Konstruksjon av databaser for aktuelle og historiske adresser

Kjell Bratbergsengen
Institutt for datateknikk og informasjonssystemer
NTNU

Sammendrag

Adresseinformasjon inngår i mange databaser – både offentlige og private. Adressedata endrer seg ofte og de må vedlikeholdes og oppdateres. En felles adressedatabase vil kunne spare arbeid og gi bedre datakvalitet. Adresse i dagligtale er både upresist og kontekstavhengig. I artikkelen tar vi for oss adressen som begrep. To aktører – det offentlige Statens kartverk har ansvaret for forvaltningen av stedsadresser, mens statseide Posten Norge AS har ansvaret for postnummer og postadresser. Både stedsadresser og postadresser endrer seg over tid.

Vi skiller mellom to typer adresse-endringer: 1) Skifte av adresse, – fra én adresse til en annen og 2)Endring av adressen i seg selv. Vi presenterer databaseutforminger som tar vare på begge disse endringene. Den beskrevne databaseutformingen er ikke ferdig. Både praktiske eksperimenter bør utføres og alternativer må utvikles. Hovedhensikten med denne artikkelen er å belyse en problemstilling.

1 Adresser – innledning

Dagligtalens adresse forståes oftest i en sammenheng og kan ha to meninger: Stedsadresse eller postadresse. Stedsadressen er en lokaliseringsadresse eller et oppmøtested. Den er særlig viktig for alle utrykningsetater. Adressen må være entyding og indikere hvordan en skal komme dit. Det er etablert regler for hvordan en adresse skal utformes. Det er Statens kartverk som har hovedansvaret for utforming og vedlikeholdet av stedsadressene i landet. Kartverket har også utarbeidet en adresseveileder for hvordan adresser skal utformes og forvaltes, se [4].

Postadressen bestemmer hvor posten skal leveres: Ved en vei eller gateadresse eller i en postboks. Postadressene forvaltes av statsaksjeselskapet Posten Norge AS. Alle postadresser må ha et postnummer. Postnumrene brukes internt i Posten til postens interne sorterings- og fordelingsmekanismer. Til postnummeret hører også en stedsangivelse. En vanlig privat postadresse består av en gate-veiadressedel og postnummer og poststed. Postnummer er en dynamisk størrelse, hvilke gater og veier som sogner til et gitt postnummer bestemmes av Posten. Endringene trer i kraft 1. oktober hvert år, [7, 8]. Posten er en kommersiell bedrift og postnummer-registeret

Denne artikkelen ble presentert på konferansen NIK-2012; se http://www.nik.no/.

må kjøpes hvert år for å være oppdatert. *Adresse* i det etterfølgende er å forstå som postadresse.

DIFI - Direktoratet for forvaltning og IKT har utarbeidet en rapport som heter "Nasjonale felleskomponenter i offentlig sektor", se [1]. Denne rapporten foreslår fem felleskomponenter:

- Enhetsregistrert - grunndata om virksomheter
- Folkeregisteret - grunndata om personer
- Matrikkelen - grunndata om eiendom
- Altinn
- Felles infrastruktur for e-ID i offentlig sektor

Vi legger merke til at *adresse* ikke eksplisitt er nevnt blant de fem felleskomponentene. Imidlertid er adresse felles informasjon i de fire første komponenentene. Etter undertegnedes mening er *adresse* en fundamental fellsmodul. Vi vil analysere forskjellige mulige utforminger av en database for adresse ut fra et slikt perspektiv.

Det legges ned et betydelig arbeid hvert år med å vedlikeholde og forbedre adressering og adressesystemer, se for eksempel [5].

Artikkelen berører fire aspekter:

1. Hvordan bygger vi opp en adressedatabase?
2. Hvordan kopler vi adresser til personer og andre adresserbare objekter?
3. Hvordan kan vi ta vare på historiske adresser som sådan?
4. Hvordan kan vi integrere aktuelle og historiske adresser for objekter (personer, firma, osv.)?

Vi holder oss til norske adresser. I et praktisk system må vi også ha rom for å kunne registrere andre adresser. Det kan for eksempel gjøres ved å lagre dem som rene tekstobjekter.

Adressen som attributt

I de enkleste informasjonssystemer er adressen et attributt, typisk er listen over idrettslagets medlemmer eller de ansatte på jobben. "Databasen" kan se slik ut, se tabell 1:

PERSONER			
PersonNr	Navn	Adresse	TelefonNr

Tabell 1: Adresse som attributt

Adresse er et attributt som i sin enkleste form inneholder et antall linjer tekst, en ny linje kan være signalisert med et komma. En mer avansert utgave vil være å definere *Adresse* som et sammensatt attributt, bestående av:

- veg eller gatenavn,
- gatenummer,
- postnummer og
- poststed

Databasen er ikke normalisert[1] da postnummer bestemmer poststed. Hvis adressen skulle være et objekt ville *Adresse* i posten PERSONER vært en fremmednøkkel til et adresseobjekt.

