Classroom log 9

V. DAKSHAYANI, IMT2014061

Functional dependencies

- Functional dependencies give a mechanism to analyze how good a database is.
- Dependency: How much knowing one attribute predicts about another
- FDs should be specified by the designer (intention), not inferred as patterns (extension)

Rules of FD inference

- Reflexive rule: $Y \subseteq X \Rightarrow X \rightarrow Y , X \rightarrow X$
- Augmentation rule: $X \rightarrow Y \Rightarrow XZ \rightarrow Y Z$; $X \rightarrow Y \Rightarrow XZ \rightarrow Y$
- Transitivity rule: $X \rightarrow Y$; $Y \rightarrow Z \Rightarrow X \rightarrow Z$
- Decomposition or projective rule: $X \rightarrow Y Z \Rightarrow X \rightarrow Y$
- Union or additive rule: $X \rightarrow Y$; $X \rightarrow Z \Rightarrow X \rightarrow Y Z$
- Pseudo transitive rule: $X \rightarrow Y$; $WY \rightarrow Z \Rightarrow WX \rightarrow Z$

The first three are called Armstrong Axioms.

Attribute closure

- The attribute closure(X^+) of an attribute X is the set of all possible attributes functionally determined by X.
- Example:
- F: {X→Y, Y→Z, X→AZ, Z→BY,C→X}

  Then X^+ = {X,Y,Z,A,B}

Computing X^+

Algorithm:

1. $X^+ \leftarrow X$
2. repeat
   (a) *old* $X^+ \leftarrow X^+$
   (b) for each FD $Y \rightarrow Z$ in F do
       if $Y \subseteq X^+$ then $X^+ \leftarrow X^+ \cup Z$
3. until (X^+ == *old* X^+)

Closure of an FD

- The set F along with all the FDs is called the closure of F denoted by F^+
- Can be thought of as canonicalization or standardization of an FD

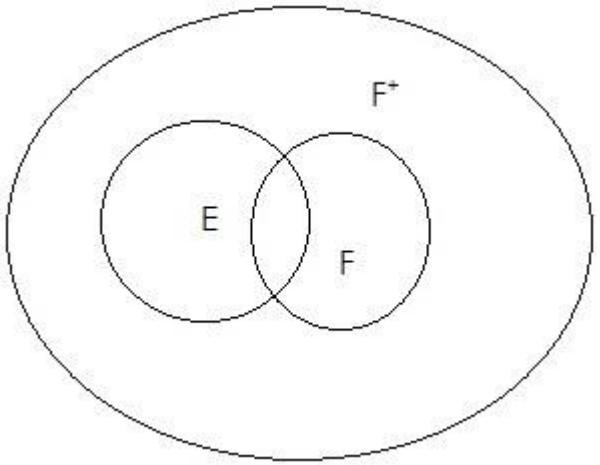
Cover and equivalence

- Cover: For any two FDs E and F, set F is said to cover set E if

  $\forall e \in E \Rightarrow e \in F^+$

- Equivalence: E and F are equivalent if $E^+ = F^+$. This also means

  $\text{covers}(F,E) \wedge \text{covers}(E,F)$



Minimal set of FDs

- Every FD has a single attribute on the right. (i.e., FD is of the form X→Y, XA →Y etc)
- Non decomposable, non replaceable FDs
- Can't remove any FD and still get an equivalent set of FDs

Normalization - Definition and purpose

- Normalization is the process of organizing the relational schema to reduce data redundancy and improve data integrity.
- It is a rule for designing the database taking dependencies between attributes into consideration.
- Hence, purpose:
    - Eliminate redundancy
    - Store data logically
    - Remove anomalies

Normalization – 1NF

- Only atomic attributes. No multivalued attributes. The second table is what we get after applying 1NF rules.
- Eg:

Student	Courses
A	Java, C++
B	HTML,C

Student	Courses
A	Java
A	C++
B	HTML
B	C

Normalization – 2NF

- A relation is in 2NF if, every non-prime attribute is fully functionally dependent on the primary key.
- A non prime attribute is not part of the candidate key.
- Eg: R = {VoterId, Name, DLNo, Area}

  {VoterId, DLNo} → {Name, Area}

  If DLNo → Area, then R is not in 2NF.

  To be in 2NF, change the structure as follows:

  R1(VoterId, DLNo, Name)

  R2(DLNo, Area)

Normalization – 3NF

- No transitive dependencies allowed.

  Eg: R = {VoterId, Name, DLNo, Area} and let DLNo → Area

  As VoterId is primary key, VoterId → DLNo holds.

  VoterId → DLNo and DLNo → Area => VoterId → Area is a transitive dependency.

  So, not in 3NF.

  To be in 3NF, change the structure as follows:

  R1(VoterId, Name, DLNo)

  R2(DLNo, Area)
- Transitive dependencies result in update and deletion anomalies.

Thank You!