Aflevering 5 - 10 ECTS

Exercise (1 - BCNF)**.**

Consider the relation $R = \{A,B,C,D,E,F,G,H,I,J\}$ and the set of functional dependencies,

$$\begin{aligned} F = \{&\{A,B\} \to \{C\},\\ &\{A\} \to \{D,E\},\\ &\{B\} \to \{F\},\\ &\{C\} \to \{B\},\\ &\{F\} \to \{G,H\},\\ &\{D\} \to \{I,J\}\}. \end{aligned}$$

a. Decompose R to BCNF.

We start from where we finished in the previous handin, with the following decomposed relations,

$$\begin{aligned} R_{11} &= \{A,D,E\}, && \text{key } A\\ R_{12} &= \{D,I,J\}, && \text{key } D\\ R_{21} &= \{B,F\}, && \text{key } B\\ R_{22} &= \{F,G,H\}, && \text{key } F\\ R_3 &= \{A,B,C\}, && \text{key } A,B. \end{aligned}$$

Lets remind ourselves of the conditions that BCNF have to satisfy,

BCNF:

For all $X \to B \in F^+$

$$B \subseteq X \quad \text{or}$$
$$X \text{ is a superkey.}$$

We see that this is satisfied for $R_{11}, R_{12}, R_{21}, R_{22}$, the only functional dependencies we have here, are the ones given by the primary keys. (We also see that this definiton of a trivial dependency differs from the one we saw earlier...). R_3 on the other hand violates the requirement, as $\{C\} \to \{B\}$ is neither trivial nor a key. To fix this we use algorithm 15.5,

Algorithm 15.5

Find a functional dependency $X \to Y$ in R that violates BCNF and replace R by two relation schemas $(R-Y)$ and $X \cup Y$.

Applying this, we get two new relations,

$$\begin{aligned} R_{31} &= \{C\} \cup \{B\} = \{B,C\} \quad \text{key } C\\ R_{32} &= R_3 - \{B\} = \{A,C\} \quad \text{key } A,C. \end{aligned}$$

These relations both satisfy BCNF.

b. Is your decomposition lossless for BCNF? Please provide your steps.

The argument is completely analagous to the ones I made in the previous handin. Lets recall what lossless means,

Property NJB

A Decomposition $D = \{R_1, R_2\}$ is lossless with respect to a set of functional dependencies F on$R$ if,

$$(R_1 \cap R_2) \to (R_1 - R_2) \in F^+ \quad \text{or}$$
$$(R_1 \cap R_2) \to (R_2 - R_1) \in F^+.$$

We essentially have to check all combinations. In the end we see that it holds for all the relations in our decomposition. I'll just show a few for good measure,

$$(R_{31} \cap R_{32}) = \{C\} \to \{B\} = (R_{31} - R_{32}) \qquad \text{is in } F^+$$
$$(R_{31} \cap R_{11}) = \{A\} \to \{D, E\} = (R_{11} - R_{31}) \quad \text{is in } F^+.$$

3. Is your decomposition dependency preserving?

In our final decompostion $R_3 \to R_{31}, R_{32}$, we lost the functional dependency $\{A, B\} \to \{C\}$. There is no way to recover this so,

$$\bigcup_i F_{R_i} \neq F.$$

The functional dependencies on our decomposed relations do not combine to yield the original functional dependencies. Therefore our decomposition is not dependency preserving.

Exercise (2 - Multi-valued Dependency)**.**

Check whether the following multi-valued dependencies,

$$\text{star\_name} \to\to \text{street}, \text{city}$$
$$\text{movie\_title} \to\to \text{star\_name}.$$

hold for the current data.

When we have some multivalued dependency (MVD) $X \to\to Y$ it means that Y is allowed to vary independently of the other attributes in R. So for the dependency to hold, it is required that whenever we see some pair of tuples (x_1, y_1, z_1) and (x_1, y_2, z_2), we should also see a pair where the ys have been interchanged. The star_name $\to\to$ street, city, does make sense intuitively. We would expect the street and city of properties owned by an actor to be independent of any movies they have acted in. We do in fact see this in the data. Mark Hamill has two adresses and has appeared in two movies, and the data is repeated as it should be. The remaining actors, have eiher only a single address or been in a single movie, so they satisfy the dependency trivially. The MVD then holds for the data.
For the second dependency movie_title $\to\to$ star_name, we see that we have tuples
Since for a given movie, actor names should be independent of the other attributes, we would expect to see a tuple (Carrie Fisher,..., Star Wars, 1980). Since we don't the MVD is violated.

Table 1.1 Excerpt of the data.

star_name	...	movie_title	year
Mark Hamill	...	Star Wars	1977
Mark Hamill	...	Star Wars	1980
Carrie Fisher	...	Star Wars	1977

Exercise (3 - NGO)**.**

As we noticed, our NGO Database was suffering from redundancy as the same person would be inserted multiple times if they support multiple NGOs, and thereby we could potentially have update and insertion anomalies. Therefore, we create two new relations, Supporter

a - Update. Update level, as the number of donations the supporter made to the NGO, for all entries in Supports using a single update command.

I made use of the following command,

```
UPDATE Supports AS s
SET s.level = (
    SELECT COUNT(d.amount)
    from Donations AS d
    WHERE d.email = s.email AND d.ngo_name = s.ngo_name
);
```

We use UPDATE to specify the relation we are changing, and then SET to specify level. We then count the number of donations by counting the number of rows in donation the email and ngo match.

b - Trigger. Ensure that the 'level' of an NGO supporter is updated correspondingly whenever they make a donation to the NGO. Remember that level is defined to be the number of times a supporter has donated to the NGO. We assume that you cannot undo a donation, and therefore level cannot decrease.

```
CREATE TRIGGER UpdateLevel
    AFTER INSERT ON Donations
    REFERENCING NEW AS nD
    FOR EACH ROW
BEGIN
    UPDATE Supports as s
    SET s.level = s.level + 1
    WHERE nD.email = s.email AND nD.ngo_name = s.ngo_name);
END
```

We create a trigger that fires on insertions into Donations. We then increment the level of the supporter whos email and ngo_name match the ones in the inserted donation.

c - Indexes. Create an index on Supports on the attribute ngo_name, so it is faster for an NGO to retrieve its supporters.
- What type of index will this be (primary or secondary)? if secondary, will it be dense, clustering or not?
- Let's make an assumption that this index is a 2-level clustered B-tree index; to find all the supporters of KDG, how many I/O operations would I need to do using this index? Describe what other assumptions you had to make to come up with this answer.

The SQL command for this is very simple,

```
CREATE INDEX index_ngo_name
ON Supports (ngo_name);
```

Since the primary key for Supports is (ngo_name, email), this index is a secondary index, as it does not contain the entire primary key. From my understanding, mySQL creates an index for the primary key, and orders the data on the disk according to this index. As a result the data isnt ordered according to our new secondary index. Since it isnt ordered, we cant do clustering, so the index will have to be dense. We instead assume that it is a 2-level clustered B-tree index. Then our tree has a root level and a leaf level. We start by reading the root node, this is 1 I/O. We then find leaf-node that contains our KDG data, and fetch the data, this i 1 I/O more. Assuming the lead nodes are large enough to contain all of the KDG data, we only have to read a single leaf node. Under these assumptions, it takes a total of 2 I/O operations to read the data.