TDT4145 Databasesystemer

Oppsummering

Vegard Aas, 2006

1 Introduksjon til databasesystemer

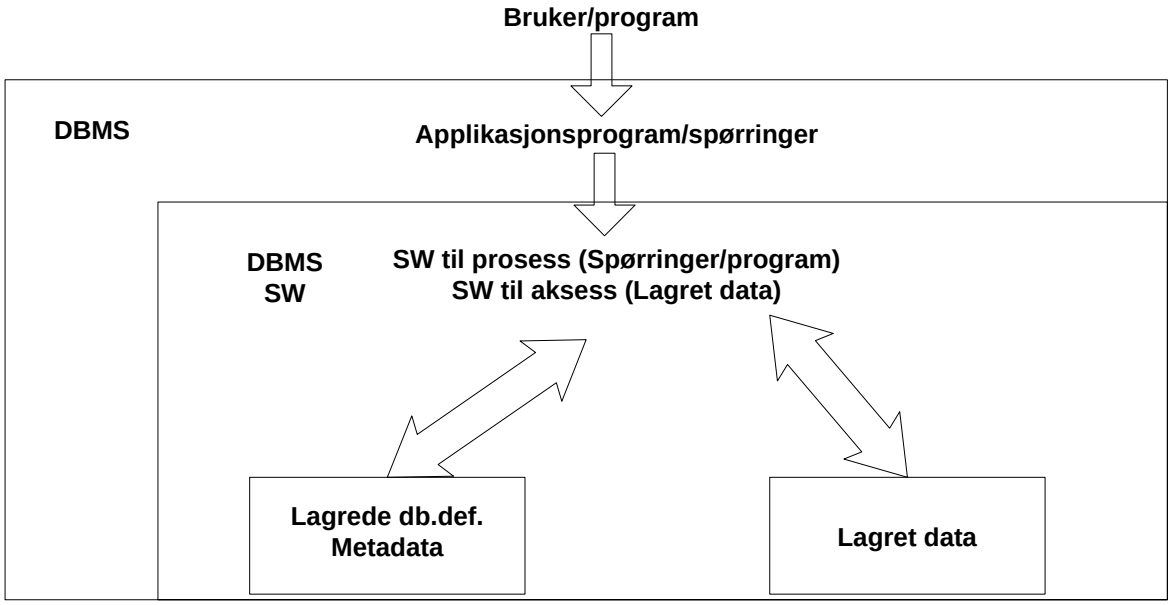
1.1 Databasesystem

Data
En mengde symboler som ikke gir mening i seg selv

Informasjon
Data som er tolket ved hjelp av metadata

Database
En samling av beslektet data som typisk beskriver relasjonene mellom relaterte entiteter
- Representerer aspekt av ved virkelige verden
- Laget for et bestemt formål, for en gruppe brukere

Databasesystem - Database Management System (DBMS)
En teknisk programvareløsning designet for å håndtere og nyttiggjøre store mengder data.
Kjennetegn:
- Kan håndtere *svært store datamengder*
- Tilbyr *effektiv aksess* til store datamengder
- Tilbyr *samtidig aksess* for flere brukere vha *låsing og serialiserbarhet*
- *Sikker og atomisk* (alle eller ingen transaksjoner gjennomføres) aksess til store datamengder
- Sørger for at *informasjon ikke går tapt* vha *recovery*
- Sørger for *datauavhengighet*: applikasjonen ikke eksponert for datarepresentasjon og -lagring)



Databaseutvikling
1. Kravspesifikasjon – beskrivelse av miniverden
2. Lage ER-modell på grunnlag av krav
3. Lage relasjonsskjema på grunnlag av ER
4. Skjemaforbedring vha. normalisering
5. Fysisk databasedesign: indekser, nøkler etc.
6. Applikasjoner og sikkerhet

Filsystem vs DBMS

- Trenger en eller flere lagringsenheter for å lagre stor datamengde
- Trenger programmert metode for å identifisere alle dataelementer
- Trenger spesielle programmer for å besvare alle spørringer fra brukeren
- Må beskytte data fra inkonsistente endringer som gjøres samtidig av ulike brukere
- Må sørge for at data gjenopprettes til konsistent tilstand dersom systemet krasjer mens oppdateringer gjøres
- Må ha sikkerhetspolitikk i forhold til hva ulike brukere skal ha tilgang til

Fordeler med DBMS

Datauavhengighet	Applikasjonsprogrammer ikke eksponert for datalagning
Effektig aksess	Vha *spørrespråk*
Integritet og sikkerhet	
Data administrasjon	Minimalisere redundans og tune lagringen til å være mest mulig effektiv
Samtidig aksess	Brukerne kan tenke på data som aksessert en om gangen
Krasjgjenoppretting	Systemfeil håndteres
Redusert utvikl.tid	

Ulemper ved DBMS

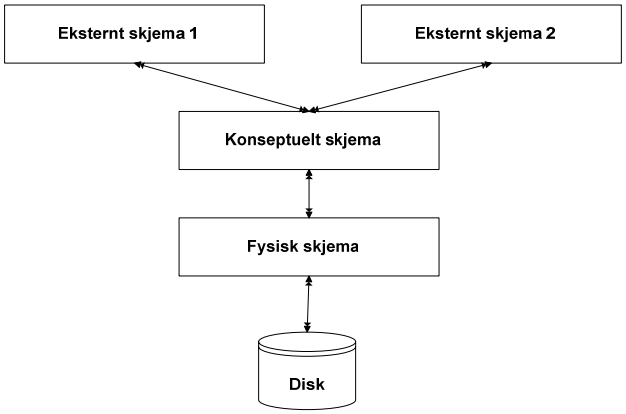
Lav ytelse	Komplekst system – kan gi lav ytelse for bestemte oppgaver
Ikke real.time	Ytelse kan være utilstrekkelig for spesialiserte applikasjoner som real-time
Begrenset funksj.	Behov for annen datamanipulering enn det som støttes av spørrespråket

Beskrivelse og lagring av data i DBMS

Datamodell	En samling høynivå beskrivelser av data som skjuler lagringsdetaljer
Relasjonsmodell	Data beskrives vha. *entiteter og relasjoner* mellom disse Består av en samling *rader/tupler*, og beskrives av *skjema*
Skjema	Spesifikasjon av relasjonsnavn samt navn og type for *felt/attributter/kolonner*
Semantisk modell	Mer abstrakt høynivå datamodell
Hierarkisk modell	IBM IMS DBMS
Nettverksmodell	
Objektorientert modell	

