Obligatorisk oppgave 4

IN2090 | Filipcl

Oppgave 1 – Lage databaser

I denne oppgaven beskriver jeg de tre tabellene i kronologisk rekkefølge, hvor jeg redegjør for antagelser og valgene jeg har tatt samt tilhørende kode for å opprette databasene (DB).

DB tog:

Ved opprettelse av tog DB har jeg gjort noen få antagelser, den inneholder en startStasjon og en Endestasjon av typen text. Disse attributtene er satt til å være NOT NULL da et tog må ha en start- og ende-stasjon. Tilsvarende er gjort med ankomstTid da et Tog må ha en estimert ankomstTid, her antar jeg ved en mulig forsinkelse at ankomstTid kan oppdateres. Til slutt kommer attributtet togNr, den er satt til typen Integer med constraints som NOT NULL og UNIQUE. Dette er fordi vi vil laget et unikt heltatt slik at man enklere kan identifisere vært tog. Deretter er togNr også satt til å være en Primary Key, som lar oss nettopp gjøre dette.

Ved opprettelsen av togNr var jeg litt usikker på hvilke constraints jeg skulle bruke, i utgangspunktet tror jeg at det holder med å sette togNr til NOT NULL og PRIMARY KEY, for å gjøre den til en unik primær nøkkel. Men for å være sikker inkluderte jeg også UNIQUE constrainten.

På grunn av primær nøkkelen kan ikke tog med samme togNr gå mer enn en etappe om dagen. Fordi det kun kan bli lagt inn et tog med et unikt togNr.

```
CREATE TABLE tog (
    togNr int PRIMARY KEY NOT NULL UNIQUE,
    startStasjon Text NOT NULL,
    endeStasjon Text NOT NULL,
    ankomstTid Time NOT NULL);
```

DB togTabell:

Ved opprettelse av togTabell DB ble attributt stasjon satt til typen text med en NOT NULL constraint. Deretter har DB både togNr og avgangsTid som primærnøkler. TogNr skal også fungere som en fremmed nøkkel på tvers av alle DB'ene, derfor settes en REFERENCE til togNr i tog DB. Avgangstid blir satt til typen time med en NOT NULL constraint. For at kombinasjonene av disse attributtene skal være en primærnøkkel, kastes de til PRIMARY KEY.

```
CREATE TABLE togTabell (
    togNr int NOT NULL REFERENCES tog(togNr),
    avgangsTid Time NOT NULL,
    stasjon Text NOT NULL,
```

```
    PRIMARY KEY (togNr, avgangsTid)
    );
```

DB plass :

Beskriver om det er ledig plass i et tog og om det er vindusplass eller ikke. Attributtene vindu og ledig er av typen Boolean, som returnerer true eller false dersom det er ledig eller er vindusplass. Når det kommer til Primærnøkkelen i denne DB er det en kombinasjon av togNr, vognNr, plassNr og dato. Dette er for å kunne identiisere en enkelt plass i en vogn på et gitt tog den datoen. Tilsvarende som i togTabellen blir disse attributtene kastet til en PRIMARY KEY, hvor togNr fungerer som fremmednøkkel for å knytte DB'ene sammen.

```
CREATE TABLE plass (
    dato Date NOT NULL,
    togNr int NOT NULL REFERENCES tog(togNr),
    vognNr int NOT NULL ,
    plassNr int NOT NULL ,
    vindu boolean NOT NULL,
    ledig boolean NOT NULL,
    PRIMARY KEY (dato, togNr, vognNr, plassNr)
    );
```

Oppgave 2 – Fder og Normalformer

Oppgave A:

Gitt relasjonen – R (A, B, C, D, E, F, G) med FD'enen Q={CDE→B,AF→B,B→A,BCF→DE,D→G}.
For å identifisere kandidatnøkler begynner jeg må å identifisere hvilke attributter som ikke forekommer på høyre side:

CDE -> B
AF -> B
B -> A
BCF -> DE
D -> G

De eneste attributtene som ikke forekommer på høyre side er C og F, kombinasjonene CF alene er ikke en kandidatnøkkel. Men kombinasjonen CFA determinerer: (CFA) = AFCBDEG, og BCF determinerer: (BCF) = BCFDEAG. Disse to kombinasjonene determinerer alle attributtene i relasjonen R og er derfor kandidatnøkler. *(Jeg er usikker på om jeg har gjort riktig ved å si at CFA er k nøkkel vil gjerne ha feedback på dette)*

Oppgave B:

For å finne høyeste normalform relasjon R tilfredsstiller, sjekket jeg først om relasjonen oppfylte 1.NF. Alle verdiene i relasjonen er atomære og identiske, så relasjonen oppfyller 1.NF.

Deretter identifiserte jeg først alle kandidatnøklene og fant kandidatnøkkelen (BCF). BCF gir et sett med non-prime attributter som er (DEAG), her er attributt A -> B og G -> D. Det betyr at A og G kun er delvis funksjonell avhengig av kandidatnøkkelen (BCF) i Relasjon R noe som ikke tilfredsstiller 2.NF. Det vil si at den høyeste normalformen relasjon R tilfredsstiller uten dekomponering er 1.NF.

Oppgave C:

For å dekomponere relasjons R tapsfritt ved å ta utgangspunkt i FDen CDE -> B, må jeg sjekke hvilke normalformer relasjonene oppfylte. I forrige oppgaven oppdaget jeg at relasjonen kun oppfyller 1.NF. Dersom relasjonene skal oppfylle 2.NF må vi dekomponere attributtene A og G som kun er delvis funksjonelt avhengig av kandidatnøkkelen, til en egen tabell med relasjon til tabellen med kandidatnøkkelen.

Bildet under viser prosessen i dekomponeringen, førts finner jeg kandidatnøkkelen (k.key), deretter finner jeg non-prim-attributtene (NA). A og G er kun delvis funksjonelle avhengige av k.key, noe som gjør at relasjonen kun er 1.NF. For å tapsfritt dekomponere relasjonene til BCNF trenger deler jeg opp relasjonen i R1 , R2 og R3. Relasjon R2 og R3 inneholder kun funksjonell avhengighet som fungerer som en supernøkkel og er derfor av normalformen BCNF.

Tilsvarende er relasjon R1 nå også bestående av kun en funksjonell avhengighet som fungerer som sin egne supernøkkel og inneholder referanser (fremmed nøkkel) til de andre relasjonene via (B og D). Alle relasjonene oppfyller BCNF og er tapsfrie i form av at vi kan få relasjon R ved å bruke INNER JOIN.