H97 |
| 101 | LO323D | Databaser | H97 |
| 102 | LO323D | Databaser | V98 |
| 103 | LO323D | Databaser | H97 |
| 104 | LO323D | Databaser | H97 |
| 105 | LO323D | Databaser | H97 |
| 101 | LO343D | Grunnkurs EDB | H97 |
| 102 | LO370D | Webteknikker | H97 |

Her ser vi at den ikke er på 2NF siden kolonnen fagnavn ikke bestemmes av hele primærnøkkelen (studnr, fagkode), men bare fagkode delen av nøkkelen. For å få denne på 2NF lager vi derfor en ny tabell for fag, der fagkode er primærnøkkelen.

Fagvalg (studnr, fagkode\*, semester)

|  |  |  |
| --- | --- | --- |
| Studnr | Fagkode | Semester |
| 102 | LO248D | V98 |
| 104 | LO248D | H97 |
| 101 | LO323D | H97 |
| 102 | LO323D | V98 |
| 103 | LO323D | H97 |
| 104 | LO323D | H97 |
| 105 | LO323D | H97 |
| 101 | LO343D | H97 |
| 102 | LO370D | H97 |

Fag (fagkode, fagnavn)

|  |  |
| --- | --- |
| Fagkode | Fagnavn |
| LO248D | Datakommunikasjon |
| LO323D | Databaser |
| LO343D | Grunnkurs EDB |
| LO370D | Webteknikker |

Disse to tabellene vil derimot oppfylle 2NF kravene, og vil derfor være på andre normalform.

Tredjenormalform

* En tabell er på tredje normalform om den oppfyller kravene for andre normalform og det ikke finnes noen *transitive determineringer* mellom kolonner som ikke er kandidatnøkler. Dette vil si at vi ikke har en tabell der en av kolonene bestemmes av en annen kolonne som ikke er med i primærnøkkelen.
* Eksempel

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Studnr | Etternavn | Fornavn | adresse | Postnr | poststed |
| 101 | Hansen | Ole | Kombinertkroken 3 | 7563 | Malvik |
| 104 | Vik | Kari | Prinsens gate 34 | 3490 | Grimstad |
| 105 | Hansen | Nils | Saupstadringen 39A | 7078 | Saupstad |
| 103 | Tormodsen | Per | Kongsveien 13 | 7223 | Melhus |
| 102 | Jensen | Karen | Elgveien 45 | 9900 | Kirkenes |

I tabellen over ser vi at kolonnen postnr bestemmer kolonnen poststed, men den er ikke en del av primærnøkkelen(studnr). For å få tabellen på 3NF lager vi en ny tabell sted, som holder oversikt over postnr og poststed.

Student (studnr, etternavn, fornavn, adresse, postnr\*)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Studnr | Etternavn | Fornavn | adresse | Postnr |
| 101 | Hansen | Ole | Kombinertkroken 3 | 7563 |
| 104 | Vik | Kari | Prinsens gate 34 | 3490 |
| 105 | Hansen | Nils | Saupstadringen 39A | 7078 |
| 103 | Tormodsen | Per | Kongsveien 13 | 7223 |
| 102 | Jensen | Karen | Elgveien 45 | 9900 |

Sted (postnr, poststed)

|  |  |
| --- | --- |
| Postnr | poststed |
| 7563 | Malvik |
| 3490 | Grimstad |
| 7078 | Saupstad |
| 7223 | Melhus |
| 9900 | Kirkenes |

Tabellene over vil være på 3NF

Boyce-Codd normalform

* BCNF sier at det ikke må finnes funksjonelle determineringer mellom kandidatnøkler, og at enhver determinant er en kandidatnøkkel
* Når en tabell har kun en mulig kandidatnøkkel og allerede er på 3NF, er den også automatisk på BCNF,
* En tabell med kun 2 kolonner alltid være BCNF gitt at 1NF er oppfylt.
* Om vi har en tabell med flere enn en kandidatnøkkel og mer enn to kolonner, vil ikke 3NF alltid gi en normalisert løsning. Da kan vi få indirekte determineringer. Derfor må vi bruke BCNF i slike tilfeller.
* Eksempel se bok side 94 til 96

Når skal vi ikke normalisere?

