Høgskolen i Østfold

EKSAMEN

Emnekode: ITF10306	Emne: Databaser
Dato: 08.01.15	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 gjennom hele oppgavesettet før du begynner å besvare spørsmålene.

Sensurdato: 29. januar 2015

Karakterene er tilgjengelige for studenter på Studentweb senest 2 virkedager etter oppgitt sensurfrist, se www.hiof.no/studentweb

Trimkonkurranser er i ferd med å bli «big business». Du er derfor blitt spurt om å lage et system eller en app som kan brukes av individuelle og grupper som skal holde styr på trimaktiviteter[1]. Hver person som skal være med på systemet må registrere seg, foreløpig tenker vi at vi kun har med **PERSON, med PersonID (primærnøkkel, kan f.eks. være en teller), brukernavn (som et fritt valgt navn/nick), etternavn og fornavn.** Man har dessuten ulike aktivitetstyper, noen eksempler er vist under.

AKTIVITETSTYPE

Typenr	**Typenavn**	**Faktor**
1	Situps	200
2	Skritt	1
3	Skog/fjell	117
…	Aerobics	110
..	Billiard	60
…	……..	….

AKTIVITET

PersonID	**Dato**	**Typenr**	**Antall**
1001	2015.01.03	1	10
1001	2015.01.03	3	100
1001	2015.01.03	2	3000
1001	2015.01.04	3	200
1003	2015.01.03	1	100
1002	…	…	…

Primærnøkler er som vanlig understreket, og typenr danner primær/fremmednøkkelkoblingen.

Med **faktor** menes i de fleste tilfelle poeng pr. minutt, f.eks. får du 200 poeng for hvert minutt du tar situps. **Antall** er antall minutter du trimmer for denne aktivitetstypen for denne datoen og personen. Det eneste unntaket her, er at man for skritt registrerer antall skritt, f.eks. 2000 skritt for PersonID 1001 den 2015.01.03. Dermed er faktor satt til 1 for skritt (dvs. vanlig gåing/gange).

Dermed blir f.eks. totalt antall poeng for person 1001 på datoen 2015.01.03:

Situps	faktor 200 x 10 minutter	= 2000	poeng
Skritt	faktor 1 x 3000 stk	= 3000	poeng
Skog/fjell	faktor 117 x 100 minutter	= 11700	poeng

Totalt blir dette 16700 poeng for denne datoen.

Oppgave 1. SQL Tid: 1 time.

a) *(Oppvarming)*
Lag en spørring som skriver ut navn og faktor på alle aktivitetstyper som finnes i systemet, med de som har høyest faktor (dvs. de «mest slitsomme») først. Typenavn og faktor skal være med.

b) Skriv ut en liste over aktiviteter for PersonID 1001, men bare aktivitetene Aerobics og aktiviteter som har faktor på minst 150. Dato, Typenavn, Antall og Poeng skal være med.

Tips: spørringen vil antagelig inneholde
```
SELECT …….Antall, Faktor * Antall as Poeng
FROM …
```

[1] Et lignende system finnes f.eks. på dytt.no, men du skal selvsagt lage noe mye bedre☺. Takk til foreleserens kone, som har gitt opplysninger om hvordan dette systemet virker. Foreleseren burde nok også ha deltatt ☹.

c) Det kan finnes personer som har meldt seg på, men ikke registrert aktivitet. Skriv i tilfelle ut for- og etternavn på disse.

d) Det spennende er jo hvem som har gjort det best i konkurransen. Skriv ut en liste med PersonID, Fornavn, Etternavn og sum poeng pr. person, sortert slik at de med høyest sum poeng kommer først.
Personer som har under 100000 i sum poeng tas ikke med.

e) Hvem er vinneren(e)? (Husk, det kan være flere med samme sum poeng). Fornavn og etternavn skal være med.

f) *(Enkelt).* For Billiard vil man endre faktoren til 80. Skriv en SQL-setning som gjør dette.

