Functional Dependencies and Finding a Minimal Cover

Robert Soulé

1 Normalization

An *anomaly* occurs in a database when you can update, insert, or delete data, and get undesired side-effects. These side side-effects include inconsistent, redundant, or missing data. Database *normalization* is the process of organizing the data into tables in such a way as to remove anomalies.

To understand how to normalize a database, we defined four types of data dependencies:

- *From the full key*: From the full primary key to outside the key
- *Partial dependency*: From part of the primary key to outside the key
- *Transitive dependency*: From outside the key to outside the key
- *Into key dependency*: From outside the key into the key

We then discussed how to removed these dependencies informally, and defined several normal forms based on the definition of these dependencies:

- *First Normal Form (1NF)*: there are a fixed number of columns.
- *Second Normal Form (2NF)*: 1NF and no partial dependencies
- *Third Normal Form (3NF)* 2NF and no transitive dependencies.
- *Boyce-Codd Normal Form (BCNF)* 1NF and all dependencies from full key.

2 Motivation

Recall the example database in class `Pay(Employee, Grade, Salary)`, in which the value of `Salary` was dependent on the value of `Grade`, and `Employee` was the primary key. In other words, we had a *from the full key* dependency $E \to GS$ and a *transitive dependency* $G \to S$.

To normalize the database, we eliminated the transitive dependency by splitting the `Pay(Employee, Grade, Salary)` table into two separate tables: `Pay(Employee, Grade)`. and `Rate(Grade, Salary)`.

This approach works, but it becomes difficult to do as the database becomes larger and more complex, and the number of functional dependencies increases. Instead of starting with a table and re-organizing it, we are going to start with the functional dependencies and synthesize the correct tables.

3 Small Example

In the `Pay(Employee, Grade, Salary)` example, we had two dependencies dependencies: $E \to GS$ and $G \to S$. These might seem simple, because the example is small, but they are not actually as simple as they could be.

Notice that if $E \to GS$, then of course, $E \to G$. In other words, if E forces a relation to be equal on G and S, then of course it forces the relation to be equal on just G. Without going into detail, we notice that we could re-write our dependencies as:

- $E \to G$
- $E \to S$
- $G \to S$

Then, (just take my word for it for now), we could further simplify the dependencies to be:

- $E \to G$
- $G \to S$

This is the simplest way that we could write the dependencies, while still preserving the same information. This form is called a *minimal cover*, and corresponds exactly to the normalized tables that we want.

4 Functional Dependencies

A *functional dependency* is a constraint between two sets of attributes in a relation. For a given relation R, if the set of attributes $A_1, \cdots, A_n$, functionally determine the set of attributes $B_1, \cdots, B_n$, then it means that if two tuples that have the same values for all the attributes $A_1, \cdots, A_n$, then they must also have the same values for all the attributes $B_1, \cdots, B_n$. This is written:

$$A_1, \cdots, A_n \rightarrow B_1, \cdots, B_n$$

Note that functional dependencies generalize the concept of a *key*.

5 Strength of Functional Dependencies

If a relation satisfies the dependency $A \rightarrow BC$, then it must also satisfy the dependency $A \rightarrow B$. Informally, if A forces a relation to be equal on B and C, then of course it forces the relation to be equal on just B. That is, $A \rightarrow BC \Rightarrow A \rightarrow B$. If a functional dependency F impies a functional dependency H, we say that F is *stronger* than H.

Be careful, though. It is easy to make mistakes. For example, if we are given the functional dependency $AB \rightarrow C$, it might be tempting to conclude that $A \rightarrow B$ and $A \rightarrow C$. However, this decomposition is incorrect, since neither A nor B alone determine C. In other words, $AB \rightarrow C \not\Rightarrow A \rightarrow C$, and $AB \rightarrow C$ is *weaker* than $A \rightarrow C$.

6 Properties of Functional Dependencies

There are often several ways to write the same functional dependencies. Two sets of functional dependencies, S and T, are *equivalent* if the same set of relation instances that satisfy S also satisfy T, and vice versa.

There are several useful rules that let you replace one set of functional dependencies with an equivalent set. Some of those rules are as follows:

- *Reflexivity*: If $Y \subseteq X$, then $X \rightarrow Y$
- *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

- *Pseudotransitivity*: If $X \to Y$ and $WY \to Z$, then $WX \to Z$
- *Composition*: If $X \to Y$ and $Z \to W$, then $XZ \to YW$

These rules are a slightly more formal way of capturing the idea of *relative strength* of functional dependencies.

