User Manual

for the

Database Normalizer

application

Elmar Jürgens 2002 - 2004
juergens@in.tum.de

Introduction

About the DATABASE NORMALIZER 

The Database Normalizer (DN) is an application that works with functional dependencies to compute normalization properties of relational database schemas.

It can determine the normal form a schema is in and compute candidate keys and equivalent tuples.
In addition to these analysis features it implements a synthesis algorithm that can create relational schemas from functional dependencies that are guaranteed to be in third normal form (3NF) and contain a minimal set of relations.

It's intended fields of use are twofold: In academic teaching, like in courses about database systems to support students, and in practice to help database developers compare different solutions for database schemas with respect to normalization.

Purpose of this document

This user manual contains all the information necessary to exploit all the functionality of the application.

Since database normalization is a relatively old field in computer science, a lot of material on it exists already and is freely available on the web. Thus, rather than rehash normalization theory yet again, this document expects the reader to have some basic understanding of it.

However, this application can well be (and really is meant to be) used to support the process of learning about database normalization.

Concepts and Terms

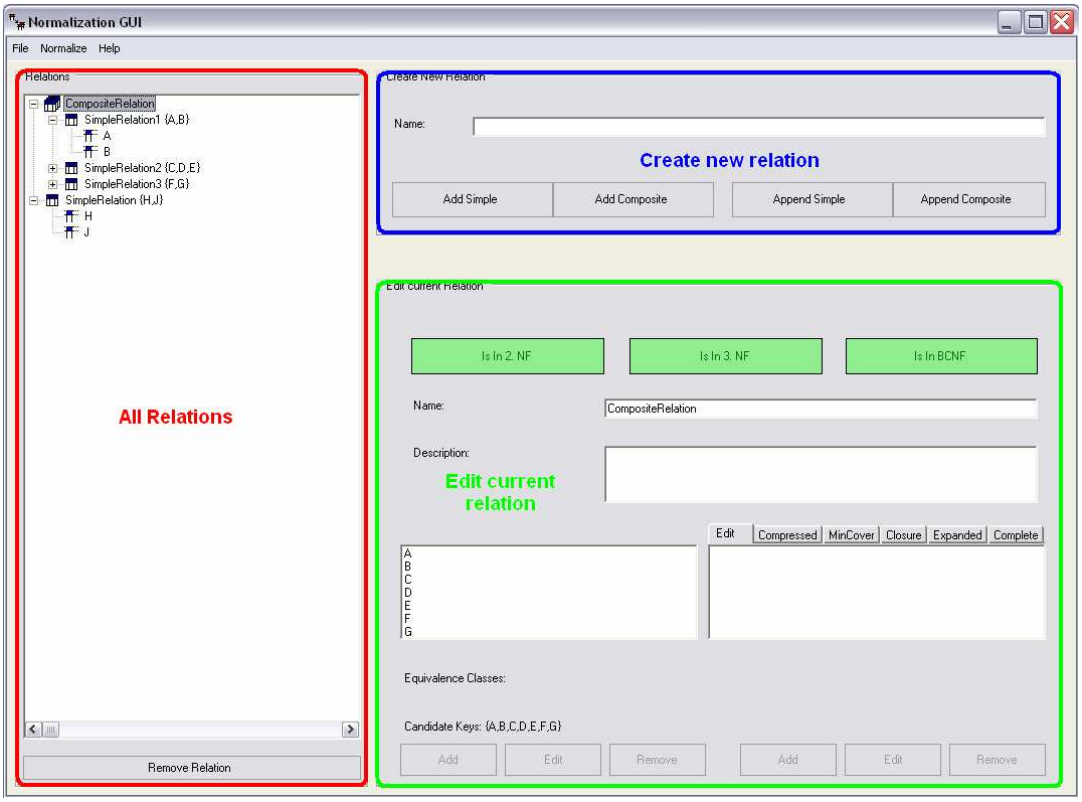
The data contained in Relational Database Management Systems (RDBMS) can be divided into two different classes: Data contained in tables, and metadata describing the schema of these tables.
Database normalization, and with it the DN and this document, only deal with the schematic level: How to form the schemas of the tables.

