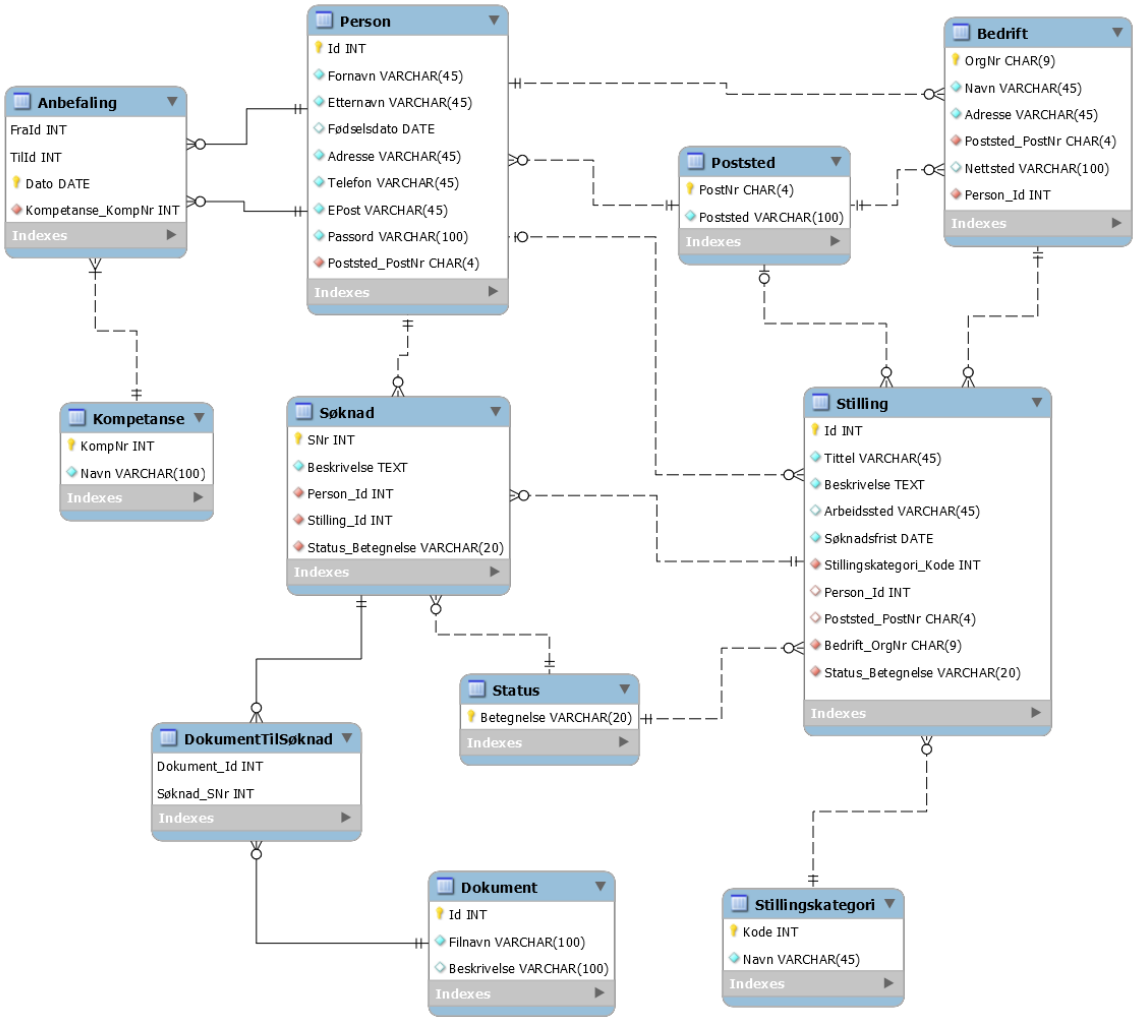
DAT1000 Databaser 1 – løsningsforslag konte august 2021

Oppgave 1



Besvarelser på oppgaver i datamodellering blir vurdert ut fra flere og sammensatte kriterier. Det er for det første snakk om å få med alle momentene som blir nevnt i oppgaveteksten, men man skal på den andre siden heller ikke ta med unødvendige ting. Videre er det viktig at man får samlet de ulike opplysningene (attributtene/kolonnene) med fornuftige datatyper i hensiktsmessige entiteter (tabeller) og får opprettet forhold mellom entiteter der det er hensiktsmessig. Det er samtidig viktig å mestre de «tekniske» sidene ved ER: alle entiteter bør ha en fornuftig primærnøkkel, man bør ta hensiktsmessige valg når det gjelder en-til-mange, en-til-en og mange-til-mange forhold samt når det gjelder identifiserende og ikke-identifiserende forhold. Attributter bør gis naturlige datatyper og være definert med **NOT NULL** der det er aktuelt. Blant annet.