Fordelen ved denne enkle utformingen, hvor adressen er et attributt - er at all informasjon om en bestemt person er på ett sted. Skal vi finne ut hvem som bor i nærheten av hverandre eller har samme adresse kan vi sortere etter adresse eller bruke en søkemotor hvis filen er tilgjengelig for "crawling"[2]. Eksempel på en SQL-setning som finner personer i samme gate er:

```
SELECT * FROM PERSONER WHERE Adresse LIKE 'Bergs All%';
```

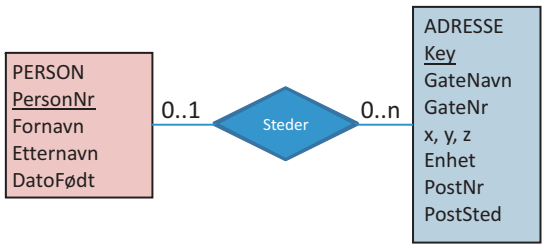
Vi finner alle som har adresse Bergs Allé. Vi valgte å stoppe foran e-en fordi mest sannsynlig finnes gatenavnet i versjoner både med og uten apostrof.

Når en person flytter vil, den gamle adressen overskrives. Gamle adresser vil ikke bli tatt vare på uten at en tar vare på hele databasen med visse mellomrom. Selv da er det mulig å miste hyppige flyttinger. Det følger heller ikke av databasen hva slags adresse dette er. En person kan ha mange adresser: bostedsadresse, ferieadresse, jobbadresse, osv.

I det etterfølgende skal vi problematisere utformingen av adresse. Vi starter med å objektifisere adressen. Deretter vil vi trekke inn tidsdimensjonen slik at vi kan ta vare på historikk og være i stand til å sette adressene inn i tiden.

2 Objektifisert adresse

En adresse er angivelse av et sted. Angivelsen er slik at det skal være lett å komme dit ved å følge den beste veien. Adresse som sådan er ikke følsom informasjon, det er *koplingen* av for eksempel personer til en adresse som kan være følsom. Den enkleste objektifisering



Figur 1: Adressen som objekt. Databasen er ikke normalisert. I tillegg til gatenavn og gatenummer er også adressen oppgitt med GPS-koordinater og enhetsbetegnelse. Enhet er enhet i en bygning, for eksempel en leilighet. Enhet skrives som Hxxyy hvor xx er etasje og yy er enhet innen etasjen.

er å la det sammensatte attributtet: gatenavn, gatenr, postnr og poststed være ett objekt med en surrogatnøkkel[3] som identifikator. Det vil gi ikke-normaliserte tabeller av samme grunner som tidligere nevnt. I figur 1 er det vist et EER-diagram for en overforenklet objektmodell av adresse.

[1] Normalisering er forklart i de fleste introduksjonsbøker om databaser. Se for eksempel [3] eller [6].

[2] En crawler er et program som løper gjennom tilgjengelige filer på "alle" maskiner for å finne data som blir søkbare i en søkemotor.

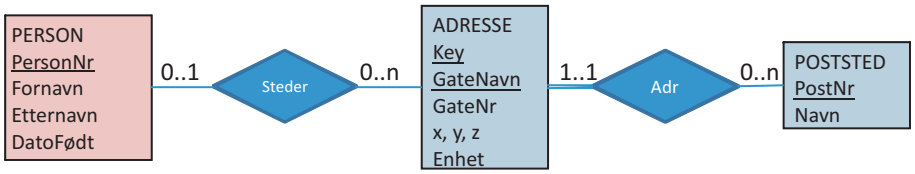
[3] Surrogatnøkkel er et entydig attributt uten ekstern betydning. Den enkleste og kanskje beste form for surrogatnøkkel er et løpenummer.

Grader av normalisering

I det etterfølgende skal vi demonstrere databaseskjema for forskjellige grader av normailisering.

Normalisert adresse, poststed i eget objekt

En normalisert utgave av ADRESSE vil bestå av to tabeller: ADRESSE og POSTSTED. Figur 2 viser EER-diagrammet. Tabellene i databasen kan utformes som følger:



Figur 2: Adressen som objekt. Databasen er normalisert

PERSON	(PersonNr, Fornavn,Etternavn, DatoFødt, *ADRESSE_Key*)
ADRESSE	(Key, GateNavn, GateNr, x, y, z, Enhet, *POSTSTED_PostNr*)
POSTSTED	(PostNr, Poststed)

I de fleste EER-diagram og oversikter over databasens tabeller (skjema) har vi tatt med PERSON som adressebruker. Som nevnt er det mange forskjellige brukere og mange typer entiteter som kan koples til en adresse, PERSONER er illustrasjonsobjekt. I skjema har vi understreket attributter som inngår i primærnøkkelen. Attributter som er - eller inngår i fremmednøkkel er skrevet i kursiv. Atributter som inngår i både nøkkel og fremmednøkkel er understreket og i kursiv.

