Høgskolen i Østfold

EKSAMEN

Emnekode: ITF10306	Emne: Databaser	
Dato: 12.05.15	Eksamenstid: 09.00 - 13.00.	
Hjelpemidler: Syntaksoversikt (vedlagt oppgaven)		Faglærer: Edgar Bostrøm
Oppgavesettet består av 4 sider inklusiv denne forsiden. Vedlegget består av 6 sider. Kontroller at oppgavesettet er komplett før du begynner å besvare spørsmålene. **Les nøye gjennom hele oppgavesettet før du begynner å besvare spørsmålene.**		
Sensurdato: 05.06.15 Karakterene er tilgjengelige for studenter på Studentweb senest 2 virkedager etter oppgitt sensurfrist, se www.hiof.no/studentweb		

Tema: Boligalarmer.

Det finnes mange firmaer som driver med boligalarmer. De installerer alarmsystemer med en sentral og et antall alarmpunkter (bevegelsessensorer, røkdetektorer etc.) i boliger. Hvis noen bryter seg inn, sendes det alarmsignal til firmaet, som så ringer avtalte telefonnummer for å sjekke om det er en "ekte" alarm eller ikke, og rykker ut hvis nødvendig. Det skal startes et nytt alarmfirma, som heter Falsk Alarm A/S. Du skal være med å lage et IT-system for disse.

Som nevnt vil en boligalarm innebære en alarmsentral og et antall alarmpunkter i hjemmet.

INSTALLASJON

Installasjonsnr	**EierID**	**Etternavn**	**Fornavn**	**Adresse**	**Postnr**	**Poststed**
7812	329	Os	Anne	Osveien 3	0231	Oslo

ALARMPUNKT

Installasjonsnr	**Punktnr**	**Komponentkode**	**Komponentnavn**	**Plassering**
7812	1	MK	Magnetkontakt	Inngangsdør i kjelleren
7812	2	MK	Magnetkontakt	Kjøkkeninngang
7812	3	BS	Bevegelsessensor	Hall, til høyre
7812	4	RØ	Røykdetektor	Barnerom
7311	1	MK	Magnetkontakt	.....

EierID, Etternavn og Fornavn gjelder den som eier installasjonen. Oftest er det samme person som bor i boligen, men det kan også være at boligen er leid ut, slik at den/de som bor der ikke er samme som eier. Adresse, Postnr og Poststed gjelder installasjonen. Plassering er en fritekst.

NB! Denne strukturen er muligens ikke ideell. Dette kommer vi tilbake til senere.

Oppgave 1. SQL. Tid: 1,0 time

a) Skriv ut en liste over de komponentene som Falsk Alarm A/S kan levere. Komponentkode og -navn skal være med, men bare en gang pr. komponent. Lista skal være sortert på komponentnavn.

b) Skriv en liste over installasjoner på postnr 1658 som har **enten MK eller BS** installert. Where-delen skal inneholde bl.a. komponentkode = "MK" OR komponentkode = "BS". Betingelsen om like Installasjonsnr skal finnes i where-delen. Lista skal inneholde Etternavn, Fornavn, Adresse.

c) Denne spørringen kan gjøres på mange måter. Blant annet kan komponentkode = "MK" OR komponentkode = "BS" skrives smartere, og vi kan bruke INNER JOIN i stedet for å bruke betingelsen om likhet i Where-delen. Skriv setningen over på nytt, men med disse to endringene.

d) En tredje løsning er å bruke SELECT ….. WHERE …. Komponentkode IN (SELECT …….). Skriv en ny versjon hvor du bruker dette!

e) Vi ønsker en liste over installasjoner som har **både RØ og MK**. Første forsøk inneholdt bl.a. WHERE Komponentkode = "RØ" AND Komponentkode = "MK". Hvorfor blir dette feil?

f) Lag en løsning som fungerer. Alle kolonner i Installasjon skal være med. (Vanskelig?). Finner du flere løsninger på denne, er det et pluss.

Oppgave 2. Mer SQL. Tid: 1,0 time

a) Lag en CREATE TABLE-setning for tabellen Alarmpunkt. Primær-og fremmednøkler skal være med. Du kan gjerne dele det i en CREATE TABLE- og en ALTER TABLE-setning.

b) Lag en INSERT-setning for å legge inn data om en ny bevegelsessensor i stua for installasjonsnr 7812. Dette blir punktnr 5.

c) Vi ønsker å gi et tilbud til alle som ikke har installert røykdetektorer enda. Lag en liste med alle de dette gjelder. Alle kolonnene fra installasjonstabellen skal være med.

d) Inntil 7 alarmpunkter er gratis, men noen har kjøpt flere punkter, og må betale ekstra månedsavgift for dette.
Lag en liste over de installasjoner (installasjonsnr, adresse, postnr, poststed) som har mer enn 7 alarmpunkter.

e) enten: Hvor stor andel (i prosent) har mer enn 7 alarmpunkter?
eller: Hvilken komponent er den mest solgte?

