UNIVERSITETET
I OSLO

Parallelle og distribuerte databaser – del III

NoSQL og alternative datamodeller

NoSQL

NoSQL er et paraplybegrep som omfatter en rekke ulike typer teknikker og databaser.

Typiske egenskaper ved NoSQL-databaser:

- Datamodell avviker fra relasjonsmodellen
- Fleksibelt skjema
  - enklere å videreutvikle databasen
- Designet for å håndtere enorme mengder data
  - store maskinclustere
- Open-source

Hvorfor alternative datamodeller

Tre ulike grunner til at alternative datamodeller kan være aktuelt:

- **Big data:** Behov for støtte til håndtering av virkelig store datavolumer. Utnyttelse av maskinclustere
  - Relasjonsdatabaser er ikke designet for dette formålet
  - Ønsker datastrukturer som er lette å håndtere for databasesystemer på clustere
  - CAP-teoremet; må avveie konsistens mot responstid
- **Informasjonsutveksling:** Behov for større fleksibilitet i datastrukturene ved utveksling av informasjon mellom systemer
  - Rikere datastrukturer
  - Behov for mindre rigid skjema
- **Grafer:** Situasjoner der relasjonsmodellen er dårlig egnet til å representere virksomhetsområdet
  - Data som er naturlig å beskrive som grafer, og særlig hvis informasjon knyttet til relasjoner mellom entitetene er vel så viktig som informasjon knyttet til entitetene
  - Komplekse datastrukturer der kompleksiteten i det vesentlige ligger i et stort antall relasjoner mellom entitetene; relasjonsmodellen gir utstrakt bruk av dyre joiner

Datamodeller under NoSQL

- **Aggregatorienterte modeller**: Et **aggregat** er i denne forbindelsen en samling av data som utgjør en naturlig enhet, og hvor spørringer og transaksjoner i all hovedsak er begrenset til ett og ett aggregat.
  - Hvert aggregat lagres typisk som en enhet. Databasesystemet benytter f.eks. hashing for å bestemme hvor aggregatene skal lagres
  - Databasesystemet tilbyr typisk bare ACID-egenskaper (serialiserbarhet) pr. aggregat; ACID-egenskaper på tvers av aggregater støttes ikke i samme grad
  - Transaksjoner eller søk som ikke passer med den valgte aggregatstrukturen, støttes i mindre grad av systemet
- **Grafmodeller**: Data er strukturert i form av entiteter (noder) og kanter som reflekterer hvordan entitetene forholder seg til hverandre.

Aggregatorienterte modeller

- **Key-value databases** (**key-value stores**): Hvert data-element er på formen (k,v) der k er en nøkkel og v en verdi.
  Søk: Oppgi k, finn tilhørende v.
- **Document-oriented databases**: Hvert «dokument» består av en nestet struktur som inneholder navn-verdi-par. Til hvert dokument hører en nøkkel.
  Søk: Finn verdien til et gitt navn/felt ved å angi en nøkkel og hvor i strukturen navnet befinner seg. Eller mer generelt, finn alle dokumenter som har et gitt mønster.
- **Wide-column databases** (**column family databases**): Data er strukturert i form av tidsstemplede navn-verdi-par. Samhørende verdier knyttes sammen i rader med en nøkkel («radnøkkel»).
  Søk: Angi radnøkkel eller et intervall av nøkkelverdier.