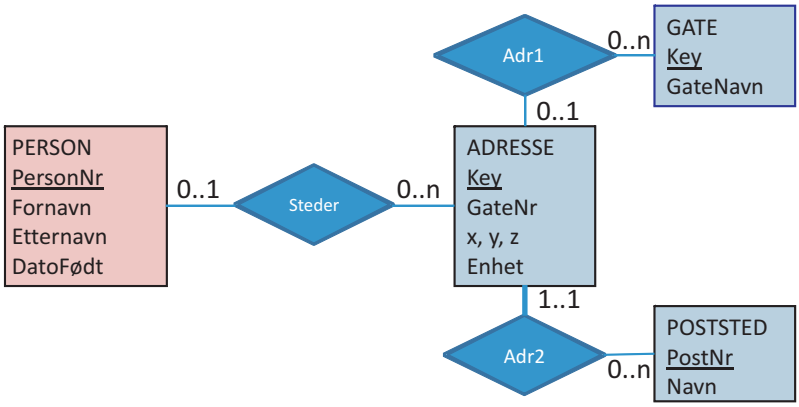
Der vi opererer med nøkkel uten ekstern betydning kaller vi identifikatoren for Key. Fremmednøkler starter som regel med navnet på den tabell de er nøkkel i. Normalisering gir viktige fordeler: Nå kan vi for eksempel endre skrivemåte for poststed ved å oppdatere kun én post. SELECT-setningen for å finne alle som bor i Bergs Allé blir nå:

```
SELECT p.PersonNr, p.Fornavn, p.EtterNavn, a.GateNavn, a.GateNr,
       o.PostNr, o.Navn
FROM PERSONER  AS p, ADRESSE AS a, POSTSTED AS o
WHERE g.GateNavn LIKE 'Bergs All%'
    AND a.Key     = p.ADRESSE_Key
    AND o.PostNr = a.POSTSTED_PostNr;
```

Objektifisert gatestrekning

Vi kan drive objektifiseringen lenger ved å objektifisere gaten. EER-diagrammet er vist i figur 3. Vi har objektifisert gate- eller veistrekningen. Vi har ikke objektifisert navnet på gaten. Det vil da være mange strekninger med samme navn. Skjema for EER-diagrammet i figur 3 er:

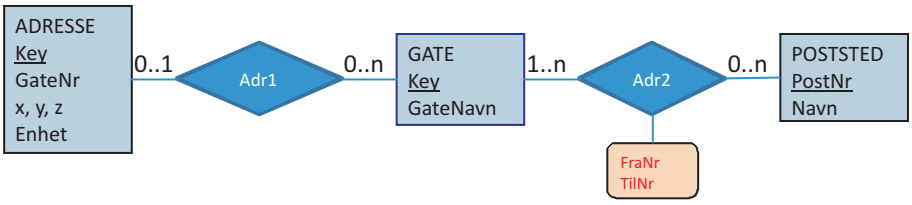
PERSON	PersonNr, Fornavn, Etternavn, *ADRESSE_Key*)
ADRESSE	(Key, *GATE_Key, POSTSTED_PostNr,* GateNr, x, y, z, Enhet)
GATE	(Key, GateNavn)
POSTSTED	(PostNr, Navn)



Figur 3: Objektifisert adresse. Her er selve gatestrekningen objektifisert. NB: Gatestrekning er ikke det samme som gatenavn.

Poststed som funksjon av gatestrekning

Det en relasjon mellom postnummer og gate. Alle gateadresser - gatenavn og gatenummer er relatert til ett og bare ett postnummer. Imidlertid kan lange gater sogne til flere postnummer, derfor er det en mange-til-mange-relasjon mellom postnummer og gate. Samme gatestrekning kan i prinsippet ha flere segmenter som er relatert til samme postnummer. Gate- og gatenummer *sammen* bestemmer altså postnummer. Med en entydig sammenheng mellom gate, gatenummer og postnummer kan vi endre EER-diagrammet i figur 3 hvor gate og postnummer er urelatert, til EER-diagrammet i figur 4. I det siste EER-diagrammet er poststed direkte relatert til gate med støtte i gatenummer.



Figur 4: Objektifisert adresse. Gatestrekningen er objektifisert. Poststed bestemmes av gate og postnummer fra og til.