Oppgave 3. Normalisering. Referanseintegritet Tid: 1,0 time

NB! a) og b) er uavhengig av hverandre.

a) Normalisering:
   i) Analyser tabellene INSTALLASJON og ALARMPUNKT med hensyn på normalform (tegn gjerne determineringsdiagram som en del av forklaringen).
   ii) Som konklusjon på analysen skal du sette opp en normalisert versjon av tabellstrukturen.

(Får du til dette riktig, har du løst det meste av oppgave 4a)).

b) Referanseintegritet:
i) hva er det/hvorfor trenger vi det, hvordan håndteres det i relasjonsdatabaser.
ii) hva betyr RESTRICT/NO ACTION (forskjellen på disse kreves ikke) / CASCADE / SET NULL.

Oppgave 4. Datamodellering. Tid: 1,0 time.

(Oppgaven er ikke vanskelig, men det er viktig å få med seg alle detaljer. Blant annet er det viktig at primærnøkler og fremmednøkler er riktige).

a) Ta utgangspunkt i strukturen i oppgave 1 og endringer i oppgave 3a. Vi skal utvide modellen litt:

Vi må ha med adresse, postnr og poststed på eieren, i tillegg til at vi har det på installasjonen[1]. Ofte vil eieren bo samme sted som alarmen er, men det er ikke sikkert. For enkelhets skyld lager vi ikke noe felles "adresseregister" for eiere og installasjoner.

Falsk Alarm A/S har en kontaktperson-liste med forhåndsdefinerte telefonnr for hver installasjon. Når en alarm utløses, ringer de til disse i prioritert rekkefølge, slik at de kan sjekke om alarmen ble utløst ved et uhell, eller om det var en ekte alarm. Eksempel på noen data:

Installasjonsnr	Prioritet	Telefonnr	Navn	
7812	1	98778900	John	
7812	2	78998799	Johanne	
7812	3	14414414	Anne	osv.
7311	1	……..	……	

Tegn datamodell, idet du tar utgangspunkt i strukturen i oppgave 1, forbedringer som du har oppdaget ut fra oppgave 3 og utvidelsene over.

b) Dersom det må rykkes ut, skal det dokumenteres hvilke aktiviteter som er blitt gjort. For å kunne systematisere dette, er det laget faste aktivitetstyper, med Kode (1,2,3…) og Aktivitetsnavn, f.eks.

1	Oppdrag startet
2	Kontaktet politi
3	Undersøkt åstedet
4	Kontaktet brannvesenet
…….	
99	Oppdrag avsluttet

Det må dessuten finnes oversikt over ansattnr, etternavn og fornavn for de som kan rykke ut. Systemet kan dermed inneholde f.eks. at ansattnr 321 og 412 var med på utrykning 12.05.2015 kl.06:15 til installasjon 7812, og at de først startet oppdraget (kategori 1), så kontaktet brannvesenet (kategori 4), deretter undersøkte åstedet (kategori 3), så kontaktet brannvesenet igjen (kategori 4), før de avsluttet oppdraget (kategori 99).

En smart forretningsidé er å definere såkalt nabovarsling. Det betyr at andre som bor i nærheten, og som også har Falsk Alarm A/S som sin leverandør blir, nabovarsler for hverandre, f.eks. ved at de har tilgang til hverandres hus. De kan avtale selv hvem som skal være nabovarsler for hverandre. Siden det kun er andre som har Falsk Alarm som sin leverandør, behøver det ikke å være de som bor aller nærmest deg. Falsk Alarm ønsker imidlertid å ha en liste over hvem som er nabovarsler til hvem[2].

Utvid modellen med dette. Drøft gjerne upresisheter i oppgaven, og ulike løsninger du kan tenke deg.

---

[1] Vi regner med at det bare er en eier pr. installasjon, selv om det er en forenkling.

[2] Smartheten i dette er naturligvis at de som har Falsk Alarm som sin leverandør hjelper hverandre. Det kan gi færre utrykninger, samtidig som det er et salgsargument.)

