Høgskolen i Østfold

EKSAMEN

Emnekode: ITF10306	**Emnenavn:** Databaser
Dato: 23.mai 2018	**Eksamenstid:** 4 timer
Hjelpemidler: Syntaksoversikt (vedlagt oppgaven)	**Faglærer:** Edgar Bostrøm

Om eksamensoppgaven og poengberegning:
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. Vedlegget finnes ved siden av oppgaveteksten, og inneholder tabellene som skal brukes i oppgaven, samt syntaksoversikt.

Du kan bruke et eget ark der du ønsker å gjøre å lage illustrasjoner, modeller etc. Når du gjør det, skriver du **SE PAPIRARK** hver gang det er aktuelt, slik at du er sikker på at sensor ser hva du har skrevet i tillegg til det som leveres elektronisk.

Tips: **vær svært nøye med å lese oppgaveteksten og kontrollere løsningen på hver av deloppgavene!**

Sensurfrist: 13. juni 2018
Karakterene er tilgjengelige for studenter på Studentweb www.hiof.no/studentweb

Database for sykehus[1]

Det meste av oppgaven gjelder et system for sykehusdata. Opplegget er naturligvis svært forenklet. Oppgavesettet må ses i sammenheng, slik at det er opplysninger i en tidligere oppgave som du også kan få bruk for i en senere oppgave.

Først og fremst skal systemet inneholde informasjon om innleggelser og diagnoser. Diagnosene klassifiseres etter en internasjonal standard, ICD10. Et par eksempler på slike koder:

DIAGNOSETYPE

Dkode	Dnavn
8332	Brudd på skinneben og leggben, …..
0066	Amøbesår i huden
….	…..

Alle pasienter som legges inn på et sykehus får et pasientidentifikasjonsnummer, PIN. Dette beholdes også dersom pasienten legges inn flere ganger. Gangnr er et løpenummer som økes med en for hver gang denne pasienten legges inn (innleggelse nr. 1, 2, 3 osv. for denne pasienten). Fødselsnr er det vanlige 11-sifrede fødselsnummeret som brukes i Norge (altså sammensatt av fødselsdato, persondel og kontrollsifre), og lagres som en streng. Fødselsnr er tatt med for å kunne rapportere inn til NAV m.m. Det regnes som innleggelse bare hvis man har vært på sykehuset i minst en natt. Dkode er diagnosekoden fra tabellen over. Andre, opplagte kolonner som adresse etc. er droppet her.

INNLEGGELSE

PIN	Gangnr	Personnavn	Innleggelsesdato	Utskrivningsdato	Dkode	Fødselsnr
43511	1	Bø, Oda	20.04.18	22.04.18	0066	14049712122
….	….	….	….	….	….	……..

Primærnøkler er som vanlig understreket. NB! Denne strukturen er ikke ideell, se oppgave 2.

[1] Takk til mine gode venner, legene Berit og Knut Hjortaas, for informasjon om sykehus, innleggelser m.m.

Oppgave 1. SQL. Tid: 1 time.

a) Pasient 43511 blir langt inn igjen 30.04.18, med samme diagnose. Lag en insert-setning for å legge inn disse dataene i INNLEGGELSE.

b) Skriv ut Dkode og Dnavn for alle diagnosene som handler om skinneben. Ordet skinneben skal altså finnes i diagnosenavnet, men behøver ikke å komme først eller sist i dette navnet. Listen skal være sortert på diagnosenavn.

c) Skriv ut hele «sykehistorien» til pasientene med fødselsnr 12018411111 og 25078522222. Utskriften skal inneholde PIN, gangnr, navn, innleggelsesdato, utskrivingsdato, diagnosekode og diagnosenavn. Listen skal sorteres på økende PIN og minkende dato (dvs. siste innleggelser først).

d) Skriv ut en liste over ubrukte diagnoser, dvs. som finnes i diagnosetabellen, men ikke i innleggelsestabellen. Diagnosekode og –navn skal være med.

e) En del pasienter som kommer inn, får samme diagnose mange ganger. Skriv ut en liste over dkode, fødselsnr, navn og antall ganger for de pasienter som har fått samme diagnose minst 5 ganger. (Noen kan også ha flere diagnoser som de har blitt lagt inn for minst 5 ganger.)

f) Det bør være et absolutt krav at en pasient er utskrevet før vedkommende blir innskrevet igjen. (Det er imidlertid ok om man blir utskrevet tidligere på dagen og måtte innskrives igjen senere på dagen.) Skriv ut eventuelle brudd på dette. (Tips: alias).

g) Hva er den mest vanlige diagnosen? Diagnosekode, -navn og antall ganger denne diagnosen har blitt brukt skal tas med.

Oppgave 2. Datadefinisjon og normalisering. Tid: 1 time.