The mathematical model for a table is a relation. A relation is a set of attributes. Attributes have a name and are ordered in the relation.
(In the database, a relation corresponds to a table, and an attribute to a column in the table.)
Furthermore, a relation contains functional dependencies (FDs) between its attributes. The attributes and functional dependencies determine the normal form of the relation.

Explanation of the User Interface

Overview

The user interface consists of three different regions.



ALL RELATIONS Region

This tree view shows all the relations that currently exist in the application. Two different kinds of relations are supported:

Simple Relation:
The simple relation is a relation as explained above: It contains attributes and relations between them. It cannot contain other relations.
Since the attributes are unstructured (that is, cannot contain lists or relations), all Simple Relations are in first normal form.

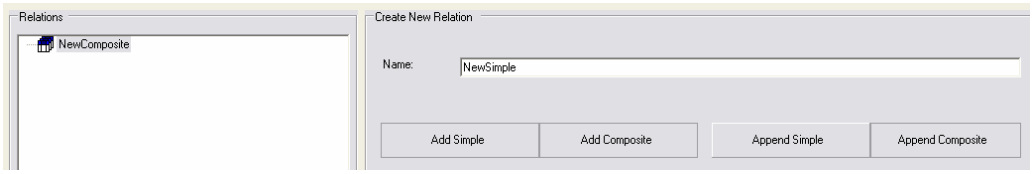
Composite Relation:
A Composite relation groups various relations (Simple Relations or Composite relations) into a logical unit.
It does not contain attributes or functional dependencies itself. Rather, its attributes and functional dependencies are the union of the attributes and functional dependencies of its contained relations.
Its normal form the lowest normal form a contained relation is in.

The composite relation is used to model a database schema. A schema is said to be in a normal form, if all of its contained relations are in the normal form, just as holds for the composite relation.
So you can use composite relations to group relations into a schema.

As you would expect, clicking on a relation opens its properties in the “Edit Current Relation” region.

CREATE NEW RELATION Region

This region is used to create a new simple or composite relation.
The append buttons are used to insert a new simple or composite relation into a composite relation. (And thus are deactivated, if a Simple Relation is selected in the All Relations View, since relations cannot be appended to a Simple Relation.)

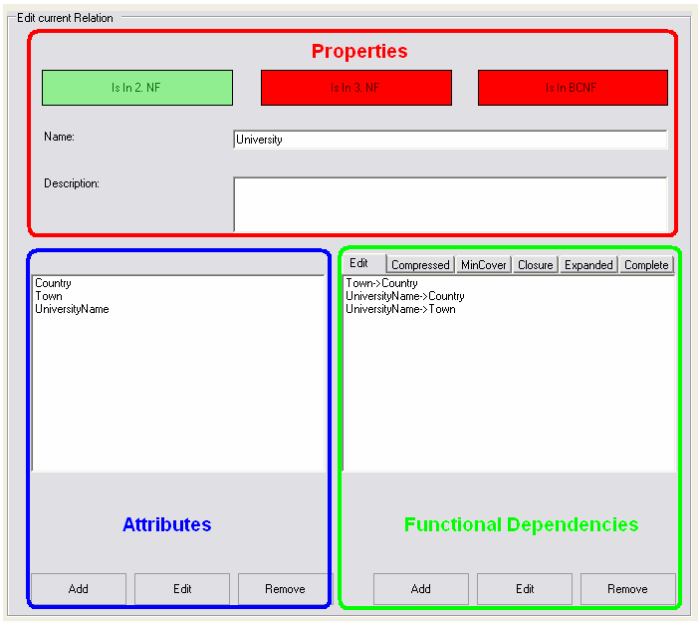


Before appending a Simple Relation to a Composite Relation

After appending a Simple Relation to a Composite Relation

EDIT CURRENT RELATION Region

This region displays and edits the properties of the relation currently selected in the All Relations View:



Properties Region
The coloured buttons display the normal form, the relation is in. Every time the relation changes, the buttons are updated automatically.

