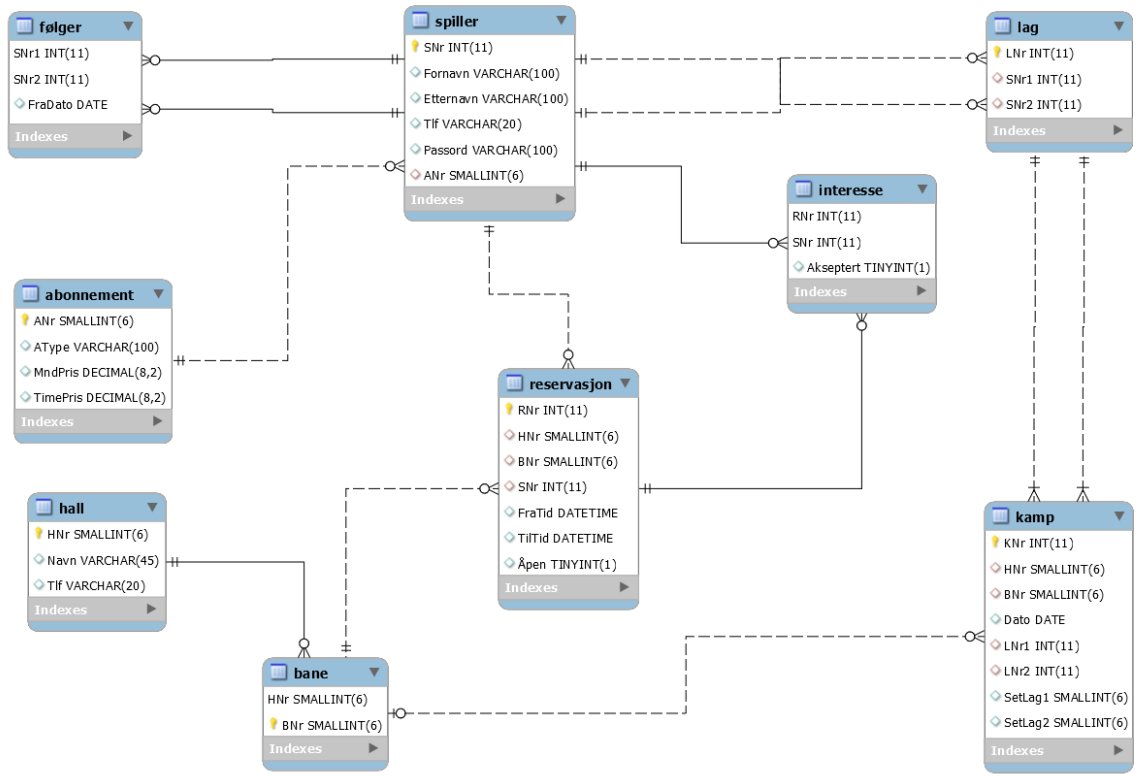
DAT1000 Databaser 1 vår 2021 – løsningsforslag

Oppgave 1



Oppgave 2

Det finnes flere gode datamodeller til oppgave 1. Vanskelighetsgraden på SQL-spørsmålene i denne oppgaven vil variere avhengig av løsningen man har valgt i oppgave 1. Dette blir det tatt høyde for under sensur.

2-a

Dette spørsmålet er ment å teste **SELECT**-spørring mot én tabell med bruk av funksjoner, sortering, jokernotasjon og sammenligning med **LIKE**, og sammensatte betingelser (**AND**).

```
SELECT SNr, CONCAT(Fornavn, ' ', Etternavn) AS Navn, Tlf
FROM Spiller
WHERE Etternavn LIKE 'M%' AND LENGTH(Etternavn)>3
ORDER BY Etternavn, Fornavn;
```

For de fleste vil denne oppgaven la seg løse med en spørring mot én enkelt tabell. Hvis det er nødvendig å koble tabeller, blir det vurdert om man gjør dette på riktig måte. Lignende helhetsvurderinger blir gjort ved vurdering av alle SQL-spørsmålene.