PERSON	PersonNr, Fornavn, Etternavn, *ADRESSE_Key*)
ADRESSE	(Key, *GATE_Key,* GateNr, x, y, z, Enhet)
GATE	(Key, Navn)
POSTSTED	(PostNr, Navn)
Adr2	(*POSTSTED_PostNr, GATE_Key,* FraNr, TilNr)

SELECT-setningen for å finne alle som bor i Bergs Allé blir nå:

```
SELECT p.PersonNr, p.Fornavn , p.EtterNavn, g.Navn, a.GateNr,
       o.PostNr, o.Navn
FROM PERSONER p, ADRESSE a, GATE g, POSTSTED o, Adr2 a2
WHERE g.Navn     LIKE   'Bergs All%'
    AND g.Key    = a.GATE_Key
```

```
    AND a.Key    = p.ADRESSE_Key
    AND a.GateNr BETWEEN a2.FraNr AND a2.TilNr
    AND a2.POSTSTED_PostNr = o.PostNr;
```

Dette er vesentlig mer komplisert enn å spørre i den første enkle persontabellen vi startet med. For å komme fram til et resultat må databasesystemet gjøre to seleksjoner og fire foreninger. Den mer krevende utførelse kan vi ikke endre på, men vi kan dekke over kompleksiteten for den som skriver rapporteringsprogrammer ved å definere "views".

Normaliseringens begrensning

Ved inspeksjon av tabellene over postnummer og tilknyttede adresser, oppdager vi fort at det er en rekke postnummer som ikke er koplet til gateadresser. Det kan være postboksadresser eller adresser til større firma. Dermed er det ikke hensiktsmessig å bruke den siste normaliseringen og vi står igjen med EER-diagrammet i figur 3 som den mest praktiske utforming av adressedatabasen.

Definering av views

Som tidligere nevnt kan vi definere en view (virtuell tabell, ekstern modell eller et utsnitt) av databasen for å lette seleksjonsoperasjoner. Et "flatt" syn på databasen hvor personer er oppført med navn og full adresse i en rad, fås av utsnittet:

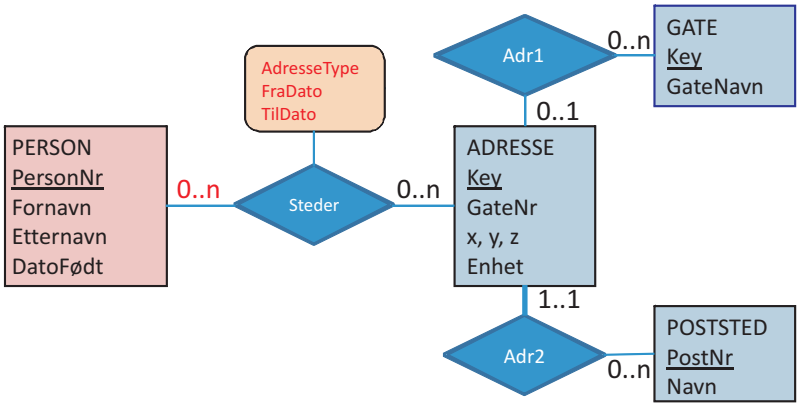
```
CREATE VIEW
    PERSONFIL (pnr,fornavn,etternavn,gatenavn,gatenr,postnr,poststed)
AS
SELECT p.PersonNr,p.Fornavn,p.Etternavn,g.GateNavn,a.GateNr,o.PostNr,o.Navn
FROM PERSONER  AS p, ADRESSE AS a, GATE AS g, POSTSTED AS o
WHERE p.ADRESSE_Key     = a.Key
    AND  a.GATE_Key     = g.Key
    AND  a.POSTSTED_Key = o.PostNr;
```

En view fører ikke til ekstra lagring av tabeller. *View-definisjonen* blir lagret. Den blir hentet fram og brukt når spørsmål som bruker den virtuelle tabellen blir utført.

3 Personer med flere typer adresser og adresser i tidsdimensjonen

I figur 5 viser vi en database hvor vi har tatt med flere typer adresser og hvor vi tar vare på adressekoplingshistorien. Det fører til at vi må opprette en ny tabell: *Steder*. Denne databasen tar ikke vare på selve *adressenes historiske utvikling*. Det behandles i avsnitt 4.

Det er relasjonen *Steder* som viser vei til bostedsadressen. Perioden for en gyldig stedstilknytning er oppgitt i relasjonens tilknyttede attributter FraDato og TilDato. Når TilDato er NULL kjenner vi ennå ikke avslutningsdato for relasjonen. Denne relasjonen kan derfor tolkes som nåverdien på adresse, altså den aktuelle adresse. FraDato kan også være NULL og en tolkning kan være at relasjonen gjelder fra "tidenes morgen" eller fra før databasen ble opprettet. Det vil imidlertid lette senere programmering hvis en finner en annen konvensjon enn NULL for ukjent start og avslutning. Ukjent start kan i stedet defineres som en tidlig dato for eksempel 1. januar 1900. Tilsvarende kan en uavsluttet periode ha en tildato som ligger langt inn i framtida. AdresseTyper bør defineres med et begrenset sett av verdier, for eksempel: bosted, hybel, arbeid, ferie, hytte, osv. Det at vi nå tar vare på flere typer informasjon må betales med en mer kompleks datauthenting selv ved enkle operasjoner. Igjen kan vi legge til rette for forenklet programmering ved å definere views.



