dDB Excercise Week 4

Fra relationships til relations.

Nu når vi har fået vores skemaer på plads, kan SQL udtrykkene til konstruktion af relationerne laves. Det foregår ved at vi tager en 1-til-1 oversættelse af skemaerne, og sørger for at indarbejde de krævede constraints. De understregede koverteres til Keys eller Uniques, som er single value constraints. Da der udelukkende kan eksistere unikke tupler af disse.
Unique dog med undtagelsen af **NULL** værdier, da disse godt kan optræde for denne type.

Til stort set alle attributer er tilføjet en constraint med **NOT NULL** som ikke tillader indsættelse af en tuppel hvor disse attributer mangler. Enkelte steder, som f.eks. **level** attibutten, må gerne være **NULL**. For enkelte relationer er der således **CHECK**'s af om disse constraints er overholdt. Teoretisk virker de ligesom assertions, ved at værdier som ikke evaluere til **TRUE**, ikke tillades at blive indsat i relationen. I vores DBMS, som er MySQL med phpMyAdmin som frontend, lader disse constraints dog ikke til at blive overholdt. Ikke destomindre er de med.

Den første transformation er simpel:

```
Wagons(ID)
```

Skemaet konverteres så til en SQL-statement:

```
CREATE TABLE Wagons(
    wagon_id       INT PRIMARY KEY
);
```

Så er det Seats entiteten som konverteres. I denne har jeg i mit E/R diagram og skema en Boolean **isaSleep**, som så har en isa-relation til Sleep entiteten. Denne Sleep entitet har fire attributter, hvor af de tre bare er duplikater fra Seats relationen, og så attributten **level**. Den entitet er tydeligvis redundant, og derfor befandt **level** sig noget smartere under selve Seats entiteten. Og hvis der ikke specificeret en level (hvis den er **NULL**) så er det ikke en soveplads.

```
Seats(Letter, Row, SeatClass, isaSleep, wagon_id)
›  SeatClass: er enten '1. klasse' eller 'standard'
›  level: Soveplads niveau 1 eller 0, eller null hvis sæde
›  wagon_id: er togvognens unikke ID
```

Da **ROW** er en kommando i SQL er attributnavnet blevet **_row** og attributten **isaSleep** fra skemaet er byttet ud med **level**.

```
CREATE TABLE Seats(
    _row            INT NOT NULL,
    letter          CHAR(1) NOT NULL,
```

```
    wagon_id        INT NOT NULL REFERENCES Wagons(wagon_id),
    UNIQUE(_row, letter, wagon_id),
    seatclass       CHAR(10) NOT NULL,
    level           INT,
    CHECK(seatclass IN ('1. klasse','Standard')),
    CHECK(level IN ('0','1',NULL))
);
```

Så laver jeg et **INDEX** over sæderne, da de bliver refereret hver gang der bliver lavet et kald til NextToSeat relationen.

```
CREATE UNIQUE INDEX SeatIndex ON Seats(_row, letter, wagon_id);
```

Og så NextToSeat relationen, som dannes ud fra flg. skema:

```
NextToSeat(r_Letter, r_Row, r_wagon_id, l_Letter, l_Row, l_wagon_id)
›  r_: er sædet til venstre for
›  l_: er sædet til højre for
```

Alle keys er referencer til Seats relationen.

```
CREATE TABLE NextToSeat(
    r_row           INT NOT NULL,
    r_letter        CHAR(1) NOT NULL,
    l_row           INT NOT NULL,
    l_letter        CHAR(1) NOT NULL,
    wagon_id        INT NOT NULL,
    FOREIGN KEY(r_row, r_letter, wagon_id)
                    REFERENCES Seats(_row, letter, wagon_id),
    FOREIGN KEY(l_row, l_letter, wagon_id)
                    REFERENCES Seats(_row, letter, wagon_id),
    CHECK(NOT((r_row = l_row) AND (r_letter = l_letter)))
);
```

Derefter Stations relationen. Den havde jeg glemt at få med i mit skema, men da jeg begyndte at tænke over strukturen, indså jeg at den var nødvendig. Den har et skema som ser sådan he ud:

```
Stations(name, location)
```