2-b

Dette spørsmålet tester likekobling, gruppering og mengdefunksjoner.

```
SELECT H.HNr, H.Tlf, COUNT(*) AS AntKamper
FROM Hall AS H INNER JOIN Kamp AS K ON H.HNr = K.HNr
WHERE DATEDIFF(CURDATE(), Dato) <= 30
GROUP BY H.HNr, H.Tlf;
```

MySQL gir korrekt resultat også uten H.Tlf i **GROUP BY**, så dette gir også full uttelling.

Alternativ løsning med **WHERE**-betingelse i stedet for **INNER JOIN** gir også full uttelling (det samme gjelder alle øvrige SQL-spørsmål):

```
SELECT H.HNr, H.Tlf, COUNT(*) AS AntKamper
FROM Hall AS H, Kamp AS K
WHERE H.HNr = K.HNr
AND DATEDIFF(CURDATE(), Dato) <= 30
GROUP BY H.HNr, H.Tlf;
```

2-c

Dette spørsmålet tester oppdateringsspørringer.

```
UPDATE Abonnement
SET MndPris = MndPris*1.05
WHERE AType = 'Hobby';
```

2-d

Dette spørsmålet tester mer komplisert bruk av likekoblinger, gruppering og «formler». Oppgaven kan nok variere i vanskelighetsgrad avhengig av hvordan datamodellen ser ut, noe man må ta særlig høyde for under sensur. For god uttelling bør man summere kostnaden for enkeltreservasjoner. Det gir liten uttelling hvis man har lagret ferdig utregnet månedskostnad i en databasetabell.

```
SELECT S.SNr, Fornavn, Etternavn,
  ROUND(SUM(TIMESTAMPDIFF(MINUTE, FraTid, TilTid))/60*TimePris
  + MndPris, 2) AS Beløp
FROM Abonnement AS A INNER JOIN
       (Spiller AS S INNER JOIN Reservasjon AS R ON S.SNr = R.SNr)
     ON A.ANr = S.ANr
WHERE YEAR(FraTid) = 2021
AND MONTH(FraTid) = 4
GROUP BY S.SNr, Fornavn, Etternavn, TimePris, MndPris;
```

Tilsvarende spørring med **WHERE**-betingelse i stedet for **INNER JOIN** gir også full uttelling.

MySQL gir korrekt resultat også med kun S.SNr i **GROUP BY**, så dette gir også full uttelling.

2-e

Her skulle man både skrive oppgavetekst og løse denne oppgaven. Dette spørsmålet tester bruk av delspørringer med **NOT EXISTS**, men også evne til å beskrive en oppgave presist og gjenkjenne hvilke oppgaver der delspørringer med **NOT EXISTS** er nyttige. **Eksempel på oppgavetekst**: Skriv en SQL-spørring som viser alle spillere som ikke har reservert noen timer.

```
SELECT *
FROM Spiller AS S
WHERE NOT EXISTS
  (SELECT * FROM Reservasjon AS R WHERE S.SNr = R.SNr);
```

2-f

Her skulle man også både skrive oppgavetekst og løse denne oppgaven. Dette spørsmålet tester bruk av gruppering og ytre koblinger, men også evne til å beskrive en oppgave presist og gjenkjenne hvilke oppgaver der gruppering og ytre koblinger er nyttige. **Eksempel på oppgavetekst**: Vis antall reservasjoner for hver spiller. Også spillere som *ikke* har lagt inn reservasjoner skal med i utskriften.

For å få riktig resultat er det viktig hvilken kolonne man bruker som argument/parameter til COUNT. Med vår tabellstruktur bør man velge en kolonne fra Reservasjon.

```
SELECT S.SNr, S.Fornavn, S.Etternavn, COUNT(R.SNr) AS Antall
FROM Spiller AS S LEFT OUTER JOIN Reservasjon AS R
     ON S.SNr = R.SNr
GROUP BY S.SNr, S.Fornavn, S.Etternavn;
```

MySQL gir korrekt resultat også med kun S.SNr i **GROUP BY**, så dette gir også full uttelling.