NB! I a), b) og c) skal du forholde deg til tabellene gitt i oppgave 1, selv om de burde vært bedre strukturert, jf. deloppgave d), e) og f).

a) Definer tabellen DIAGNOSETYPE (dvs. med en create table-setning).

b) For tabellen INNLEGGELSE kan det finnes flere kandidatnøkler. Drøft hvilke kandidatnøkler denne tabellen har.

c) Definer tabellen INNLEGGELSE.
Minst en av kandidatnøklene i tillegg til primærnøkkelen skal defineres.
Fremmednøkkel skal også defineres.
Definisjonen kan gjerne deles i flere deler.

---

d) Ta for deg tabellen INNLEGGELSE. Kan det finnes brudd på 1NF i denne? Begrunn svaret.

e) Anta at tabellen INNLEGGELSE er på 1NF. Forklar hvorfor den bryter med 2NF, og normaliser deretter strukturen til 2NF. Dersom det ligger spesielle forutsetninger i den

analysen du gjør, bør dette påpekes.

f) Hvilken normalform er strukturen på etter denne normaliseringen?
Det kreves begrunnelse, gjerne i form at et determineringsdiagram, men liste over determineringer kan også benyttes.

(Eventuelt determiningsdiagram kan tegnes med tegneverktøyet eller på eget ark).

Oppgave 3. Datamodell. Tid: 1 time.

Modellen tegnes på eget ark. Attributtliste kan eventuelt skrives her i stedet for sammen med modellen. Kommentarer o.l. kan skrives her.

Vi skal utvide systemet litt. Vi begrenser fremdeles systemet til å gjelde pasienter som er innlagt, slik at vi ikke tar med forhold som angår poliklinikk eller annen kortbehandling.
Maksima, identifikator/primærnøkler og fremmednøkler skal være med. Minima, verb/roller bør være med, og må være med dersom det er nødvendig for å forstå modellen. Del a) og b) kan tegnes i samme modell.

a)
Systemet må inneholde en felles oversikt over alle leger (både sykehusleger, allmenpraktiserende og andre leger). Denne oversikten skal inneholde en legekode/nr, legenavn og adresse. Når pasienter legges inn, er det på grunnlag av en henvisning fra en lege (som regel sin fastlege, i denne sammenhengen kalt/i rolle som henvisende lege). Noe av det første som skjer er dessuten at pasienten blir undersøkt av en lege (som regel en sykehuslege, i denne sammenhengen kalt/i rolle som innleggende lege). For hver innleggelse skal altså systemet ha med både hvem som er henvisende lege og hvem som er innleggende lege. Innleggelsen skjer dessuten til en bestemt avdeling (her skal vi ha med avdelingsnr og avdelingsnavn – og oversikt over avdelinger skal finnes i systemet). Dessuten skal vi ha med hvilken sykehuspost vedkommende legges inn på (kun et nummer).

Når en pasient legges inn, får vedkommende en diagnose. Det kan imidlertid være slik at personen både får en hoveddiagnose og flere andre diagnoser (f.eks. at hoveddiagnosen er lungebetennelse, men med tilleggsdiagnoser pusteproblem, ribbensbrudd og hoven ankel). Alle slike diagnoser finnes naturligvis i en egen oversikt, jf. også oppgave 1. Vi skal altså både ha mulighet for å registrere en hoveddiagnose og flere tilleggsdiagnoser for innleggelsen.

Lag en datamodell for dette, med utgangspunkt i strukturen fra oppgave 1, forandringer fra oppgave 2, samt de utvidelser som er beskrevet her.

b)
Mens pasienten er innlagt, vil hun/han normalt gjennomgå en eller flere behandlinger på sykehuset. Vi vil ha en kort "fritekst"- beskrivelse av hver slik behandling, startdatoen for denne, og vi vil også vite hvilke av sykehuspersonalet som var involvert, og i hvilken rolle. Med rolle for den ansatte menes operasjonssykepleier, assisterende sykepleier, anestesilege el.l. Det kan hende at f.eks. samme sykepleier er assisterende en gang, men operasjonssykepleier en annen gang. Rollene er dermed ikke nødvendigvis faste pr. person. Det trengs en oversikt over alle ansatte som kan være med i slike behandlinger, samt en oversikt over hvilke roller som er mulige.

På samme måte som for diagnoser finnes en klassifisering av behandlingstyper, i form av et 3-,4- eller 5-sifret tall, f.eks.

Behandlingstypekode	Behandlingstypenavn
00399	Overflateanestesi med bupivakaim
07008	Explorasjon av nerve eller plexus, hovedstamme på legg.
08721	Paralgin forte, 5 tabl. pr. dag.