Nivå av abstraksjon

Eksterne skjema	Samling visningsmodi for tilpasset visning gjennom *visninger (views)*.
Konseptuelle skjema	Beskriver lagrede data i henhold til datamodellen til DBMS
Fysisk skjema	Beskrivelse av hvordan data lagres fysisk. *Filorganisering og indeksering* I en relasjonsdatabase beskrives alle relasjoner som er lagret i databasen
Datadefinisjonsspråk	Data definition language – DDL Definerer eksterne og konseptuelle skjema
Systemkatalog	Inneholder informasjon om konseptuelle, eksterne og fysiske skjema



Dataavhengighet

Brukerapplikasjoner skal være isolert fra endringer i datalager og struktur. Dette gjøres gjennom tre abstraksjonsnivå: eksternt, konseptuelt og fysiske skjema. Dette gir *logisk* og *fysisk* datauavhengighet

Spørringer

Spørringer	Spørsmål som stilles DBMS om lagrede data
Spørrespråk	Spesialisert språk spørringene skrives i
Relasjonskalkulus	Formelt spørrespråk basert på matematisk logikk med presise definisjoner
Relasjonsalgebra	Formelt spørrespråk basert på *operatorer* for å manipulere relasjoner
DML	D*ata manipulation language*: Innsetting, endring og spørring mot data

Transaksjonshåndtering

Gjenoppretting	DBMS må sørge for at all data gjenopprettes ved systemkrasj Delvise endringer må da angres
Transaksjon	Enkel eksekvering av brukerprogram i DBMS *Delvise transaksjoner* kan ikke tillates, og effekten av en gruppe transaksjoner må være ekvivalent *seriell eksekvering* av alle transaksjonene. *Tidsplaner* brukes for å tillate *samtidig aksess* vha. *låseprotokoller*
Låseprotokoller	Et sett regler for transaksjonene for å sikre *serialiserbarhet* *Delte låser* kan holdes av to ulike transaksjoner på samme objekt *Eksklusiv lås* holdes av kun en transaksjon om gangen
Ufullstendige trans	For å hindre ufullstendige transaksjoner ved systemkrasj benyttes en *log* Alle skrivehendelser må lagres i loggen før endringene gjøres i databasen *WAL (Write ahead log)*
Gjenopprettingspunkt	Loginformasjon tvinges til disk med jevne mellomrom for raskere gjenopprett

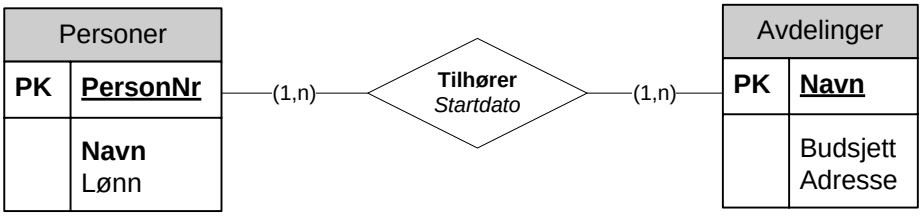
Databasearkitektur

Spørringsoptimerer	*Query optimizer* bruker *informasjon om lagrede* data til å lage en
Eksekveringsplan	Plan for gjennomføring av spørring, representert vha *relasjonsoperatorer*
Aksessmetoder	DBMS lagrer data i *filer* som er *samlinger sider eller poster*
Buffer manager	Henter inn *sider fra disk til hovedlager* som svar på spørringer
Disk space manager	Håndterer selve l*agringen og lesingen fra disk*
Transaction manager	Sørger for at *transaksjoner og låsing* går i henhold til protokoll
Lock manager	Holder informasjon om *forespørsler og tildeling av låser*
Recovery manager	Opprettholder *log* for systemgjenoppretting

2 ER datamodell

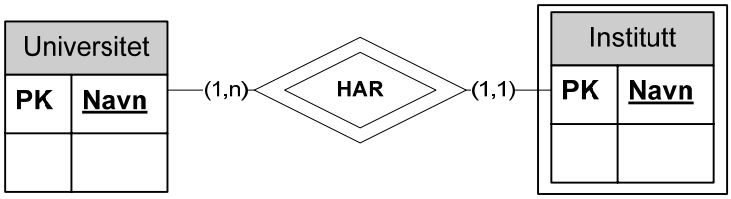
ER-modell

Relasjonsmodellen	Ser på *entiteter* som representerer objekter og *relasjoner* mellom disse
Relasjonsskjema	Beskrivelse av kolonnenavn og datatyper
Relasjonsinstans	En tabell
Entitet	Representasjon av objekt, for eksempel Person
Attributt	Egenskap til entitet, for eksempel Navn, Epost
Enitetsett	Samling av lignende entiteter
Nøkkel	Et sett av attributter som unikt identifiserer en entitet
Relasjon	Assosiasjon mellom to eller flere entiteter, sett av n-tuper: $\{(e_1,\ldots, e_n) \mid e_1 \in E_1,\ldots, e_n \in E_n\}$ Kan inneholder *beskrivende attributter* som AnsattSiden *Graden* er antall felter i relasjonen
Multiplisitet	En-til-en, mange-til-en eller mange-til-mange



Svake entitetssett

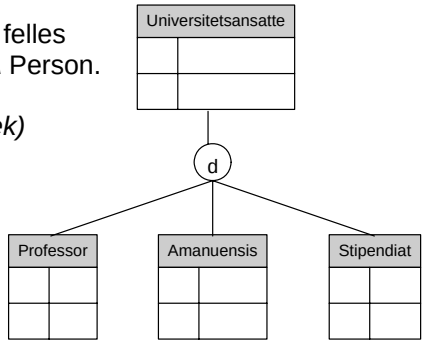
Identifiseres ved hjelp av andre entiteter og må være *(1,1)*. Realiseres ofte vha. egne id-nøker



Subklasser

En entitetsklasse kan være et spesialtilfelle av en annen, hvor de har en del felles attributter men enkelte som er ulike for subklassene. Denoteres Student *ISA* Person.

Delvis subklassing	Entiteter kan være del av superklasse
Total subklassing	Alle entiteter *må være med i subklasser (dobbel strek)*
Disjunkt subklassing	En entitet kan *kun være i en subklasse (d)*
Overlappende	Entiteter kan være med i flere subklasser (o)
Konvertering	Subklasser kan konverteres til relasjoner: 1) Inkluderer alle attributter til rotklassen i subkl. 2) Ser på entitetene som objekter av klasse 3) Bruker NULL-verdier og lager alt i en relasjon



Aggregering

Viser at et relasjonssett (identifisert ved et stiplet rektangel) deltar i et annet relasjonssett

3 Relasjonsalgebra

Settoperatorer

U	*Union*, tar bare med en forekomst av distinkte verdier
$\cap$	*Snitt*
-	*Differanse*
X	*Kryssprodukt/Kartesisk produkt* RxS returnerer alle felt i R, S

Seleksjonsoperator

Velger ut et resultat hvor alle tuplene tilfredsstiller villkåret satt i seleksjonen.

