Theory of Normal Forms
Functional Dependencies in Databases

Functional Dependencies

Functional dependencies allow identify *redundancy* in relational database schemas.

A **functional dependency (FD)** on a relation R is a statement of the form:

"If two tuples agree on all attributes $A_1, \ldots, A_n$ then they must also agree on all attributes $B_1, \ldots, B_m$."

A functional dependency is formally denoted as

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$$

We also say "$A_1, \ldots, A_n$ *functionally determine* $B_1, \ldots, B_m$".

Keys and Superkeys

Functional dependencies allow us to formalize the definition of a **key** in a relational table.

A set of one or more attrbiutes $A_1, \ldots, A_n$ of relational table R is a **key** of R if:

1. $A_1, \ldots, A_n \rightarrow B$ **for all attributes** B of R. (i.e., $A_1, \ldots, A_n$ functionally determine all attributes of R).
2. No proper subset of $A_1, \ldots, A_n$ functionally determines all attributes of R.

Informally: a **key** is a minimal collection of attributes which uniquely identifies all tuples in a relation. A relation can have multiple keys, but for two different keys of the same relation, one cannot be a proper subest of the other.

A **superkey** of a relation R is any set of attributes that contains a **key**.

Reasoning about Functional Dependencies

FD Equivalence. Two sets S and T of functional dependencies over some relation R are **equivalent** if the *set of all instances of R satisfying S is* ***the same*** *as the set of all instances of R satisfying T.*

FD Implication. A set of FDs S **follows** a set of FDs T over some relation R if *and instance of R that satisfies S also satisfies T.*

Rules for Manipulating Functional Dependencies

1. Trivial FDs. Let R be a relation, and let $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$ be some of R's attributes. Then the following FD always holds:

$$A_1, \ldots, A_n \rightarrow B_1, \ldots B_m$$

2. Splitting/Combining Rule. Functional dependencies with multiple attributes in right-hand side can be simplified:

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \equiv A_1, \ldots, A_n \rightarrow B_1, \ldots, A_1, \ldots, A_n \rightarrow B_m$$

Basically, it means, that we only need to be establishing functional dependencies with a signle attribute on the right-hand side.

3. Trivial Dependency Rule. Let R be a relation. Let $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$ and let $\{C_1, \ldots, C_k\} \cap \{A_1, \ldots, A_n\}$. Then, the following FDs are equivalent:

$$A_1, \ldots, A_n \rightarrow B_1, \ldots B_m, C_1, \ldots, C_k \equiv A_1, \ldots, A_n \rightarrow C_1, \ldots C_k$$

4. Transitive Rule. $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m, B_1, \ldots, B_m \rightarrow C_1, \ldots C_k \Longrightarrow A_1, \ldots, A_n \rightarrow C_1, \ldots C_k$

Armstrong's Axioms

Armstrong's axioms are a complete system of FD derivation. Here **complete** means that using Armstrong's axioms, it is possible to derive from a collection of FDs **all** FD that logically follow from them.

There are three axioms, two of which parallel the trivial dependency and transitivity rules listed above.

1. **Reflexivity.** Let R be a relation. Let $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$ and let $\{C_1, \ldots, C_k\} \cap \{A_1, \ldots, A_n\}$. Then, the following FDs are equivalent:

   $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m, C_1, \ldots, C_k \equiv A_1, \ldots, A_n \rightarrow C_1, \ldots C_k$

2. **Augmentation.**

   $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m \Longrightarrow A_1, \ldots, A_n, C_1, \ldots, C_k \rightarrow B_1, \ldots B_m, C_1, \ldots, C_k,$

   for any $C_1, \ldots, C_l$.

3. **Transitivity.**

   $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m, B_1, \ldots, B_m \rightarrow C_1, \ldots C_k \Longrightarrow A_1, \ldots, A_n \rightarrow C_1, \ldots C_k$

Algorithms For Functional Dependencies

Closure

Closure of a set of attributes. Let R be a relation, and $Q = \{A_1, \ldots, A_n\}$ be a subset of R's attributes. The closure of Q, denoted Q^+ is a set of attributes, which are functionally determined by Q, given a set S of functional dependencies on R.

FD Closure algorithm. The following algorithm computes the closure of a set of attributes.

```
Algorithm FD-Closure(R Relational_Schema, Q Set_of_Attributes, S Set_of_FDs)
     returns Set_of_Attributes;

// Step 1. split all rules in S
for each f in S
 begin
    if  (right side of f has multiple attributes)
    then replace f in S with FDs that have only on attribute on the right;
 end;

// Step 2. Include set Q in the closure

X := Q;

// Step 3. Keep including new attributes for as long as FDs allow

repeat
  X' := X;
  for each f: B1,...,Bn -> C in S
  begin
     if {B1,...,Bn} is a subset of X
     then  X := X union {C};
  end;
until (X' == X);  // repeat until no new attributes can be added to X

return X;
```