* Av og til kan det være ønskelig å velge en lavere normalform. En av disse grunnene kan være ytelse. Om spørringene skal utføres raskt slipper du med en lavere normalform å joine/forene tabellene for å få den originale tabellen.
* Om man velger å ikke normalisere burde man ha en god grunn og ha en god forklaring på hvorfor man har valgt å droppe det.

# Kapittel 4 – SQL – opprette en tabell og enkle spørringer

## **Læringsmål**

## - Hvorfor vi bruker SQL

## - Hvordan vi oppretter tabeller

## - Hvordan vi lager integritetsregler

## - Hvordan vi skriver SQL-spørringer som gjør endringer i databasen

## - Hvordan vi skriver SQL-spørringer som henter data fra tabeller

## - Hvordan vi håndterer NULL-verdier

Oppretting av tabeller | Datadefinisjonsspråk

**CREATE TABLE *navnPåTabell***

**(**

**Navnp*åkolonne1 datatype(x),***

**Navnp*åkolonne2 datatype(x),***

**Navnp*åkolonne3 datatype(x),***

**CONSTRAINT navnPÅTabell\_pk PRIMARY KEY (nøkkelattributt1, nøkkelattributt2)**

**);**

* Alle kolonner i en tabell må ha et domene. Dette gjøres ved å bruke datatyper. En datatype kan f.eks. være CHAR om vi skal ha en fastsatt lengde på noe som telefonnr er alltid 8 tall om de er norsk. Da kan vi skrive inne telefon\_nr CHAR(8). Om lengden kan variere som f.eks. navn kan vi bruke VARCHAR som vil kan variere innenfor et visst intervall.
* At vi bruker CONSTRAINT ... PRIMARY KEY (...) viser at vi legger til en entitetsregel,
* Om vi vet at vi skal bruke en kolonne som fremmednøkkel må vi husket at navnet på kolonnen må være lik i begge tabellene

**Eksempel**

CREATE TABLE sted

(

🡪 denne kan ikke være null pga den skal brukes som en foreign key i en annen tabell

postnr CHAR (4) NOT NULL,

poststed VARCHAR (50),

CONSTRAINT sted\_pk PRIMARY KEY (postnr)

);

* Når tabellene vi skal ha er opprettet må vi tilføye referanseintegritetsregler. Altså vi må implementere fremmednøkler.
* Om vi har en tabell og skal legge inn en ny rad i tabellen så må verdien vi fyller ut i den kolonnen som er fremmednøkkelen allerede eksistere. Dette fører til at vi har mer regler når det kommer til sletting av data.

**Eksempel**

ALTER TABLE student

ADD CONSTRAINT student\_fk FOREIGN KEY (postnr)

REFERENCES sted (postnr);

* Om vi har flere primærnøkler i en tabell så må vi ha like mange fremmednøkler slik at vi får knyttet tabellene sammen.
* NB! En fremmednøkkel alltid må være en primærnøkkel i en annen tabell. Dette betyr at når vi skal opprette referanseintegriteten, så må den kolonnen som skal være fremmednøkkel, allerede være opprettet som primærnøkkel i den tabellen det referer til (mor-tabellen). Altså må entitetsintegriteten implementeres først. Derfor er rekkefølgen vi oppretter tabellene i viktig.
* I tillegg til entitets- og referanseintegritet i tabellene kan vi også implementere dataintegritet. F.eks. om vi vil ha unike navn på fag i en tabell eller at en tabell med karakterer skal gå fra 1 til 6

**Eksempel**

ALTER TABLE fag

ADD CONSTRAINT karakter

CHECK(karakter BETWEEN 1 AND 6);

ALTER TABLE fag

ADD CONSTRAINT fag\_entydig UNIQUE(fagnavn);

* Man kan også legge til kolonner eller fjerne kolonner med ALTERTABEL

**Eksempel**

ALTER TABLE student

ADD favorittfag VARCHAR(30);

ALTER TABLE student

DROP favorittfag VARCHAR(30);

* Om vi vil endre betingelsene for en eksisterende kolonne kan dette gjøres med MODIFY

**Eksempel**

ALTER TABLE student

MODIFY etternavn VARCHAR(34);

* Vi kan slette hele tabeller med kommandoen DROP TABEL *navnpåtabell,* men husk at vi må slette fremmednøkkel må slettes før vi får lov til å slette tabellen den referer til.