Hver slik behandling kan bestå av flere deler, f.eks. ved at man først får en bedøvelse (anestesi, f.eks. kode 00399 over), operasjon, deretter medikamentell behandling (medisiner) og fysioterapi. Man vil dermed også ha behov for å vite startdato på hver slik del av behandlingen. Blant annet for medikamentell behandling vil også sluttdato være aktuelt.

Utvid datamodellen til å ta med disse forholdene.

Oppgave 4. Teorioppgave. Tid: 1 time.

Om transaksjoner: samtidighet, hva er transaksjoner, hvorfor trengs det, ACID, låsing, 2PC, vranglås (hva det er, hvordan oppdage det, hvordan unngå det), optimistisk låsing.

(NB! Skriv en omfattende forklaring - optimal tidsforbruk er 1 time)

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 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.

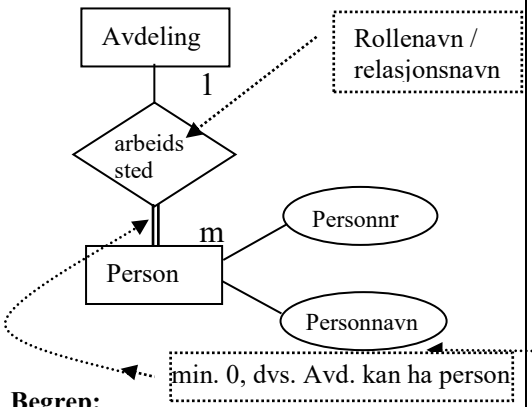
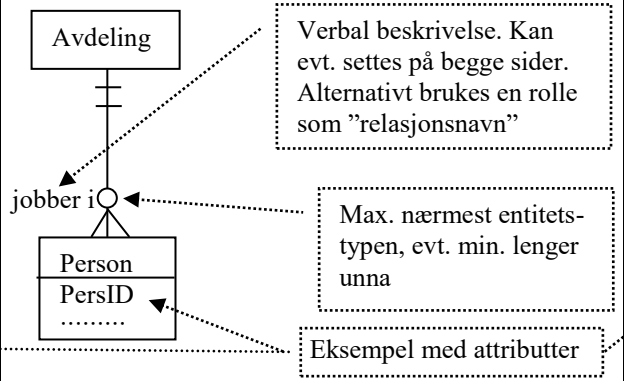
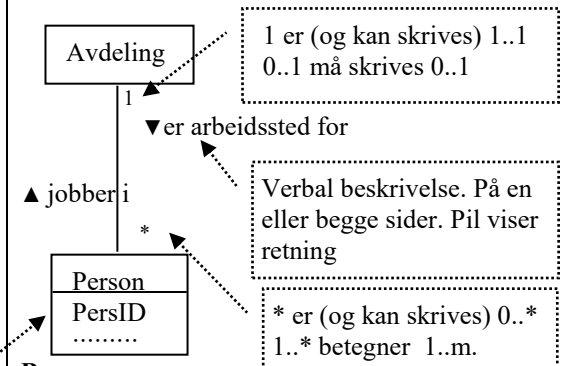
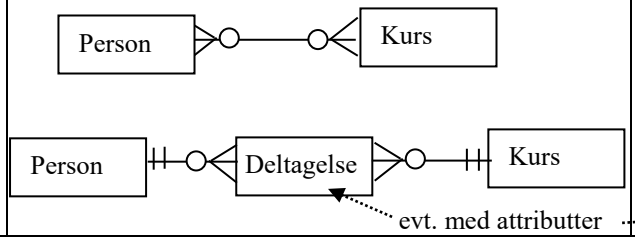
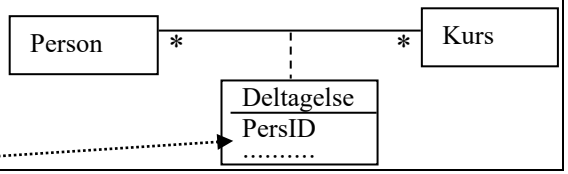
Noen variasjoner og begrensninger fra et system til et annet.

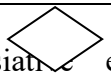
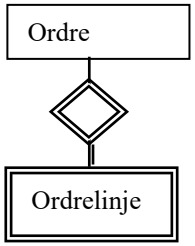
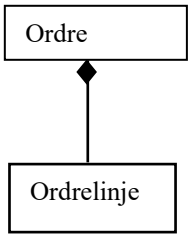
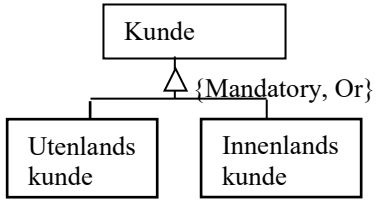
Annet:

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: <u>understreking</u> fremmednøkkel: <u>prikket linje</u>, *, 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. Assosiativ 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.