2-g

Dette spørsmålet tester bruk av views og også bruk av egenkoblinger. I løsningsforslaget er tabellen Kamp koblet med to «kopier» av Lag og fire «kopier» av Spiller. Man kan også løse denne oppgaven med delspørringer. Oppgaven kan nok variere i vanskelighetsgrad avhengig av hvordan datamodellen ser ut, noe man må ta særlig høyde for under sensur.

```
CREATE VIEW ÅretsKamper AS
  SELECT K.*,
    S1.Etternavn AS Etternavn1, S2.Etternavn AS Etternavn2,
    S3.Etternavn AS Etternavn3, S4.Etternavn AS Etternavn4
  FROM Kamp AS K,
    Lag AS L1, Lag AS L2,
    Spiller AS S1, Spiller AS S2, Spiller AS S3, Spiller AS S4
  WHERE K.LNr1 = L1.LNr AND K.LNr2 = L2.LNr
  AND L1.SNr1 = S1.SNr AND L1.SNr2 = S2.SNr
  AND L2.SNr1 = S3.SNr AND L2.SNr2 = S4.SNr
  AND YEAR(K.Dato) = YEAR(CURDATE());
```

Tilsvarende spørring med **INNER JOIN** notasjon gir også full uttelling.

Oppgave 3

3-a

Vi ser altså på følgende tabell:

- Bestilling(PNr, Navn, Pris, SNr, Fornavn, Etternavn, Tidspunkt, Antall)

Ulempen med den foreslåtte tabellen er at informasjon om produkter (Navn og Pris) blir gjentatt for alle bestillinger av samme produkt (PNr) og informasjon om spillere (Fornavn og Etternavn) blir gjentatt for alle bestillinger av samme spiller (SNr).

Dette skaper redundans (dobbeltlagring) og medfører at oppdatering av tabellen blir mer komplisert. En god begrunnelse bør være konkret, slik som dette, og ikke bare si at tabellen inneholder redundans.

Funksjonelle avhengigheter:

- PNr, SNr, Tidspunkt -> ALLE KOLONNER
- PNr -> Navn, Pris
- SNr -> Fornavn, Etternavn

Kandidatnøkkel: PNr, SNr, Dato

Den første avhengigheten er uproblematisk (starter i kandidatnøkkelen). Begge de to andre er brudd på 2NF, vi behandler den fra PNr først.

- Bestilling(PNr, SNr, Fornavn, Etternavn, Tidspunkt, Antall)
- Produkt(PNr, Navn, Pris)

Deretter den fra SNr, som gir en tredje tabell Spiller. Resultattabeller med primærnøkler understreket og fremmednøkler merket med stjerne:

- Produkt(<u>PNr</u>, Navn, Pris)
- Spiller(<u>SNr</u>, Fornavn, Etternavn)
- Bestilling(<u>PNr*, SNr*, Tidspunkt</u>, Antall)

3-b

En god besvarelse bør inneholde relevante eksempler på følgende kommandoer mot databasen man har laget i oppgave 1:

- CREATE USER
- CREATE ROLE
- GRANT
- REVOKE

Det kan være naturlig å opprette roller for spillere og for (ulike typer av) ansatte. Begrunnelsen bør få fram at brukere ikke bør gis flere rettigheter enn de trenger, f.eks. trenger ikke spillere rettighet til å endre i databasetabellene Hall, Bane og Abonnement.

3-c

En god besvarelse bør forklare de fleste av følgende teknikker/mekanismer og dessuten gi konkrete og relevante eksempler mot egen database fra oppgave 1:

- Datatyper
- Primærnøkler
- Fremmednøkler, og kanskje spesielt bruk av fremmednøkler mot «kodetabeller» som kun inneholder én kolonne med listen av lovlige verdier.
- NOT NULL
- UNIQUE
- CHECK-regler

Kanskje vil noen også nevne triggere, selv om dette ikke er pensum i emnet.