Legge inn data/endre data | Datamanipuleringsspråk

Manipulering av radene i en tabell

* Legge inn data
  + INSERT INTO tabellnavn (kolonne1, kolonne2, kolonne3) VALUES (verdi1, verdi2, verdi3);
* Oppdatere data
  + UPDATE tabellnavn SET kolonne = verdi;
* Slette data
  + DELETE FROM tabellnavn WHERE betingelse;
* Commit – Rollback
  + Når vi bruker en flerbrukerdatabase så må vi utføre en commit etter hver spørring, og om vi oppheve spørringen vår kan vi gjøre ett roleback

Utvalgsspørringer

* Hva skal med i SELECT-listen?
  + Om vi vil ha alt fra tabellen skriver vi kun SELECT \*, om vi kun skal ha noen kolonner kan vi spesifisere hvilke kolonner etter SELECT
  + I tillegg kan vi lage nye navn på kolonnene vi henter ut
    - F.eks. SELECT studnr, fagkode, karakter AS tall\_karakter FROM fagvalg.
  + Om vi ikke vil ha duplikater i en kolonne kan vi skrive DISTINCT foran kolonne navnet
    - F.eks. SELECT DISTINCT studnr FROM fagutvalg
* Avledete kolonner
  + Dette er kolonner som ikke er en del av strukturen til basistabellen, men som blir opprettet i resultattabellen.
    - F.eks. SELECT fagkode, fagnavn, karakter, karakter\*3 AS studiepoeng FROM fag;
  + Man bruker operatorene +,-,\*,/ til å kalkulere verdiene til en avledet kolonne. Kolonnenavnet opprettes med AS.
* Utvalg
  + Når vi skal gjøre et utvalg bruker vi WHERE. Denne krever at vi setter en betingelse.
  + Disse er de lovlige betingelsesoperatorerene i SQL
    - = (lik)
    - > (Større enn)
    - < (Mindre enn)
    - <= (Mindre eller lik)
    - >= (Større eller lik)
    - != (Ulik)
  + Vi kan også bruke boolske uttrykker for å lage sammensatte betingelser. Da kan vi bruke
    - AND (må oppfylle begge betingelsene, prioriteres over OR)
    - OR (Returnere det som oppfyller en av betingelsene)
    - NOT (Returnerer de som ikke oppfyller noen av betingelsene)
  + Om vi vil finne rader som oppfyller en betingelse i et intervall kan vi bruke BETWEEN x AND y
    - F.eks. SELECT \* FROM fag WHERE karakter BETWEEN 2 AND 4
  + IN operatoren lar deg spesifisere flere verdier i WHERE betingelsen.
    - F.eks. om du er ute etter å finne alle kunder i landene Frankrike, Sverige og Russland kan vi skrive:
      * SELECT \* FROM kunder WHERE land IN ('Frankrike', 'Sverige', 'Russland');
    - Vi kan også bruke NOT IN for å finne de kundene som ikke er i de landene
  + Om vi vil finne noe der vi leter etter likhet i form av tekst bruker vi LIKE operatoren. Med denne kan vi finne f.eks. folk som begynner på en spesifikk bokstav eller slutter på en spesifikk bokstav.

Sortering

* Vi kan sortere tabellen vi får ut når utfører en spørring, for å gjøre dette bruker vi operatoren ORDER BY.
  + ORDER BY kolonne DESC (Sorterer i synkende rekkefølge)
  + ORDER BY kolonne ASC (Sorterer i stigende rekkefølge)
  + Om vi vil sortere alfabetisk kan vi f.eks skrive ORDER BY etternavn, fornavn. Da vil den først sortere etternavnene alfabetisk så etter navn alfabetisk.

Beregningsfunksjoner

* Arbeider på en hel kolonne.
* SQL har 5 beregningsfunksjoner
  + AVG() - Returnerer gjennomsnittet av verdiene i en gitt kolonne. Gjelder bare kolonner som inneholder numeriske verdier)
  + COUNT() – Returnerer antall verdier i en gitt kolonne (telle antall rader).
  + MAX() – Returnerer den største verdien i en gitt kolonne.
  + MIN() - Returnerer den minste verdien i en gitt kolonne.
  + SUM() – Returnerer summen av verdiene i en gitt kolonne. Gjelder bare for kolonner med numeriske verdier. Overser rader som inneholder NULL-verdier.