$\sigma_{villkår}(relasjon) = ?$

$\sigma_{lengde<80}(skip)$

Projeksjon

Velger ut et resultat ved å vise kun enkeltkolonner

$R1 = \Pi_{kolonne1,kolonne2}(R_2)$

Lineær notasjon vs operatortre

$R = \pi_{navn}\left(\sigma_{lengde<80}(skip)\right) = \{\text{Navn1, Navn2}\}$

R

π_{navn}

$\sigma_{lengde>80}$

Navn1, Navn2

Sammenslåing (Join)

Slår sammen relasjoner ved å sammenligne tupler og *kombinere informasjon* fra to eller flere relasjoner. Kan også defineres som et kryssprodukt etterfulgt av seleksjoner og projeksjoner.

Betinget sammenslåing $R \triangleright\!\triangleleft_c S = \sigma_c(RxS)$

$R \triangleright\!\triangleleft_{R1.ID<R2.ID} S = \sigma_c(RxS)$

Naturlig sammenslåing $R \triangleright\!\triangleleft S = \sigma(RxS)$ R.A=S.A AND R.B=S.B AND... for alle attributter

Utvidet relasjonsalgebra

δ Eliminerer duplikat fra bag

τ Sorterer tupler

γ Gruppering og aggregering

$\overset{o}{\triangleright\!\triangleleft}$ Ytre join: Har også med tupler som ikke joiner med andre og tilfører eventuelt null-verdier. Left/right

Aggregeringsoperatorer

AVG(R)	Gjennomsnitt
SUM(R)	Sum
COUNT(R)	Teller tupler
MAX(R)	Maksimum
MIN(R)	Minimum

3 Structured Query Language (SQL)

Høynivåspråk for:
- Skjemadefinisjon (opprette tabeller)
- Spørringer (SELECT)
- Databaseoppdateringer (INSERT, UPDATE, DELETE)

SELECT FROM WHERE

SELECT sett med attributter
FROM sett med tabeller
WHERE villkår med tupler

Relasjonsalgebra	SELECT L FROM R WHERE C → $\pi_L(\sigma_c(R))$
Semantikk	1. Kartesisk produkt med alle tupler i tabellene 2. (Bag) σ med villkåret i WHERE-delen 3. (Utvidet/bag) π med attributter i SELECT-delen
Underspørring	SELECT-FROM-WHERE kan brukes inne i en WHERE eller FROM EXISTS (gir boolean), IN (gir boolean), op ALL og op ANY kan brukes i betingelse SELECT tittel FROM bok WHERE utgittår < ALL(SELECT utgittår FROM bok)
Mengdeoperasjoner	Kan bruke UNION, INTERSECT og EXCEPT i spørringene
Sammenslåing	NATURAL JOIN kan brukes som spørring eller for å definere FROM
NULL-verdier	Tupler som ikke har noen verdi får NULL, og ved sammenligning UNKNOWN
Outer join	OUTER JOIN slår sammen relasjoner i tillegg til å inkludere de som faller utenfor, hvor det settes inn NULL

Bag-operasjoner

SQL ser relasjoner som bagger, hvor *et element kan forekomme flere ganger*. Kalles også *multisett*, og er *ikke sortert*. Kan eliminere duplikat ved å bruke SELECT DISTINCT eller SELECT ALL for å tillate alle.

Aggregering

Man kan *sammenfatte* (aggregere) informasjon om verdiene i en kolonne ved å bruke *SUM, AVG*, *MIN, MAX* eller *COUNT*, samt *partisjonere* tuplene før aggregeringen ved å bruke *GROUP BY.*

```
SELECT COUNT(ISBN)
FROM Bok
GROUP BY forfatterPersonID
WHERE forfatterPersonID = 123
```

Sammenligning

=	Case sensitiv
LIKE '_tekst%'	_ ett tegn, % ett eller flere tegn

Modifisering

INSERT INTO	INSERT INTO R(A1, A2..) VALUES(v1, v2) INSERT INTO R v1, v2, vn
DELETE FROM	DELETE FROM R WHERE villkår
UPDATE	UPDATE R SET A1=v1, A2=v2 etc.
Konkatinering	Sammenkjeding: UPDATE Vin SET land = land || '(EU') WHERE land = 'Spania' Gir land = Spania(EU)

Indekser

- Indeks på attributt A gjør det mulig å raskt finne tupler som har en bestemt verdi for A
- CREATE INDEX navn ON relasjon(Attributt)
- Bruk av indekser gjør spørringer, men forsinker oppdateringer
- Opprettes en ny tabell Index(Attributt), og søkes gjennom denne for eksempel ved binærsøk
- SELECT DISTINCT bruker i gjennomsnitt (n/2) tid
- SELECT leter gjennom hele tabellen og bruker n tid
- Indeks vedlikeholdes automatisk ved oppdatering og brukes automatisk ved spørringer

Views – virtuelle tabeller

- *Eksternt skjema* som defineres med utgangspunkt i andre tabeller og views.
- Akesseres *som en baserelasjon*
- CREATE VIEW navn AS SELECT a1, a2, FROM R1, R2 WHERE villkår
- Når views brukes: DBMS tolker spørringa om til en basetabell
- Kan også sette rettigheter og brukes som *sikkerhetsmekanisme*
- Problemer knyttet til *oppdatering*

4 Restriksjoner og triggere

Nøkkelrestriksjoner

En attributt eller et sett av attributter kan deklareres til å være en nøkkel ved UNIQUE eller PRIMARY KEY.

```
CREATE TABLE Forfatter(
                    PersonNr INT(11) PRIMARY KEY,
                    Fornavn CHAR(20),
                    Etternavn CHAR(20),
                    Kjønn CHAR(6)
);
```

Fremmednøkler

Vi kan definere at en verdi som opptrer som et attributt eller sett av attributter også må finnes i de tilsvarende attributtene i en annen relasjon vha. REFERENCES eller FOREIGN KEY.

```
REFERENCES R(A)    Default: ON DELETE NO ACTION
                   Fjerne tuppel med fremmednøkkel: ON DELETE CASCADE

CREATE TABLE Bok(
            ISBN CHAR(9) PRIMARY KEY,
            Tittel CHAR(20)
            ForfatterPersonNr INT(11),
            FOREIGN KEY (ForfatterPersonNr) REFERENCES Forfatter(PersonNr)
                ON DELETE SET NULL
                ON UPDATE CASCADE
);
```