Note that a *trivial* functional dependency is one in thich the constraint holds for any instance of the relation. For example, $X \to X$ always holds.

7 Closure

Given a set of functional dependencies F, and a set of attributes X, the *closure* of X with respect to F, written X_F^+, is the set of all the attributes whose values are determined by the values of X because of F. Often, if F is understood, then the closure of X is written X^+.

For example, given the following set, M, of functional dependencies:

1. $A \to B$
2. $B \to C$
3. $BC \to D$

Then we can compute the closure of A with respect to M in the following way:

i $A \to A$ (by reflexivity rule)

ii $A \to AB$ (by (i) and 1)

iii $A \to ABC$ (by (ii), 2, and transitivity rule)

iv $A \to ABCD$ (by (iii), and 3)

Therefore, $A_M^+ = ABCD$.

8 Cover

We say that a set of functional dependencies F *covers* another set of functional dependencies G, if every functional dependency in G can be inferred from F. A *minimal cover* is the smallest set of functional dependencies that covers G. We won't prove it, but every set of functional dependencies has a minimal cover. Also, note that there may be more than one *minimal cover*.

We find the minimal cover by iteratively simplifying the set of functional dependencies. To do this, we will use three methods:

Simplifying an FD by the Union Rule Let X, Y, and Z be sets of attributes. If $X \to Y$ and $X \to Z$, then $X \to YZ$

Simplifying an FD by simplifying the left-hand side Let X and Y be sets of attributes, and B be a single attribute not in X.

Let F be:

$$\cdots$$
$$XB \to Y$$

and H be

$$\cdots$$
$$X \to Y$$

If $F \Rightarrow X \to Y$, then we can replace F by H. In other words, if $Y \subseteq X_F^+$, then we can replace F by H.

For example, say we have the following functional dependencies (F):

- $AB \to C$
- $A \to B$

And we want to know if we can simplify to the following (H):

- $A \to C$
- $A \to B$

Then $A_F^+ = ABC$. Since, $Y \subseteq X_F^+$, we can replace F by H.

Simplifying an FD by simplifying the right-hand side Let X and Y be sets of attributes, and C be a single attribute not in Y.

Let F be:

$$\cdots$$
$$X \to YC$$

and H be

$$\cdots$$
$$X \to Y$$

If $H \Rightarrow X \rightarrow YC$, then we can replace F by H. In other words, if $YC \subseteq X_H^+$, then we can replace F by H.

For example, say we have the following functional dependencies (F):

- $A \rightarrow BC$
- $B \rightarrow C$

And we want to know if we can simplify to the following (H):

- $A \rightarrow B$
- $B \rightarrow C$

Then $A_H^+ = ABC$. Since, $BC \subseteq X_H^+$, we can replace F by H.

9 Finding the Minimal Cover

Given a set of functional dependencies F :

1. Start with F
2. Remove all trivial functional dependencies
3. Repeatedly apply (in whatever order you like), until no changes are possible
    - Union Simplification (it is better to do it as soon as possible, whenever possible)
    - RHS Simplification
    - LHS Simplification
4. Result is the minimal cover

Small example Applying to algorithm to EGS with

1. $E \rightarrow G$
2. $G \rightarrow S$
3. $E \rightarrow S$

Using the union rule, we combine 1 and 3 and get

1. $E \rightarrow GS$
2. $G \rightarrow S$

Simplifying RHS of 1 (this is the only attribute we can remove), we get:

1. $E \rightarrow G$
2. $G \rightarrow S$

Done!

10 An algorithm for (almost) 3NF Lossless-Join Decomposition

We can now use the following algorithm to compute our 3NF Lossless-Join Decomposition:

1. Compute F_m, the minimal cover for F
2. For each $X \rightarrow Y$ in F_m, create a new relation schema XY
3. For every relation schema that is a subset of some other relation schema, remove the smaller one.
4. The set of the remaining relation schemas is an almost final decomposition

This algorithm is almost correct, because it may be possible to compute a set of relations that do not contain the key of the original relation. If that is the case, you need to add a relation whose attributes form such a key.

For example, consider the relation `R(A,B)` that has no functional dependencies. The relation has only one key AB, but our algorithm, without the key, would produce no relations. So, we need to add the relation `R(A,B)`.

It is easy to check if your relation contains the key of the original relation. Simply compute the closure of the relation with respect to all the functional dependencies and see if you get all the attributes in the original relation.