Figur 5: Type adresse og tidsdimensjonen er tatt med i koplingen mellom adresse og objekt - person i dette tilbfelle. Vi kan nå lagre aktuelle og historiske adresser av forskjellige typer.

Modifisert database-skjema

Registrering av historiske og ulike typer adresser til samme objekt (person) gjør at vi må modifisere den opprinnelige databaseutformingen. Siden vi går fra en-til-mange-kopling til mange-til-mange-kopling må vi opprette en egen tabell for relasjonen. Fra PERSON har vi fjernet fremmednøkkelen til ADRESSE. Alle adressereferanser til person er nå samlet i tabellen **Steder**. Hvis en person skal kunne bo flere perioder på samme adresse må FraDato eller TilDato inngå i nøkkelen. Vi har valgt FraDato fordi FraDato er kjent når den nye forbindelsen opprettes.

PERSON	(PersonNr, Fornavn, Etternavn, DatoFødt)
ADRESSE	(Key, GATE_Key, GateNr, *POSTSTED_PostNr*)
GATE	(Key, GateNavn, x, y, z, Enhet)
POSTSTED	(PostNr, Poststed)
Steder	(*PERSON_PersonNr, ADRESSE_Key*, AdresseType, FraDato, TilDato)

Definisjon av utsnitt for aktuell bostedsadresse

Under følger definisjonen av et utsnitt for å beholde et enkelt grensesnitt til programmer som operer på aktuell bostedsadresse. At vi bare kan ha én aktuell bostedsadresse kan vi ikke modellere inn i skjemaet, det er en kontroll som - hvis den skal gjøres - må gjøres i en egen SQL-setning. Vi har brukt konvensjonen at gjeldende adresse har en avslutningsdato som ligger i framtiden.

```
CREATE VIEW
PERSONFIL (pnr,fornavn,etternavn,gatenavn,gatenr,postnr,poststed)
AS
SELECT p.PersonNr,p.Fornavn,p.Etternavn,g.GateNavn,a.GateNr,o.PostNr,o.PostSted
FROM PERSONER  AS p, Steder AS s, ADRESSE AS a, GATE AS g,
     POSTSTED AS o
WHERE p.PersonNr       = s.PERSON_PersonNr
    AND s.AdresseType  = 'bosted'
    AND s.TilDato      > TODAY
    AND s.ADRESSE_Key  = a.Key
```

```
AND a.GATE_Key     = g.Key
AND a.POSTSTED_Key = o.PostNr;
```

4 Adressers endring over tid

Før vi starter arbeidet med å utforme en database som tar vare på selve adressens historiske utvikling, må vi mer nøye se på adressedata slik de behandles fra forvalternes side.

Stedsadresse som den er definert av Statens kartverk

Statens kartverk har satt opp regler for hvordan en stedsadresse skal bygges opp, se adresseveilederen [4]. Sentral i Kartverkets system er *parsellen*. En parsell er en vei eller gate, eller en del av en vei eller gate. En parsell identifiseres med en *adressekode* - AdrKode i EER-diagrammet i figur 6. Adressekoden er et nummer mellom 1000 og 99998 i følge adresseveilederen. Nummeret er entydig innen en kommune. Koden beholdes selv om parsellen skifter navn. En parsell krysser ikke en kommunegrense. Hvis vegen fortsetter kan imidlertid navnet på vegen være det samme. En adressekode som går ut av bruk skal ikke gjenbrukes. Hva som skjer ved f.eks. kommunesammenslåing er ikke beskrevet i veilederen. Ved kommunesammenslåing er det stor sannsynlighet for at en rekke adressekoder mister sin entydighet. Adressekode er parsellens eksterne identifikator som kan brukes i andre systemer, for eksempel til kartproduksjon. Siden en parsell ikke krysser kommunegrenser vil en parsell entydig høre til en og bare en kommune på et gitt tidpunkt.

KNr i KOMMUNE er et nummer mellom 0101 og 1999. De to første siffer angir fylke og de to siste siffer angir et kommunenummer. Numrene er unike til en hver tid, men både navneendring og gjenbruk kan finne sted, se for eksempel informasjon om kommunenummer i Wikipedia.