SQL-syntaks – noen elementer

- Syntaksoversikten gjelder SQL2.
- Oversikten er ikke fullstendig og heller ikke helt presis, men er forhåpentligvis til hjelp.
- [ ] brukes om frivillige elementer, det er altså ikke med i SQL-språket.
- ¦ brukes som eller, det er altså ikke med i SQL-språket.
- { ….. } start, hhv. slutt, " ".
- < ….> brukes for å beskrive et språkelement. Disse beskrives eller er beskrevet tidligere i syntaksbeskrivelsen eller følger av det generelle mønsteret fra andre.
- **Fet skrift** brukes om faste språkelementer

Create / alter / drop table-setning

Create table

CREATE TABLE <tabellnavn> **(**<kommaseparert tabelldefinisjonsliste>**)**;

<kommaseparert tabelldefinisjonsliste>:

- liste med en eller flere elementer som er enten <kolonnedefinisjon> eller <skrankedefinisjon>
- hvis listen består av flere elementer, er det komma mellom disse.
- listen må ha minst en <kolonnedefinisjon>, har som regel også minst en <skrankedefinisjon>

<kolonnedefinisjon>:

- <kolonnenavn> <datatype> [**NOT NULL**] [**DEFAULT** <verdi>] , samt eventuell <skrankedefinisjon>, men uten (den første) kommaseparerte kolonnelisten.

<skrankedefinisjon> (det finnes noen flere enn de som er omtalt her)

- [**CONSTRAINT** <skrankenavn>] **PRIMARY KEY (**<kommaseparert kolonneliste>**)**
- [**CONSTRAINT** <skrankenavn>] **FOREIGN KEY (**<kommaseparert kolonneliste>**) REFERENCES** <tabell> (<kommaseparert kolonneliste>) [**ON UPDATE** <ref.oper.>] [**ON DELETE** <ref.oper.>]
- [**CONSTRAINT** <skrankenavn>] **UNIQUE (**<kommaseparert kolonneliste>**)**
- [**CONSTRAINT** <skrankenavn>] **CHECK (**<betingelse>**)**

<kommaseparert kolonneliste>:

- en eller flere kolonner. Hvis det er flere kolonner er disse adskilt med komma

<ref.oper.>: (dvs. referanseintegritetsoperasjon)

- {**RESTRICT** ¦ **NO ACTION** ¦ **CASCADE** ¦ **SET NULL**}

Alter table

ALTER TABLE <tabellnavn>
{**ADD** ¦ **DROP**} {[**COLUMN**][3] <kolonnedefinisjon> ¦ <skrankedefinisjon>};

Noen systemer mangler **DROP.**

Drop table

DROP TABLE <tabellnavn>;

[3] Skal være med for noen systemer, skal utelates for andre.

Select-setninger.

Select-setning uten gruppering

SELECT [**DISTINCT**] <kommaseparert resultatiste>
FROM <kommaseparert tabellliste>
[**WHERE** <betingelse>]
[**ORDER BY** <ordnet kolonneliste med sortering>];

<kommaseparert resultatliste>:

- kommaseparert liste, hvor hvert element er en av
  - en kolonne
  - en beregning m.m.
  - en select-setninger som returnerer en verdi for hver verdi av de andre i listen.
- et element kan gis et eget navn (alias). Mest vanlig for å gi resultatet av en beregning et folkelig navn. Skrives <kolonne> / <beregning> **AS** <NyttNavn>.

<kommaseparert tabelliste>:

- enkleste form er en enkelt tabell eller en liste av tabeller med komma mellom
- et element i denne kan også være alias, på formen <tabellnavn> [**AS**] <aliasnavn>. Alias må brukes hvis man trenger to eller flere benevnelser for samme tabell.
- elemenene i denne kan være **INNER JOIN**, **LEFT [OUTER] JOIN** eller **RIGHT [OUTER] JOIN** .
  Eks.: <tabell1> **LEFT OUTER JOIN** <tabell2> **ON** <tabell1>.<kolonne1> = <tabell2>.<kolonne2>
- inner, left og right join kan også nestes i flere nivåer.

