**ST. XAVIER’S COLLEGE**

**Maitighar, Kathmandu**



**Data Base Management System**

**Theory Assignment #9**

**Submitted by**

Yub Raj Basnet

013BScCSIT048 (4th Semester)

**Submitted to**

|  |  |
| --- | --- |
| Er. Sanjaya Kumar Yadav  (Lecturer, St. Xavier’s College ) |  |

# Functional Dependencies

# Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

A Functional Dependencies is a relationship between an attribute "Y" and a determinant (1 or more other attributes) "X" such that for a given value of a determinant the value of the attribute is uniquely defined.

* X is a determinant
* X determines Y
* Y is functionally dependent on X
* X → Y
* X →Y is trivial if Y ⊆ X

## Example:

Let R be  NewStudent(*stuId, lastName, major, credits, status, socSecNo*)

FDs in R include

* *{stuId}→{lastName}*, but not the reverse
* *{stuId} →{lastName, major, credits, status, socSecNo, stuId}*
* *{socSecNo} →{stuId, lastName, major, credits, status, socSecNo}*
* *{credits}→{status}*, but not *{status}→{credits}*

## Closure of a Set of Functional Dependencies:

**Armstrong's Axioms**

If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

**Reflexive rule**

If alpha is a set of attributes and beta is\_subset\_of alpha, then alpha holds beta.

**Augmentation rule**

If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.

**Transitivity rule**

Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

**Trivial Functional Dependency**

**Trivial**

If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.

**Non-trivial**

If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.

**Completely non-trivial**

If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

# Decomposition

## Introduction

Decomposition is the process of breaking down in parts or elements. Decomposition in database means breaking tables down into multiple tables. From Database perspective means going to a higher normal form.

## Lossless-Join Decomposition

We claim the above decomposition is lossless. How can we decide whether decomposition is lossless?

Let *R* be a relation schema.

Let *F* be a set of functional dependencies on *R*.

Let **R1** and **R2** form a decomposition of *R*.

The decomposition is a lossless-join decomposition of *R* if at least one of the following functional dependencies are in F+ :

tex2html_wrap_inline1630

tex2html_wrap_inline1632

Why is this true? Simply put, it ensures that the attributes involved in the natural join ( tex2html_wrap_inline1634 ) are a candidate key for at least one of the two relations.

This ensures that we can never get the situation where spurious tuples are generated, as for any value on the join attributes there will be a unique tuple in one of the relations.

We'll now show our decomposition is lossless-join by showing a set of steps that generate the decomposition:

First we decompose *Lending-schema* into

*Branch-schema = (bname, bcity, assets)*

*Loan-info-schema = (bname, cname, loan#, amount)*

Since *bname* tex2html_wrap_inline1526 *assets bcity*, the augmentation rule for functional dependencies implies that

*bname* tex2html_wrap_inline1526 *bname assets bcity*

Since *Branch-schema* tex2html_wrap_inline1640 *Borrow-schema* = *bname*, our decomposition is lossless join.

Next we decompose *Borrow-schema* into

*Loan-schema = (bname, loan#, amount)*

*Borrow-schema = (cname, loan#)*

As *loan#* is the common attribute, and

*loan#* tex2html_wrap_inline1526 *amount bname*

This is also lossless-join decomposition.

## Dependency Preservation

Getting lossless decomposition is necessary. Losing dependecny means that the corresponding constraint can be checked only through natural join of the approprite resultant relation in the decomposition so it is necessary to keep dependencies.

A decomposition D={R1, ..Rm} of R is dependency preserving wrt a set F of FDs if (F1∪…∪Fm)+=F+

Where Fi means the projection of the dependency set F onto Ri.

Fi =ΠRi(F+) denotes a set of FDs X → Y in F+ such that all attributes in X ∪ Y are contained in Ri:

Fi=ΠRi(F+) ={ X→Y| {X,Y}⊆ Ri and X→Y ∈ F+ }

**Example:**

R=(A,B,C), F={A->B, B->C}

Decomposition of R: R1 = (A,C) R2=(B,C)

Does this decomposition preserve the given dependencies?

**Solution**

In R1 the following dependencies hold: F1’ ={A->A, C->C, A->C, AC->AC}

In R2 the following dependencies hold: F2’ = {B>B, C->C, B->C, BC->BC}

The set of non-trivial dependencies hold on R1 and R2: F’:={B->C, A->C}

A->B can not be derived from F’ so this decomposition is NOT dependency preserving.