Gruppering

* Gruppering går ut på å gruppere rader for å beregne verdier med utgangspunkt i en beregningsfunksjon. Man bruker GROUP BY etterfulgt av kolonne navnet, og dataen blir dermed grupperes etter verdiene i denne kolonnen. Om vi har flere kolonnenavn med i uttrykket, vil den prioritere avhengig av rekkefølgen.
* F.eks.
  + SELECT ting FROM tabell(er) [WHERE betingelse] GROUP BY grupperingskolonner [HAVING betingelse]
  + Det som står i [] er frivillig å ha med.

Delspørringer

* Delspørring eller nøstet spørring kan brukes i SELECT-, SELECT...INTO-, INSERT...INTO-, DELETE- og UPDATE spørringer.
* IN brukes ofte sammen med delspørringer, dette er som regel om vi vil ha flere rader, siden ellers vil vi bruke boolske uttrykk og disse vil kun returnere en rad.
* Delspørringen kommer i en parentes etter WHERE og betingelsen.

NULL-verdier

* I SQL har vi en treverdiet, altså vi kan ha sant, usant og ukjent. NULL gir oss info om at verdien i kolonnen er ukjent, noe som kan føre til problemer om vi ikke er obs når vi skriver SELECT setninger.
* Boolske uttrykk og NULL-verdier
  + Ved bruk av boolske uttrykk vil ikke radene med kolonnen det spørres om som inneholder NULL returneres. Dette må man være obs på!
* Test på NULL-verdier
  + IS og IS NOT brukes som en test mot NULL-verdier.
* Aggregering og NULL-verdier
  + Om vi bruker COUNT() kommandoen og velger en spesifikk kolonne vil ikke de radene med NULL inkluderes, dermed må vi bruke COUNT(\*) om vi vil ta med NULL verdiene.
* GROUP BY og NULL-verdier
  + Alle NULL-verdier håndteres som like når vi grupperer.
* NULL-verdier ved bruk av DISTINCT og beregninger
  + Vil kun returnere NULL-verdien en gang.
* Delspørringer og NULL-verdier
  + Delspørringer som inneholder NULL-verdi rader vil ikke bli returnert.

# Kapittel 5 – SQL – flertabellspørringer og utsnitt

## **Læringsmål**

## - Hvordan vi skriver SQL-spørringer som henter data fra flere tabeller

## - Hvordan vi skriver SQL-spørringer som oppretter utsnitt

## - Hvordan vi skriver SQL-spørringer som legger inn og endrer data i utsnitt

## - Hvordan vi skriver SQL-spørringer som henter data fra utsnitt

Flertabellspørringer (foreningsspørring)

