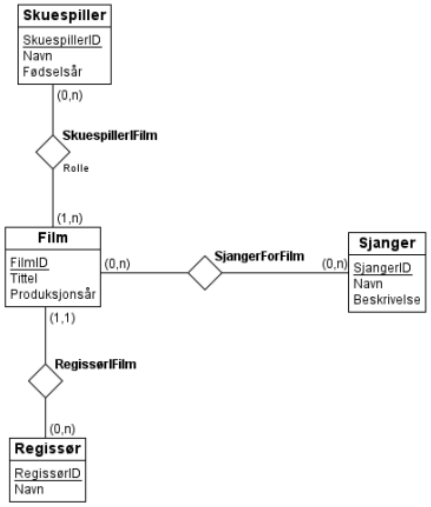
1. **Lage SQL-tabeller og legge inn data**

Gitt følgende relasjonsskjema og ER-diagram:

Film (FilmID, Tittel, Produksjonsår, RegissørID)
SjangerForFilm (FilmID, SjangerID)
Sjanger (SjangerID, Navn, Beskrivelse)
Skuespiller (SkuespillerID, Navn, Fødselsår)
SkuespillerIFilm (FilmID, SkuespillerID, Rolle)
Regissør (RegissørID, Navn)



a. **Vi ønsker at dersom man sletter eller oppdaterer en rad i Film-tabellen, skal tilsvarende referanser til denne raden også slettes eller oppdateres. Dette betyr at hvis man for eksempel sletter en film, skal SkuespillerIFilm- og SjangerForFilmtabellene også slette de radene der referanser til denne filmen inngår. Hvordan kan man spesifisere en slik restriksjon i SQL?** Dette kan spesifiseres ved å sette opp en constraint i table 'film' for både skuespiller, Sjanger og Regissør. Eks:

CONSTRAINT `Reg_FK` FOREIGN KEY (`RegissørID`) REFERENCES `regissør` (`RegissørID`) ON DELETE CASCADE ON UPDATE CASCADE

b. **Skriv nå kode som konstruerer disse tabellene i SQL. Velg selv passende datatyper for attributtene og husk primær- og fremmednøkler. Ta også med kravet presentert i a).**

c. **Skriv SQL-setninger som legger inn følgende data i databasen:**
Regissør(1, "Peyton Reed") insert into regissør values(1, "Peyton Redd")
Regissør(2, "Tom Shadyac") insert into regissør values(2, "Tom Shadyac")
Film(1, "Yes Man ", 2008, 1) insert into Film value(1, "Yes Man", 2008, 1)
Skuespiller("1 , "Jim Carrey", 1962) insert into skuespiller value(1, "Jim Carrey", 1962)
SkuespillerIFilm(1,1, "Carl") insert into Skuespillerifilm value(1,1,"Carl")

d. **Skriv en SQL-setning som oppdaterer navnet til Jim Carrey til James Eugene Carrey**
update skuespiller set navn = "James Eugene Carrey" where navn = "Jim Carrey"

e. **Skrive en SQL-setning som sletter Tom Shadyac fra databasen.**
delete from regissør where navn = "Tom Shadyac"

2. **Writing select statements in SQL**
Vi skal nå fortsette med filmdatabasen fra oppgave 1. Dokumentasjonen for spørringer i MySQL kan være nyttig om du står fast på denne oppgaven1 . Vi anbefaler at du går på Blackboard og henter inn kode fra et script, ov3_filmdata, som setter inn en del mer data i tabellene for deg. Alternativt kan du koble deg opp til en database som kjører på en server på IDI. Se eget skriv på Blackboard for mer informasjon: mysql_filmbase.pdf. Skriv spørringer i SQL som...
   a. **Henter ut filmID, tittel, produksjonsår og regissørID på alle filmer.**
   select * from film
   b. **Finner navn på alle skuespillere født senere enn 1960.**
   select * from skuespiller
   where fødselsår > 1960
   c. **Finner navn på alle skuespillere født på 80-tallet, sortert i alfabetisk rekkefølge.**
   select * from skuespiller
   where fødselsår > 1979
   and fødselsår < 1990
   order by navn;
   d. **Finner titlene på alle filmene og de tilhørende rollene som "Morgan Freeman" har spilt.**
   select film.tittel, skuespillerifilm.rolle
   from skuespiller
   inner join skuespillerifilm using (skuespillerid)
   inner join film using (filmid)
   where navn = "Morgan Freeman"
   e. **Henter ut de distinkte titlene på filmene hvor regissøren og en av skuespillerne i filmen har likt navn.**
   select distinct film.tittel
   from film
   inner join regissør on
   film.RegissørID = regissør.RegissørID
   inner join skuespillerifilm on
   film.filmid = skuespillerifilm.FilmID
   inner join skuespiller on
   skuespillerifilm.skuespillerid = skuespiller.skuespillerid
   where regissør.navn = skuespiller.navn

