NY OG UTSATT EKSAMEN

Emnekode: ITF10306	Emne: Databaser
Dato: 08.01.13	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 gjennom hele oppgavesettet før du starter.	
Sensurdato: 30. januar 2013 Karakterene er tilgjengelige for studenter på studentweb senest 2 virkedager etter oppgitt sensurfrist.	

Tema: *Studieadministrasjon.*

Vi skal se på et studieadministrativt system for en høgskole. Det som tas opp her, er naturligvis svært begrenset og forenklet.

Høgskolen er delt opp i flere avdelinger, under har vi eksempeldata bare fra to av de. Hvert emne er knyttet til bare en avdeling, og undervises bare en gang i året, og det er eksamen bare en gang i året for hvert emne[1]. Samme emne (dvs. med samme emnekode, -navn, studiepoeng osv.) undervises gjerne i mange år, og undervises ved samme avdeling hele tiden. Stp er en forkortelse for studiepoeng. Studant er antall studenter på kurset dette året. Lærerkoder er initialer på den eller de lærerne som er involvert, adskilt med skråstrek. Hvilke lærere som er involvert kan naturligvis variere fra år til år.

Et utkast til noen av tabellene følger under. Primærnøkler/identifikator er understreket.

STUDENT

Studentnr	**Etternavn**	**Fornavn**
111	Jensen	Jens
345	Hansen	Hanne

EMNE

Emnekode	**Årstall**	**Emnenavn**	**Stp**	**Avdkode**	**Avdnavn**	**Studant**	**Lærerkoder**
INF160	2011	Databaser	10	IT	Informasjonsteknologi	78	EBO/POB/THN
NOR150	2012	Norsk 1	15	LU	Lærerutdanning	17	KPI
INF160	2012	Databaser	10	IT	Informasjonsteknologi.	53	POB/EBO
INF120	2012	OOP	10	IT	Informasjonsteknologi	40	BS/POB

[1] NB! Ved HiØ er det noen unntak på dette, men hold deg til teksten som er gitt i oppgaven!

EKSAMEN

Studentnr	Emnekode	Årstall	Eksamensdato	Karakter
111	INF160	2011	02.05	F
111	INF160	2012	12.05	C
345	INF160	2012	12.05	B
345	INF120	2012	18.05	B

Oppgave 1. SQL. Tid: 1 time.

a) Lag en liste med alle emner i avdeling IT, samt emner ved avdeling LU med emnekode som begynner på NOR. Alle kolonner i tabellen Emne skal være med.

b) Lag en liste over alle eksamenene som student 111 har tatt, med de siste eksamenene først. Lista skal inneholde studentnr, etternavn, fornavn, emnekode og –navn, samt eksamensår, -dato og karakter.

c) Lag en liste over studenter som ikke har vært oppmeldt til noen eksamen i 2011.

d) Lag en liste over studenter som har tatt en eller flere eksamener både ved IT og ved LU.

e) En person som har minst 180 studiepoeng (stp) kan på visse vilkår få utstedt vitnemål som bachelor. Lag en liste med studentnr, for- og etternavn og antall studiepoeng på de som har minst 180 stp.

f) Lag en liste med studentnr, etternavn og fornavn for studenter som har fått karakteren "A" på alle sine eksamener i 2011.

Oppgave 2. Mer om tabellene. Utsnitt/views. Tid: 1 time.

a) Lag en CREATE TABLE-setning for å definere tabellen Eksamen, inkl. primærnøkkelen.

b) Det bør defineres opp referanseintegritet mellom Eksamen og Emne. Lag en ALTER TABLE-setning for å gjøre dette. Benytt CASCADE for endring, RESTRICT for sletting.

c) Lag en INSERT-setning for å legge inn student nr. 543, Otto Ottosen, i Student-tabellen.

d) Lag et utsnitt som tilsvarer spørringen i 1e. Det holder med å skrive CREATE VIEW-delen, du behøver ikke å gjenta SELECT-delen fra 1e.

e) Hva brukes utsnitt/views til? Lag en punktvis oversikt, med korte kommentarer for hvert punkt..

f) Hva vil det si at et utsnitt er oppdaterbart?

Oppgave 3. Normalisering. Tid: 1 time.