Key-value-databaser

- Datastruktur: Par på formen (k,v) der k er en nøkkel og v en verdi.
  - Hver verdi utgjør et aggregat og lagres som en enhet
- Søk: Oppgi k, finn tilhørende v.
  - Applikasjonene må selv håndtere (vite om og utnytte) eventuell aggregatstruktur
  - I utgangspunktet bare én måte å søke på (en og en nøkkelverdi)
  - *Ordered key-value databases* tillater søk på intervaller av nøkkelverdier
- Skjemaløs
  - Hver enkelt verdi kan ha en hvilken som helst kompleks struktur
  - Databasesystemet har ingen innsikt i (ingen skjemainformasjon om) disse strukturene

Berkeley DB

- Embedded. Programvaren distribueres i form av et lite javabibliotek
- Transaksjonsstøtte: Tofaselåsing
  - Enbruker/flerbrukermodus og recoverymodus mm. kan velges når databasen åpnes
  - Applikasjonene kan, ved å håndtere låser selv, definere transaksjoner som benytter eksterne ressurser sammen med databaseressursene
- Indekser:
  - Primærindekser på “key” i key-value-parene
  - Kan definere sekundærindekser på felter i “value”
- Aksessmetoder:
  - B^+-trær hvor bruker kan definere sorteringsfunksjonen selv
  - Extended linear hashing hvor bruker kan definere hashfunksjonen selv
  - Recno: Record-aksessmetode; kan bl.a. brukes til å aksessere linjene i en tekstfil som om de er records
- Eksempler på bruk: Oracle NoSQL, Sendmail, Subversion
- Open-source under GNU AGPL

Datamodell

- **Tabeller** og **records** defineres ved hjelp av javaklasser
  - En tabell pr. klasse. Klassen merkes med `@Entity`
  - Objektene ("value" i key-value) lagres som records i tabellen
  - Et attributt i klassen ("key" i key-value) benyttes som primærnøkkel. Attributtet merkes med `@PrimaryKey`
  - Kan definere sekundærindekser på andre attributter (`@SecondaryKey`)
- Kan gjenfinne en record ved å angi tilhørende nøkkelverdi
- Kan gjennomløpe alle recordene i en tabell med en cursor

Definisjon av tabell/records

```
@Entity
public class Vare {
    @PrimaryKey
    private String vareId;

    @SecondaryKey(relate=MANY_TO_ONE)
    private String vareNavn;

    private float pris;

    public void setVareId(String data) {vareId = data;}
    public void setVareNavn(String data) {vareNavn = data;}
    public void setPris(float data) {pris = data;}

    public String getVareId() {return vareId;}
    public String getVareNavn() {return vareNavn;}
    public float getPris() {return pris;}
}
```

Databaseoperasjoner

1. Åpne databasen med lese- og skriveaksess:

```
static File minEnvPath = new File("...");
EnvironmentConfig minEnvConfig = new EnvironmentConfig();
StoreConfig storeConfig = new StoreConfig();
minEnvConfig.setReadOnly(false);
storeConfig.setReadOnly(false);
Environment minEnv = new Environment(minEnvPath, minEnvConfig);
EntityStore store = new EntityStore(minEnv, "VareLager", storeConfig);
```

2. Opprett aksessorer:

```
PrimaryIndex<String,Vare> vareIdIndeks =
    store.getPrimaryIndex(String.class, Vare.class);
SecondaryIndex<String,String,Vare> VareNavnIndeks =
    store.getSecondaryIndex(vareIdIndeks, String.class, "vareNavn");
```

3. Legg inn records:

```
Vare v = new Vare();
v.setVareId(...); v.setVareNavn(...); v.setPris(...);
vareIdIndeks.put(v); ...
```

4. Finn en record basert på primærnøkkelen:

```
Vare u = vareIdIndeks.get("M5X22"); ... bruk u ...
```

5. Finn records basert på sekundærnøkkelen (ved hjelp av en cursor):

```
EntityCursor<Vare> linksSkruer = vareNavnIndeks.subIndex("Linksskrue").entities();
for (Vare w : linksSkruer) {... bruk w ...}
linksSkruer.close();
```