Og relationen:

```
CREATE TABLE Stations(
    name            VARCHAR(50)  PRIMARY KEY,
    location        VARCHAR(50)  NOT NULL
);
```

Hver gang der skal bruges et startpoint eller endpoint skal der laves opslag i Stations, derfor laves der et **INDEX** dertil.

```
CREATE UNIQUE INDEX StationNameIndex ON Stations(name);
```

Herefter er det skemaet for Connections som har flg. skema.

```
Connections(Departure, Arrival, ChildRebate, OldRebate, Std.Rebate,
            StartName, EndName, wagon_id)
```

Det er jeg også i stand til at følge, og producere en relation ud fra. Da rabatterne er procentsatser, skal deres værdi befinde sig mellem 0 og 100. Og alle priserne mellem alle byerne er 42,- Jeg er ikke klar over om man logisk kan sammenligne `TIMESTAMP`'s, men da MySQL, som tidligere nævnt, ikke rigtigt retter sig efter disse tjeks, har det ikke noget problem med det.

```
CREATE TABLE Connections(
    departure        TIMESTAMP,
    arrival          TIMESTAMP,
    start_name       VARCHAR(50) REFERENCES Station(Name),
    end_name         VARCHAR(50) REFERENCES Station(Name),
    UNIQUE(departure, arrival, start_name, end_name),
    childrebate      INT NOT NULL,
    oldrebate        INT NOT NULL,
    std_rebate       INT NOT NULL,
    wagon_id         INT NOT NULL,
    CHECK(start_name <> end_name)
    CHECK(departure < arrival),
    CHECK( 0 >= (childrebate AND oldrebate AND oldrebate) <= 100)
);
```

E-Tickets er en simpel relation, med kun en ID attribut.

```
CREATE TABLE E_Tickets(
    eticket_id      INT PRIMARY KEY
);
```

Herefter Passengers relationen ud fra skemaet

```
Passengers(ID, Name, Age, OrdreNo)
›  Age: er et positivt tal
›  ID: er unikt pr. person.
›  OrdreNo: er de billetter som er købt i denne persons navn.
```

Med flg. relation

```
CREATE TABLE Passengers(
    id              INT NOT NULL PRIMARY KEY,
    name            CHAR(255) NOT NULL,
    age             INT NOT NULL,
    ordreno         INT NOT NULL REFERENCES E_Tickets(eticket_id)
);
```

Til sidst danner jeg relationerne ud fra mine relationships. Her er alle attributterne referencer fra mine entity sets.

```
CREATE TABLE Spans(
    departure       TIMESTAMP NOT NULL,
    arrival         TIMESTAMP NOT NULL,
    start_name      VARCHAR(50) NOT NULL,
    end_Name        VARCHAR(50) NOT NULL,
    FOREIGN KEY(departure, arrival, start_name, end_name)
                    REFERENCES Connections(depart, arrive, start_name,
                    end_name),
    eticket_id      INT NOT NULL REFERENCES E_Tickets(eticket_id)
);

CREATE TABLE RunsOn(
    wagon_id        INT NOT NULL REFERENCES Wagons(wagon_id),
    departure       TIMESTAMP NOT NULL,
    arrival         TIMESTAMP NOT NULL,
    start_Name      VARCHAR(50) NOT NULL,
    end_Name        VARCHAR(50) NOT NULL,
    FOREIGN KEY(departure, arrival, start_name, end_name)
                    REFERENCES Connections(depart, arrive, start_name,
                    end_name)
);

CREATE TABLE Reservations(
    eticket_id      INT NOT NULL REFERENCES E_Tickets(eticket_id),
    departure       TIMESTAMP NOT NULL,
    arrival         TIMESTAMP NOT NULL,
    start_Name      VARCHAR(50) NOT NULL,
    end_Name        VARCHAR(50) NOT NULL,
    FOREIGN KEY(departure, arrival, start_name, end_name)
                    REFERENCES Connections(depart, arrive, start_name,
                    end_name),
    _row            INT NOT NULL,
    letter          CHAR(1) NOT NULL,
    wagon_id        INT NOT NULL,
    FOREIGN KEY(_row, letter, wagon_id)
                    REFERENCES Seats(_row, letter, wagon_id)
);
```

Jeg synes det er blevet til noget værre mudder. MySQL understøtter ikke **`REFERENCES`** til at opretholde integriteten med. Og MySQL v.4 har heller ikke **`TRIGGERS`**, så jeg kan på ingen måde opretholde en konsistent database. Og ligeledes er det noget af et møjsomt arbejde at tilføje og ændre på de forskellige relationer og attributter.