Merk: løsningen av denne oppgaven vil danne en stor del av grunnlaget for modellen i oppg. 4. Oppgave 3c og d er nok litt komplisert, men er samtidig svært nyttig for å få en korrekt datamodell. På den annen side vil du kunne lage en riktig datamodell i oppgave 4 via sunn fornuft selv om du ikke helt har fått til oppgave 3. Se derfor oppgave 3 og 4 i sammenheng!

Vi antar at alle kolonnene, bortsett fra kolonnen lærerkoder i Emne er atomiske. Dermed er tabellen Emne unormalisert.

a) - Forklar begrepet atomisk i en databasesammenheng.
   - Det er likevel slik at det i en del tilfelle kan være tvil om det er naturlig å betrakte en kolonne som atomisk eller ikke. Gi eksempler på dette fra kolonner i tabellene over, og begrunn kort hvorfor det kan være tvilstilfelle.

b) Hvilken normalform er tabellen Eksamen på? Svaret skal begrunnes. Lag gjerne determineringsdiagram.

c) Normaliser tabellen Emne til 1 NF, og tegn deretter determineringsdiagram (evt. lag en liste med alle determineringer for denne). Både determineringer som er OK og eventuelle som ikke er ok skal være med.

d) Drøft tabellen Emne videre med hensyn til normalisering. Dersom den ikke er normalisert fullt ut, skal du vise normaliseringen trinn for trinn fram til BCNF.
(Hint: det er bl.a. viktig å skille mellom emnet som sådan og hvert år som emnet undervises).

Oppgave 4. Datamodellering. Tid: 1 time.

Du trenger ikke å vise minima, men verb/roller skal tas med der er det er nødvendig for å klargjøre modellen. a) og b) kan godt tegnes i samme modell.

a) Tegn opp en datamodell med utgangspunkt i tabellene i oppgave 1, men ta hensyn til endringene du har gjort i oppgave 3, og eventuelle andre endringer som du ser burde gjøres i den opprinnelige strukturen. Forklar hvorfor du har gjort de endringene du har gjort.

b) Vi ønsker å gjøre følgende utvidelser:
   - Det må finnes en oversikt over navnene på de ulike lærerne, ikke bare initialer.
   - Selv om et emne har flere lærere, er det alltid slik at en av lærerne er hovedansvarlig for kurset.
   - Det er slik at enkelte emner bygger på hverandre, helt eller delvis. Det er laget koder og navn på ulike "grader av avhengighet": F = forutsetter (betyr at man må ha bestått et tidligere emne for å ta dette emnet), B = bygger på, L = lurt å ha tatt. Disse kodene skal finnes i systemet, og for hvert emne som bygger på et annet emne skal vi ha med "grad av avhengighet". Eksempel på slike avhengigheter:

Emnekode	Bygger_på	Avhengighetsgrad
INF230	INF160	F
INF230	INF140	L
INF350	INF230	L

   - Som kjent har hver avdeling flere studier, f.eks. Bachelor i økonomi, Bachelor i Informasjonssystemer, Bachelor i Markedsføring. Systemet må ha en oversikt over disse studiene, inkl. en forkortelse for hver av disse (f.eks. ØKØ = Bachelor i økonomi). De fleste studentene er opptatt kun til ett slikt studium, men det er ingen ting i veien for at de kan være opptatt til flere. Systemet skal inneholde hvilke(t) studium/studier hver student er opptatt, inklusive hvilket år hun/han ble opptatt på studiet/ene.

BONUS-OPPGAVE (som betyr at du ikke blir trukket dersom du ikke løser den, men du kan få litt pluss dersom du klarer å løse den).
Et emne kan tilhøre flere studier (f.eks. hører emnet "Databaser" inn under studiene "Informasjons-systemer", "Informatikk - design og utvikling av IT-systemer" og "Dataingeniør").Vi tenker oss at vi innfører en regel om at du bare kan ta eksamen i emner som hører til det studiet/de studiene som du er tatt opp til. Utvid med et forslag som gjør at du kan kontrollere dette, og drøft eventuelle vanskeligheter med dette forslaget. Denne delen bør tegnes og kommenteres uavhengig av resten av denne oppgaven.