<betingelse>:

- består av en eller flere <enkeltbetingelse> evt. med **AND** eller **OR** mellom.
- paranteser brukes på vanlig måte, **AND** binder sterkere enn **OR**

<enkeltbetingelse>:

- er et utsagn som, for en gitt rad i from-setningen, resulterer i enten sant eller usant.
- ofte <kolonnenavn> = <verdi>, men kan også være >, >=, <>, <=
- hvis du ikke bruker INNER / LEFT / OUTER JOIN er det viktig å ha med <tabell1>.PK = <tabell2>.FK
- **BETWEEN** <startverdi> **AND** <sluttverdi>
- søking i starten av en streng (trunkert søking): <kolonne> **LIKE** '<startstreng>%'
- søking i om delstrengen finnes i kolonnen: <kolonne> **LIKE** '%<delstreng>%'
- **NOT** brukes til å negere en enkeltbetingelse eller sammensatt betingelse. Binder sterkere enn **AND** og **OR**.
- <kolonne> **IS [NOT] NULL** brukes for å sjekke om en kolonne er NULL, evt. ikke er NULL.
- delspørringer med IN / NOT IN:
  <kolonne> **[NOT] IN**
  (SELECT <enkeltkolonne> .....)
- delspørringer med EXISTS / NOT EXISTS:
  [NOT] EXISTS
  (**SELECT** .....)
- **ALL** og **ANY** brukes på resultatet av en delspørring.
  - **ALL** er sann hvis alle i delspørringen oppfyller kriteriet. Usant hvis delspørringen er tom.
  - **ANY** er sann hvis noen (en eller flere) oppfyller kravet. Sant hvis delspørringen er tom. **SOME** er ekvivalent med ANY.
  - Tips: **WHERE** <kolonne> >= **ALL** (**SELECT** <kolonneliste> .....) er det samme som **WHERE** <kolonne> = (**SELECT max**(<kolonne>) .....)

<ordnet kolonneliste med sortering>:

- som kolonneliste, men i sorteringsprioritet, og hver kolonne kan etterfølges av **ASC** eller **DESC**.
- hvis det ikke oppgis sortering, blir sorteringen i stigende rekkefølge.

Select-setning med gruppering / aggregering

For det som er felles for alle select-setning henvises det til 0.

SELECT <kommaseparert resultat- eller aggregeringsliste>
FROM <kommaseparert tabelliste>
[**WHERE** <betingelser>]
[**GROUP BY** <kommaseparert resultatliste>]
[**HAVING** <betingelse for gruppe>]
[**ORDER BY** <ordnet kolonneliste med sortering>];

<kommaseparert resultat- eller aggregeringsliste>:

- NB! hvert element er enten et element fra group by-listen eller en <aggregeringsfunksjon>.

<aggregeringsfunksjon>:

- {**count(*)**|**count**(<kolonne>)|**sum**(<kolonne>)|**max**(<kolonne>)|**min**(<kolonne>)|**avg**(<kolonne>)| mfl.}
- <kolonne> kan også være en beregning
- noen systemer har også mulighet for **count** (**distinct** <kolonne>)), teller altså opp antall ulike.
- hvis vi ikke har med **GROUP BY** gjelder aggregeringen for hele tabellen

<betingelse for gruppe>:

- bare aktuelt dersom man har **GROUP BY**.
- betingelse som gjelder gruppen, innholder ofte en aggregeringsfunksjon, f.eks. count(*) > 1, sum(<kolonne>) = (select sum(……)
- kan inneholde **AND**, **OR**, **NOT** osv., på samme måte som <betingelse>

INSERT / UPDATE / DELETE

INSERT-setning

INSERT INTO <tabell> [(<kommaseparert kolonneliste>)]
{ **VALUES** (<kommaseparert verdiliste>) | <select-setning> } ;

UPDATE-setning

UPDATE <tabell>
SET <kommaseparert kolonne/verdi-liste>
[**WHERE** <betingelse>];

- I noen systemer kan <tabell> i stedet være en begrenset form for <kommaseparert tabellliste>

<kommaseparert kolonne/verdi-liste>:

- hvert element består av <kolonne> = <konstant> eller <kolonne> = <beregninget verdi, f.eks. på grunnlag av tidligere verdi>
- oftest bare en slik kolonne/verdi-kombinasjon, men kan være flere.

DELETE-setning

DELETE
FROM <tabell>
[**WHERE** <betingelse>];

Create / drop view

Create view

CREATE VIEW <utsnittsnavn> [(<kommaseparert kolonneliste>)]
AS
<select-setning>;

- kolonnelisten er nødvendig hvis det ikke er fullt samsvar mellom kolonnenavn i select-setningen og utsnittet.

