ST. XAVIER’S COLLEGE

**Maitighar,Kathmandu**

**(Affiliated to Tribhuvan University)**



**Database Management System**

**Lab Assignment #10**

**Submitted By**

Aashish Raj Shrestha

B.Sc. CSIT

Year II/IV Semester

013BSCIT002

**Submitted To**

Er. Sanjay Kumar Yadav

Lecturer,

Department of Computer Science

St. Xavier’s College

Maitighar, Kathmandu

**Submitted On**

October 4, 2015

1. **Functional Dependencies**
   * Functional dependencies are constraints on the set of legal relations. It defines attributes of relation, how they are related to each other.
   * It determines unique value for a certain set of attributes to the value for another set of attributes that is functional dependency is a generalization of the notation of key.
   * Functional dependencies are interrelationship among attributes of a relation.
   1. **Basic Concepts**

**DEFINITION:**

For a given relation R with attribute X and Y, Y is said to be functionally dependent on X, if given value for each X uniquely determines the value of the attribute in Y. X is called determinant of the functional dependency (FD) and functional dependency denoted by X→ Y.

Example: consider a relation supplier

Supplier (supplier\_id#,sname,status,city)

Here, sname, status and city are functionally dependent on supplier\_id. Meaning is that each supplier id uniquely determines the value of attributes supplier name,supplier status and city This can be express by

Supplier.supplier\_id→supplier.sname

Supplier.supplier\_id→supplier.status

Supplier.supplier\_id→supplier.city

Or simply,

supplier\_id→ sname

supplier\_id→ status

supplier\_id→city

**Application of Functional dependencies**

Functional dependencies are applicable

* To test the relation whether they are legal under a given set of functional dependency.
  + Let r is a relation and F is a given set of functional dependencies. If r satisfies F, then we determine that r is legal under a given set of functional dependency F
* To specify the constraints for the legal relation
  + We say that f holds on R if all legal relations on R satisfy the set of functional dependencies F.

**Types of Functional Dependencies**

**Trivia functional dependency**

Functional dependencies are said to be trivial if it satisfied by all relations.

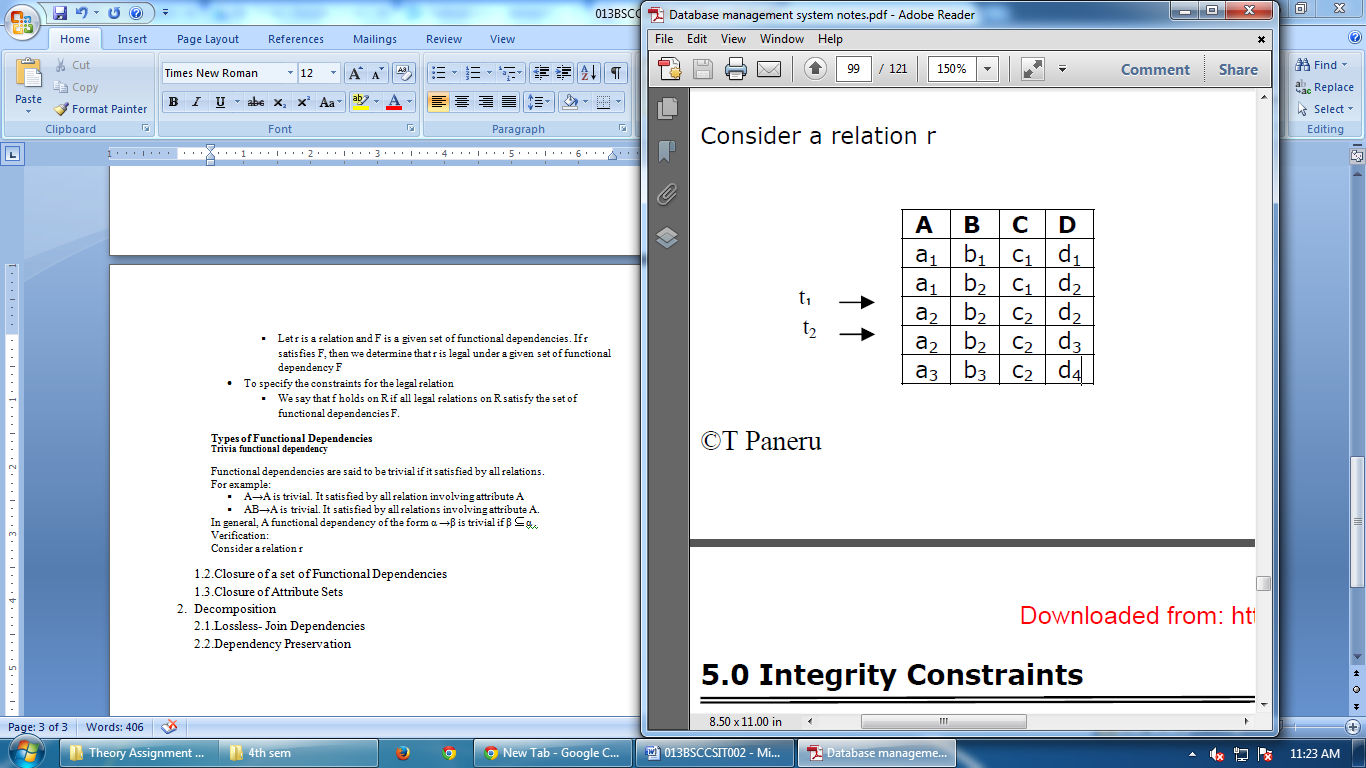
For example:

* + A→A is trivial. It satisfied by all relation involving attribute A
  + AB→A is trivial. It satisfied by all relations involving attribute A.