Det er vanskelig å beskrive presist hva som skal til for at en datamodell skal bli vurdert til f.eks. en A eller C eller F, for noen besvarelser kan ha med nesten alle momentene fra oppgaveteksten, men inneholde flere store, tekniske feil, mens andre besvarelser kan være teknisk sett gode, men ikke ha med mange av problemstillingene. Noen ganger kan det være krevende å tolke datamodeller riktig

(slik det var ment), fordi ulike personer (studenter) velger forskjellige ord for å beskrive samme ting. Som regel vil det dessuten finnes flere, ulike og fullgode løsninger på en gitt datamodelleringsoppgave.

Gitt alle disse forbeholdene: For å få uttelling tilsvarende karakteren C bør man i hovedsak mestre de tekniske sidene ved ER og også ha med de viktigste momentene. Men man kan altså gjøre noen feil, kanskje til og med en grov feil som f.eks. å «snu» et en-til-mange forhold – hvis man gjort dette riktig andre steder i modellen. For å få karakteren A bør (så å si) alle momentene være med og det bør kun være små tekniske feil. For å få karakteren E kan det være flere grove feil og store mangler, men man må likevel levere et *ER-diagram* som viser entiteter, attributter og forhold for den aktuelle problemstillingen.

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

Alle personer født mellom 1970 og 1985 på postnummer 3200 og 3800.

Denne oppgaven tester enkle spørringer med bruk av **AND** og **OR**.

```
SELECT *
FROM Person
WHERE YEAR(Fødselsdato) BETWEEN 1970 AND 1985
AND (Poststed_PostNr = '3200' OR Poststed_PostNr = '3800');
```

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

Aktive stillinger som inneholder «SQL» som del av beskrivelsen. Sortert med hensyn på søknadsfrist (de nyeste først) og deretter på tittel.

Denne oppgaven tester (dobbel) sortering og bruk av **LIKE**-sammenligning.

```
SELECT Id, Tittel, Søknadsfrist
FROM Stilling
WHERE Beskrivelse LIKE '%SQL%' AND Status_Betegnelse = 'aktiv'
ORDER BY Søknadsfrist DESC, Tittel ASC;
```

2-c

Alle kompetansebekreftelser så langt i år. Ta med navn og alder på personen som har fått anbefalingene, dato og selve kompetansen i resultatet.

Denne oppgaven tester likekoblinger og bruk av datofunksjoner.

```
SELECT P.Id, P.Fornavn, P.etternavn, A.Dato,
  YEAR(FROM_DAYS(DATEDIFF(CURDATE(), Fødselsdato))) AS Alder,
  K.Navn
FROM Person AS P INNER JOIN
       (Anbefaling AS A INNER JOIN Kompetanse AS K
        ON A.Kompetanse_KompNr = K.KompNr)
     ON A.TilId = P.Id
WHERE YEAR(Dato) = YEAR(CURDATE());
```

Spørringen kan alternativt skrives med vanlige **WHERE**-betingelse. Det gir også full uttelling (her og i andre oppgaver der man må skrive likekoblinger).

```
SELECT P.Id, P.Fornavn, P.etternavn,
  YEAR(FROM_DAYS(DATEDIFF(CURDATE(), Fødselsdato))) AS Alder,
  K.Navn
FROM Person AS P, Anbefaling AS A, Kompetanse AS K
WHERE A.TilId = P.Id AND A.Kompetanse_KompNr = K.KompNr
AND YEAR(Dato) = YEAR(CURDATE());
```

Her får man tilnærmet full uttelling også for (litt for enkle) beregninger av alder, som f.eks.

```
  YEAR(CURDATE())-YEAR(Fødselsdato)
  DATEDIFF(CURDATE(), Fødselsdato) / 365
```

2-d

Antall (aktive) søknader for hver ledige stilling som nå er aktiv.

Denne oppgaven tester gruppering og mengdefunksjoner.

```
SELECT Stilling.Id, Stilling.Tittel, COUNT(*) AS AntallSøknader
FROM Stilling INNER JOIN Søknad
     ON Stilling.Id = Søknad.Stilling_Id
WHERE Stilling.Status_Betegnelse = 'aktiv'
AND Søknad.Status_Betegnelse = 'aktiv'
GROUP BY Stilling.Id;
```

Her kan man ta med flere kolonner fra Stilling.

2-e

Utsett søknadsfristen med en uke for alle ledige stillinger (status «aktiv») ved en gitt bedrift. Velg bedrift selv.

Denne oppgaven tester bruk av **UPDATE** (og datofunksjoner).

```
UPDATE Stilling
SET Søknadsfrist = DATE_ADD(Søknadsfrist, INTERVAL 1 WEEK)
WHERE Bedrift_OrgNr = '123456789'
AND Status_Betegnelse = 'aktiv';
```

2-f

Skriv en SQL-oppgave som tester evne til å lage views som kobler data fra flere tabeller.

Denne oppgaven tester bruk av views og likekoblinger, samt evne til å se sammenhengen mellom oppgaveformulering og SQL-kode.

Eksempel: Lag et view som viser søkerlistene for alle IT-stillinger.