Deferable

Alle restriksjoner kan settes:

```
DEFERABLE          INITIALLY DEFERRED Begrensningen sjekkes etter transaksjonen
                   INITIALLY IMMEDIATE Sjekk før modifikasjon
NOT DEFERABLE
```

Attributtbasert restriksjon

En restriksjon settes på en attributt ved å legge til CHECK

```
Kjønn CHAR(6)
    CONSTRAINT KjønnSjekk CHECK(Kjønn IN ('Mann', 'Kvinne'))
```

Tuppelbasert restriksjon

```
CREATE TABLE Forfatter(
    PersonNr INT(11) PRIMARY KEY,
    Fornavn CHAR(20),
    Etternavn CHAR(20),
    Kjønn CHAR(6),
    CHECK(Kjønn ='Mann' OR Etternavn NOT LIKE 'Lars')
);
```

Assertions

Lager restriksjon utenfor tabell og kan involvere en eller flere tabeller

```
CREATE ASSERTION Forfatter
    CHECK (Kjønn = 'Mann' OR Etternavn NOT LIKE 'Lars');
```

Sammenligning av restriksjoner

Type	Deklarert når	Aktivert når	Garantert å holde?
Attributtbasert CHECK	Med attributt	På innsetting i tabellen eller oppdatering av attributten	Ikke hvis det er subspørringer
Tuppelbasert CHECK	Element for en tabell	På innsetting i tabellen eller oppdatering av attributten	Ikke hvis det er subspørringer
Assertion	Element for en database	På alle forandringer i alle tabeller nevnt i assertion	Ja

Triggers

Hvorfor: Attributt-/tuppelbaserte sjekker ikke garantert å holde, assertions ikke nok effektivt

Hvordan: *Event-Condition-Action-Rules*

- Hendelser: Typisk DB-modifisering
- Vilkår: Generelt SQL-uttrykk med boolsk verdi
- Aksjon: Vilkårlig SQL-setning

```
CREATE TRIGGER Navn
        AFTER INSERT ON R
        REFERENCING NEW AS Navn2
                FOR EACH ROW
                WHEN(villkår)
                        BEGIN
                                INSERT INTO
                                        VALUES
                        END;
```

5 Representasjon av data

Terminologi

Post	Record, tuppel, rad
Blokk	Side, page
Tabell	Relasjon

Ekstern lagring

Filorganisering Måten poster arrangeres på eksternt lager
Enhet som leses er *side*, en DBMS-parameter typisk på 4-8KB
Kostnaden ved side- *I/O* (input fra disk og output fra minne til disk) dominerer
Alle poster har en *record id (rid)* som identifiserer den i en fil

Sekundærlager Magnetiske disker med kapasitet på flere gigabyte. De har flere sirkulære plater av magnetisk materiale med *spor* (eng: tracks) som kan lagre bits. Platene roterer rundt en sentral "spindle". Et spor i en bestemt radius fra sentrum av platen utgjør en *sylinder*.

Blokker Spor (tracks) blir delt inn i sektorer som er separert med umagnetisert materiale (gaps). Sektorer er enhetene for skriving og lesing fra disk. *Blokker er logiske enheter* for lageret brukt av en applikasjon slik som DBMS. Blokker består typisk av flere sektorer.

Diskkontroller: Prosessor som kontrollerer en eller flere diskenheter. Den er ansvarlig for å flytte diskens hode til den rette sylinderen for å lese eller skrive et forespurt spor. Den kan også håndtere konkurrerende forespørsler for diskaksess og bufrer blokkene som skal bli skrevet eller lest. Diskkontrolleren kommuniserer med minnet og prosessoren via en buss.

Diskaksesstid: Tiden mellom en forespørsel om å lese/skrive til dette er utført.

Søketid:	Flytte lesehodet til riktig sylinder
Rotasjonsforsinkelse:	Tid før ønsket sektor er under lesehodet (l/2)
Overføringstid:	Selve I/O

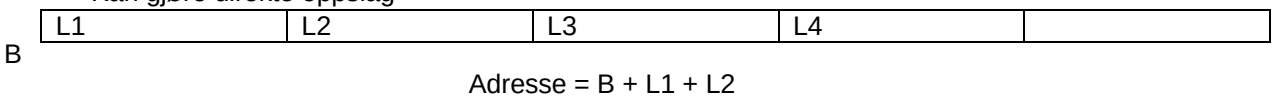
Når vi skal modifisere en blokk må vi 1) lese blokken inn i minnet 2) gjøre endringer i minnet 3) skrive nytt innhold tilbake til disk 4) hvis ønskelig – verifisere korrekt skrevet.
Moores lov: Prosessorhastighet og disk/minnekapasitet fordobles hver 18. Mnd.

Buffer manager Algoritmer brukes for hensiktsmessig innlesing til primærminnet for å spare I/O-tid
Tofase, multivei flettesortering (mest brukt).

Representasjon av dataelementer

Felter (Fields) *Primitive* dataelementer med fast lengde gis et antall byte i sekundærlager.
Felter av varierende lengde lagres som *sekvens av bytes med endemarkør*

Poster Hel tuppel. Inneholder flere felter samt en header med timestamp, lengde og tabinfo
Informasjon om felttyper lagres i systemkatalog
Kan gjøre direkte oppslag



Adresse = B + L1 + L2

Variabel lengde:
1. *Teller og stoppsymbol*
2. *Liste med felter*

Filorganisering

Fillaget lagrer poster i samling av filer fordelt på ulike diskblokker. Holder orden på hvilke sider som er allokert til hver fil samt tilgjengelig plass.

Heap-file — *Uordnet fil, rålager* hvor postene er lagret tilfeldig over filens sider.
Rid spesifiserer hvilke poster som ligger hvor.
Bra på *tabellsum*, dårlig på *direkte aksess* og *range-søk*

Indeks: Datastruktur som organiserer data slik at den raskt kan finne en tuppel ved å søke gjennom en liten mengde. Baseres på typisk attributt – søkenøkkel.
Vha. *trær* eller *hashing*
Data lagres i indeks på følgende måter:

1. En post k* *inneholder faktiske data* med søkenøkkel k
2. Post er et *(k; rid)*-par
3. Poste er *(k; rid-list)*

Clustret indeks: Dataposter er i eller nær samme rekkefølge som indeksene
Filer er vanligvis ikke sortert ettersom dette er svært kostbart å vedlikeholde. Svært bra på *range søk*.