f. Finner antallet skuespillere som har et navn som starter på "C".

```
select distinct count(skuespiller.navn)
from skuespiller
where navn like 'C%'
```

g. For hver sjanger finner sjangernavnet og antallet filmer av den sjangeren.

```
select navn, count(tittel) as AntallTittler
from sjanger
inner join sjangerforfilm on sjangerforfilm.SjangerID = sjanger.SjangerID
inner join film on film.filmID = sjangerforfilm.filmId
group by sjanger.navn
```

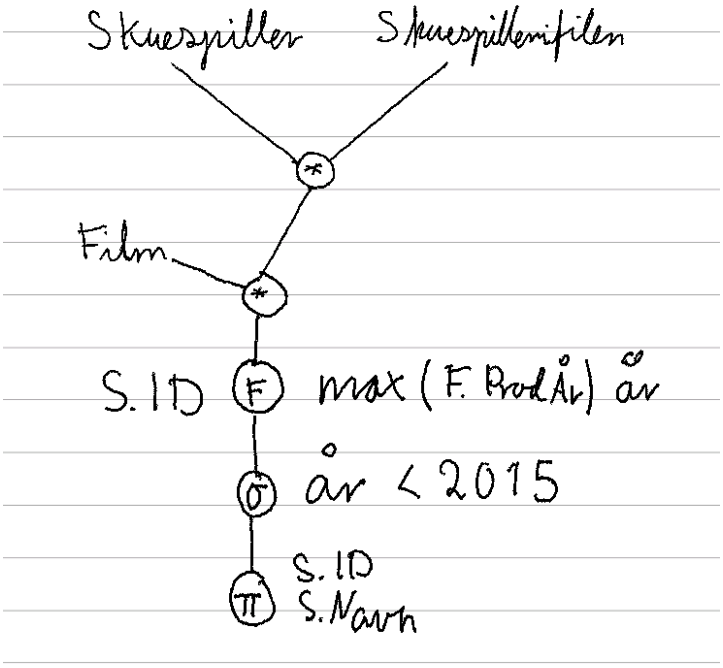
h. Finner navnet på skuespillere som har spilt i filmen "Ace Ventura: Pet Detective", men aldri i filmen "Ace Ventura: When Nature Calls".

```
select navn, tittel
from skuespiller
        inner join skuespillerifilm on skuespillerifilm.SkuespillerID =
skuespiller.SkuespillerID
        inner join film on film.filmID = skuespillerifilm.filmID
where film.tittel = "Ace Ventura: Pet Detective"
        and skuespiller.navn not in (select navn
                from skuespiller
                        inner join skuespillerifilm on
skuespillerifilm.SkuespillerID = skuespiller.SkuespillerID
                        inner join film on film.filmID = skuespillerifilm.filmID
                where film.tittel = "Ace Ventura: When Nature Calls")
```