6. Lukk databasen:

```
store.close();
minEnv.close();
```

Dokumentorienterte databaser I

- Datastruktur:
  - Par på formen (k,v) der k er en nøkkel og v er et **dokument**, dvs. en nestet (vanligvis treliknende) struktur av navn-verdi-par
  - Hvert dokument utgjør et aggregat og lagres som en enhet
  - Terminologien stammer fra semistrukturerte datamodeller, jf. «XML-dokument». XML og JSON er formalismer egnet til å beskrive nestede strukturer
- Søk:
  - Kan søke på nøkkelverdier og/eller felter i dokumentene
  - Kan hente ut deler av et dokument
  - Databasesystemet kan bygge indekser basert på felter i dokumentene

Dokumentorienterte databaser II

- Skjemaløs
  - Hvert enkelt dokument kan ha en vilkårlig nestet struktur (innenfor den klassen av strukturer som databasesystemet tillater) uten at denne må deklareres i forkant
  - Strukturene kan endres dynamisk
  - Databasesystemet kan traversere strukturene basert på mønstere angitt i et søk
  - Databasesystemet har i utgangspunktet ikke informasjon om dokumentenes struktur, så applikasjonene må selv kjenne dokumentstrukturen for å kunne formulere søk på felter i dokumentene

MongoDB

- Skjemaløs - må ikke deklarere noen dokumentstruktur i forkant
- Transaksjonsstøtte:
  - ACID-egenskaper pr. dokument
  - Granularitetslåser: Tar IS eller IX på database- og kolleksjons-nivå (*kolleksjoner*: se neste side) og S eller X på dokumentnivå
- Indekser:
  - Alle dokumenter har et _id-felt som det bygges primærindeks på
  - Kan definere sekundærindekser på kombinasjoner av felter i dokumentene
- Skalerbarhet: Sharding, replikering, lastbalansering
- Eksempler på systemer som bruker MongoDB: eBay, LinkedIn, SAP
- Open-source under GNU AGPL

Datamodell

- Hvert **dokument** tilhører en **kolleksjon** (**collection**)
  - En kolleksjon svarer til en tabell i en relasjonsdatabase
  - **Capped collection**: Sirkulær FIFO-kø av dokumenter
  - Dokumentstruktur: BSON (binær JSON)
  - Dokumenter kan nestes (**embedded documents**) og/eller lenkes sammen ved hjelp av pekere (**document references**)
- Spørringer:
  - Angi en kolleksjon og et mønster; resultatmengden er de dokumentene i kolleksjonen som matcher mønsteret
  - Kan returnere utvalgte felter i resultatdokumentene
  - Aggregering: Via MapReduce-funksjonalitet

BSON-dokument

```
{
  tittel: "Simula BEGIN",
  forfatter: ["Graham M. Birtwistle", "Ole-Johan Dahl",
              "Bjørn Myrhaug", "Kristen Nygaard"],
  arstall: 1973,
  plassering: [
                { bibliotek: "UiO Informatikkbiblioteket",
                  samling: "UREAL/INF Boksamling",
                  sted: "D.3.2 Simula/Bir"
                },
                { bibliotek: "UiO HumSam-biblioteket",
                  sted: "ISO E53/6316"
                },
                { bibliotek: "UiO Realfagsbiblioteket",
                  sted: "Kj:4522"
                }
              ]
}
```

Opprettelse av en database

```
use minDB
db.boksamling.insert( {_id: ObjectId("7147819833000201"),
                       tittel: "Simula BEGIN",
                       forfatter: [...] ,
                       arstall: 1973,
                       plassering: [ { bibliotek: ...}, ...]
                      }
                    )
db.boksamling.createIndex( { "plassering.bibliotek": 1 } )
```

- Hvis databasen `minDB` og/eller kolleksjonen `boksamling` ikke finnes allerede, blir de opprettet av `insert`-operasjonen
- Hvis et dokument mangler _id-feltet, blir feltet opprettet automatisk og gitt en entydig verdi. Det bygges automatisk primærindekser for hver kolleksjon på _id-feltet
- `createIndex` definerer en sekundærindeks på feltene `bibliotek` i arrayen `plassering`