B+-tre indeks: Binærtre hvor *løvnoder inneholder dataposter* og er *lenket*, interne noder inneholder indekser. Antall I/O likt *lengden på sti fra rot til løv, pluss kvalifiserende løvnoder*.
Balansert slik at alle stier fra rot til løvnoder er like lange, høyde vanligvis 3-4
Fan-out: gjennomsnittlig antall barn for ikke-løvnode med n^h . Fan-out på 100 og høyde 4 gir 100 millioner løvnoder, som kan gjennomsøkes på 4 I/O.
Bra på range- og sekvensielt søk, samt direkte aksess, dårlig på oppdatering

Vanlige tall er 67% fyllingsgrad i blokkene, indekspost 12 bytes (4 bytes primærnøkkel + 4 byte blokkid + 4 byte dictionarypeker), 50 bytes overhead i blokken. Skal vi skrive en ny blokk gir dette typisk 1 I/O for lesing (ett nivå i treet), 2 I/O for lesing og 2 I/O for skriving. 10ms for å lese/skrive.

B-link — Har også *linker mellom interne noder*
Kalles også *Lehman-Yaho*

Hash-basert — Poster gruppert i *bøtter* som består av en hovedside og *overflytssider*
Hashfunksjon: h(r) = bøtte hvor r ligger
Gjennomsnittlig *67 % fyllgrad*
Svært bra på direkte oppslag, veldig dårlig på range-søk
Statisk hashing: Antall primære sider konstant, event. overflytsblokker
Sekvensielt/lineær søk innen hver blokk
Utvidbar hash: Directory med pekere til bøtter. Dobler ved å doble directory
Bøtte identifiserers ved *d* siste bit

Kostnadsmodell B: Antall sider/blokker, R antall records/tupler, D: gj. lesetid C prosesseringstid
Tar ikke hensyn til DB-buffer, CPU-tid etc.

Filtype	*Scan*	*Likhetssøk*	*Intervallsøk*	*Innsetting*	*Sletting*	*Svakhet*	*Fordel*
Heap	B(D+RC)	0,5(D+RC)	B(D+RC)	2D+C	Søk + D		Innsetting
Sortert	B(D+RC)	Binærsøk: $D\log_2 B + C\log_2 R$	$D(\log_2 B + \#\text{ pgs})$	Søk + BD	Søk + BD		Range på statiske data
Clustered	1.5B(D+RC)	$D\ \log_F 1.5B + C\ \log_2 R$	$D\ \log_F 1.5B + C\ \log_2 R$	Søk + D	Søk + D	Plassutnyttelse	Alt annet
Unclustered tree index	BD(R+0.15)	$D(10\log_F 0{,}15B)$	$D(10\log_F 0{,}15B + \#\text{treff})$	$D(3+\log_F 0.15B)$	Søk + 2D		

Unclustered hash index	BD(R+0.125)	2D	BD	4D	Søk + 2D	Likhets-søk

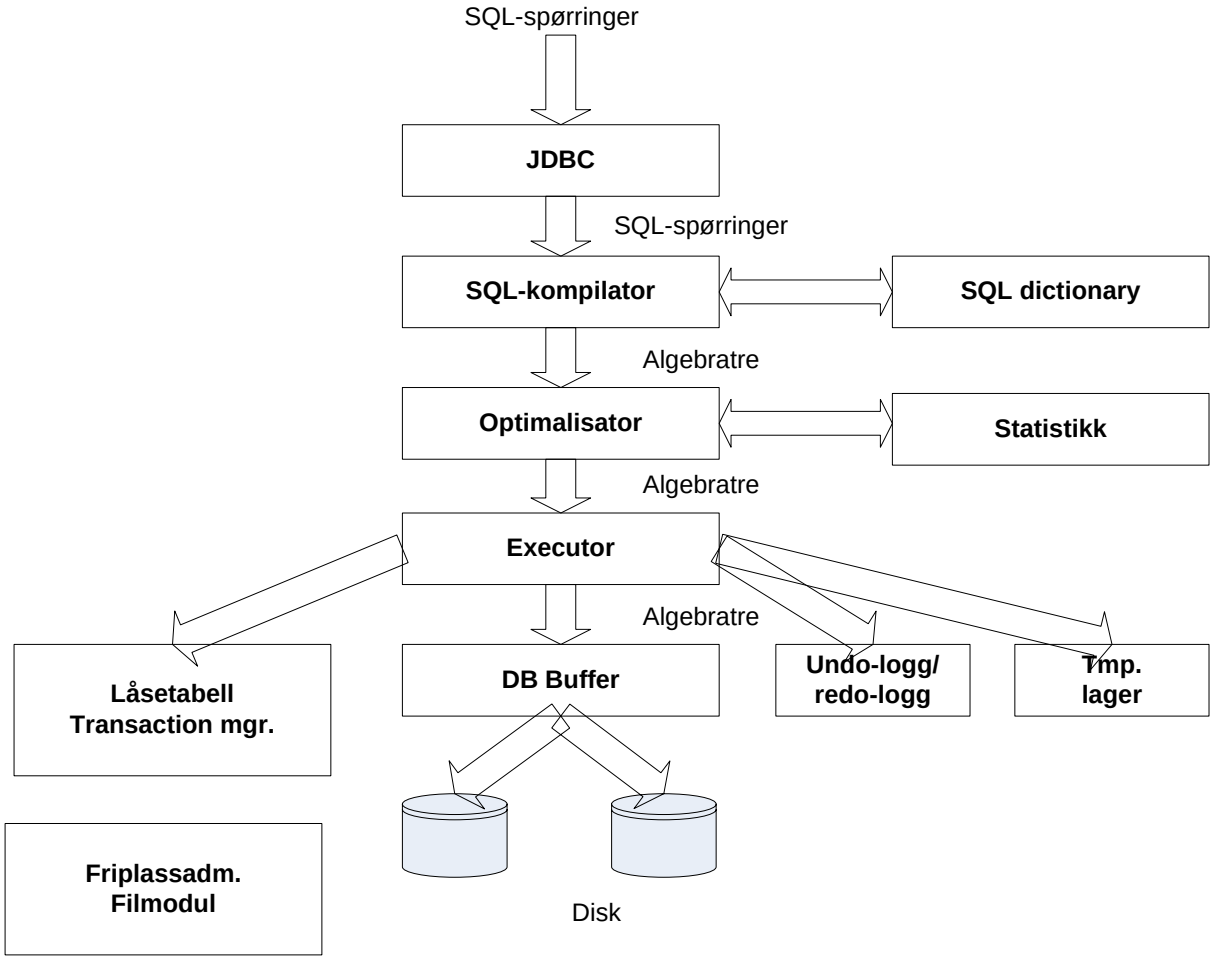
Vurdering:

Vanlige operasjoner
Hyppighet av operasjoner
Andel spørringer vs endringer
Dataegenskaper

6 Databasearkitektur

Egenskaper ved databasesystem (DBMS)

1. Håndterer store datamengder
   - Indekser
2. Støtter effektiv aksess til store datamengder
3. Støtter samtidig deling av data
   - Låsing → Transaksjoner
   - Ulike rettigheter
   - Constraints, triggers, stored procedures
4. Sikrer atomisk aksess
   - Fullstendige transaksjoner
5. Persistens



Buffer Manager

Buffer Manager styrer tilgang til ledige minneblokker og svarer på forespørsler om hovedlageraksess.

