SchemaWizard - User documentation

Vojtěch Vondra - Java NPRG013

Terminology and an Introduction to database relations

In relational database theory, a **relation** is a set of tuples (d_1, d_2, ..., d_j), where each element d_n is a member of D_n, a data domain.[1] Each distinct domain used in the definition of a relation is called an **attribute**, and each attribute is usually named.[1] In addition to attributes, we define Functional dependencies, attribute-based integrity constraints defined by the user.

Example:

Name	Position	Hourly salary
John Doe	assistant	200
Jane Doe	consultant	250

Name, Position and *Hourly salary* are attributes of the schema. We do not know the relationship between hourly salaries and individual people. In this example, two possible dependencies are possible:

- Name + Position → Hourly Salary
- Position → Hourly salary

Each tells us more about the context of the relation, with the first rule two different people on the same position could have a different salary, opposed to the second rules which says that a position defines the employee's salary.

A **key** (sometimes called a candidate key) is a minimal set of attributes, which through functional dependencies clearly defines all other attributes. Each set of values selected by a key identifies a complete tuple in the relation.

Normal forms are sets of criteria which classify relations into categories based on their consistency and integrity. SchemaWizard works with **BCNF** (Boyce-Codd Normal Form) and has an interface which can split the relation into smaller ones which hold the necessary condition:

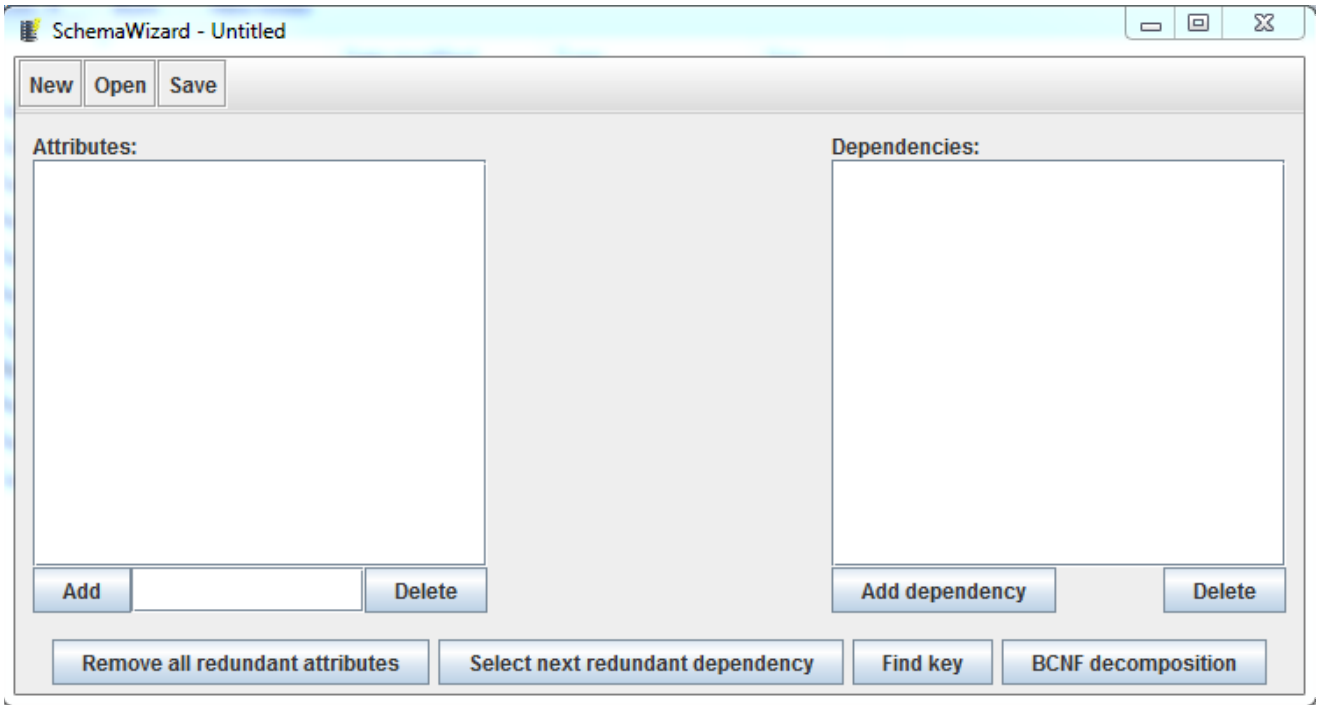
- For every functional dependency X → Y, X is a **key**.

SchemaWizard is a program which enables you to define attributes and dependencies and work with the created schema.

[1] http://en.wikipedia.org/wiki/Relation_(database)

Main dashboard

When the application is launched, you will be presented with the interface for editing a schema.



