

Functional Dependencies

Out: October, 7, 2009

In: Due whenever possible

A Definitions

Much of the difficulty of learning functional dependencies is getting all the terminology straight.

Be aware that the textbook is a great resource for this information.

- **Armstrong's Axioms.** These are a set of rules we use to compute new functional dependencies that are logically implied by other functional dependencies. They are (with examples):

- Reflexivity: If α is a set of attributes, and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.

Example: $AG \rightarrow G, AG \rightarrow A, AG \rightarrow AG$.

We refer to these as *trivial* functional dependencies.

- Augmentation: If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$.

Example: Supposing $A \rightarrow C$ and B, D are attributes, then $AB \rightarrow CB, AD \rightarrow CD, ABD \rightarrow CBD$

- Transitivity: If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$.

Example: Supposing $A \rightarrow B, B \rightarrow H$, then $A \rightarrow H$.

While these three are axiomatic, we often use other rules that are provable via Armstrong's axioms:

- Union rule: if $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$.

- Decomposition rule: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ hold.

- Pseudotransitivity rule: $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \sigma$ holds, then $\gamma\alpha \rightarrow \sigma$ holds.

We usually use Armstrong's axioms and their derivatives to compute closures on sets of functional dependencies or closures on attributes.

- **Closure of a set of FD's.** Given a set of functional dependencies F , the closure of F (denoted as F^+) is the set of all functional dependencies logically implied by the elements of F . F^+ will contain every FD that could be derived or computed from the original set F , and can be easily thought of as the comprehensive, or 'maximal' set of FD's from a given set.

As an example, suppose $F = \{A \rightarrow B, B \rightarrow H\}$. Since $A \rightarrow B$ and $B \rightarrow H$, $A \rightarrow H$. While $A \rightarrow H$ isn't present in F , it is logically implied by F 's contents, so F 's closure F^+ will contain it (and every other FD you could compute).

- **Closure of an attribute over a set.** Let α be a set of attributes in a relation R , which has a set of functional dependencies F . The closure of α over F , denoted as α^+ , is the set of all FD's in F^+ with α on the left-hand side. One way to think of it would be all attributes dependent on α .

An example would be the set $F = \{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$. In this case the closure of A over F is A^+ , and would be $A \rightarrow BCD$ (by union rule, and transitivity).

- **Extraneous attributes.** An attribute in a functional dependency is considered extraneous if it can be removed from that functional dependency without affecting the closure that is calculated from the set. For example, if $F = \{A \rightarrow B, BC \rightarrow DC\}$, the attribute C is extraneous in the FD $BC \rightarrow DC$ since it can be removed to produce $B \rightarrow D$, and the closure F^+ is unaffected by the change.

This definition isn't the most formal, but hopefully suffices (the formal explanation is harder to grasp, see section 7.4.3 of the textbook for more information).

- **Canonical Cover.** The canonical cover F_c over a set of FD's F is such that all FD's in F_c logically imply the FD's in F , and visa-versa (which is to say, they contain the same information). Furthermore, the canonical cover (F_c) must fulfill the following two properties:

- No FD in F_c can contain an extraneous attribute.
- The left side of each FD in F_c must be unique.

Described intuitively, a canonical cover is a way of expressing the same information as an equivalent set of functional dependencies, but with the fewest rules possible. Note that a canonical cover isn't unique, a set of functional dependencies may produce many equivalent canonical covers (each one, however, will contain the same number of rules).

- **Lossless Decomposition:** Let R be a schema, and R_1 and R_2 be a decomposition of that schema. (R_1, R_2) is a lossless decomposition if

$$\Pi_{R_1} \bowtie \Pi_{R_2} = R$$

If the join results in a relation which isn't R , the decomposition is lossy. This is best illustrated by example: if the schema

<i>ssn</i>	<i>name</i>	<i>smile_quality</i>
225-91-6211	Randy McRanderson	Toothy
553-44-3312	Jennifer Ealey	All Gums
112-88-5531	Jennifer Ealey	Grilles

Were decomposed into the two relations

<i>user_id</i>	<i>name</i>
225-91-6211	Randy McRanderson
553-44-3312	Jennifer Ealey
112-88-5531	Jennifer Ealey

<i>name</i>	<i>smile_quality</i>
Randy McRanderson	Toothy
Jennifer Ealey	All Gums
Jennifer Ealey	Grilles

Then the join of the two would be

<i>user_id</i>	<i>name</i>	<i>smile_quality</i>
225-91-6211	Randy McRanderson	Toothy
553-44-3312	Jennifer Ealey	All Gums
553-44-3312	Jennifer Ealey	Grilles
112-88-5531	Jennifer Ealey	Grilles
112-88-5531	Jennifer Ealey	All Gums

Which is not equal to the original, since it has 5 tuples instead of three. The ‘loss’ of information is present in the fact that we no longer know which Jennifer Ealey has the grilles, and which one smiles entirely with her gums: the distinction was lost in the decomposition.

The most common way to prevent lossy joins is to consider primary keys when decomposing schemas. Here it is clear that the name was not an appropriate unique designator for each person, meaning the join failed.

- **Dependency Preservation.** Let R_1, R_2, \dots, R_n be a decomposition of R , and F is a set of functional dependencies of R . Let F_1, F_2, \dots, F_n be subsets of functional dependencies of F where each F_i contains all FD’s of F^+ whose elements are all in R_i (so if R is split up into R_1, R_2, R_3 , F_1 contains all the FD’s in F^+ where each attribute of is present in R_1 , etc.).

A decomposition is dependency preserving if $F' = F_1 \cup F_2 \cup \dots \cup F_n$, and $F'^+ = F^+$. More informally, if each table in the decomposition contains every element in a set of functional dependencies, and the union of those compute to the same closure as the original set F , the decomposition is dependency-preserving.

An intuitive way to see if a table is dependency preserving is to compute a canonical cover, and make sure that each there is a table that contains both the left-hand side and the right-hand side of every FD in that cover. Since it is a canonical cover, its closure will evaluate to F^+ . If there exists a dependency whose left-hand side is in the table but not the right-hand side, the union of F_1, F_2, \dots will not contain that dependency, and the preservation will not be preserved.

This is another definition whose intuition is simple, but harder to grasp when you try to define it formally. The utility of dependency preservation is that you can verify each element in a dependency without having to perform joins.

• **Normal Forms.** A normal form specifies a set of desirable properties a schema may have. The normal forms we expect you to know are:

- Boyce-Codd Normal Form (BCNF). A schema with functional dependencies F is in Boyce-Codd Normal Form if, for all dependencies of the form $\alpha \rightarrow \beta$ in F^+ , at least one of the following holds:
 - * $\alpha \rightarrow \beta$ is a trivial FD (meaning $\beta \subseteq \alpha$).
 - * α is a superkey for the schema R .

Recall that a subset of elements K are a superkey for a relation R if, given two tuples t_1 and $t_2 \in R$, if $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$. In English, this means that any two non-equal tuples can't share the same value for attributes K .

A database design is in BCNF if each of its relation schemas is in BCNF.

- Third Normal Form (3NF). The requirements for 3NF are the same as BCNF, but with an additional option:
 - * Each attribute A in $\beta - \alpha$ is contained in a candidate key for R .

Recall that a candidate key is a minimal superkey - a superkey which contains no subsets which are also superkeys.

Note that every schema in BCNF is also in 3NF, but not the other way around. This makes BCNF a stronger, more stringent form than 3NF.