Arkitektur	1) Buffer Manager styrer hovedminnet direkte	
	2) Buffer Manager allokerer buffer i virtuelt minne som OS styrer	
Strategier	Avhengig av I/O aksessmønster:	
	1) *Last Recently Used (LRU)*	Kaster den det er lengst siden ble lest/skrevet
	2) *First In First Out (FIFO)*	Den blokken som har vært lengst skal ut
	3) *Klokking*	Blokker settes med 0/1 og for hver Klokkesykel letes det etter en blokk med 0, hvor alle foregående settes fra 1 til 0.
	Pinned	Blokker som ikke kan kastes direkte

Friplassadministrasjon

Benytter *egen partisjon* til å *mappe ledige blokker* på databasediskene

Filmodul

(fillId, RotBlokkId)
Eget filsystem på topp av OS

Systemkatalog

(tabell, fillId)
(indeks, fillId)
Inneholder *informasjon om hva som finnes på disk*, og ligger i minnet samt på disk
For hver indeks inneholder den *struktur*
For hver relasjon inneholder den *navn, filnavn, filstruktur, type, posisjon*

7 Query-evaluering

Plan
- *Algebratre* for utføring, *algoritme for hvert nivå*
- Hver operator implementert som *pull interface*
- *Pipelining*
- Evaluerer *ulike planer* for hver spørring
- I praksis: *unngår verste planer*

Indeksering — Brukes ved seleksjon eller join for å finne tupler som tilfredsstiller betingelse

Iterering — Skanner alle tupler

Partisjonering — Inndeling av tupler (aggregering) ut fra søkenøkkel
Sortering og hashing

Systemkatalog — *Tabell:* tabellnavn, filidentifikator, filstruktur, attributtnavn- og typer, indeksnavn, restr.
Indeks: navn og struktur, nøkkelattributter
Generelt:
- Kardinalitet: antall tupler i tabell
- Størrelse: Antall sider for hver tabell
- Indekskard. Antall distinkte nøkkelverdier for hver indeks
- Indeksstørrelse: Antall sider for hver indeks
- Indekshøyde: Antall ikke-løvnoder for treindeks
- Indeksbredde: Min. og maks. nøkkelverdi

Oppdateres periodisk og histogrammer lagres

Aksessvei — Metode for *uthenting av tupler*
Komplekse seleksjoner skrives om til konjunktiv normalform
Finner mest selektive aksessvei og skanner så disse indeksene
Kostnad avhenger av # kvalifiserte tupler, samt clustering

CNF — *Konjunktiv normalform*: Enkel seleksjon hvor alle betingelser er på formen *attr OP val*
Hash-indeks mathcer på *attributt = verdi*
Tre-indeks matcher på *attributt OP verdi*, hvor OP er enhver sammenligningsop.

Pipelining — Operator sendes direkte (on-the-fly) til neste *uten å mellomlagre* i en midlertidig tabell, noe som kalles *materialisering*

Join

Simple nested loop — Naiv algoritme som for hver tuppel i tabell 1 prober alle tupler i tabell 2
Kompleksitet n^2

Index nested loop

Kompleksitet *n*
Antar hashbasert indeks på felles post

Eks. Leser tabell + for hver post * oppslag i indeks + fil
1000 + 100 0000*(1.2+1) = 221 000 I/O

Sort-merge join — Kompleksitet *nlogm*
Kan sortere en stor fil i 2,3 eller 4 pass avhengig av primærlager
Antall I/O = antall pass * 2 (les/skriv)

Eks. Sorter tabell 1 + sorter tabell 2
2 * 2 * 1000 I/O + 2 * 2* 500 I/0
+ Merge: 1000 + 500

Til sammen 7 500 I/O

Push-down selection Seleksjon gjøres før join for å gjøre denne på færre tupler

Hash join

8 Transaksjoner

Samtidighetskontroll (Concurrency Control)

Støtter deling og samtidig aksess av data gjennom *transaksjoner* og *låsing*
Samtidighetsdiagram

ACID-egenskaper

Atomic	En transaksjon kjøres enten *fullstendig eller over hodet ikke* (roll-back)
Consistent	Transaksjoner overholder *konsistenskrav*: Primary key, references
Isolation	Samtidige spørringer skjer uavhengig av hverandre
Durability	Endringer er permanente etter commit (systemgjenoppretting)

COMMIT/ABORT

1) En transaksjon slutter med COMMIT og oppdatering har vært suksessfull
2) En transaksjon slutter med ROLLBACK
AUTOCOMMIT Hver SQL-setning er en egen transaksjon

Historie

Liste (sekvens) av aksjoner fra en mengde transaksjoner:
Read (A)
Write (A)
ABORT
COMMIT

Seriell historie	Aksjoner fra forskjellige transaksjoner flettes ikke
Serialiserbar	Samme effekt på databasen som en seriell historie
Konfliktserial	To historier er *konfliktekvivalente* hvis de kan bli like ved swapping av aksjoner uten konflikt. En historie er *konflitserialiserbar* om den er ekv. en seriell historie
Read uncom	Read uncommitted: W-R-konflikt med *dirty read* T_1: R(A), W(A), R(B), ABORT T_2: R(A), W(A)
Unrep reads	Unrepeatable reads: *R-W-konflikt* T_1: R(A), R(A), W(A) T2: R(A), W(A)

Isolasjonsnivåer

Mer samtidighet ↑ READ UNCOMMITTED
READ COMMITTED
REPEATABLE READ
SERIALIZABLE (default)
↓ Mer isolasjon – ”Korrekthet”

Samtidighet 1. Mye diskaksess, bruker CPU til andre transaksjoner så lenge
2. Parallelle maskiner

	Dirty read	*Unrepeatable read*	*Fantom*
Read uncommitted	x	x	X
Read comitted	-	x	x
Repeatable read	-	-	x
Serializable	-	-	-

Strict locking

Strict 2PL – tofaselåsing

1. Hvis en T ønsker å lese/skrive et objekt så må det settes en *delt/eksklusiv* lås på objektet først
2. Alle låser holdt av en T slippes samtidig når denne er ferdig
Tillater kun serialiserbare historier

Implementasjon	- Låsetabell i primærlager - Låsetyper: postlåser, blokklåser, *tabellåser, verdiområdelåser, predikatlåser* *Unngå fantomer*
Vranglås	Flere transaksjoner venter gjensidig på hverandre. Løsninger: *Wait-for-graphs*: finn sykler og aborter noen transaksjoner *Aborter hvis time-out* *Ordne elementer*: låsing av rekkefølgen av elementer *Tidsstempel* på alle låser

9 Håndtering av systemfeil

ARIES

Gjenopprettingsalgoritme som virker med steal, no-force

1. Analyse
Identifiserer *dirty pages* i buffer pool og *aktive transaksjoner* i transaksjonstabell

2. Redo
Repeterer alle aksjoner og gjenoppretter databasen til tilstand før krasj

3. Undo
Angrer alle aksjoner til transaksjoner som *ikke comittet*

Write ahead logging
Endringer i et objekt *lagres først i log*, og log må skrives til stabilt lager før endringen i databasen skrives til disk.