Managing attributes and dependencies

On the left and right, you will see the list of added attributes and functional dependencies. Attributes can contain numbers and letters.
Add attributes by entering its name and clicking the *Add* button. Add dependencies by clicking the *Add dependency* button. This will show a dialog with two columns. From each column select the attributes for the left and right side of the rule appropriately.
Both attributes and dependencies can be deleted by selecting them from the list and clicking *Delete*.

Saving schemas

Schemas are saved in readable plain-text files. You can use the *Open* and *Save* dialogs to load a previously saved database schema.
Attributes and functional dependencies are saved, BCNF decomposition is not.

Features

Remove all redundant attributes

This feature will check all functional dependencies and remove all attributes that are not necessary on the left side, simplifying them.

Example:
Given attributes A B C D E and the following rules:

- A B C → D
- A → E
- E →C

If we select a value for an A attribute, it identifies a A E tuple and transitively an A E C tuple. Thus the C attribute in the first rule is redundant and will be removed resulting in A B → D.
This example can be found in the *red_attr.db* file.

Remove redundant dependencies

It is desirable to have as little dependencies as possible. This feature helps determine, which dependencies can be derived from others.
When the user clicks *Select next redundant dependency*, it is highlighted in the list and the user can delete it. If none exists, a message will be shown.
Example:
Given a similar example as in the last case - let us have attributes I J K L M and rules:

- J →M
- J I → L
- J → K
- L → I
- L → K
- L M → J
- M → K

The third dependency is redundant, as it can be obtained by chaining the J → M and M → K rules.
You can load this schema with the example from *red_dep.db*.

Find keys

When the user clicks the *Find key* button. A list of keys, one one each line will be displayed.
Keys were explained in the introduction. Each line consists of a set of attributes which are enough to identify a whole tuple containing all attributes in the relation.
Without any dependencies a trivial key exists, one containing all the attributes.

BCNF decomposition

When creating a relational database schema, we want individual relations to hold the criteria of normal forms. It prevents data duplication and ensures data integrity.

SchemaWizard can help with splitting the defined relation into smaller ones, which are in BCNF using the Decomposition algorithm. In each step a dependency not meeting the requirement that its left side is a key is used to decompose the relation.

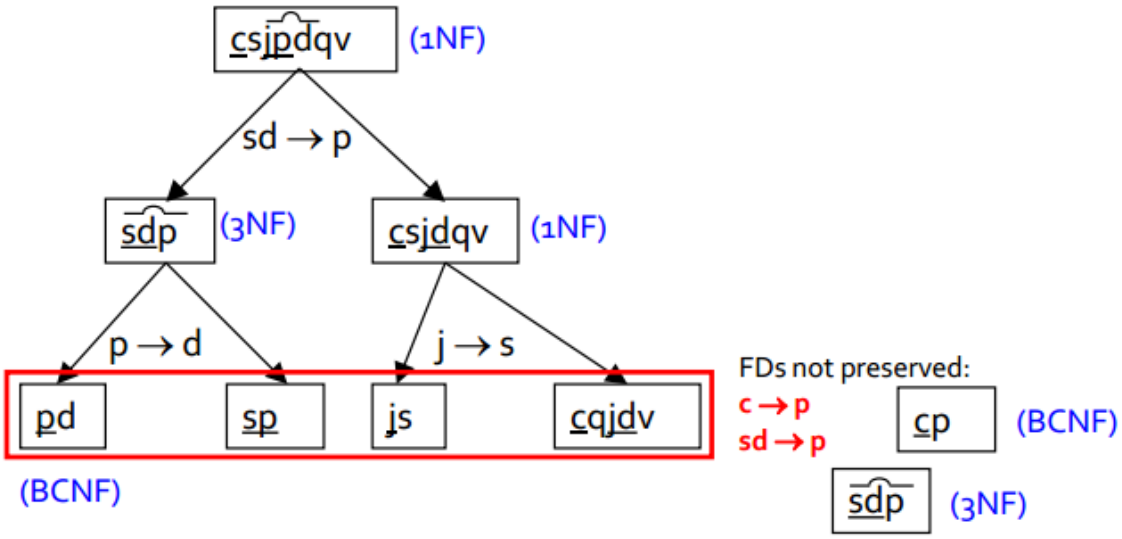
In each step, select the dependency to use and click *Decompose.* If a higher level node is decomposed, the whole tree below will be automatically refreshed.

When you click decompose, it takes the selected rule and puts all of its attributes in a separate relation, which might or might not be in BCNF. If not you can continue to decompose the schema.

Example

This example is taken from Tomas Skopal's slides and can be loaded from the *contracts.db* file.

Contracts(A, F)
A = {c = ContractId, s = SupplierId, j = ProjectId, d = DeptId, p = PartId, q = Quantity, v = Value}
F = {c → all, sd → p, p → d, jp → c, j → s}



Step by step you can reach this result: