**ST.XAVIER’S COLLEGE**

MAITIGHAR, KATHMANDU



Database Management System

Assignment #10

Submitted By:

Aabhash Dhakal

013BSCCSIT001

2nd year/ 4th semester

Submitted to:

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer  Department of Computer Science |  |

# Functional Dependencies

* 1. Basic Concepts

Functional Dependencies are formal tool for analysis of relational schemas. They enable us to detect and describe some of the above -mentioned problems in precise terms. They are constraints between two sets of attributes from the database.

Formally, a functional dependency, denoted by X -> Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples, t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].

Functional Dependencies and keys are used to define normal forms for relations.

Functional dependencies allow us to express constraints that cannot be expressed using superkeys.

Consider the schema:

*Loan-info-schema = (branch-name, loan-number, customer-name, amount).*

We expect this set of functional dependencies to hold:

*loan-number → amount*

*loan-number → branch-name*

but would not expect the following to hold:

*loan-number → customer-name*

We use functional dependencies to test relations to see if they are legal under a given set of functional dependencies. If a relation r is legal under a set F of functional dependencies, we say that r satisfies F.

Similarly, we use functional dependencies to specify constraints on the set of legal relations; we say that F holds on R if all legal relations on R satisfy the set of functional dependencies F.

* 1. Closure of a set of functional dependencies

Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F. The set of all functional dependencies logically implied by F is the closure of F. We denote the closure of F by F+.

We can find all of F+ by applying Armstrong’s Axioms: –

* if β ⊆ α, then α → β (reflexivity)
* if α → β, then γα → γβ (augmentation)
* if α → β and β → γ, then α → γ (transitivity)

These rules are sound and complete.

We can further simplify computation of F+ by using the following additional rules.

* If α → β holds and α → γ holds, then α → βγ holds (union)
* If α → βγ holds, then α → β holds and α → γ holds (decomposition)
* If α → β holds and γβ → δ holds, then αγ → δ holds (pseudotransitivity)

The above rules can be inferred from Armstrong’s axioms.

R = (A, B, C, G, H, I)

F = {A → B

A → C

CG → H

CG → I

B → H}

Some members of F+

– A → H

– AG → I

– CG → HI

* 1. Closure of Attribute sets

Define the closure of α under F (denoted by α+) as the set of attributes that are functionally determined by α under F:

α → β is in F+ ⇔ β ⊆ α+

Algorithm to compute α+, the closure of α under F

result := α;

while (changes to result) do

for each β → γ in F do

begin if β ⊆ result then result := result ∪ γ;

end

**2. Decomposition**

2.1 Lossless-Join Decomposition

A decomposition D = {R1, R2,..., Rm} of R has the lossless join property with respect to the set of dependencies F on R if, for every relation r of R that satisfies F, the following holds,

\*(πR1(r), ..., πRm(r)) = r,

where \* is the natural join of all the relations in D.

The word loss in lossless refers to *loss of information*, not to loss of tuples.

2.2 Dependency Preservation

Getting lossless decomposition is necessary. But of course, we also want to keep dependencies, since losing a dependency means, that the corresponding constraint can be check only through natural join of the appropriate resultant relation in the decomposition. This would be very expensive, so, our aim is to get a lossless dependency preserving decomposition.

Example:

R=(A, B, C), F={AÆB, BÆC}

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

Does this decomposition preserve the given dependencies?

Solution:

In R1 the following dependencies hold:

F1’={AÆA, CÆC, AÆC, ACÆAC}

In R2 the following dependencies hold:

F2’= {BÆB, CÆC, BÆC, BCÆBC}

The set of nontrivial dependencies hold on R1 and R2: F':= {BÆC, AÆC} AÆB cannot be derived from F’, so this decomposition is NOT dependency preserving.