Queries:

De queries der blev oprettet som relationel algebra, bliver som følger når de er oversat til SQL:

1) *How many seats are still available (i.e. not booked) on today's 10am direct connection between Vejle and Fredericia?*

```
SELECT COUNT(Connections.wagon_id) AS FreeSeats
FROM Connections
NATURAL JOIN Seats
WHERE
(
_row != (
      SELECT _row
      FROM Reservations
      WHERE start_Name = 'Vejle'
      AND end_Name = 'Fredericia'
      AND departure 10am $TODAY
      AND wagon_id = Connections.wagon_id )
OR letter != (
      SELECT letter
      FROM Reservations
      WHERE start_Name = 'Vejle'
      AND end_Name = 'Fredericia'
      AND departure = 10am $TODAY
      AND wagon_id = Connections.wagon_id )
)
```

Tidsbegrænsningerne er pseudosyntaks i det ovenstående og de efterfølgende, da jeg ikke lige ved hvordan MySQL sammenligner `TIMESTAMP` typer.

2) *Find all ways together with their price to reach Aalborg from Frederikshavn today with at most one stop in between.*

```
SELECT Depart.start_name AS _FROM,Arrive.start_name AS
_STOP,Arrive.end_name AS _TO
FROM Connections AS Depart, Connections AS Arrive
WHERE Depart.start_name = 'Aalborg' AND
      Depart.end_name = Arrive.start_name AND
      Arrive.end_name = 'Frederikshavn' AND
      Depart.departure = $TODAY AND
      Arrive.arrival = $TODAY AND
      Depart.departure < Arrive.arrival AND
      Depart.wagon_id = Arrive.wagon_id
```

3) *When should I leave Aarhus the latest if I want to be in Randers before 2PM today with at most one stop?*

```
SELECT max(A.departure)
FROM Connection AS _FROM, Connection AS _TO
WHERE _FROM.start = "Aarhus"
AND _FROM.end = _TO.start
AND _TO.end = "Randers"
AND _TO.arrival < 2pm $TODAY
```

4) *Find 2 neighbouring seats (if possible) all the way along the following route: Aarhus-Vejle-Kolding anytime today provided the waiting time in Vejle is less than 1 hour.*

```
SELECT *
FROM Connections as A, Connections AS B
WHERE
      A.start = "Århus" AND
      A.slut = B.start = "Vejle" AND
      B.slut = "Kolding" AND
      (B.departure - A.arrival) < 1 hour
      .
      .
      .
      .
      _
```

Nummer 4 her kunne jeg simpelthen ikke få til at lykkes. Men det jeg gerne ville var, at finde alle sæderne fra i alle vognene fra Århus til Vejle. Og dernæst joine med relationen NextToSeat for at få de sæder som er naboer dertil. Dernæst ville jeg finde ud af hvilke, hvis nogen, der var optagede, og vælge dem fra.

Nye Queries

1) *Give the total number of tickets sold for each destination reachable from Vejle during the next 7 days. Sort them in decreasing order of price and include only destinations that are not fully booked.*

```
SELECT Reservations.start_Name,COUNT(*) AS n
FROM Connections
INNER JOIN Reservations
USING (wagon_id, departure, arrival, start_Name, end_Name)
WHERE Reservations.start_Name = 'Vejle'
AND n < 4
GROUP BY start_Name
```

2) *Give a list of stations with their full names, their total number of arrivals, and total number of departures during next week. Order them in lexicographical order.*

```
SELECT DISTINCT start_name AS
station,
      (SELECT COUNT( * ) AS number_of_departures
      FROM `Connections`
      WHERE start_name = station) AS
starts_from,
      (SELECT COUNT( * ) AS number_of_arrivals
      FROM `Connections`
      WHERE end_name = station) AS
ends_at
FROM Connections
ORDER BY station
```