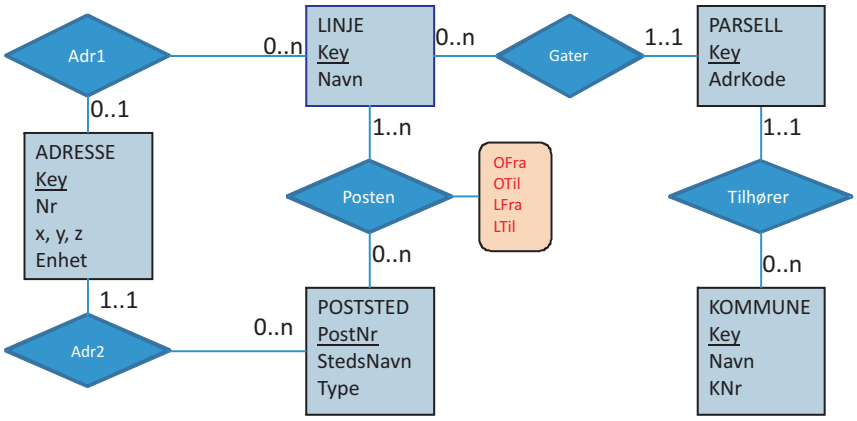
Postnummerets betydning

Som tidligere nevnt er Posten Norge AS forvalter av postnummer. Posten bestemmer hvilke postnummer en gatestrekning skal ha. Posten er ikke sensitiv for kommunegrenser, slik at koplingen mellom gate og postnummer gjelder en gatestrekning bestemt ut fra gatenummer. For gatestrekninger gjelder ulike nummerserier for høyre og venstre side av veien.

Men postnummer er ikke bare koplet til gate- eller veistrekninger. Større virksomheter kan ha eget postnummer, NTNU og SINTEF er eksempel på slike. Flere virksomheter kan dele et postnummer, adresseringen kan da se slik ut: Statoil Norge, 7005 Tronheim. En serie postbokser har samme postnummer. Ved postkontor med mange bankbokser kan hver serie ha forskjellige nummer. For eksempel har Florø følgende serier: Postnummer 6901 for postbokser 1-219, 6902: (220-419) og 6903 (420-669). I POSTSTED er Type en intern informasjon som forteller hva postnummeret står for: Vei eller gateadresse, postboksadresse, seviceboks, stedsnavn med postnummer, post i butikk, postmottaker med eget postnummer. EER-diagrammet for en database hvor vi tar hensyn til interne behov hos adresseforvalterne er vist i figur 6. Hovedteksten i første linje i adressen hentes fra entiteten LINJE. LINJE kan inneholde et gate- eller veinavn, navnet på en virksomhet, ordet Postboks, osv., det avhenger av hvilken type adresse vi har. Vi har også nøytralisert attributtet Nr i ADRESSE. Nr kan være et gatenummer, men det kan også være et postboksnummer.

Sammenhengen mellom LINJE og POSTSTED er gjennom relasjonen Posten. Relasjonen har fire attributter knyttet til seg. Hvis LINJE inneholder et veinavn og alle gatenummer i denne veien sogner til samme postnummer settes OFra=1 og LFra[4]=2.

[4]O står for odde og L for like. Gatenummeret er oddetall for adresser på høyre side og liketall for venstre side av veien når en går i retning av stigende nummer.



Figur 6: EER-diagrammet viser et øyeblikksbilde av koplingene mellom linje 1 og linje 2 i adressen, samt koplingen mellom gate (LINJE), parsell og kommune.

OTil og LTil settes begge til MAXGNR som er en konstant med høy verdi som aldri blir oversteget av noe gatenummer i databasen. Hvis bare en del av veien sogner til postnummeret, må det oppgis to serier, en for høyre og en for venstre side av veien. OFra må ha en verdi fordi den må inngå i nøkkelen til relasjonen Posten da forskjellige segmenter av samme gate kan sogne til samme postnummer.

Siden LINJE ikke alltid inneholder et gatenavn er det ikke alle LINJE-objekter som relaterer seg til PARSELL. Det forklarer kardinaliteten 0..n mellom Gater og LINJE. Databaseskjema for EER-modellen i figur 6 blir:

ADRESSE	(Key, Nr, x, y, z, Enhet, *LINJE_Key, POSTSTED_PostNr)*
LINJE	(Key, Navn)
PARSELL	(Key, AdrKode, *LINJE_Key, KOMMUNE_Key)*
KOMMUNE	(Key, Navn, KNr)
POSTSTED	(PostNr, StedsNavn, Type)
Posten	(*LINJE_Key, POSTSTED_PostNr,* OFra, OTil, LFra, LTil)

De lagrede data gir mulighet for konsistenskontroll. Nr i ADRESSE må ligge innenfor intervallet slik det er spesifisert i koplingen Posten. I PARSELL kunne vi også ha lagt inn flere data, nemlig hvilke gatenummer parsellen dekker. Igjen må vi ha en serie for hver side av veien. Informasjon om postnummeret og postnummerets endring over tid er i stor grad basert på [2].

5 Adressens historiske utvikling

Både entiteter og relasjoner kan endres. Derfor må vi ha med gyldighetsperiode for begge typer registreringer.