Oppgave 2. Databasestruktur Tid: 1 time.

a) Definer tabellen AKTIVITET, inklusive primær- og fremmednøkler. Bruk gjerne flere setninger (CREATE TABLE/ALTER TABLE).

b) Det er vanlig å sette på referanseintegritetsformer (RESTICT/NO ACTION/CASCADE/SET NULL). Forklar hva dette er. Kort: hva ville du ha brukt i denne strukturen?.

c) Analyser tabellen AKTIVITET skritt for skritt for å finne ut hvilken normalform denne er på.

d) Utsnitt / view:
- Forklar kort hva det er.
- Det er en ting som er spesielt for denne oppgaven, som gjør at det ville vært en fordel å ha brukt utsnitt. Hva?

Oppgave 3. Datamodellering Tid: 1 time.

a) Et slikt system bør bestå av mye mer enn det vi har sett på nå. Det bør bl.a. være slik at ulike firmaer kan registrere seg, og at de kan sette i gang flere konkurranser (f.eks. 15.10.14-15.12.14, så en ny 10.01.15-10.02.15). Konkurransen gjelder kun ett firma. Vi må altså ha med firma (med firmanr, navn, postnr og poststed), konkurransene for hvert av firmaene (konkurransenr, fra- og til.-dato), samt hvem som er ansvarlig for denne konkurransen. Data om vedkommende skal finnes i tabellen PERSON. Det er forresten slik at det kan lages konkurranser uavhengig av firmaer. Dette skal markeres i modellen.

Dessuten skal personene lage grupper i hver konkurranse, og slik at gruppene skal kunne konkurrere seg i mellom. Vi må dermed ha med et gruppenr (løpenummer) og et (morsomt) navn på gruppa, selvsagt knyttet til hvilken konkurranse denne gruppa er for. I tillegg til hvilke deltagere som er med i gruppa, skal det utpekes en gruppeleder for hver gruppe, og dette skal finnes med i systemet. Her antar vi at personen må registrere seg på nytt (med ny PersonID hvis hun/han skal være med i flere konkurranser).

Ta utgangspunkt i det som er sagt tidligere i oppgavesettet og i denne deloppgaven, og tegn datamodell.

b) *(fortsett helst på samme datamodell).*

Det ble foreslått å lage mulighet for å registrere puls, vekt, blodsukker, blodtrykk m.m. (dette er en fast liste, men som skal kunne utvides) pr. dag for hver deltager. Ta med dette i modellen.

Som sagt i innledningen er dette «big business», og vi håper at en rekke **leverandører av sportsutstyr** m.m. kommer til å knytte seg til dette, og gi rabatt. Det er da om å gjøre for de firmaene som deltar i de ulike konkurransene å gjøre avtaler med ulike leverandører for ulike varetyper, slik at deltagerne kan kjøpe varer på slike rabattavtaler. To eksempel på slike rabattavtaler kan være at Borregaard A/S i konkurransen som går 15.02.15 til 15.03.15 har fått forhandlet seg fram til 25% rabatt på alle sportsklær fra Bergans og 20% på alt sportsutstyr fra G-sport. Leverandørene (med nummer, navn og URL), de ulike varetypene (kode og navn), samt hvilke rabatter det er gitt av hver leverandør for hver konkurranse for de ulike varetypene skal være med.

c) ***Bonus*** *(teller positivt hvis du har løst denne, men ikke negativt hvis du ikke har løst den):*
Det bør være slik at hvis en person har registrert seg, så kan hun/han meldes på så mange konkurranser vedkommende ønsker, og at vedkommende ikke behøver å være knyttet til noen gruppe i det hele tatt. Lag en skisse på endringer som må gjøres i modellen, og eventuelle vanskeligheter som det medfører. En god diskusjon av dette er nødvendig hvis dette skal gi bonus.

Oppgave 4. Indekser Tid: 1 time.

a) Forklar hva indekser er og hva de brukes til i en relasjonsdatabase.

b) Lag en SQL-setning for å lage en unik indeks på Brukernavn i PERSON.

c) Hva er et B-tre?