Spørringer

- Finn bøker på informatikkbiblioteket fra før 1975. Skriv for hver ut tittel, forfatter og årstall, sortert på årstall:

```
db.boksamling.find(
  {
    arstall: { $lt: 1975 },
    "plassering.bibliotek": "UiO Informatikkbiblioteket"
  },
  { tittel: 1, forfatter: 1, årstall: 1 }
).sort( { årstall: 1 } )
```

- Aggregering: Finn antall bøker i informatikkbiblioteket fordelt på årstall:

```
db.boksamling.mapReduce(
  function() { emit( this.årstall, this.tittel ); },
  function(key, values) { return Array.count( values ) },
  {
    query: { "plassering.bibliotek": "UiO Informatikkbiblioteket" },
    out: "antallbøker"
  }
)
```

Key-value-databaser versus dokumentorienterte databaser

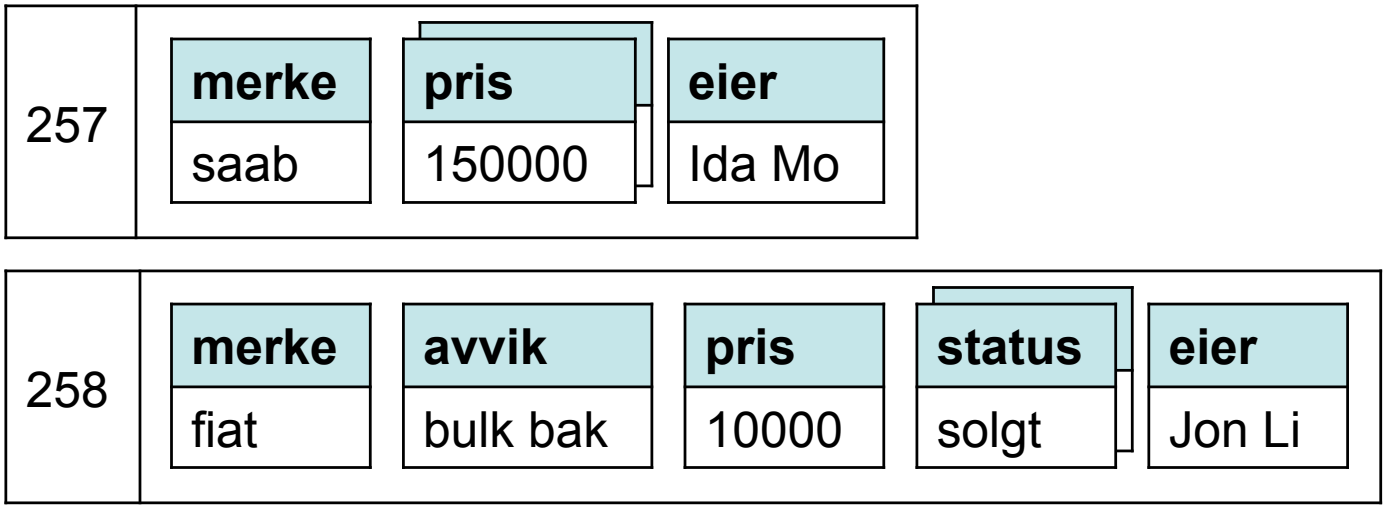
- Glidende overgang mellom de to betegnelsene
  - Noen databasesystemer omtales som key-value-databaser i enkelte sammenhenger og som dokumentorienterte i andre
  - Noen key-value-databaser kan suppleres med metadata som databasesystemet benytter til å få tilgang til/innsikt i strukturene og utnytter til å snevre inn søket
  - Dokumentorienterte databaser er key-value-databaser hvis man ikke er interessert i å søke på strukturene, men bare ønsker å få tak i fullstendige dokumenter

Fra relasjonsdatabaser til wide-column-databaser

- Tabell i en relasjonsdatabase:

