

## Høyskolen Kristiania

### Eksamen

### DB1100 DATABASES

Tillatte hjelpemidler: Ingen

Varighet: 180 minutter

Dato: 3. desember 2018

1 vedlegg, inkludert i oppgavesettet: "MySQL, utvalgte datatyper/syntaks"

---

*Denne eksamen utgjør 100% av karakteren i DB1100.*

*Der databasetype er relevant for spørsmålet tar oppgavene utgangspunkt i en MySQL database.*

*Du kan anta at verktøyet MySQL Workbench benyttes der verktøy er relevant.*

*Pass på at du disponerer tiden riktig i forhold til vektingen av de ulike oppgavene.*

*Oppgi eventuelle forutsetninger du tar.*

## Oppgave 1: SQL – 50%

Et firma som passer hunder mens eierne er på ferie (en kennel) holder oversikt over hundeeiere, deres hunder og hvilken rase hver hund er. Dette lagres i en database, som består av følgende:

hundeeier

- eier\_id{PK}      int      not null
- fornavn      varchar(100)      not null
- etternavn      varchar(100)      not null
- fødselsdato      date      ---
- epostadresse      varchar(256)      not null

hund

- hunde\_id{PK}      int      not null, auto\_increment
- navn      varchar(100)      not null
- fødselsdato      date      ---
- rase\_id {FK}      int      ---
- eier\_id {FK}      int      not null

hunderase

- rase\_id{PK}      int      not null, auto\_increment
- navn      varchar(100)      not null
- verdensdel      enum      ---

## Videre informasjon:

- `verdensdel` i `hunderase` er en enum. Enum er en datatype vi kan definere mulige verdier for selv, for denne er det 7 mulige verdier: 'Nord-Amerika', 'Sør-Amerika', 'Europa', 'Asia', 'Afrika', 'Oseania' og 'Antarktis'. Denne beskriver i hvilken verdensdel rasen har sitt opphav.
- En hund må registreres med én eier.
- En hund kan registreres uten å oppgi rase.

Bruk informasjonen over som grunnlag for følgende SQL oppgaver (se gjerne vedlegget for tips til SQL syntaks):

- a) Skriv en SQL-spørring som henter ut all informasjon fra tabellen `hundeeier`.
- b) Skriv en SQL-spørring som gir oss antall hunder som er registrert i databasen. Kall svar-kolonnen for «Antall\_hunder».
- c) Skriv en SQL-spørring som henter ut navn på hunderaser som har opphav fra verdensdelen 'Nord-Amerika'. Sorter hunderasenes navn i stigende rekkefølge.
- d) Lag en SQL-spørring med bruk av subquery som henter ut navn og fødselsdato for alle hunder som har opphav fra verdensdelen Asia.
- e) Lag en SQL-spørring som gir oss **alle** hunder (navn og fødselsdato), og i de tilfellene der hunderase er oppgitt, også deres hunderase-navn. Bruk en form av join (LEFT/RIGHT/INNER) til dette.
- f) Kennelen vet at de har 9 hunder som heter Lassie i databasen. De lurte på om det er andre hundenavn som går igjen hos flere hunder. Lag en SQL spørring som gir oss hundenavn og antall hunder som har dette navnet. Lag spørringen slik at den bare viser navn og antall når antallet hunder som heter dette er mer enn 1.
- g) Lag et view (Navn: «fellesnavn») for spørringen over. Kjør deretter en spørring mot viewet.
- h) Skriv en SQL-statement som sletter alle hunder som eies av hundeeier med `eier_id = 8`.
- i) Skriv en SQL-statement som endrer epostadresse for hundeeier med `eier_id = 3`. Riktig epostadresse skal være: `ola.olsen@mymail.com`.
- j) Kennelen tar imot en ny kunde. Kunden leverer en hund med en rase som ikke finnes i databasen fra før. Lag **tre SQL statements** som legger inn informasjon om `hundeeier` (Eier\_id: 12345, Fornavn: Per, Etternavn: Petterssen, Fødselsdato: 22. januar 1993, Epostadresse: `per.petterssen@somemail.no`), `hund` (Navn: Passop, Fødselsdato: 12. desember 2015) og `hunderase` (Navn: Schipperke, Verdensdel: Europa). Du kan forutsette at neste hunderase som legges inn i databasen får `rase_id=35`. **Statementene du skriver ned skal stå i samme rekkefølge som du planlegger å kjøre dem.**