Drop view

DROP VIEW <utsnittsnavn>;

Indekser

CREATE [**UNIQUE**] **INDEX** <indeksnavn> **ON** <tabell> (<ordnet kolonneliste med sortering>);
DROP INDEX <indeksnavn>;

Noen systemer har andre mekanismer i tillegg.

Gi / frata rettigheter til tabeller, laging av brukere, databaser m.m.

GRANT <rettigheter> **ON** <tabell el.l.> **TO** <bruker/gruppeliste> [**WITH GRANT OPTION**];
REVOKE [<rettigheter> ¦ **GRANT OPTION**] **FROM** <tabell el.l.> **TO** <bruker/gruppeliste>;

<rettigheter>:
kommasepartert liste med en eller flere av **SELECT**, **INSERT**, **UPDATE** (<kolonnenavn>), **DELETE**, **ALL** m.fl..

<bruker/gruppeliste>:
kommasepart liste med en eller flere brukere eller grupper. I tillegg finnes ofte noen standardgrupper, som PUBLIC og DBA.

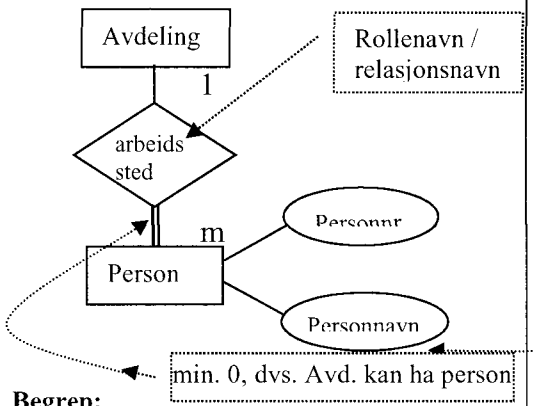
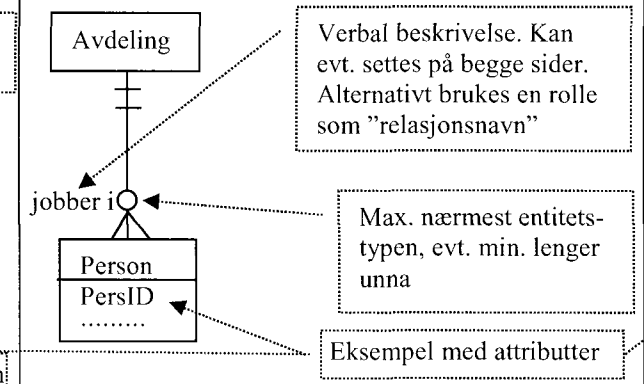
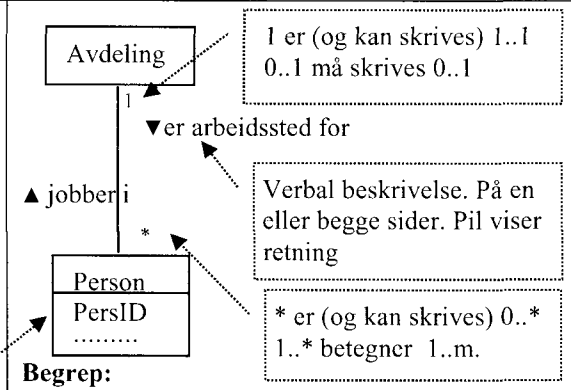
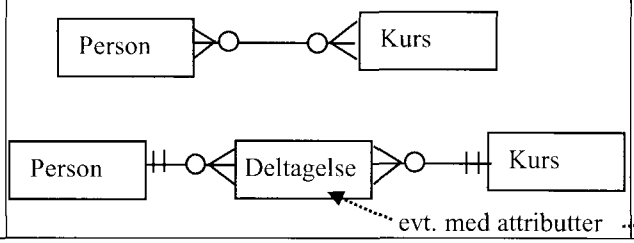
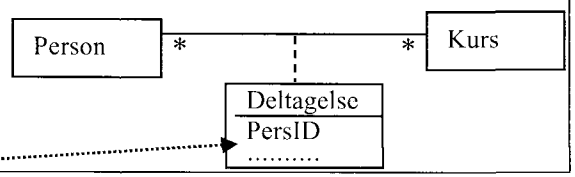
Noen variasjoner og begrensninger fra et system til et annet.