Id	merke	hjul	motor	avvik	pris	status	eier
257	saab	NULL	NULL	NULL	150000	NULL	Ida Mo
258	fiat	NULL	NULL	bulk bak	10000	solgt	Jon Li

- Rader i en wide-column-database:



Wide-column-databaser I

- Datastruktur:
  - En **celle** er et tidsstemplet par $(n,v)_t$ der n er et kolonnenavn, v er en verdi og t er tidsstempelet. Noen systemer bruker begrepet «kolonne» om det som her betegnes som en «celle»
  - En **rad** er et par (k,cc) der k er en **radnøkkel** og cc en mengde av samhørende celler
  - En **kolonnefamilie** er en samling av kolonner. Hva som mer presist ligger i begrepet kolonnefamilie, avhenger av databasesystemet
  - Kolonnefamilier må deklareres i forkant, mens kolonner kan opprettes dynamisk
  - Antall kolonnenavn kan være svært høyt, derav betegnelsen «wide-column»
- Hva som utgjør aggregat og lagringsenhet, avhenger av databasesystemet

Wide-column-databaser II

- Søk:
  - Angi radnøkkel, eventuelt et intervall av radnøkler og ytterligere «kvalifikasjoner» (f.eks. kolonnenavn = verdi)
- Fleksibelt skjema:
  - Hver rad kan inneholde et vilkårlig antall celler
  - Kan dynamisk legge til eller fjerne kolonner

Apache Cassandra

- Fleksibelt skjema:
  - Må deklarere *kolonnefamilier* (se neste side)
  - Må ikke deklarere hva slags kolonner en kolonnefamilie kan inneholde
- Transaksjonshåndtering:
  - ACID-egenskaper pr. *partisjon* (se neste side)
  - Transaksjoner kan velge isolasjonsnivåer på tvers av partisjoner
  - Distribuerte transaksjoner benytter Paxos Consensus
- Indekser: Primær- og sekundærindekser
- Skalerbarhet: Peer-to-peer-modell; hvert cluster har noder organisert i en ring
- Eksempler på bruk: Facebook, eBay, Netflix, Finn
- Open-source under Apache Licence

Datamodell

- **Rad/record**: Samling av **kolonner**
  - **Kolonne**: Navn-verdi-par m/tidsstempel for versjonering (svarer til en **celle**)
  - **Superkolonne**: Kolonne der verdien er en samling (**map**) av navn-verdi-par
  - **Kolleksjon**: Kolonne der verdien er en mengde, liste eller map
- **Kolonnefamilie** = **tabell**: Samling av rader
  - **Primærnøkkel/radnøkkel** defineres på ett eller flere kolonnenavn
  - **Sammensatte primærnøkler** (dvs. med flere kolonnenavn) brukes til å splitte store tabeller i **partisjoner**. Radene i en partisjon lagres samlet (**clustered**), sortert på primærnøkkelen
  - Kan definere sekundærindekser på kolonne- og kolleksjonsnavn
- **Keyspace**: Samling av kolonnefamilier (svarer til en relasjons-database)

Deklarasjon av et skjema

```
CREATE KEYSPACE minDB
 WITH replication = { 'class' : 'SimpleStrategy',
                      'replication_factor' : 2 };
use minDB;

CREATE TABLE boksamling (
 bibl text,
 id int,
 isbn text,
 tittel text,
 forfatter list<text>,
 utlaan map<timestamp, text>,
 primary key (bibl, id) );
```