* Dersom vi vil ha en resultattabell som inneholder data fra mer den en tabell, må vi bruke joins, der vi andgir mer enn en tabell i FROM-setningen og spesifiserer i WHERE -setningen hvilke kolonner som tabellen forenes på
* Det kartesiske produkt
  + Det kartesiske produktet (CROSS JOIN) er det vi får når vi forener to tabeller ved å multiplisere hver rad i den ene tabellen med alle radene i den andre tabellen.
  + Brukes ikke særlig ofte pga. den er tungvinn og gir ikke noe særlig bra oversikt over infoen i tabellen
  + F.eks. SELECT \* FROM tabell1 CROSS JOIN tabell2;
* Naturlig forening
  + Naturlig forening eller indre forening inkluderer ofte en likhetsforening.
  + I en naturlig forening fjernes alle duplikatkolonner
  + F.eks. SELECT kolonne1, kolonne2, kolonne3 FROM tabell1 INNER JOIN tabell2 ON tabell1.primærnøkkel = tabell2.fremmednøkkel;
    - I tabll1 er kolonnen primærnøkkel eller en del av en sammensatt primærnøkkel det som kobler tabell1 til tabell2, siden det vil være fremmednøkkelen i tabell2.
    - Vi må ha alias på tabellene for å kunne skille de navnene på kolonnene som er like.
      * F.eks. student s og sted st, der vi bruker s.postnr = st.postnr for å koble tabellene sammen.
  + Vi kan også bruke USING for å koble tabellene sammen
    - F.eks. SELECT navn, etternavn, by FROM student s INNER JOIN sted st USING(postnr) ORDER BY etternavn, navn
    - Vi ser at da bruker vi ikke alias dott notasjonen
  + Om vi vil ha en naturligforening for flere tabeller enn to må vi koble de på en felles nøkler.
  + Se eksempel 159
* Selvresultat
  + Vi bruker selvresultat (self-join) når vi ønsker å forene en tabell med seg selv
  + Se side 160 for eksempel
* Ytterforening
  + Det er ofte behov for å hente informasjon som er fordelt på to tabeller og liste opp også de radene i den ene tabellen som ikke matcher noen rader i den andre tabellen. Dvs. vi vil få NULL-verdier der det ikke matcher istedenfor at det ikke dukker opp.
  + Bruker LEFT JOIN for å lage venstre ytterforening.
    - En venstre ytterforening tar med alle radene fra den første, altså den venstre tabellen, selv om det ikke finnes tilsvarende verdier i noen av radene i den andre tabellen, altså den høyre tabellen
  + Bruker RIGHT JOIN for å lage høyre ytterforening
    - En høyre ytterforening tar med alle radene fra den andre, altså den høyre tabellen, selv om det ikke finnes tilsvarende verdier i noen av radene i den første tabellen, altså den venstre tabellen
  + En ytterforening kan bli nøstet inne i en naturlig forening, men ikke motsatt.
  + Om vi vil ha med alle radene som ikke matcher bruker vi FULL JOIN
  + Se eksempler 161
* Foreninger på ikke-nøkkelkolonner
  + Man kan også forene tabeller over kolonner som ikke er nøkkelkolonner. Utgangspunktet må være minst to tabeller som har en felles ikke-nøkkelkolonne
* EXISTS
  + EXISTS eller NOT EXISTS brukes bare i forbindelse med delspørringer.
  + Dette funker ved at den ytre spørringen sender verdier til den indre spørringen for å se om verdien matcher betingelsene som er spesifiserte i den indre spørringen. Vi må bruke alias på kolonnenavnene om kolonnenavnet forekommer i mer enn en tabell.
  + I den indre spørringen blir det SELECT \*, siden spørringen vil bli vurdert om den er sann eller usann basert på om den returnerer rader eller ikke.
  + Ved EXISTS blir verdien sann fra den indre spørringen om den returnerer minst en rad, og usann om den ikke returnerer noen. Det motsatte gjelder for NOT EXSISTS. EXSIST tester ikke på tomme datasett.

Delspørring eller forening?

* Delspørringer og foreninger vil ofte gi forskjellige resultater men som en tommelfingerregel:
  + Vi bruker delspørringer når vi ikke har behov for å skrive ut data fra kolonner som tilhører tabellene i den indre spørringen.
  + Vi bruker forening når vi ønsker å skrive ut data også fra kolonner i den indre spørringen.
  + Se side 165 for eksempel

Sett-operasjoner

* Vi kan knytte to tabeller sammen med UNION. Dette forutsetter at tabellene er unionkompatible.
  + Altså antall kolonner og kolonnesekvensen må stemme overens i begge tabellene.
  + Kolonnenavnene kan være forskjellige, og kolonnene trenger ikke å ha samme størrelse og datatype.
* Alle duplikatrader fjernes fra en union setning. Med mindre vi bruker UNION ALL
* F.eks. SELECT \* FROM tabell1 UNION SELECT \* FROM tabell2 ORDER BY kolonne;
* Vi kan bruke snitt (INTERSECT) og MINUS
  + Snitt finn felles rader
  + Minus finner ikke er felles for tabell1 i tabell2

Virtuelle tabeller

* En virtuell tabell (view), også kalt utsnitt eller vindu, representerer et subsett bestående av kolonner og/eller rader og/eller uttrykk fra en eller flere basistabeller eller andre utsnitt.
* Fordelen med et view er at vi får større grad av logisk dataavhengighet. Dette betyr at den virtuelle tabellen er upåvirket av endringer i basistabellen. Den er immun mot restrukturering.
* Ved bruk av views kan vi tilpasse til ulike brukergrupper. Dette kan gjøre databasen mer sikkerhetsvennlig og forbedre dataintegriteten. Personer skal f.eks. ha tilgang til å endre info i en hel tabell, men kun deler av den, da kan views brukes.
* Om vi vil bruke views til å endre data i basistabellen kommer det med noen begrensninger. Den kan ikke inneholde uttrykk, funksjoner, mengdeoperatorer, DISTINCT, GROUP BY eller ORDER BY.
  + En god regel er at man ikke bruker et view til å legge inn nye rader, slette eller endre rader i en basistabell. Spesielt ikke når viewen bygger på flere basistabeller som er koblet sammen.
  + Se eksempel side 170
* Om vi vil bruke et view til å endre en basistabell kan vi kun endre en ikke-nøkkelkolonne.
  + Sletting vil ikke funke eller å legge til nye rader
* Vi kan bruke WITH CHECK OPTION om vi vil forhindre nye rader som ikke passer inn i utsnittet blir lagt inn, eller at gamle rader får en uønsket verdi.