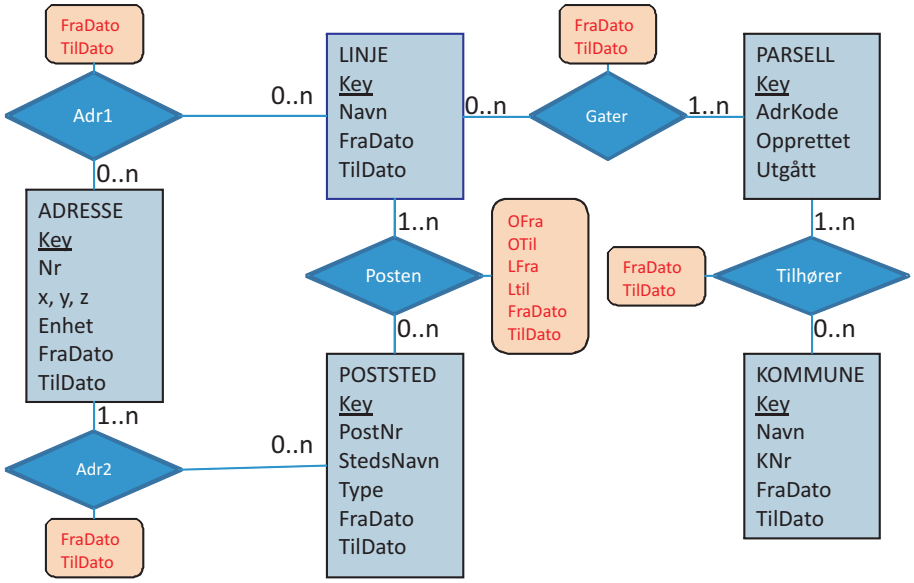
Når en parsell blir opprettet og eventuelt når den går ut av bruk, registreres i parsellobjektet. Da har vi ikke fått med at selve innholdet i parsellen kan endre seg. En parsell har et antall gatenummer knyttet til seg. Over tid kan en parsell endre innhold, det mest vanlige er at den forlenges og dermed utvides antall gatenummer. Vi kan velge å opprette et nytt parsellobjekt for hver endring. Termene *Opprettet* og *Utgått* i objektet bør henspeile på når *adressekoden*, dvs. parsellen som parsell ble opprettet og eventuelt når den går ut av bruk.

Entiteten KOMMUNE er heller ikke stabil, kommuner endrer navn og kommunekode kan endres. Entiteten får en gyldighetsperiode, og for hver endring opprettes det et nytt

objekt. Denne utvidelsen gir riktig database for hver tidsperiode. Databasen inneholder ingen direkte data som sier at en kommune er sammenslutning av kommunene x og y, det må tolkes ut fra dataene.

Vi har åpnet for at et postnummer kan brukes flere ganger i forskjellige betydninger. Det er nødvendig når vi ikke har en garanti for at postnummeret ikke blir gjenbrukt. Da er det mest hensikstmessig med en nøytral nøkkel - et løpenummer. PostNr går inn som en vanlig atributt og objektet har fått sin gyldighetsperiode.

I LINJE kan vi registrere at en veistrekning skifter navn eller at et navn endrer skrivemåte. Da opprettes et nytt LINJE-objekt. Det opprettes også nye koplinger. Det er gyldighetsperioder både på entiteter og koplinger, det gir muligheter for konsistenskontroll. Relasjonsmodellen som tilsvarer EER-diagrammet i figur 7 er:



Figur 7: EER-diagrammet viser en detaljert historiske utvikling av entiteter og relasjoner.

ADRESSE	(Key, Nr, x, y, z, Enhet, Fradato, TilDato)
LINJE	(Key, Navn, Fradato, TilDato)
PARSELL	(Key, AdrKode, Opprettet, Utgått)
KOMMUNE	(Key, Navn, KNr, FraDato, TilDato)
POSTSTED	(Key, PostNr, StedsNavn, Type, FraDato, TilDato)
Posten	(*LINJE_Key,POSTSTED_Key,*OFra,OTil,LFra,LTil,FraDato,TilDato)
Adr1	(*ADRESSE_Key, LINJE_Key,* Fradato, TilDato)
Adr2	(*ADRESSE_Key, POSTSTED_Key,* Fradato, TilDato)
Gater	(*PARSELL_Key, LINJE_Key,* FraDato, TilDato)
Tilhører	(*PARSELL_Key, KOMMUNE_Key,* Fradato, TilDato)

For å bruke våre gamle program med uforandret grensesnitt kan vi definere et view for aktuelle bostedsadresser. For å finne aktuelle bostedsadresser bruker vi tabellen **Steder** slik den er definert etter EER-diagrammet i figur 5:

```
CREATE VIEW
AKTUELLE_BOSTEDSADRESSER (gatenavn, gatenr, postnr, poststed)
AS
```

```
SELECT l.Navn, a.Nr, p.PostNr, p.StedsNavn
FROM LINJE l, ADRESSE a, POSTSTED p, Steder s, Adr1 a1, Adr2 a2
WHERE   s.AdresseType  = 'bosted'
    AND s.TilDato      > TODAY
    AND s.ADRESSE_Key  = a.Key
    AND a.Key          = a1.ADRESSE_Key
    AND l.Key          = a1.LINJE_Key
    AND TODAY          < a1.TilDato
    AND a.Key          = a2.ADRESSE_Key
    AND p.Key          = a2.POSTSTED_Key
    AND TODAY          < a2.TilDato;
```

Hvis AKTUELLE_BOSTEDSADRESSER ble brukt til å lage et snapshot ville kostnadene bli 4 seleksjoner og 5 foreninger i tabeller av noe størrelse. Hvis det er snakk om å finne én adresse vil foreningene erstattes med oppslag.