Logisk oppbygning (uten tidsstempler)

```
boksamling: ← tabell/kolonnefamilie   rad
  (bibl, id): ('ifibib', 71478198330002201)
    isbn: 'ISBN0884053407'      ← kolonne
    tittel: 'Simula BEGIN'      ← kolonner
    forfatter: ['Graham M. Birtwistle', ...]
    utlaan:                     ← superkolonne/map
      '2002-01-05': 'Jo Å'      ← subkolonne
      '2005-09-17': 'Liv Bø'    ← subkolonne
  (bibl, id): ...               ← rad
```

```
INSERT INTO boksamling (bibl, id, tittel, forfatter, utlaan)
 VALUES ('ifibib', 71478198330002201, 'Simula BEGIN',
          [ 'Graham M. Birtwistle', 'Ole-Johan Dahl',
            'Bjørn Myrhaug', 'Kristen Nygaard' ],
          { '2002-01-05' : 'Jo Å', '2005-09-17' : 'Liv Bø' }
        );

INSERT INTO boksamling (bibl, id, arstall)
 VALUES ('ifibib', 71478198330002201, 1973);
```

Spørringer

- Finn bøker på informatikkbiblioteket, begrens svarmengden til maksimalt 10:

```
SELECT *
FROM boksamling
WHERE bibl = 'ifibib'
LIMIT 10;
```

- Ingen aggregering, men kan gjøre COUNT(*) på resultatmengden:

```
SELECT COUNT(*)
FROM boksamling
WHERE bibl = 'ifibib';
```

- Finn alle bøker av Ole-Johan Dahl: Må ha sekundærindekser på kolonner og kolleksjoner for å kunne søke på dem

```
CREATE INDEX ON boksamling (forfatter);
SELECT tittel, bibl
FROM boksamling
WHERE forfatter CONTAINS 'Ole-Johan Dahl';
```

Apache HBase

- Open-source-versjon av **Google BigTable**
- Fleksibelt skjema:
  - Må deklarere *tabeller* og *kolonnefamilier* (se neste side)
  - Må ikke deklarere hva slags kolonner en kolonnefamilie kan inneholde
- Transaksjonshåndtering:
  - ACID-egenskaper pr. rad
  - Read committed på tvers av rader
- Indekser:
  - Primærindekser på radnøklene
  - Bloomfiltere på kolonnene (probabilistisk datastruktur)
- Skalerbarhet: Bruker Hadoop til distribuert lagring på clustere
- Eksempler på bruk: Facebook, Yahoo!, Linkedin, Netflix

Datamodell

- **Rad**: Samling av **celler**. Hver rad er entydig identifisert ved en **radnøkkel**
- **Celle**: kolonnenavn-verdi-par m/tidsstempel for versjonering
- **Tabell**: Samling av rader, sortert på radnøkkelen
- **Kolonne**: Samling av celler
  - Svarer til det som kalles en kolonne i relasjonsmodellen
  - Kolonnenavnet er på formen **kolonnefamilie:kvalifikator** (**columnfamily:qualifier**)
- **Kolonnefamilie**: Samling av samhørende kolonner i en tabell
- Store tabeller splittes i **regioner/tablets** basert på radnøkkelen
  - Svarer til det som kalles partisjoner i Cassandra
  - Innen en region er kolonnene i hver kolonnefamilie samlokalisert fysisk

Eksempel på en tabell[1]

kolonnefamilier → **person** **jobb**

rad-nøkkel	**person: navn**	**person: adresse**	**person: telefon**	**jobb: kontor-nummer**	**jobb: telefon**	**jobb: avdeling**
5237	Lise Høeg	0271 Oslo		4510	55555	Regnskap
5377	Carl Ask		23456789			Kantine

Logisk oppbygning
(uten tidsstempler)

```
ansatte:                              ← tabell
  rad: '5237'                         ← rad
    person:                           ← kolonnefamilie
      navn: 'Lise Høeg'               ← kvalifikator+verdi
      adresse: '0271 Oslo'            ← kvalifikator+verdi
    jobb:                             ← kolonnefamilie
      kontornummer: '4510'
      telefon: '55555'
      avdeling: 'Regnskap'
  rad: '5377'
    person:
      ...
```