# Kapittel 6 – Datamodellering

## **Læringsmål**

## - Hvordan vi kan designe en database ved hjelp av en ER (Entity-Relationship)-modell

## - De grunnleggende begrepene i ER-modellen

## - Hvordan vi designer ER-modeller med UML (Unified Modeling Language)-notasjonen

## - Hvordan vi utvider ER-modellen med begrep hentet fra objektorientering (EER-modell)

Entitetstyper og entiteter

* En entitetstype vil være som en klasses i et klassediagram. Entitetstypen har klare avgrensninger og finnes beskrevet i problembeskrivelsen som substantiv som av brukeren oppfattes å ha en avhengig eksistens.
  + F.eks. om vi har entitetstypen sted vil entiteter være Oslo eller Bergen.
    - En entitet er et objekt som kan identifiseres entydig. Entiteter har noe til felles, tilhører samme entitetstype.
* Sterke entitetstyper
  + En sterk entitetstype er en entitetstype som eksisterer uavhengig av andre entitetstyper.
  + En sterk entitetstype har en egen, entydig nøkkel representert av {PK} i attributtslisten, som identifiserer hver entitet
* Svake entitetstyper
  + En svak entitetstype er avhengig av en annen entitetstype for sin egen identitet. Den er identitetsavhengig av en annen entitetstype.
  + En svak entitetstype kan ikke identifiseres av sine egne attributter alene, derfor må vi bruke en fremmednøkkel sammen med en/flere av attributtene for å lage en primærnøkkel.

Attributt

* Attributter sier noe om egenskapene til entitetene i entitetstypen
* Vi skriver «+» foran disse i entitetstypediagrammet.
* Sammensatt attributt
  + Et sammensatt attributt er et attributt som er sammensatt av flere enkeltattributter
  + Ofte er det vanlig å utelate det sammensatte attributtet i ER-modellen og heller legge inn enkeltattributtene direkte
* Flerverdiatributt
  + Et flerverdiattributt har et sett med verdier for et enkeltattributt
  + For å indikere en flerverdiattributt i modellen setter vi [0..\*], der vi indikerer at den kan ha en verdi fra 0 til mange
  + Problemet med denne er når vi skal oversette til relasjonsmodell, og for å løse dette kan vi opprette en egen tabell med det vi har flere verdier av som PK og referanse til tabellen vi skal bruke den i som FK.
* Avledet attributt
  + Representerer en verdi avledet fra verdien til ett eller et sett av tilhørende attributter.
  + Disse indikeres i modellen med /-tegnet foran navnet, og vil ikke fysisk bli lagret på databasen men vil bli funnet ved hjelp av f.eks. en spørring i SQL.
* Nøkkelattributt
  + Brukes til å entydig identifisere entiteter av typen som har samme PK som entitetstypen. Disse brukes for å opprettholde integriteten til dataene i databasen.