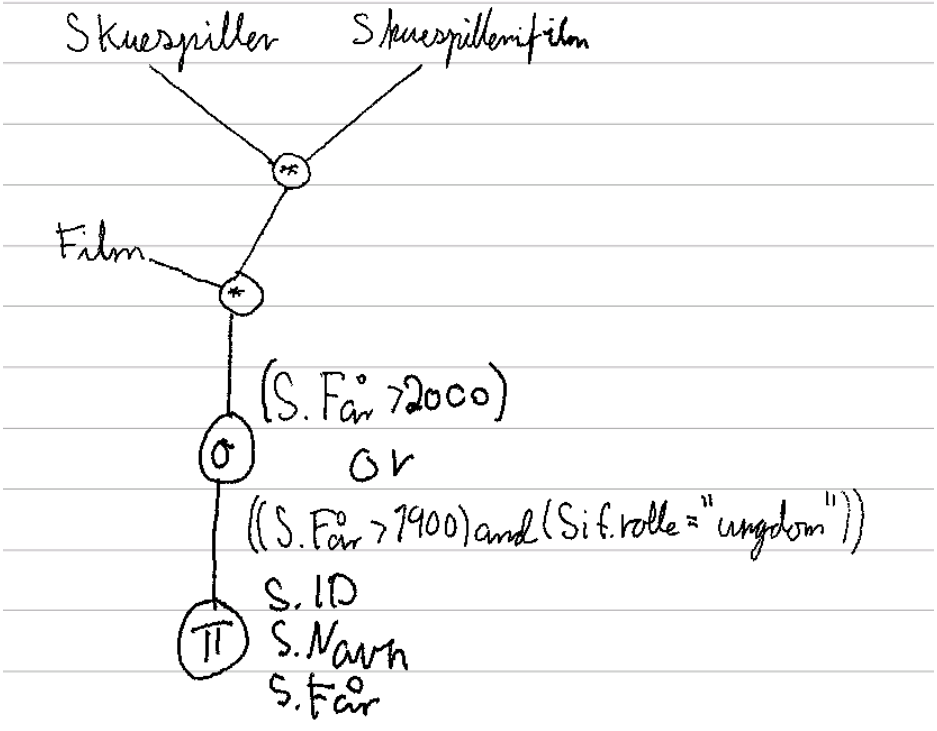
i. For hver film finner navnet på filmen, filmID og gjennomsnittlig fødselsår på skuespillerne i filmen. Vi ønsker kun å få med de filmene som har gjennomsnittlig fødselsår større enn gjennomsnittlig fødselsår for alle skuespillerne i databasen. (Hint: Her kan det være lurt med en underspørring i en HAVING-del etter aggregeringen).

```
select distinct tittel, f.FilmID, avg(Fødselsår) as GjennomsnittligFødselsår
from film f
inner join skuespillerifilm sif on f.filmid = sif.FilmID
inner join skuespiller s on sif.SkuespillerID = s.SkuespillerID
group by tittel
having GjennomsnittligFødselsår > (
        select avg(Fødselsår)
        from film f
                inner join skuespillerifilm sif on f.filmid = sif.FilmID
                inner join skuespiller s on sif.SkuespillerID = s.SkuespillerID
)
```

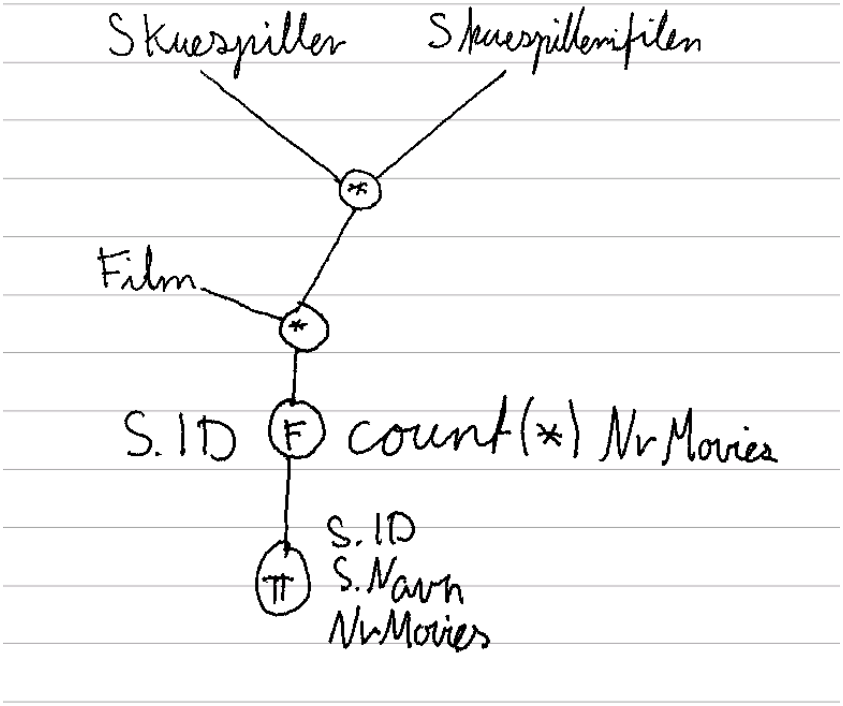
3. **Flere spørringer i relasjonsalgebra**
    a. **Finner skuespillerID og navn på de skuespillere som ikke har spilt i en film produsert etter 2014.**



    b. **Henter skuespillerID, navn og fødselsår på skuespillerne som er født senere enn 2000, og de skuespillerne som er født senere enn 1990 og har spilt i en film som en ungdom (Rolle = "Ungdom").**



c. **For hver skuespiller finner skuespillerID, navnet og antall filmer vedkommende har spilt i.**