[1]Tabellen viser ikke tidsstempler

HBase shell-kommandoer

- Deklarasjon av et skjema:

```
> create 'ansatte', {NAME => 'person'}
> put 'ansatte', '5237', 'person:navn', 'Lise Høeg'
> put 'ansatte', '5237', 'person:adresse', '0271 Oslo'
> alter 'ansatte', {NAME => 'jobb', VERSIONS => 5}
> put 'ansatte', '5237', 'jobb:kontornummer', '4610'
> put 'ansatte', '5237', 'jobb:kontornummer', '4510'
```

- Spørringer:

```
> get 'ansatte', '5377', {COLUMN = 'person:navn'}
> scan 'ansatte',
   {COLUMNS => ['person:navn', 'person:adresse',
                'jobb:telefon'],
    LIMIT => 10,
    STARTROW => '3902'}
```

Dokumentorienterte databaser versus wide-column-databaser

- Dokumentorienterte databaser tillater generelle treliknende strukturer med vikårlig dybde; wide-column-databaser har tre til fire nivåer
  - Større frihet i valg av struktur gir større fleksibilitet hva gjelder å representere virksomhetsområdets modell direkte i databasen
  - Rikere struktur tillater potensielt en rikere klasse av spørringer
  - Enklere struktur gir mer effektive søk
- Begge kan bygge indekser på utvalgte navn i strukturene for å effektivisere søk

Key-value-databaser versus wide-column-databaser

- Hvis vi bare søker på radnøkler, kan strukturen til en wide-column-database betraktes som en key-value-database
- Sett fra fugleperspektiv kan derfor wide-column-databaser, på samme måte som dokumentorienterte, fremstå som key-value-databaser
- I noen sammenhenger blir derfor alle de modellene vi her betegner som aggregatorienterte, gitt merkelappen key-value-databaser

Grafdatabaser

- For data som er naturlig å beskrive og traversere som grafer
  - Hver node har en indre struktur som angir nodens egenskaper
  - Kantene angir relasjoner mellom nodene
  - Kantene kan på samme måte som nodene være informasjons-bærere; som et minimum har de et navn eller en type som reflekterer hva slags noderelasjoner de representerer
- Kan gjøres skjemaløst, dvs. må ikke spesifisere i forkant nodenes indre struktur eller hvilke typer relasjoner de kan ha seg imellom
  - Nye noder og kanter (med nye indre strukturer) kan introduseres dynamisk
  - Eksisterende noder og kanter kan utvides med nye egenskaper
- Søk: Angi hvordan grafen skal navigeres
  - Grafen traverseres direkte via pekere til nabonoder (traverseringen krever ingen indekser og ingen joinoperasjoner)

Neo4j

- Skjemaløs - må ikke definere noen struktur i forkant
- Spørrespråk: Cypher
  - Deklarativt, mønsterbasert
- Transaksjonsstøtte: Default isolasjonsnivå er read committed
  - Neo4j har et Java-API som tilbyr låser; transaksjoner kan oppnå høyere isolasjonsnivåer ved å håndtere låsene selv
- Skalerbarhet: Kan ha clustere med master-slave-modell der slavene er speilinger av masterdatabasen
- Eksempler på bruk: EBay, HP
- Neo4j Community Edition er open-source under GNU GPL

Datamodell

- **Node**
- **Relasjon** (**relationship**):
  - Rettet kant mellom to noder
  - Kan traverseres begge veier
- **Property:** Navn-verdi-par
  - Både noder og relasjoner kan ha properties
  - Navnet er en tegnstreng
  - Verdien er hentet fra en basal datatype (int, char, ..) eller en array over en basal datatype (int[], char[], ..)
- **Label:** Brukes til å angi typen eller rollen til en node
  - En node kan ha null eller flere labels
- **Relasjonstype**: Brukes til å karakterisere en relasjon
  - Hver relasjon har nøyaktig én type