In general, A functional dependency of the form α →β is trivial if β ⊆ α .

Verification:

Consider a relation r



t1[AB]=a2b2, t2[AB]=a2b2 agree

t1[A]=a2 t2[A]=a2 agree

Here, t1[AB]=t2[AB] ⇒ t1[A]= t2[A]. This implies AB→A is satisfied.

**Fully functionally dependency**

For a given relation schema R, FD X→Y, Y is said to be fully functionally dependent on X if there is no Z (where Z is a proper subset of X) such that Z→Y.

Example: Let us consider relational schema R=(A,B,C,D,E,H) with the FDs

F={A→BC,CD→E,C→E,CD→E,CD→AH, ABH→BD,DH→BC}

* Here, the FD A→BC is left reduced, so clearly, BC is fully functionally dependent on A (because there is no possible proper subset of only element A)
* Here, the FDs CD→E, C→E where E is functionally dependent on CD and again E is functionally dependent on subset of CD. That is C (i.e. C→E). Hence E is not fully functionally dependent on CD.

Example: Consider a relation sales

Sales (product\_id#,sales\_date#,quantity,product\_name) With the following functional dependencies

F={product\_id,sales\_date→quantity, product\_id→quantity, product\_id→product\_name}

* Here,. FDs product\_id,sales\_date→quantity, product\_id→quantity, quantity is not fully functional dependent on product\_is,sales\_date.
* Here, functional dependency product\_id→product\_name, product\_name is fully functional dependent on product\_id.

**Partial functional dependency**

For a given relation schema R with set of functional dependency F on attribute of R. Let K as a candidate key in R. if X is a proper subset of K and X and X→A then A is said to be partially dependent on K.

Example: Consider a relation schema ‘student\_course\_info’

student\_course\_info(name#,course#,grade,phone\_no,major,course\_department)

with the following FDs

{name→phone\_no,major

course→course\_department,

name,course→grade

}

Here {name,course} is a candidate key. Here grade is fully functionally dependent on {name,course}. If there is a possible FD name→grade then we cannot say grade is fully functionally dependent on {name,course}. Here phone\_no, major and course\_department are partially dependent on {name,course}

**Transitive dependency**

For a given relational schema R with set of functional dependency F. Let X and Y be the subset of r anf Let A be the attribute of R s.t. X ⊄ Y, A ⊄ XY. If the functional dependencies {X→Y, Y→A} implies by F (i.e. X→Y→A) then A is said to be transitively dependent on X.

Example:

Let us consider relational schema ‘prof\_info’

prof\_info=(prof\_name#,department\_name, head\_of\_department) with the set functional dependency

F={prof\_name→department\_name, department\_name→head\_of\_department}

Here prof\_name→department\_name→head\_of\_department so head\_of\_department is transitively dependent on the key prof\_name.

Example:

Let R=(A,B,C,D,E) and FDs F={AB→C,B→D,C→E}

Here AB acts a candidate key and E is transitively dependent on the key AB, Since AB→C→E).

* 1. **Closure of a set of Functional Dependencies**

For a given set of functional dependencies F, there are certain other functional dependencies that are logically implies by F. (i.e. if A→B and B→C, then we can write A→C). the set of all functional dependencies logically implies F is the closure of F. Closure of F is denoted by F+.

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

* if β ⊆ α then α → β or α →α (reflexive)
* if α → β then γ α →γ β (augmentation)
* if α →β and β →γ then α →γ (transitivity)

Example: Let R=(A,B,C,G,H,I)

F={A→B, A→C,CG→H,CG→I,B→H}

Compute closure of F+.

Closure of F+ computed as follow:

* A→H
* by transitivity A→B and B→H
* AG→I
* By augmenting A→C with G we get AG→CG and then by transitivity with CG→I we get

AG→I

* CG→HI
* From CG →H and CG→I “union rule” can be inferred from definition of functional dependency to Augmentation of CG→I to infer CG→CGI, argumentation of CG→H to infer CGI→HI, and then transitivity.

Hence, F+={ A→A,B→B,C→C,H→H,G→G,I→I,A→B,

A→C,CG→H,CG→I,CG→HI,B→H,A→H,

AG→I,CG→Hi

}

here , first six FDs obtain by reflexive axiom.

We can further simplify the the computation of F+ by using the following addition rule.

(a) if α → β holds and α → γ holds, then α → β γ (Additivity or union rule)

(b) if α → β γ holds then α → β holds and α → γ holds (projectivity/decomposion)

(c) if α → β holds and γ β →δ holds then α γ →δ holds (pseudotransitivity)

Examples: Let R=(A,B,C,D) and F={A→B,A→C,BC→D} then compute F+.