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 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 **Feil! Fant ikke referansekilden.**.

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.

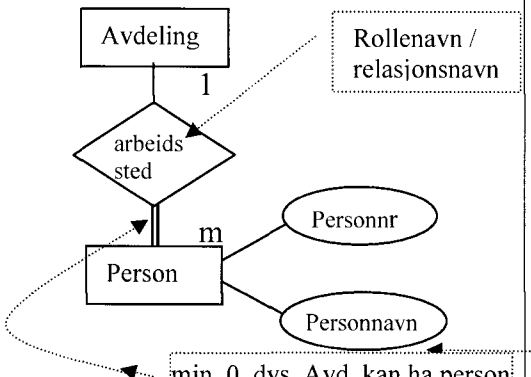
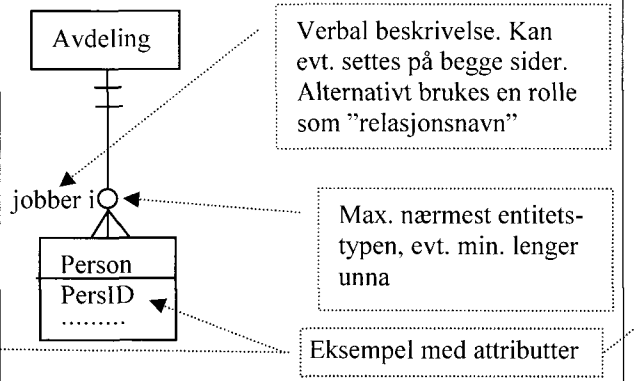
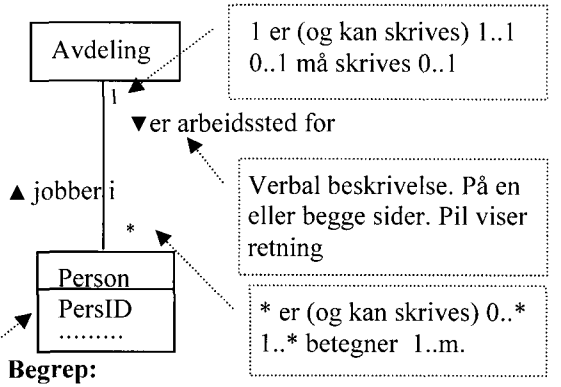
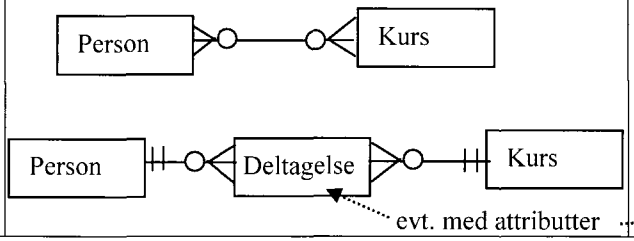
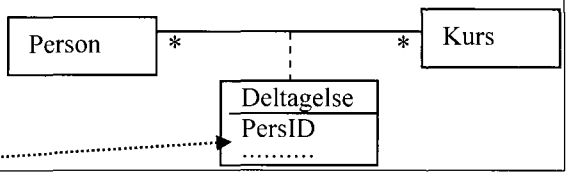
Noen variasjoner og begrensninger fra et system til et annet.

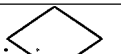
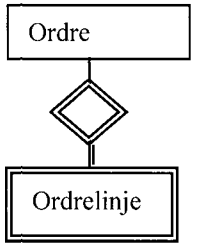
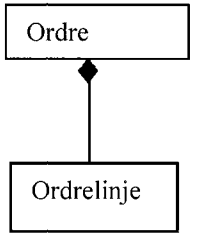
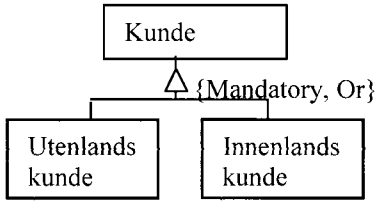
Annet:

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.