Sammenhengstyper

* En sammenhengstype symboliseres med en linje som trekkes mellom de aktuelle entitestypene. Navnet vi gir sammenhengstypen er typisk et verb og en preposisjon, som «bor på» eller «tilhøre», eller «arbeider for», som forteller hva sammenhengen beskriver.
* Vi har også en multiplisitet som beskriver antall sammenhenger en entitet i en entitetstype kan ha med entiteter i en annen entitetstype.
* Multivegs sammenhengtyper
  + Mest vanlig er todimensjonale sammenhengsstyper. Det er sammenhengen mellom entiteter i to entitestyper.
  + Eksempel side 189
* Flervegs sammenhengstyper
  + Tredimensjonale sammenhengstyper, sammenhenger mellom entiteter i tre entitetstyper. Se eksempel 189
  + Vi bruker dette mest når vi har mange til mange sammenhengstyper og prøver å bruke binært om vi har en til en mellom en av de tre
  + Se eksempel side 189.
* Hierarkier
  + I rekursive sammenhengstyper kan entitene i den samme entitetstypen opptre i forskjellige roller.
  + Denne vil gjerne ha en fremmednøkkel til sin primærnøkkel
  + Se eksempel side 191
* Multiplistet
  + Multiplisenten bestemmer antall mulige sammenhenger mellom entiteter i en sammenhengstype.
  + Vi har alltid en multiplisitet i en sammenhengsstype.
  + Der symbolet til venstre er 1 vil avhengigheten være total og entiteten er dermed eksistensavhengig.
  + Der symbolet til venstre er 0 vil avhengigheten være delvis og entiteten vil ikke nødvendigvis er knyttet til en entitet i den andre entitetypen.
  + En-til-en
    - 1..1
      * Totalavhengighet altså må fremmenøkkelen være på siden der 1..1 er.
    - 0..1
      * Delvisavhengighet
    - Eksempel side 193.
  + En-til-mange
    - Vanligste sammenhengstypen i ER-modeller
    - Oppgis slik: x..y
      * 0..\*
      * 2..4
      * Osv.
    - Noen ganger hender det at vi vil ha en egen attributt som ikke er direkte knyttet til noen av entitestypene
      * Attributtene vil alltid havne på mange siden når vi oversetter til relasjonsmodell
    - Se side 196 for eksempler
  + Mange-til-mange
    - Vanlig at sammenhengstypen har egne attributter
      * Dette er en koblingstabell som vi ha en sammensattnøkkel bestående av de to tabellene den kobler sammen sin PK, som også er FKer

Konstruksjon av ER-modeller

* Rekkefølgen i modellereringsprosessen:

|  |  |  |
| --- | --- | --- |
| Trinn | Datastruktur | Restriksjoner |
| 1 | Bestem generelle entitetstyper | Bestem typer og multiplisiteten til sammenhenstypen |
| 2 | Spesialiser entitetstypen | Bestem typer og restriksjoner til spesialiseringen |
| 3 | Bestem sammenhengstyper | Finn svake entitetstyper |
| 4 | definer attributter og nøkler | * ¨ |
| 5 | kontroller modellen:  Finnes det generalisering/ spesialisering?  Kan vi slå sammen sammenheng super?  Finnes det sammenheng typer som bør modelleres med egne attributter?  Finnes det flerverdiattributter? |  |

Objektorgjentering i ER-modeller

* Generalisering
  + prosessen som går ut på å redusere forskjellene mellom entitet typer ved å identifisere deres fellestrekk.
  + Generalisering er det bottom-up tilnærming
  + Den generelle entitettypen blir superentitetstypen
* Spesialisering
  + Top-down tilnærming, hvor en først finner superentitetstyper og deretter tilhørende spesialiserte subentitetstyper.
  + Vi kan si at spesialisering utdyper forskjellene mellom entiteter
* Super- og subentitetstyper
  + Restriksjoner
    - Disjunkt
      * som betyr at den entitet i superentitettypen ikke kan være medlem av mer enn én av subentitettypene i spesialiseringen
    - Overlappende
      * som betyr at en entitet kan være medlem av mer enn én av subentitetstypene i spesialiseringen
    - Total deltakelse
      * som betyr at hver entitet i superentitettypen å være medlem i subentitettypen
    - Delvis deltakelse
      * som betyr at en entitet i superentitettypen ikke behøver å tilhøre noen av subentitettypene i spesialiseringen

|  |  |  |
| --- | --- | --- |
| Avhengighet | Disjunkt (OR) | Overlappende (AND) |
| Delvis | Deltar i 0 til 1 subenetitetstype | Deltar i 0 til mange subenetitetstyper |
| Total | Deltar i minst 1 subenetitetstype | Deltar i 1 til mange subenetitetstyper |

* + Nøkler i generalisering og spesialisering
    - Superentitet nøkkelen blir både primær nøkkel og fremmed nøkkel i subentitettypene
* Aggregering og komposisjon
  + Aggregering brukes for å vise hvordan enkel entiteten henger sammen i en helhet
  + En komposisjon uttrykker et sterkt eierforhold mellom delene og helheten
    - Dette er for eksempel om noe ikke kan eksistere uten noe annet for eksempel som at en avisannonse ikke eksisterer hvis avisen ikke eksisterer

# Kapittel 8.2 – Transaksjoner