* Since A→B and A→C then by union rule A→BC.
* Since BC →D, then by projective/decomposition B→D, C→D. Again by transitivity A→B &

B→D ⇒ A→D and A→C and C→D ⇒ A→D.

* + - Hence, F+ ={A→A, B→B, C→C, D→D, A→B, A→C, BC→D, B→D, C→D, A→D}
  1. **Closure of Attribute Sets**

{A1,A2, . . Am} such that the FD X→Ai for Ai∈X+ follows from F by the inference axioms for functional dependencies.

Example:

Let X=BCD and F={A→BC,CD→E,E→C,D→AEH,ABH→BD,DH→BC}. Compute the closure X+ of X

under F.

* + initialize X+:=BCD.
  + Since left hand side of the FD CD→E is a subset of X+ (i.e CD⊆X+), X+ is augmented by the right hand side of the FD (i.e. E) thus now X+:=BCDE.
  + Similarly, D⊆X+, the right hand side of the FD D→AEH is added to X+. Hence now X+:=ABCDEH.
    - Now X+ cannot be augmented any further because no FDs left hand side is subset of X+.

**Application of Attribute Closure**

1**. Testing superkey**

To test α is a superkey we compute α + and check whether α + contains all attributes of R. if so α is a superkey, otherwise not.

**2. Testing functional dependencies**

To check a functional dependency α → β holds check whether β ⊆ α +. If so α →β ; otherwise not.

1. Decomposition
   1. Lossless- Join Dependencies

The **lossless join** property is a feature of decomposition supported by normalisation. It is the ability to ensure that any instance of the original relation can be identified from corresponding instances in the smaller relations.

Lossless means functioning without a loss. In other words, retain everything.

Important for databases to have this feature.

Formal Definition

* Let R be a relation schema.
* Let F be a set of functional dependencies on R.
* Let and 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+

1) R1 ∩ R2 ∩ R1

2) R1 ∩ R2 ∩ R2

In Simpler Terms,

R1 ∩ R2 ∩ R1

R1 ∩ R2 ∩ R2

If R is split into R1 and R2, for the decomposition to be lossless then at least one of the two should hold true.

Projecting on R1 and R2, and joining back, results in the relation you started with.

If R is the original relation and R1 and R2 are relations obtained by the decomposition of R, the decomposition can be said only if R can be got back from R1 and R2.

In other words:

decomposition of a relation R into 2 relations R1 and R2 is called lossless if and only if:

content(R1) \* content(R2) = content(R)

where \* could be any relational operator such as Cartesian product

EXAMPLE:

You certainly need the two relations R1(A,B) and R2(C, D) that you outline in the lossless decomposition, but you've lost the crucial information about which A values are associated with which C values that was present in the original R(A, B, C, D). So you also need R3(A, C) to keep all the original information.

Relation R

A B C D

1 2 13 14

2 2 13 14

3 1 12 15

Relation R1

A B

1 2

2 2

3 1

Relation R2

C D

13 14

12 15

Join R1 and R2 (Cartesian product); bogus rows marked ☜

A B C D

1 2 13 14

1 2 12 15 ☜

2 2 13 14

2 2 12 15 ☜

1 3 13 14 ☜

3 1 12 15

Since this join is not the same as R, the proposed decomposition is not lossless.

Relation R3

A C

1 13

2 13

3 12

Join R1, R2, R3

A B C D

1 2 13 14

2 2 13 14

3 1 12 15

Since this result relation is the same as the original R, the decomposition into R1, R2, and R3 is lossless.

|  |
| --- |
|  |

* 1. **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.

**Example:**

R=(A, B, C), F={AÆB, BÆC}

Decomposition of R: R1=(A, B) R2=(B, C) Does this decomposition preserve the given dependencies?

**Solution:**

In R1 the following dependencies hold:

F1={AÆB, AÆA, BÆB, ABÆAB}

In R1 the following dependencies hold:

F2= {BÆB, CÆC, BÆC, BCÆBC} F’= F1’ U F2’ = {AÆB, BÆC, trivial dependencies} In F’ all the original dependencies occur, so this decomposition preserves dependencies.

**Definition:**

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

Where Fi means the projection of the dependency set F onto Ri. Fi =Π Ri(F+) denotes a set of FDs X → Y in F+ such that all attributes in X ∪ Y are contained in Ri:

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

We do not want FDs to be lost in the decomposition.

Always possible to have a dependency-preserving decomposition D such that each Ri in D is in 3NF.

Not always possible to find decomposition that preserves dependencies into BCNF.

**Example:**

R(A, B, C, D), F={A → B, B → C}

Let S(A,C) be a decomposed relation of R.

What dependencies do hold on S?

**Solution:**

Need to compute the closure of each subset of {A,C}, wrt F+ Compute

{A}+ = {ABC} – C is in S – so A → C holds for S Compute

{C}+ – {C}+ = C,

no new FD Compute {AC}+ – {AC}+ = ABC, no new FD

Hence, A → C is the only non-trivial FD for S, Π S(F+)={A→C, + trivial FDs}