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**][2] <kolonnedefinisjon> ¦ <skrankedefinisjon>};

Noen systemer mangler **DROP.**

Drop table

DROP TABLE <tabellnavn>;

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

Select-setninger.

Select-setning uten gruppering

SELECT [DISTINCT] <kommaseparert resultatliste>
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.

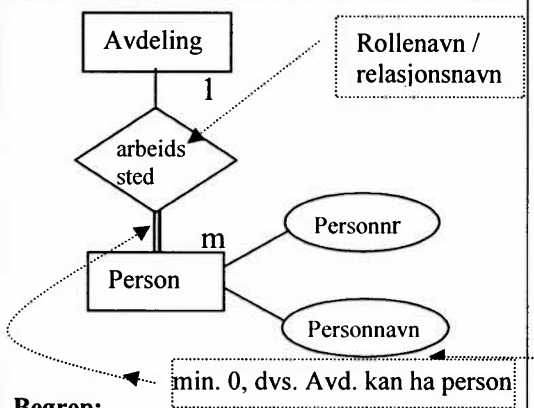
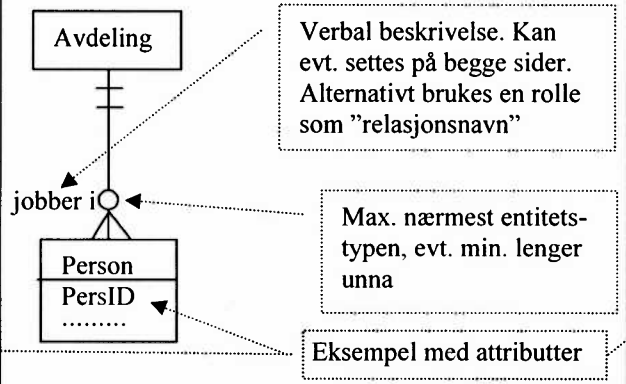
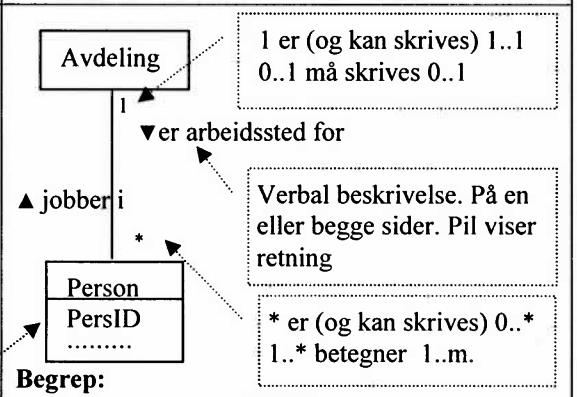
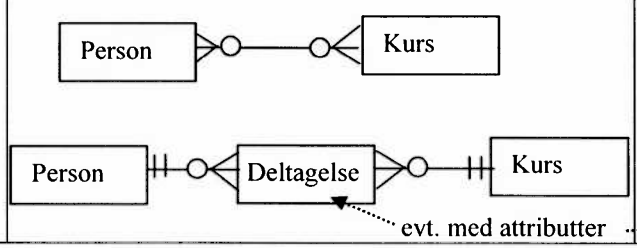
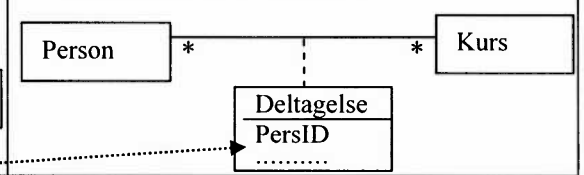
Noen variasjoner og begrensninger fra et system til et annet.

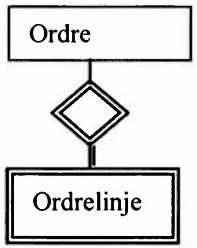
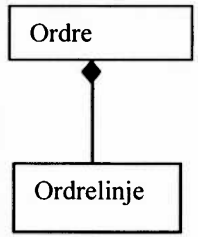
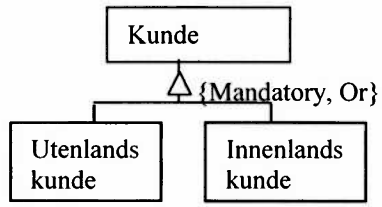
Annet:

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.