Opprettelse av noder og kanter

```
CREATE
  (a:Person {navn:"Anne", født:1997}),
  (b:Person {navn:"Bjørn", fødselsdato:191148}),
  (c:Person {navn:"Carl", status: "gift",
             interesser:["skiturer", "dykking"],
             email:"carl@gmail.com"}),
  (a)-[:SLEKTSKAP {type:"datter", status:"adoptert"}]->(b),
  (a)-[:SLEKTSKAP {type:"niese"}]->(c)
```

```
MATCH (x:Person {navn:"Bjørn"}), (y:Person {navn:"Carl"})
CREATE (x)-[r:SLEKTSKAP {type:"bror"}]->(y)
RETURN r
```

Spørringer

- Slektninger av slektninger:

```
MATCH (p:Person {navn:"Anne"})-[:SLEKTSKAP]-(s1),
      (s1)-[:SLEKTSKAP]-(s2)
RETURN s2
```

- Felles slektninger:

```
MATCH (pers1)-[:SLEKTSKAP]-(slekt),
      (pers2)-[:SLEKTSKAP]-(slekt)
WHERE pers1.navn = "Anne" AND pers2.navn = "Carl"
RETURN slekt
```

- Korteste sti (begrenset til maks 5 relasjoner):

```
MATCH (p1:Person {navn:"Hilde"}), (p2:Person {navn:"Geir"}),
      sti = shortestPath((p1)-[*..5]-(p2))
RETURN sti
```

- Antall slektskap (når retningen på relasjonen er signifikant):

```
MATCH (a:Person)-[:SLEKTSKAP]->(b:Person)
RETURN a.navn, count(*)
ORDER BY count(*) DESC
```

RDF - Resource Description Framework

- RDF er en W3C-standard for informasjonsmodellering. All informasjon er kodet som **fakta**, dvs. tripler bestående av **subjekt**, **predikat** og **objekt**. Eksempel:

```
@prefix ee: <http://www.mn.uio.no/ifi/eksempel/> .
ee:a ee:harNavn "Anne" .
ee:a ee:erFødt 1997 .
ee:a ee:erNieseAv ee:c .
```

- Graftolkning: Triplene svarer til rettede kanter i en graf
  - Subjektet og objektet identifiserer noder (**ressurser**)
  - Predikatet beskriver en relasjon mellom nodene
- Spørrespråk: SPARQL
- RDF-databaser: Kan lagre informasjonen "flatt" i en **triplestore**
- RDF er en *standard*, så RDF er ikke låst til et produkt eller en leverandør. RDF-baserte systemer: Apache Jena, AllegroGraph, GraphDB, ...

Grafmodeller versus andre datamodeller

- Grafmodeller vs. relasjonsmodellen:
  - Traversering av en graf er svært mye billigere enn join; benytter direktepekere til nabonoder
  - Arbeidsbelastning forskyves fra queryeksekvering til innsetting og vedlikehold av data
- Grafmodeller vs. aggregatorienterte modeller:
  - Grafdatabaser er ikke nødvendigvis designet med tanke på skalering til den størrelsesorden som aggregatorienterte modeller kan håndtere; krever tildels monolittiske systemer
  - Konsistens kan ivaretas på tvers av noder i grafen
  - Spørrespråk følger helt andre prinsipper enn i aggregatorienterte modeller

Hvilken NoSQL-datamodell?

- Big data: Aggregatorienterte modeller (key-value-databaser, dokumentorienterte databaser, wide-column-databaser)
- Informasjonsutveksling: Dokumentorienterte databaser
- Grafer: Grafmodeller

Introduksjon til NoSQL

Martin Fowler har en god introduksjon til NoSQL
(fra GOTO Aarhus Conference 2012):

http://www.youtube.com/watch?v=qI_g07C_Q5I
(Varighet: 55 min.)

Temaer:

- Historikken bak NoSQL
- Datamodeller i NoSQL
- Distribusjonsmodeller (sharding, replikering)
- ACID vs. BASE
- CAP-teoremet
- Når og hvorfor bruke NoSQL