COSC6340: Database Systems
Project: Database Normalization in SQL

Instructor: Carlos Ordonez

1 Introduction

You will develop a Java program that generates SQL code to certify normal forms (NF), or normalize a database, for a set of tables (relation) and a potential "candidate key" for each table (unknown if it is indeed a candidate key). If the normalization option is selected, you will have to propose a lossless decomposition. That is, initially the user is trying to explore if a given set of columns are a candidate key or not. If the answer is yes, then the program tells the user if the table is already normalized with respect to that key, if the answer is no, then the program must propose a lossless decomposition for the normalization option, or just answer no for the certification option. This kind of tool is useful when the data modeler has no clear idea about the contents of columns in big tables, and requires to modify the tables to be in 3NF. This program is a single phase project. However, you are welcome to ask the TA for feedback of your project earlier than the due date.

2 Program requirements

Your program will generate SQL code based on input parameters defined in a text file. Basically, you will initially have to check normal forms against a given candidate key and several nonkey (nonprime) attributes. You can assume the "candidate" key has at most two attributes and there are at most 10 nonkey (nonprime) attributes. As another option in the program, you maybe required to create new tables to propose a normalized database based on the NF validation.

At the minimum, your program must certify NFs correctly to get a passing grade in this project. Full credit is obtained by getting the certification and normalization sections correct.

1. For 1NF, you must check duplicates and a clean key. No atomic value checking since a table in the DBMS will be given as input. The given key may be composite. Therefore, every attribute must be clean. Bear in mind you are given a candidate key and the table itself already has another valid primary key. However, this "candidate key", proposed by the user, may have duplicates or nulls. In short, you are checking whether you have a potential new key. If the validation fails, and the decomposition option is selected, you will have to remove the null and duplicate rows in a new table.

2. For 2NF check partial functional dependency (FD) on the candidate key. If there is a simple key given then, the table is in 2NF. If you have a composite key, you must analyze every subset of columns for partial FDs against every subset of the composite key. Rows with nulls should be skipped. You can assume the candidate key has at most 2 attributes. If the validation fails, and the decomposition option is selected, you should create additional tables to propose the lossless decomposition in 2NF. This is accomplished by moving the defined attributes of the source table into a new table, and using the subset of the candidate key as the new primary key of this table.

3. For 3NF, as a simplification, we will only require you to check all the transitive dependencies between single nonkey attributes. That is, all transitive (if any) FDs of the form $A \rightarrow B$, where A and B are columns in your input table, should be analyzed. As in the 2NF, you maybe required to create additional tables, using a similar strategy, to propose a lossless decomposition that is in 3NF.

2.1 Programming details

1. Your program will have an input parameter "option" to specify if it is required only a certification of the NFs, or it is required to perform a lossless decomposition (1=certify;2=decompose).
2. You need to use JDBC to submit queries, but your program should be able to generate an SQL script file for all the queries used in your program. The user must provide an input text file and an evaluation option parameter (certification or decomposition). If any of these parameters are missing, your program should return an error message.
3. Always remember to drop any derived table in advance.
4. Your program should store the final decomposition tables in the text file called like your input file with a suffix (_3NF).

2.2 Theory

Remember a relation in 3NF is also in 2NF and 1NF and a relation in 2NF in also in 1NF. Therefore, if the program is only certifying, when a lower NF is not satisfied the program should not continue checking further NFs (since that would be unnecessary). If the program has to perform lossless decomposition, and a table is not in 1NF, you will have to create a new table in 1NF, and proceed with it for the rest of the validations (2NF and 3NF) for the derived table. Note that a table that is not in 2NF, will require derived tables to be created, and these derived tables should be also in 3NF. Your program will get just one key without specifying other (secondary) candidate keys. You are required to check normal forms based on that key only. SQL code generation: You can use any combination of SELECT statements, temp tables and views. You can name your tables/columns any way you want, and assume unique column names in the database. The only constraints that you have are not to modify the input table in any way and the output table should be called "NF". The additional tables created as a result of a lossless decomposition should be named as the original table and a suffix "_#", where # is the number of the derived table.

3 Program call and Output

Here is a specification of input/output.

