Homework 3: Design Improvements

Björn Þór Jónsson

Introduction to Database Design, Spring 2019

1 Normalization – Basic Examples

1.1) Consider a table $R(\underline{A}, \underline{B}, C, D, E)$ with the following dependencies:

$$\begin{aligned} AB &\rightarrow C \\ A &\rightarrow DE \\ C &\rightarrow B \end{aligned}$$

Select the true statements:

(a) AB is the only key of R.

(b) $AB \rightarrow C$ is a trivial functional dependency.

(c) Normalizing to 3NF (but not to BCNF) results in exactly two relations.

(d) The relation can normalized to BCNF without losing dependencies.

1.2) Consider a table $R(\underline{A}, B, C, D, E)$ with the following dependencies:

$$\begin{aligned} C &\rightarrow D \\ C &\rightarrow A \\ D &\rightarrow E \\ A &\rightarrow BCDE \end{aligned}$$

Select the true statements:

(a) A is the only key of R.

(b) $BCD \rightarrow D$ is an unavoidable functional dependency.

(c) Normalizing to BCNF results in exactly two relations.

(d) The relation can normalized to BCNF without losing dependencies.

1.3) Consider a table $R(A, B, C, \underline{D}, \underline{E})$ with the following dependencies:

$$\begin{aligned} A &\rightarrow B \\ DE &\rightarrow ABC \\ DE &\rightarrow D \\ A &\rightarrow C \end{aligned}$$

Normalize R to BCNF and write down the resulting relations here:

2 Index Selection

Consider the following relation with information on parts (*stock* is the quantity in stock):

Part (id, descr, price, stock, ...)

For each of the queries below, select the index(es) that a good query optimiser is likely to use to process the query. You may select 1 or more options (including "no index"). Assume that all indexes are non-clustered.

(a) Part(id)

(b) Part(stock)

(c) Part(price)

(d) Part(stock, price)

(e) Part(stock, price, id)

(f) Part(stock, price, descr)

(g) No index

The queries are:

2.1) Query 1

```
select id
from Part
where stock > (select max(price) from Part);
```

2.2) Query 2

```
select id, descr
from Part;
```

2.3) Query 3

```
select stock
from Part
where price = 23;
```

2.4) Query 4

```
select id, descr, price
from Part
where stock > 35;
```

3 Normalization – Detailed Examples

Attached to the quiz is a script to create and populate five independent relations, each of which has seven columns and a primary key, with a few thousand rows. For this project, though, we only consider the relations Rentals and Boats.

Each relation models a potential real-life database design situation, but with some design problems that must be addressed. In short, each of the relations has embedded a set of functional dependencies. You must a) find these dependencies and b) use them to guide decomposition of the relations. For this project, though, we only consider the 3NF and BCNF normal forms based on functional dependencies.

For each relation, take the following steps:

1. Find all the FDs in the relations, given the constraints and assumptions above.

   Note: We strongly recommend to make a script in your favourite programming language to generate the SQL script, based on a list of column names, and then run this script using `psql`.

2. Decompose the relation until each sub-relation is in BCNF, or in 3NF but not BCNF, while preserving all non-redundant FDs. Write down the results schema description in a simple Relation(columns) format.

3. Write the detailed SQL commands to create the resulting tables (with primary keys and foreign keys) and populate them, by extracting the relevant data from the original relations.

   Note: In this homework, create a new relation also for the "original" relation, so that you can compare the decomposition result with the original relation. Thus, you do not need to alter any table at any point.

4. Select the correct normal form for the decomposed schema.

In this project, assume that the following simplifying assumptions hold for each of the relations (this is the "reality" that you check them against):

- The relations must each be considered in isolation. The columns have short names that hint at their meaning, but you should not count on implicit FDs derived from the expected meaning of the columns. In short, the column names may trick you!

- Assume that all functional dependencies in each relation (aside from primary key dependencies and dependencies derived from that) can be found by checking only FDs with one column on each side, such as $A \rightarrow B$.

  Note: As discussed in lectures, you can use SQL queries to detect potential FDs. Using the assumptions above, it is indeed relatively simple to create a script to output queries for all possible checks for FDs, which you can then run with `psql`, thus automating the detection process.

- If you find a functional dependency that holds for the instance given, you can assume it will hold for all instances of the relation.

  Exception: When an ID column and a corresponding string column both determine each other, consider that only the ID column determines the string column, not vice versa. For example, if both $CID \rightarrow CN$ and $CN \rightarrow CID$ are found to potentially hold, then consider that only $CID \rightarrow CN$ is valid.

- The only dependencies you need to consider for decomposition are a) the dependencies that can be extracted from the data based on the assumptions above, and b) the given key constraints.