T-202-GAG1: Project 5

1 Readings and Lectures

Ramakrishnan & Gehrke: Chapter 19. Lectures: V09A and V10A.

2 The Project

The goal of this project is normalization: to transform a set of relations into the highest normal form possible, while preserving all existing dependencies.

You are given a script (CREATE.sql) to create four independent relations (Persons, Boats, Projects and Courses), each of which has six or seven columns and a primary key. You are also given a script (FILL.sql) to fill the four relations with data; on average each relation contains about eight thousand rows.

All four relations model potential real-life database design situations, but with some design problems that must be addressed. In short, each of the four relations has embedded a set of functional dependencies and/or multi-valued dependencies. You must a) find these dependencies and b) use them to guide the decomposition of the relations.

3 Assumptions

For this project, we consider 1NF, 2NF, 3NF, BCNF (based on functional dependencies) and 4NF (based on multi-valued dependencies).

Persons, Projects and Courses model a fairly straightforward situation from real life, but a note is in order about the Boats relation. For the purposes of the ship and boat registry, Iceland was originally divided into several regions and each region is assigned a unique two-character code. Examples are RE for “Reykjavík”, AE for “Akureyri and Eyjafjörður”, and ÞH for “Þingeyjarsýslur and Húsavík”. Each boat was then assigned a number, which was unique within that region; together the location code and the number thus formed a key. For example, the two boats I worked on as a teenager had identifiers ÞH202 and ÞH51. Note that while the regional code generally covers larger areas than zip codes, no zip code crosses regional code boundaries. The Boats relation models a registry design based on these requirements. (Note that while these codes still exist, all ships and boats are now also assigned a unique ID.)

Furthermore, in this project the following simplifying assumptions hold for each of the relations:

- The four 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 → B.
- Assume that all multi-valued dependencies have only one column on the right side. The left side may have more than one column, however. An example is AB →→ C. (Recall the MVDs only occur in relations with at least three columns in all keys.)
- Assume that no join dependencies are embedded in the relations; we do not consider 5NF in this project.
- If you find a (functional or multi-valued) dependency that holds for the instance given, assume it will hold for all instances of the relation.
- 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.

As discussed in lecture V10A, you can use SQL queries to detect potential FDs and MVDs. Using the assumptions above, you can indeed create a script to generate **all** possible checks for FDs and MVDs, thus automating the detection process.

4 Normalization Process

For each relation, you must determine the following:

1) All the (functional and multi-valued) dependencies represented in the relation (based on the assumptions above);
2) All the keys of the relation (based on the detected dependencies);
3) The minimal cover of the set of functional dependencies; and
4) The current normal form of the relation.

Then you must decompose the relation until each sub-relation is in either 3NF or BCNF, as well as in 4NF, while preserving all FDs of the minimal cover.

For each relation resulting from the decomposition, you must determine the same four items as above.

You must document all these steps and outcomes and write a report. Then you must create the resulting tables and populate them, by extracting the relevant data from the original relations.

5 Naming and Filling Relations

For easier grading, you must assign unique and consistent names to decomposed relations, which can then be easily and consistently checked. Here is how!

Each relation that you create must be named by concatenating the name of the original relation and the names of all the columns it contains, with "_" between, in creation order. For example, if the original relation Person was partially

decomposed into relation containing the columns ID, DID and DN, the resulting relation must be named Persons_ID_DID_DN.

To populate the new relations, use the INSERT INTO … SELECT … notation in SQL (e.g., see http://www.w3schools.com/sql/sql_insert_into_select.asp).

6 Deliverables

The project is a group project, with three (or at minimum two) students per group. The deadline is at 23:59 on Thursday November 3.

Late submissions will not be accepted, so make sure to submit your solutions on time. And note that one-student "groups" are not allowed!

Submit three files:

a) A PDF file (REPORT.pdf) describing, for each relation, the normalization process, as described in Section 4 above.

b) A text file (CREATE.txt) containing SQL commands to create the resulting database tables.

c) A text file (FILL.txt) containing SQL commands to fill the resulting database tables from the tables in the original database.

Correctly normalizing and clearly reporting on each of the four tables gives 25% of the final grade.