Ta med navn og fødselsdato til søkerne, samt stillingsnummer, søknadsfrist og stillingstittel.

```
CREATE VIEW Søkerliste AS
  SELECT St.Id, St.Tittel, St.Søknadsfrist,
         P.Fornavn, P.Etternavn, P.Fødselsdato
  FROM Person AS P, Søknad AS S,
       Stilling AS St, Stillingskategori AS K
  WHERE S.Person_Id = P.Id
  AND S.Stilling_Id = St.Id
  AND St.Stillingskategori_Kode = K.Kode
  AND K.Navn = "IT";
```

2-g

Alle stillingskategorier som ikke er i bruk.

Denne oppgaven tester delspørringer og **IN**-operatoren.

```
SELECT *
FROM StillingsKategori
WHERE Kode NOT IN
  (SELECT DISTINCT Stillingskategori_Kode FROM Stilling);
```

Her kan man også bruke **NOT EXISTS**.

2-h

Registrere en ny ledig stilling. Velg eksempeldata selv. Gjør rede for eventuelle forutsetninger som gjelder fremmednøkler.

Denne oppgaven tester bruk av **INSERT**.

Jeg har valgt autonummerert primærnøkkel i Stilling. Jeg forutsetter at verdier for stillingskategori, kontaktperson, postnummer, organisasjonsnummer og status allerede er lagt inn i tilhørende tabeller.

```
INSERT INTO
  Stilling(Tittel, Beskrivelse, Arbeidssted, Søknadsfrist,
    Stillingskategori_Kode, Person_Id, Poststed_PostNr,
    Bedrift_OrgNr, Status_Betegnelse)
VALUES
  ('Utvikler', 'Vi søker en ...', NULL, '2021-08-30',
    7, 313, '3200', '123456789', 'ny');
```

Oppgave 3

3-a

Vi ser altså på følgende tabell:

- Avtale(Dato, KlSlett, RegNr, Merke, Modell, Beskrivelse, EierTlf, EierNavn)

Eksempler på redundans:

- For gjentatte avtaler på samme bil (samme RegNr), blir merke og modell gjentatt.
- For gjentatte avtaler med samme bileier (EierTlf), blir navnet (EierNavn) gjentatt.

Følgende funksjonelle avhengigheter gjelder:

- Dato + KlSlett + RegNr → alle kolonner
- RegNr → Merke + Modell
- EierTlf → EierNavn

Kandidatnøkkel: Dato + KlSlett + RegNr

Man kan argumentere for at navnet Ole Mo er sammensatt og ikke «atomært». Det virker hensiktsmessig å dele opp EierNavn i Fornavn og Etternavn, da får vi følgende tabell:

- Avtale(Dato, KlSlett, RegNr, Merke, Modell, Beskrivelse, EierTlf, Fornavn, Etternavn)

Den første avhengigheten er uproblematisk (starter i kandidatnøkkelen).

Avhengigheten RegNr → Merke + Modell bryter med 2NF. Vi splitter tabellen og får:

- Avtale(Dato, KlSlett, RegNr, Beskrivelse, EierTlf, Fornavn, Etternavn)
- Bil(RegNr, Merke, Modell)

Avhengigheten EierTlf → Fornavn bryter med 3NF. Vi splitter avtaletabellen igjen og får sluttresultatet – med primærnøkler og fremmednøkler:

- Avtale(<u>Dato, KlSlett, RegNr*</u>, Beskrivelse, EierTlf*)
- Bil(<u>RegNr</u>, Merke, Modell)
- Eier(<u>EierTlf</u>, Fornavn, Etternavn)

(Vi har her bare beholdt navnet Avtale på «hovedtabellen» i hvert steg og det er jo også et naturlig navn på denne tabellen i sluttresultatet.)

3-b

En god besvarelse bør ta for seg flere konkrete kolonner (fra oppgave 1) med ulike datatyper og diskutere typiske valg som f.eks. tall eller tekst, heltall eller desimaltall, flyttall eller fast antall desimaler, maksimalstørrelse på tekst, fast eller variabel lengde på tekst, valg mellom ulike dato/tid-kolonner.

3-c

En forretningsregel der lovlige verdier skal begrenses til en konkret liste kan med håndteres med fremmednøkler. Eksempler fra oppgave 1: Kompetanse, Status og Stillingskategori. Fordelen med denne teknikken er at en bruker av databasen kan redigere listen. En typisk **CHECK**-regel kunne f.eks. være å sikre at alle epostadresser er på et lovlig format (inneholder alfakrøll og punktum på de riktige stedene).

Eksempel på en forretningsregel som ikke lar seg håndtere med noen av disse to teknikkene: Sørg for at alle som søker på en gitt stilling er over 18 år, altså at datoen i kolonnen Person.Fødselsdato er minst 18 år forut i tid sammenlignet med datoen i kolonnen Søknad.Søknadsfrist – for tilfeller der denne personen faktisk søker på denne stillingen.