## Oppgave 2: Modellering – 20%

Du skal modellere en database i forbindelse med utviklingen av en ny streamingtjeneste for TV serier. Oppdragsgiver beskriver behovet slik:

*Vi ønsker å lagre data for TV serier. Vi må lagre originaltittel og hvilket år serien ble lansert (vist første gang). Vi må også kunne lagre en anbefalt aldersgrense slik at vi kan filtrere seriene basert på alderen til den som skal streame de. En serie kan bestå av flere sesonger. Hver sesong har et sesongnummer (sesong 1, sesong 2 etc), en originaltittel og et lanseringsår. Hver sesong består av flere episoder. Hver episode har et episodenummer (episode 1, episode 2, etc.), en originaltittel, en premieredato og en lengde (hvor lenge en episode varer).*

*Vi ønsker også muligheten til å lagre beskrivelser av seriene. Vi skal i starten ha mulighet for beskrivelser på norsk og engelsk per serie, men vi vil sannsynligvis trenge flere språk etter hvert. Brukeren må kunne velge hvilket språk han/hun foretrekker, så vi ser for oss at språk må være en egen greie. Legg til funksjonalitet som støtter beskrivelser på flere språk for flere serier, om det er mulig å få til.*

**Oppgave:** Tegn en modell for din foreslåtte løsning. Du kan selv velge om du vil benytte kråkefot eller UML notasjon. Velger du kråkefot trenger du ikke skille mellom identifiserende og ikke-identifiserende forhold. (UML notasjon har uansett ikke skille på dette.) Modellen din skal inneholde:

- Entitetene og deres attributter.
- Primærnøkler og fremmednøkler.
- Relasjonene mellom entitetene.
- Multiplisiteten (deltagelse og kardinalitet) for relasjonene.
- Hvis nødvendig, koblingsentiteter.

## Oppgave 3: Normalisering – 20%

Et firma som leverer bredbånd leverer også streaming av filmer (video-on-demand) som en ekstra tjeneste til sine abonnenter. Firmaet har en tabell med data som beskriver hvilke filmer som er lånt av hvilke kunder. Et utdrag av tabellen er vist nedenfor.

kundenummer	film_id	kundenavn	filmtittel
8	37	Morten Hanssen	Tatt av vinden
9	15	Lene Jenssen	Pretty woman
11	24	Hans Hanssen	Terminator 2
12	15	Andre Jenssen	Pretty woman
12	24	Andre Jenssen	Terminator 2
12	37	Andre Jenssen	Tatt av vinden

Tabellen har en sammensatt primærnøkkel (kundenummer, film\_id). En kunde kan identifiseres ved sitt kundenummer, og har navn likt kundenavn. En film kan identifiseres ved sin film\_id, og har tittel lik filmtittel.

- a) Forklar redundans og nevnt to tilfeller av dette i tabellen over.
- b) Forklar hvorfor tabellen ikke er på 2NF, og normaliser tabellen slik at vi oppnår 2NF.

Firmaet har også en tabell som holder informasjon på deres ansatte. Et utdrag av tabellen er vist nedenfor.

ansattnummer	fornavn	etternavn	avdelingsnummer	avdelingssted
3	Per	Persson	3	Hovseter
5	Ole	Olsen	4	Drammen
8	Liv	Hanssen	3	Hovseter
9	Beate	Jensen	2	Fredrikstad

En ansatt kan identifiseres ved sitt ansattnummer som er primærnøkkel i tabellen. Tabellen holder informasjon om den ansattes fornavn og etternavn. Det registreres også hvilken avdeling den ansatte jobber i. Avdelingen har et unikt avdelingsnummer og avdelingsstedet forteller oss hvor i Norge avdelingen holder til.

- c) Forklar hvorfor tabellen ikke er på 3NF, og normaliser tabellen slik at vi oppnår 3NF.
- d) Oppgi regelen for Boyce-Codd NormalForm (BCNF), og forklar hva begrepene i denne regelen betyr.

