**ST. XAVIER’S COLLEGE**

**Maitighar, Kathmandu**



**Data Base Management System**

**Theory Assignment #9**

**Submitted by**

Linus Dhakal

013BScCSIT022 (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.

**Lossless Join Decomposition:**

Definition:

Let { R1, R2 } be a decomposition of R (R1 U R2 = R); the decomposition is lossless if for every legal instance r of R:

r = ΠR1(r) |X| ΠR2(r)

**Testing Lossless Join:**

* Lossless join property is necessary if the decomposed relation is to be recovered from its decomposition.
* Let R be a schema and F be a set of FD’s on R, and α = (R1, R2) be a decomposition of R. Then α has a lossless join with respect to F iff

R1 ∩ R2 -> R1 (or R1 - R2 ) or R2 ∩ R1 -> R2 (or R2 - R1 ) where such FD exist in Closure of F.

PS This is a sufficient condition, but not a necessary condition.

**Example:**

From the previous example : R = (ABC) F = {A -> B}

R1 = (AB), R2 = (AC)

R1∩ R2 = A, R1- R2 = B

check A -> B in F ? Yes. Therefore lossless

R1 = (AB), R2 = (BC)

R1∩ R2 = B, R1 - R2 = A , R2 - R1= C

check B -> A in F ? NO

check B -> C in F ? NO

So, this is lossy join.

**DEPENDENCY PRESERVATION**

A desirable property in database design is dependency preservation. We would like to check easily that updates to the database do not result in illegal relations being created. It would be nice if our design allowed us to check updates without having to compute natural joins

Let Fibe the set of dependencies F+that includes only attributes in Ri

* A decomposition is dependency preserving, if (F1∪F2∪…∪Fn)+= F+
* If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

**Testing Dependency Preservation**

* To check if a dependency α→β is preserved in a decomposition of *R*into *R*1, *R*2, …, *R*n

*result* = α

**repeat**

**for each** *Ri* in the decomposition *t*= (*result* ∩*Ri*)+ ∩*Ri, result = result* ∪*t*

**until** *result* does not change

* If *result* contains all attributes in β, then the functional dependency

α→β is preserved

* We apply the test on all dependencies in *F* to check if a decomposition is dependency preserving
* This procedure takes polynomial time

**Example**

* R = (A, B, C )

F = {A →B, B →C}

Key = {A}

* R is not in BCNF
* Decomposition R1 = (A, B), R2 = (B, C)