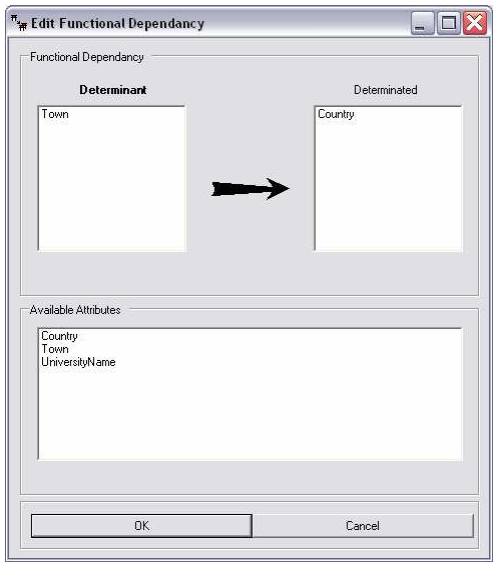
In the text fields, name and description can be entered for documentation purposes.

Attributes region
Add to, change or remove attributes from the relation.

Attention: Attribute names are application wide. Thus, if you want to use the same attribute in different relations, (i.e. as a foreign key,) you simply give it the same name. (Names are not case sensitive.)
In some cases however, this may not be what you want. The attribute “Name” for example, might appear in many different applications, and all names might be independent from each other.
So if you want different name scopes, a simple workaround is to add a prefix to the attributes name:
Instead of using the attribute “Name”, use “**Person.**Name” and “**University.**Name”.

Functional Dependencies Region
Create functional dependencies or change them.
(If you want to use attributes in a functional dependency, you must create it first. So the standard work cycle will be: Create relation, add all attributes and then add functional dependencies.)

A simple dialog is used to create or edit functional dependencies:



Functional dependencies can be displayed in different forms:

Edit: As FDs have been entered

FDs with same left side are compacted into one

Minimal cover of FDs that has still the same closure

Transitive Closure of the FDs withour reflective parts.

(that is: attributes from the left side do not appear on the right side)

FDs contain only one attribute on the right side. (Opposite of Compressed)

Complete transitive closure of the set of FDs

Functional dependencies can only be edited on the “Edit” tab page. The buttons are disabled on all other tab pages.

The available attributes list contains all attributes of the current relation. The two lists next to the arrow, “Determinant” and “Determinated”, contain the attributes on the left and right side of the functional dependency. Double clicking an attribute in the available Attributes list adds it to the functional dependency. (To the active list that is printed in bold. In this example, that is the left side).
Double clicking an attribute in one of the lists next to the arrow removes it from the dependency.
Clicking on a list next to the arrow makes it the active list.

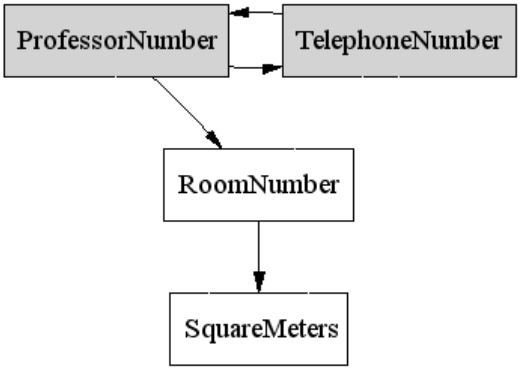
Equivalence classes
Tuples are sets of attributes. Two tuples are said to be in the same equivalence class, when they have the same transitive closure, that is, if they transitively determine the same attributes. Example:

Attributes:

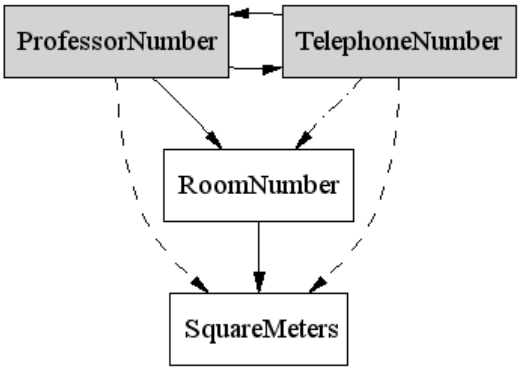
- ProfessorNumber
- TelephoneNumber
- RoomNumber
- SquareMeters
-

Functional Dependencies

- ProfessorNumber -> TelephoneNumber
- TelephoneNumber -> ProfessorNumber
- ProfessorNumber -> RoomNumber
- RoomNumber -> SquareMeters



Without transitive dependencies



With transitive dependencies as dashed arrows

In the graph that contains transitive dependencies it is easy to see that both Professor and TelephoneNumber have the same transitive closure. In this case, since both determine every attribute in the relation, both are candidate keys.