Repeterer historie ved redo
Ved gjenoppretting *repeteres alle aksjoner som var i gang* før krasj slik at systemet kommer tilbake i eksakt samme tilstand.

Logg endringer ved undo
Endringer som gjøres i undo logges i tilfelle nytt systemkrasj

Loggen

Historie med alle aksjoner utført av DBMS, filstruktur på stabilt lager.
Nyeste del (log tail) lagret i minnet og tvinges periodisk til stabilt lager
Hver loggpost kan hentes på *en I/O* ut fra *log sequence number (LSN)*
Loggposter inneholder *prevLSN, transID, type og LSN*
Oppdateringsposter inneholder i tillegg *pageID, length, offset, before-image og after-image*
Alle sider i databasen inneholder *pageLSN som er den siste logposten til siden*

1. Analyse

1.1 Finner siste *begin_checkpoint*
1.2 Finner hvilke *sider i buffer pool som er dirty* (alle sider som er endret siden begin_checkpoint)
1.3 Identifiser *aktive transaksjoner* som må fjernes

Dirty pages og transaction tabeller settes til status ved sjekkpunktet
Hvis end log record for T finnes, fjernes T fra transaksjonstabellen, ettersom denne ikke lenger er aktiv
Hvis andre log records finnes legges T til transaksjonstabellen

2. Redo

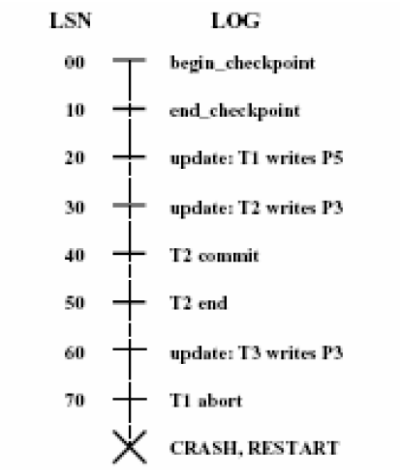
Gjennomføres *oppdateringer av alle transaksjoner*, enten de er committed eller ikke. Skanner loggen fra *siden med minst recLSN i dirty pages-tabellen*, redo hvis ikke:
- Påvirket side er ikke i dirty pages-tabellen
- recLSN > LSN for record som sjekkes
- PageLSN >= LSN til log record

Hvis redo:
- Action gjennomføres
- PageLSN på siden settes til LSN for den gjenopprettede log-posten

3. Undo

Går gjennom transaksjonstabellen fra settet med tapertransaksjoner og velger den med *størst lastLSN*

Eksempel



Analyse

transID	*lastLSN*	*Status*	*pageID*	*recLSN*
T1	70	U	P3	30
T2	50	C	P5	20
T3	60	U		

Redo

Starter på siden med minst recLSN i dirty pages: P5 på recLSN 20
LSN 20 P5 oppdateres
LSN 30 P3 hentes og pageLSN sjekkes. Hvis siden ble skrevet til disk før krasj (pageLSN>=30) endres ingenting, ellers blir endringer redone.
LSN 60 P3 oppdateres

Undo

Starter på LSN 70 (størst i transaksjonstabellen)
LSN 70 Legger til LSN20
LSN 60 Undo P3 legg til CLR
LSN 20 Undo P5 legg til CLR

10 Normalisering

Ønsker å *redusere redundans* ved å omforme til standardiserte skjema

Problemer
1. *Redundans*: repetisjon av samme info
2. *Oppdatering:* må oppdatere flere
3. *Sletting*
4. Innsetting

Redundans
- Redundant lagring: informasjon lagres unødvendig flere ganger
- Anormaliteter: -oppdatering
  - innsetting
  - sletting

Funksjonelle avhengigheter

$A_n \rightarrow B$
$A_1, A_2, \ldots, A_n$ funksjonelt bestemmer B
To tupler med like verdier for A_n vil ha samme B.
Eks. Person(pnr, navn, postnr, poststed)
Pnr → navn, postnr, poststed
Postnr → poststed

Normalisering Skjema som sikrer at enkelte problemer ikke oppstår
Person(pnr, navn, postnr)
Poststed(postnr, poststed)

Nøkler $\{A_1, A_2, \ldots, A_n\}$ er en nøkkel for R hvis
1. Attributtene $A_1, \ldots, A_n$ *funksjonelt bestemmer alle andre attributter* i R
2. Ingen delmengde av $\{A_1, .., A_n\}$ bestemmer funksjonelt: *minimal nøkkel*

Supernøkkel	Et sett attributter som funksjonelt avgjør alle attributtene i relasjonen. Dvs. en nøkkel som IKKE er minimal
Kandidatnøkkel	Alle nøklene i en relasjon
Primærnøkkel	En brukerbestemt nøkkel
Nøkkelattributt	Attributt som er med i minst en av kandidatnøklene

Delvis FA Restriksjon på hvilke tupler som kan finnes samtidig i en relasjon
Full funksj. avh.: x → y er full FA hvis vi ikke kan fjerne noe i x og ha (x-{A})→y
Eks. snr, pnr → navn (delvis FA)
Snr → navn (full FA)

Triviell FA Attributt er en *delmengde av attributtene som gir FA*
R(a,b,c), ab → b

Tillukning $F^* = \{x \rightarrow y \mid F \exists x \rightarrow y\}$ Alle funksjonelle avh. som kan utledes fra F

Refleksiv	Hvis $x \supseteq y$ så $x \rightarrow y$
Augmentering	Hvis $x \rightarrow y$ så $xz \rightarrow yz$
Transitivitet	Hvis $x \rightarrow y \wedge y \rightarrow z$ så $x \rightarrow z$
Triviell FA	$x \rightarrow y$ der $y \subseteq x$
Union	Hvis x → y og x → z så x → yz Hvis x → yz så x → y og x → z
Supernøkler	*Tillukning av supernøkler er alle attributter i relasjonen*