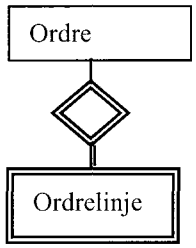
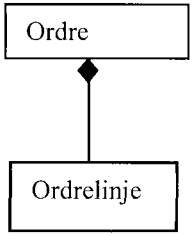
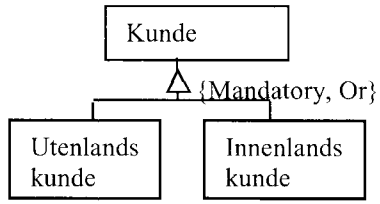
Annet:

Muligheter for å lage / ta bort brukere etc., **CREATE USER**, gjerne sammen med **IDENTIFIED BY** <passord>. Tilsvarende **DROP USER**.
Muligheter for å lage nye databaser, **CREATE DATABASE** <databasenavn>
I noen systemer: laging av typer, domener etc.

Datamodellnotasjon i 3 dialekter: Chen, kråkefot og nedskalert UML.

En del detaljer og variasjoner er utelatt.

	Chens ER	**Kråkefot**	**nedskalert UML**
Grunnleggende. For alle dialekter: • attributter kan tas med eller utelates (avh. av hvor langt i prosessen og hvor stor modellen blir) • ditto for domener/datatyper • det finnes varianter for å vise min./max.	 **Begrep:** Entitet(styper), relasjon(styper), attributter.	 **Begrep:** Entitet(styper), relasjon(styper), attributter.	 **Begrep:** Entitet(styper) eller objektklasser, (multiplisitets)assosiasjoner, attributter.
Er repetisjoner tillatt?	Ja, på konseptuelt nivå	Nei – splittes ut i egne entitetstyper	Ja, på konseptuelt nivå
Eventuelle primær- og fremmednøkler	Tas gjerne ikke med	Hvis det tas med: Markeres f.eks. med primærnøkkel: understreking fremmednøkkel: prikket linje, *, el.l.	Hvis det tas med: markeres gjerne med {PK} hhv. {FK} bak attributtnavnet. Hvis (del av) begge deler: {PK,FK}
Entitetisering	Kan gjøres, men vanligvis settes det bare på attributter på relasjonen. Bare nødvendig ved 2. ordens entitetisering (entitetisering av noe som allerede er entitetisert eller kunne vært entitetisert).	Gjøres dersom "relasjonen skal inneholde attributter". 	Kan gjøres, men bare nødvendig ved det som ellers ville vært 2. ordens entitetisering. Assosiative entitetstyper m/ attributter kan legges på: 

n-ære relasjonstype / assosiasjoner (n >2)	Innebygdt i notasjonen, ingen forskjell på binære og n-ære.	Evt. entitetisering gjøres først, deretter henges nye entitetstyper på den nye entitetstypen.	Bruk  for å knytte dem sammen. Assosiative entitetstyper kan brukes
Avhengighet av andre entitetstyper (en entitet er avhengig av eksistensen av en annen entitet)	 kalles svak entitet / weak entity	Markeres ved at fremmednøkkelen er en del av primærnøkkelen (på mange-siden)	 kalles komposisjon. Finnes også en mindre sterk kobling som kalles aggregering (markeres med ◇ i stedet for ◆) .
Arv	Finnes ikke, må i tilfelle beskrives som 1:1, men gir ikke egentlig arv.	Finnes ikke, må i tilfelle beskrives som 1: 1, men gir ikke egentlig arv.	 I tillegg: kan beskrive kombinasjoner av mandatory/optional og om en overordnet kan kobles til max. 1 eller til flere underordnede (or eller and), se over. Kan også være arv med"ett barn", f.eks. bare "Kunde" og "Utenlandskunde".
Forhold til normalisering	Må evt. gjøre utsplittinger av repetisjoner	Er normalisert	Må gjøre evt. utsplittinger av repetisjoner
Overføring til relasjonsdatabaser	Overføres til kråkefot el.l. først (fra konseptuelt til logisk nivå) Alternativt: Legg på primær- og fremmednøkler Evt. repetisjoner må tas bort. Entitetstyper blir til tabeller. Relasjoner som gjelder 1:m tas bort, relasjoner som gjelder m:m blir egne tabeller.	Evt. mange-til-mange må entitetiseres. Ellers: entitetstyper blir til tabeller	Evt. repetisjoner må tas bort. Entietstyper/objektklasser blir til tabeller. Assosiasjonsattributter i m:m blir egne tabeller, andre m:m entitetiseres. Høyere ordens relasjonstyper blir til tabeller. Arv må omformuleres (flere alternativer finnes, ingen er helt gode). Dersom man bruker ORDB-utvidelser i systemer som har dette, kan arv implementeres.