Hva har vi oppnådd?

Vi har fått en mer kompleks database, men den vanlige bruken av den er lite berørt da vi kan lage utsnitt som dekker over den ekstra kompleksiteten. Vi må lagre mer data, men samtidig er historiske data en del av den operative databasen. Vi har redusert datavolum ved at vi har unngått redundans. Databasen er konsistent ved at data er relatert til sin tid. Det viktigste er at vi ikke har mistet historiske data. Digital lagring gjør det lett å overskrive data når databasene er konstruert for å lagre kun aktuelle data.

Skal en ta vare på gamle data må vi ta vare på kopier av databasen. Etter hvert vil kopier bli glemt. Kopier må skrives om regelmessig, for eksempel hvert 3. år fordi magnetisering taper seg, og en må også følge med i format- og utstyrsutviklingen. Gamle formater og gamle lagringsenheter går ut, og data må skrives over på nye enheter for fortsatt å være tilgjengelig. Mange historiske kopier blir fort til sammen en stor datamengde, mye større enn om historiske data integreres i den operative databasen. Derfor vil en database som integrerer historiske data ikke kreve mer plass enn en enkel database med sine historiske kopier.

Vi kan drive detektivarbeid i gamle data: Det er ikke så vanskelig å finne ut "Hvor har Hans Eriksen, født 17. mai 1914 bodd?Det er overkommelig å skrive en SQL-setning som skulle gi noen svar:

```
SELECT p.PersonNr,l.Navn, a.Nr, o.PostNr, o.StedsNavn,s.FraDato,s.TilDato
FROM PERSON P, LINJE l, ADRESSE a, POSTSTED o, Steder s, Adr1 a1, Adr2 a2
WHERE   p.Fornavn      = 'Hans'      AND p.Etternavn    = 'Eriksen'
    AND DatoFødt       = '17-MAY-14' AND s.AdresseType  = 'bosted'
    AND s.ADRESSE_Key  = a.Key       AND a.Key          = a1.ADRESSE_Key
    AND a1.LINJE_Key   = l.Key       AND a.Key          = a2.ADRESSE_Key
    AND p.Key          = a2.POSTSTED_Key
    ORDER BY p.PersonNr ASC,s.FraDato ASC;
```

Denne SQL-setningen vil nok ikke gi oss alle svarene. Vi vil finne ut at postnummer er en forholdsvis ny oppfinnelse. Kanskje vil vi også oppdage inkonsistenser i databasen.

6 Konklusjon og videre arbeid

Vi har delt adresseproblematikken i flere adskilte moduler: Oppbygging, vedlikehold og ivaretaken av aktuelle og historiske adresser er et problemområde for seg. Adresser kan brukes i mange og uavhengige sammenhenger. En viktig kopling er selvfølgelig personer og adresser. Hvordan koplingen skal realiseres, hvordan den kan benyttes i mange

sammnehnger er et egen problemområde. Her er også personvernet et en meget viktig faktor.

En norsk adressedatabase er ikke stor, selv ikke om historiske data er inkludert i den operative basen. Hvis en database skal betjene alle behov stilles det krav til ytelse og driftssikkerhet. Beregning av ytelseskrav og eksperimentering med realiseringer vil være sentralt i et videre arbeid.

Referanser

[1] DIFI. Nasjonale felleskomponenter i offentlig sektor forslag til hvordan nasjonale felleskomponenter bør styres, forvaltes, finansieres og utvikles. Technical report, DIFI, 2010.

[2] Terje Eidsaunet. Personlig kommunikasjon, 2012. Posten Norge AS.

[3] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, Sixth Edition*. Pearson, 2011.

[4] Adresseveiledergruppen i Statens kartverk. Om tildeling og forvaltning av adresser etter matrikkellova - adresseveileder. http://www.statkart.no/?module=Files;action=File.getFile;ID=45443, 2012. Publiseres bare på web.

[5] Statens kartverk. http://www.kartverket.no/nor/matrikkel/matrikkellov-og-regelverk/adresse/norsk-adresseforum/.

[6] Bjørn Kristoffersen. *Databasesystemer*. Universitetsforlaget, 2009.

[7] Bring mail. http://www.bring.no/bring-mail/forside/adresser, 2012.

[8] Frode Wold. Personlig kommunikasjon, 2012. Posten Norge AS.