1NF 1. Normalform
1) Attributtenes domene må inneholde bare *atomiske verdier*, ikke mengder eller lister
2) Verdien til et attributt må være enkeltverdier

2NF 2. Normalform

Alle ikke-nøkkel-attributter fullt FA av nøkkel – delvise FA ikke tillatt

3NF — 3. Normalform

1) A e Xeller
2) X supernøkkel eller
3) A del av nøkkel for R

For alle ikke-triville avhengigheter $A_1, A_2, \ldots A_n \rightarrow B$
1) $A_1, .., A_n$ er en supernøkkel for R eller
2) B er et nøkkelattributt, dvs. del av en nøkkel
Det vil alltid finnes en dekomp. til 3NF som er FA-bevarende, men kan ha redundans

BCNF — Boyce-Codd normalform

For alle X → A
A e X eller
X supernøkke
Forskjell mellom 3NF og BCNF: *overlappende kandidatnøkler*
Eksamen (pnr, snr, fag, karakter) 3NF
Student(pnr, snr), Eksamen(pnr, fag, karakter) BCNF

Eks. FA ikke bevart:
R(SBD), SB → D, D → B

4NF — En relasjon R er på 4NF hvis

X er en supernøkkel for R
Y e X
XY = R

Tapsløs join — *Ikke adderende/overlappende join*

En dekomponering av $R = \{R_1, R_2, \ldots, R_n\}$ er tapsløs hvis det for alle R_i som tilf. F er slik at $\pi_{R_1}(R) \bowtie \cdots \bowtie \pi_{R_m}(R) = R$

Teorem: D*ekomponering gir tapsløs join dersom de felles attributtene er en supernøkkel for en eller flere av projeksjonene*

FVA, MVD — Flerverdiavhengigheter

To sett attributter i en relasjon har sett av verdier som forekommer i alle komb.
R(x,y,z), FVA X ->>Y
Om vi har t1(1,2,3) og t2(1,3,5) må vi også ha (1,2,5) og (1,3,3) for å ha FVD

Normalformer

	1NF	**2NF**	**3NF**	**BCNF**	**4NF**
Atomiske verdier	x	x	x	x	x
Alle ikke-nøkkel-attributter fullt FA av nøkkel		x	x	x	X
Alle ikke trivielle FAer har X som enten supernøkkel i relasjonen eller Y er nøkkelattributt i relasjonen			x		
Alle ikke-trivielle FAer har X som supernøkkel i relasjonen				x	
Alle ikke-trivielle MVDer (X→ >Y) har X som supernøkkel i relasjonen					x

Eksempel på hvordan finne normalform

Relasjoner: R(a, b, c, d)
FAer for R:
a → b
b → a
c → d

S(a, b, c)
a → b
b → a

Kandidatnøkl.	ac, bc	ac, bc
1NF	Alle verdier i tuplene er atomiske OK	Alle verdier i tuplene er atomiske, OK
2NF	d ikke-nøkkelattributt, delvis FA ac, NEI	Ingen ikke-nøkkelattrib., OK
3NF	ikke 2NF, NEI	Sjekker alle ikke-trivielle FA: a → b b er nøkkelattrib, b → a er nøkkelattrib., OK
BCNF	ikke 3NF, NEI	Alle ikke-trivielle FA må ha X som supernøkkel, NEI

Finne normalform med FVA

Relasjoner	T(a,b,c)	U(a,b)
MVA	a ->> b a ->> c	a ->> b
Kandidatnøk.	abc	a
Supernøkl.	abc	a, ab
4NF	a->>b, a ikke supernøkkel, NEI	a ->>b, a supernøkkel OK

11 Informasjonsgjenfinning (IR)

IR vs DBMS

	Informasjonsgjenfinning	**DBMS**
Søk	Nøkkelord	SQL
Datastruktur	Ustrukturert	Strukturert
Oppdatering	Hovedsakelig lesing	20 % oppdatering
Resultat	Top k-resultat (rangering)	Fullt resultat

Tekstsøking

IR datamodell
1 Dokument *bag med ord* {Dette er et dokument om databaser}
2 *Fjerner stopp-ord*: ”er”, ”på” etc {dette, databaser, dokument}
3 *Stemming*: lagrer kun grunnstamme: {dette, database, dokument}

Boolsk tekstsøk: ”Dokument” AND ”Database”

Rangering: Resultatsett blir rangert etter treffkvalitet

Vektorrom-modellen (VSM)

- Teller opp forekomster av ord

Termfrekvens: Frekvens av term *k* i dokument *i*: tf_{ik}

TF-IDF
Invers dokumentfrekvens: ord som er med i mange dokumenter blir lite nyttige
Term *k* som er med i n_k dokumenter av totalt *N* dokumenter: $\log(N/n_k)$
Vekt av term k i dokument *i*: $w_{ik}'=tf_{ik}*\log(N/n_k)$

Lengdenormalisering

$$w_{ik} = \frac{tf_{ik}\log(N/n_k)}{\sum_{k=1}^{t}(tf_{ik})^2\left[\log(N/n_k)\right]^2}$$

Dokumentlikhet og søk

Likhet

$$sim(Q,D_i) = \sum_{k=1}^{m} w_{qk} * w_{dk}$$

$$w_k = tf_k * \log(N/n_k)$$

Søkekvalitet
Recall: Andel av relevante dokumenter som er funnet: |Ra| / |R|
Precision: Andel av dokumenter funnet som er relevant |Ra|/|A|

11 Semistrukturerte data (XML)

Skjemaløst, selvbeskrivende
Datamodell: Representert som *annotert graf*
Path expression: Stiuttrykk (/paper/author/firstname → Sett med element)

XML

XML beskriver innholdet

XQUERY — Standard spørrespråk for XML-data

```
for
        $L in doc("bib.xml")//author/lastname
return <RESULT>{$L} </RESULT>
```

FLWOR — For Let Where Order Result

Seleksjon

```
for
        $p in doc("bib.xml")/bib/paper/
        where $p/year="2006"
return
        <RESULT>    <FIRSTNAME>Kjetil</FIRSTNAME>
                    <LASTNAME>Norvag</LASTNAME>
        </RESULT>
```

Sortering i XQ — Default er samme rekkefølge som i dokumentet

```
for
        $p in doc("bib.xml")
        order by lastname
return
        <RESULT>{$p/lastname}</RESULT>
```

Impl i rel. Db — Forskjellig fra system til system
I Oracle 9: XML-dokument som attributter eller mappet til tabell
Spørringer: SQL med XPath-utvidelser