## Oppgave 4: Transaksjoner og ACID – 10%

ACID-egenskapene har navnene: Atomicity, Consistency, Isolation og Durability. Disse begrepene brukes i sammenheng med transaksjoner i et DBMS.

- a) Forklar hva som menes med en transaksjon i DBMS sammenheng, og hva BEGIN TRANSACTION, COMMIT og ROLLBACK gjør.
- b) Forklar hva som menes med hver av de fire ACID-egenskapene: Atomicity, Consistency, Isolation og Durability.

**- Slutt på oppgavesettet –**

# Vedlegg: MySQL, utvalgte datatyper/syntaks

[] angir valgfrie elementer og | angir alternativer.

Dette er en noe forenklet syntaks-oversikt som forhåpentligvis skal hjelpe dere i oppgavene på denne eksamen.

## DATATYPER

CHAR( (lengde) )	Tekst med fast lengde, fra 0 til 255 tegn
VARCHAR( (lengde) )	Tekst med variabel lengde, fra 0 til 65 535 tegn
INT( (lengde) )	Heltall i området -2 147 483 648 til 2 147 483 647
ENUM( v1, v2... )	Definert verdisett
DATETIME	Tidsinformasjon som rommer både dato og klokkeslett.
DATE	Dato. Eks: '2018-11-08'
DECIMAL( A, B )	Desimaltall med A siffer og B desimaler . Eks: DECIMAL( 5, 2 ) kan romme tall i området -999.99 - 999.99

## FUNKSJONER

COUNT( * )	Antall
AVG( kolonne_navn )	Gjennomsnitt
SUM( kolonne_navn )	Sum
MIN( kolonne_navn )	Minimum
MAX( kolonne_navn )	Maksimum

## OPERATORER

=	Lik (ikke m/wildcards!)
<> eller !=	Forskjellig fra
<	Mindre enn
>	Større enn
<=	Mindre eller lik
>=	Større eller lik
LIKE	Lik, godtar wildcards
IN	I gitt utvalg
BETWEEN	Utvalg, følges av AND
_	Wildcard, enkelt tegn
%	Wildcard, flere tegn
IS [NOT] NULL	Null / ikke null

## LOGISKE OPERATORER

AND	Og
OR	Eller
NOT	Ikke

## SYNTAKS

### SELECT

```
SELECT      kolonnenavn [[AS] navn]
FROM        tabellnavn [[AS] alias]
[JOIN       tabellnavn [[AS] alias] ON felles nøkkel]
[WHERE      betingelse]
[GROUP BY   grupperingsuttrykk [HAVING betingelse]]
[ORDER BY   kolonnenavn]
```

### VIEW

```
CREATE [OR REPLACE] VIEW viewnavn [(alias)]
AS kriterier
```

### CREATE TABLE

```
CREATE TABLE tabellnavn
(
kolonnenavn datatype [UNIQUE|NOT NULL|DEFAULT|AUTO_INCREMENT|...],
...,
[[CONSTRAINT navn] PRIMARY KEY (kolonnenavn) ],
[[CONSTRAINT navn] FOREIGN KEY (kolonnenavn) REFERENCES tabellnavn (kolonnenavn) ]
)
```

### ALTER TABLE

```
ALTER TABLE tabellnavn
[ADD kolonnenavn datatype],
[CHANGE kolonnenavn_nå kolonnenavn_ny datatype_ny],
[DROP COLUMN kolonnenavn]
```

### DROP TABLE

```
DROP TABLE tabellnavn
```

### INSERT INTO

```
INSERT INTO tabellnavn
VALUES (verdi1, verdi2, verdi3, ...)
eller
INSERT INTO tabellnavn (kolonnenavn1, kolonnenavn2, ...)
VALUES (verdi1, verdi2, ...)
```

### UPDATE

```
UPDATE tabellnavn
SET kolonnenavn1 = verdi1, kolonnenavn2 = verdi2, ...
[WHERE betingelse]
```

### DELETE FROM

```
DELETE FROM tabellnavn WHERE betingelse
```