Menu entries

File Menu
Create new, store or load a file.
All data is stored in xml files that could be used by other applications.
Apart from these explicit store and load functions, the application persists its state on shutdown and loads it on start up. (Using the file ApplicationState.xml in the application directory)
So if you forget to save your relations, they will still be there if you start the application the next time.

Normalize Menu
Offers two different synthesis normalization algorithms.

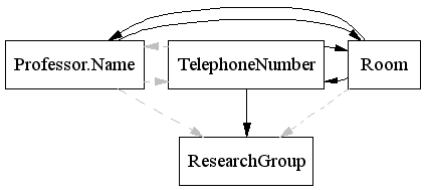
Simple Synthesis is a naïve normalization algorithm that does not take equivalent classes of attribute tuples into account.
Consistent Synthesis is an improved algorithm that lets you select the representers for classes of equivalent tuples.

Example:
Attributes:

- Professor.Name
- TelephoneNumber
- Room
- ResearchGroup

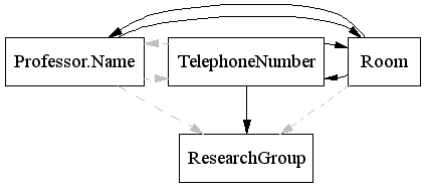
Functional Dependencies:

- Professor.Name -> Room
- Room -> Professor.Name
- TelephoneNumber -> Room
- Room -> TelephoneNumber
- TelephoneNumber -> ResearchGroup



Possible minimal cover

Since in this example the attributes Professor.Name, TelephoneNumber and Room are equivalent, there are many different minimal covers possible: The equivalence can be represented by various combinations of dependencies, and the ResearchGroup can depend on any attribute of the equivalence class without changing the closure of the set of functional dependencies.



Same closure, yet different minimal cover

The result of the simple synthesis algorithm depends on the minimal. Since the algorithm does not really care which minimal cover it computes, the outcome is rather randomly.
The consistent synthesis algorithm however takes the equivalence classes into account and always generates the same relational schema.

Help Menu
Displays a short about box.

Samples

In 2NF but nor in 3NF

Attributes:

- UniversityName
- Town
- Country

Functional Dependencies:

- UniversityName -> Country
- UniversityName -> Town
- Town -> Country

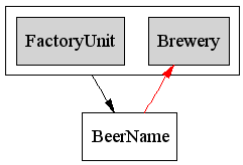
In 3NF but not in BDNF

Attributes:

- BeerName
- Brewery
- FactoryUnit

Functional Dependencies:

- BeerName -> Brewery
- Brewery, FactoryUnit -> BeerName



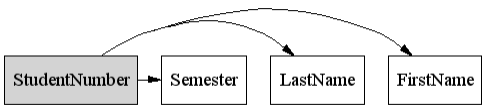
In BCNF

Attributes:

- StudentNumber
- Semester
- FirstName
- LastName

Functional Dependencies:

- StudentNumber -> Semester
- StudentNumber -> FirstName
- StudentNumber -> LastName



Conclusion

Possible future extensions

Implementation of the analysis algorithm for normalization:

The application does only implement a synthesis based algorithm for database normalization. It starts with dependencies and creates relations. It reaches BCNF if that can be done without breaking up dependencies, if not, it only reaches 3NF.
In database theory exists another normalization algorithm that starts with a single denormalized relation and decomposes it reaching BCNF. (Breaking up dependencies if need be.)

Graphical visualization of functional dependencies using graph layout tools like dot

Functional dependencies can be displayed graphically using a graph like notation as used in the samples. Arbitrary relations could be visualized relatively simply using a tool like dot that takes care of the layout of the graph.

Extraction of relational schemas and basic functional dependencies from existing databases

Relational databases keep metadata about their data tables. This metadata contains information about the relations, attributes and keys. This schematic information could be read out of the database into the application to facilitate the work with normalization rules in real world applications.

Creation of a library of normalization samples to support database lectures

A large number of samples eases the understanding of normalization theory.
Problems and solutions for them could be gathered in a central place to provide a starting point for students.