4. **Introduksjon til normaliseringsteori**
   a. **Dette er et eksempel på dårlig design. Det er fordi lagrer vi mye av den samme informasjonen mange ganger (redundans), når vi sannsynligvis kan strukturere tabellen på en smartere måte for å unngå dette. Vi får også problemer med innsetting, oppdatering og sletting av data. Hvis vi for eksempel sletter en person, kan vi risikere å helt miste informasjon om et fakultet.**

      Hvis fakultetet EDI endrer navn til DI (institutt for datainnovasjon), slik at både fakultetskoden og fakultetsnavnet må endres, hvor mange felter trenger å oppdateres?
      10 Felt

   b. **Kan du foreslå et alternativt design som gjør at vi trenger å oppdatere færre felter hvis EDI endrer navn? (Tips: Normaliseringsteorien presenterer som regel løsninger som innebærer å splitte en tabell opp i flere deltabeller. I dette tilfellet blander vi personinformasjon med fakultetsinformasjon.)**

      Du kan anta at informasjon om fakultetsnavn og fakultetsbygg vil være like for samme verdier av Fakultetskode. Dvs. at vi har en funksjonell avhengighet Fakultetskode -> Fakultetsnavn, FakultetsBygg

PersonID	**Navn**	**Telefonnr**	***Fakultetskode***
270393	Ola Johannes	73735667	EDI
...	...	...	..

Fakultetskode	**Fakultetsnavn**	**Fakultetsbygg**
EDI	Institutt for energi og datainnovasjon	Oasen
IØA	Institutt for økonomiske arkitekturer	Solklossen

5. **Functional dependencies, keys and closures**
   a. **Anta følgende tabell. Hvilke påstander kan umulig stemme? Begrunn svarene dine.**

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2
a_2	b_1	c_2	d_2
a_3	b_2	c_3	d_3
a_3	b_2	c_1	d_1
a_4	b_2	c_4	d_2

Skriver kanskje når eksemplet ikke umuliggjør påstanden

i. **A -> A** Ja (Den er triviell)
ii. **A -> B** Nei (a_1 -> b_1 og a_1 -> b_2 ulik b_1)
iii. **A -> C** Nei (Samme som over, a_x kan ikke implisere 2 forskjellige verdier)
iv. **AB -> C** Nei (vi har to like kombinasjoner av a_1 og b_1 på rad 1 og 2 som impliserer to ulike c_x verdier)
v. **C -> D** Kanskje (hver c_x impliserer en entydig verdi for d_x)

vi. **D -> C** Nei (d_2 impliserer to forskjellige verdier c_2 og c_4)

vii. **ABCD er en supernøkkel for tabellen** Ja (Dette er trivielt siden dersom A, B, C og D er nøkkelattributter gjelder dette.

viii. **ABC er en supernøkkel for tabellen** Kanskje (Dette gjelder kun om D ikke er en nøkkelattributt, og vi har at **C -> D** Kanskje)

ix. **A er en kandidatnøkkel for tabellen** Nei (Vi har at **A -> B** Nei og **A -> C** Nei, vi sjekker **A -> D** Nei, dermed kan A være en supernøkkel engang)

x. **AC er en kandidatnøkkel for tabellen** Kanskje (**AC -> D** Kanskje, **D -> B** Nei, **AC-> B** Ergo AC er kanskje en supernøkkel, er den minimal?
A er ikke en nøkkel
B er ikke en nøkkel
C er ikke en nøkkel
D er ikke en nøkkel
=> Dersom AC er en supernøkkel, er den også minimal slik at den er en kandidatnøkkel)

b. **X+ kalles tillukningen av X og er mengden av alle attributter som er funksjonelt avhengig av X (basert på en mengde funksjonelle avhengigheter F). Gitt tabellen R = {A, B, C, D} og F = {A -> C, B-> D, ABC -> D}. Finn A+, D+, BC+ og AB+.**
A+ = AC
D+ = D
BC+ = BCD
AB+ = ABCD