ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System Assignment #10**

**Submitted by:**

Binod Marikhu  
013BSCCSIT013

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav Lecturer, St. Xavier’s College |  |

**Date of Submission: October 8, 2015**

1. **FUNCTIONAL DEPENDENCIES**
   1. **Basic Concepts**

* 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.

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.

Consider a relation Supplier

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

Here, sname, status and city are functionally dependent on supplier\_id. It means that each supplier id uniquely determines the value of attributes supplier name,supplier status and city This can be expressd 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

* 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:

o if β⊆αthen α→β or α→α (reflexive)

o if α→β then γα→γβ (augmentation)

o 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:

o A→H

o by transitivity A→B and B→H

o AG→I

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

o CG→HI

o From CG →H and CG→I “union rule” can be inferred from definition of functional dependency ot

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,

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→ γholds, then α→ βγ (Additivity or union rule)

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

(c) if α→ βholds and γβ→δ holds then αγ→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**

The closure of X under a set of functional dependencies F, written as X+, is the set of attributes {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 CDX⊆+), X+ is augmented by the right hand side of the FD (i.e. E) thus now X+:=BCDE.

• Similarly, DX⊆+, the right hand side of the FD D→AEH is added to X+. Hence now X+:=ABCDEH.

• Now X+ can not be augmented any further because no FDs left hand side is subset of X+.

**Application of Attribute Closure**

1. Testing superkey

To test αis 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 →; otherwise not. αβ

1. **DECOMPOSITION**

Decomposition is to break down large and complicated relation into a no. of simple and small relations which helps in minimization of data redundancy. It can be considered principle to solve the relational model problem.

***Definition***

The decomposition of relation schema R= (A1, A2, . ,An) is a set of relation schema { R1, R2, ----- Rm}, such that Ri ⊆ R 1 ≤ i≤ m and R∀1 RU2 . . URUm = R.

That is all attributes of an original schema (R) must appear in the decomposition (R1, R2).

That is, R= R1UR2. if R R≠1UR2 then such decomposition called lossey join decomposition. That is, R≠ΠR1(R) ΠR2(R). Decomposition should ***lossless join decomposition****.*

A decomposition of relation schema R into R1 and R2 is lossless join iff at least one of the following dependencies is in F+.

R1IR2→R1

R1IR2→R2

Example 1: The problems in the relational schema branch\_loan (illustrated in above example) can be resolved if we replace it with the following relation schemas.

Branch (# branch\_name, branch\_city, assets)

Loan (customer\_name, loan\_number, branch\_name, amoun)

Example 2: Consider the relation schema to store the information a student maintain by the university.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Course** | **Phone\_no** | **Major** | **Prof** | **Grade** |
| John | 353 | 374537 | Computer Science | Smith | A |
| Scott | 329 | 427993 | Mathematics | James | S |
| John | 328 | 374537 | Computer Science | Adams | A |
| Allen | 432 | 729312 | Physics | Blake | C |
| Turner | 523 | 252731 | Chemistry | Miller | B |
| John | 320 | 374537 | Computer Science | Martin | A |
| Scott | 328 | 727993 | Mathematics | Ford | B |

* 1. **Lossless – Join Dependencies**

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+ :

R1 ∩ R2 → R1

R1 ∩ R2 →R2

It ensures that the attributes involved in the natural join ( R1  ∩ R2 ) 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.

**Example:**

1. 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 a lossless-join decomposition.
   1. **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.