Database Management Systems, Spring 2024

Homework 5: Practical Normalization

by Björn Þór Jónsson & Hildur Davíðsdóttir

1 Introduction

The goal of this homework assignment is to train you in normalization based on functional dependencies. To that end, we give you 5 relations with a variety of normalization issues, and your task is to improve their design. We solve one of the relations to give you a template for the solution; the remaining four relations are for you to normalize. If you have any questions, please ask us on Piazza.

2 Deliverables

This project is a group project, with **3** students per group (you must form your own groups). The deadline is at **23:59 on Thursday, April 11**. Individual submissions will not be accepted unless you have been granted permission by the teachers via email to bjorn@ru.is or hildurd@ru.is. Late submissions will **not be accepted.** Each group must submit two files to **Gradescope**: An SQL file named **DDL.sql** and a PDF file named **Report.pdf**.

The DDL file must run as a whole and not produce any extra output. If you wish to have some text inside the SQL file, for example for labelling items or for clarification, you must have that as comments. A template for both files is given in Section 5.

3 Instructions

Each relation in the SQL script models a potential real-life database design situation, with some design flaws that must be addressed. In short, each of the relations has embedded a set of functional dependencies (FDs), which are not stated explicitly. You must a) find these dependencies and b) use them to guide the decomposition of the relations to 3NF/BCNF normal forms, using the methodology covered in class.

3.1 Steps (for each relation):

1. Find all the important FDs in the relation, given the constraints and simplifying assumptions that are listed in detail below.
   Note: We strongly recommend using your favourite programming language to generate an SQL script with all the FD detection queries, using the SQL query template covered in slide 46 of the lecture. The program could take as input a list of column names, and output all the required SQL queries into a text file. You can then run the text file using psql. Below is a Python example of such a generator.

```python
from itertools import permutations


query_template = """
    insert an FD detection query template here
"""


tables = [
    ("Relation A", ("Attr1", "Attr2", "Attr3", "Attr4")),
]



def main():
    with open("CHECKFD.sql", "w") as file:
        for table, attributes in tables:
            for a, b in permutations(attributes, 2):
                query = query_template.format(R=table, A=a, B=b)
                file.write(query)

if __name__ == "__main__":
    main()
```

2. Decompose the relation until each sub-relation is in 3NF/BCNF, while preserving all important FDs. Write down in your report the resulting schema description in a simple Relation(columns) format shown in the template below, with the primary key of each relation underlined.

3. Describe the highest normal form of each resulting relation of the decomposed schema in your report. This will be either 3NF or BCNF.
4. Write the detailed DDL to create the resulting tables (with primary keys and foreign keys) using the method used in the template shown below.
5. Populate the new tables, by extracting the relevant data from the original relation.

4. Simplifying assumptions

In this project, assume that the following simplifying assumptions hold for each of the relations:

- 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 FDs 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.
- If you find an FD 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 → CN and CN → CID are found to potentially hold, then consider that only CID → 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. As covered in the lecture, you can ignore trivial, unavoidable, and redundant FDs in the normalization process.

5. Template solution

Here, we outline the information needed in your solution files for each relation. We do that by showing the solution information needed for the Project relation. Note that the example solution presents a very simple naming convention for relations and does not rename attributes; you should follow the same approach. Using other naming conventions or creatively naming relations **makes grading harder and will result in a lower grade**.

For reference, the original Project relation is as follows:

Project(ID, PID, SID, SN, PN, MID, MN)

5.1 PDF report

The PDF file should contain one section for each relation, with subsections with (a) information on the FDs that are deemed important for normalisation, and (b) the outcome of the normalisation process. For the Project relation, this would be as follows:

1. Projects relation

<u>1.1 Important FDs:</u>

PID → PN
SID → SN
ID → MID
MID → MN

<u>1.2 Decomposition and Normal Form:</u>

Projects1(<u>MID</u>, M) in BCNF
Projects2(<u>ID</u>, MID) in BCNF
Projects3(<u>PID</u>, PN) in BCNF
Projects4(<u>SID</u>, SN) in BCNF
Projects0(<u>ID</u>, <u>PID</u>, <u>SID</u>) in BCNF

Naming convention:

- For every new table: Original table name + number starting from 1
- For the remains of the original table: Original table name + 0

5.2 SQL file

For each relation, the DDL file must contain commands to (a) create the new decomposed relations and (b) fill them with data. The relations should be created and then filled in the same order as the decomposed relations in the report. The SQL file as a whole must run without errors using psql. For the Project relation, the SQL text would be as follows:

```
-- Projects
CREATE TABLE Projects1 (
  MID INTEGER NOT NULL,
  MN VARCHAR(50) NOT NULL,
  PRIMARY KEY (MID)
);

CREATE TABLE Projects2 (
  ID INTEGER NOT NULL,
  MID INTEGER NOT NULL,
  PRIMARY KEY (ID),
  FOREIGN KEY (MID) REFERENCES Projects1 (MID)
);



CREATE TABLE Projects3 (
    PID INTEGER NOT NULL,
    PN VARCHAR(50) NOT NULL,
    PRIMARY KEY (PID)
);

CREATE TABLE Projects4 (
    SID INTEGER NOT NULL,
    SN VARCHAR(50) NOT NULL,
    PRIMARY KEY (SID)
);

CREATE TABLE Projects0 (
    ID INTEGER NOT NULL,
    PID INTEGER NOT NULL,
    SID INTEGER NOT NULL,
    PRIMARY KEY (ID, PID, SID),
    FOREIGN KEY (ID) REFERENCES Projects2 (ID),
    FOREIGN KEY (PID) REFERENCES Projects3 (PID),
    FOREIGN KEY (SID) REFERENCES Projects4 (SID)
);

INSERT INTO Projects1
SELECT DISTINCT MID, MN
FROM Projects;

INSERT INTO Projects2
SELECT DISTINCT ID, MID
FROM Projects;

INSERT INTO Projects3
SELECT DISTINCT PID, PN
FROM Projects;

INSERT INTO Projects4
SELECT DISTINCT SID, SN
FROM Projects;

INSERT INTO Projects0
SELECT DISTINCT ID, PID, SID
```

```
FROM Projects;
```