- syntax for call: "java Normalize database=dbxyz.txt;option=1".
- Example of input file (dbxyz.txt):

```
Employee(employeeid(k),name,salary)
Department(deptid(k),subdeptid(k),deptname,deptSalesAmt)
```

- Option 1 requires a lossless decomposition and option 2 only a NF certification. If the connection to the DBMS fails, you have to send an error message. If the connection to the DBMS is successful, you should generate the "NF.sql" file regardless the selected option, and dump the SQL generated through your program.

- The SQL of your generated script should be properly formated and indented.
- A summary result table, called NF, with 5 rows should be created in the DBMS. The first column should specify the table name, the second column should include the evaluated NF, the third column should say if the table was or not in the specified form, the following column should state the reason of why the form was violated, and the final column should include your lossless decomposition suggestion. You should include all violations (define the column as char(200)), and evaluation of the derived tables if required. Call the output table "NF" and you can do a "SELECT * FROM NF;" as your last statement.
- Use JDBC to control processing of each normal form. Display the final NF certification or normalization getting the results with JDBC and displaying them on the screen.
- You can include an optional GUI, but output in simple text form is the only requirement. You can get +1 if you include a nice/intuitive GUI (with a README file). Although, you can only get this extra credit if your program performs at least the certification section correctly.

4 Examples of input and SQL

The input is one table R with PK i and a composite "candidate" key. Notice i is NOT considered in the normalization certification, but it is used just to have a practical valid PK.

R

i	K1	K2	A	B
1	1	1	1	0
2	2	1	1	0
3	3	2	2	0
4	4	2	2	0
5	1	2	2	0
6	3	1	1	0

If the program is called with "java Normalize database=dbxyz.txt;option=2", then the input/output for R is:

```
dbxyz.txt
--------------------------------------
R(K1(k),K2(k),A,B)
```

NF

Table	Form	Y_N	Reason	Lossless decomposition
R	1NF	Y		
R	2NF	N	$K2 \rightarrow A$, $K2 \rightarrow B$, $K1 \rightarrow A$	*R_1(K1(k),K2(k)), R_2(K2(k),A), R_3(K2(k),B)
R_1	1NF	Y		
R_1	2NF	Y		
R_1	3NF	Y		
R_2	1NF	N	Duplicates	R_2_1(K2(k),A)
R_2_1	1NF	Y		
R_2_1	2NF	Y		
R_2_1	3NF	Y		
R_3	1NF	N	Duplicates	R_3_1(K2(k),B)
R_3_1	1NF	Y		
R_3_1	2NF	Y		
R_3_1	3NF	Y		

*Note: It is important to appreciate that depending on the strategy followed during the development of your program, you may have alternative (but correct) tables with a lossless decomposition in each step. For example: $R\_1(K1(k), K2(k))$, $R\_2(K2(k), A)$, $R\_3(A(k), B)$ is another valid final decomposition, or in the second step, R can also be decomposed into $R\_1(K1(k), K2(k))$ and $R\_2(K2(k), A, B))$ -although R_2 is not in 3NF-. *Remember* that a lossless decomposition is the one that can obtain the original table without losing any records. Therefore, in our previous example, the R_1 table is required, despite the fact, that it does not contain any nonkey attributes.

```
dbxyz_3NF.txt
--------------------------------------


**************************************
Normalization for R
**************************************

R_1(K1(k),K2(k))
1,1
2,1
3,2
4,2
1,2
3,1

R_2_1(K2(k),A)
1,1
2,2

R_3_1(K2(k),B)
1,0
2,0
```

The following table S, which only requires a NF certification, is in 3NF (option=1;input file:S(C(k),D)) and the result table should have YES for 1NF, 2NF and 3NF:

S

i	C	D
1	1	0
2	2	1
3	3	0

A simple SQL script that simply computes the number of rows, insert the distinct values of the primary key from table R into another table E (we assume this table is empty and exists), and computes the number of rows of this new table is below. This script is generated using the user-specified parameters. The program simply concatenates parameters and SQL keywords to automatically form SQL statements.

```
SELECT count(*)
FROM employee;

INSERT INTO E
   SELECT distinct employeeId
   FROM employee;

SELECT count(*)
FROM E;
```