ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



DATABASE MANAGEMENT SYSTEM

THEORY ASSIGNMENT #10

**Submitted by:**

Lokendra Puri  
013BSCCSIT023

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer |  |

Department of Computer Science

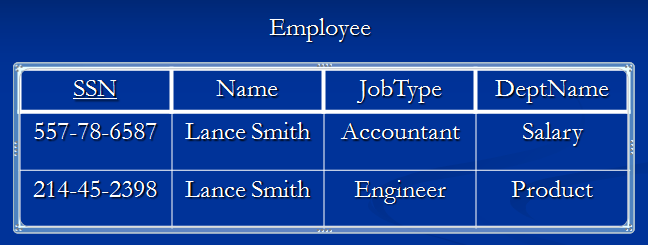
Date of Submission: October 8, 2015

1. **FUNCTIONAL DEPENDENCIES**

**BASIC CONCEPTS**

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.

****

Note: Name is functionally dependent on SSN because an employee’s name can be uniquely determined from their SSN. Name does not determine SSN, because more than one employee can have the same name..

**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*.

*E.g. If A* → *B and B* → *C, then we can infer that A* → *C*

F*+ (closure of F)*

*Set of all functional dependencies logically implied by F*

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.

EXAMPLE:

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

*if* β *⊆ α, then α →* β ***(reflexivity)***

*if α →* β, *then γ α → γ* β ***(augmentation)***

*if α →* β, *and* β *→ γ, then α → γ* ***(transitivity)***

**CLOSURE OF ATTRIBUTE SETS**

For an attriute set , + =*Closure* of under *F* *is the set of attributes that are functionally determined by under F:* → βis in *F*+ β ⊆ +

Algorithm to compute + : *result* := ;  
 while (changes to *result*) do  
 for each β→ γ in *F* do  
 begin  
 if β⊆ *result* then  *result* := *result* ∪ γ   
 end

**EXAMPLE:**

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

F = {A → B  
 A → C   
 CG → H  
 CG → I  
 B → H}

(AG)+

*1.* result = AG

*2.* result = ABCG (A *→* C *and* A *→* B)

*3.* result = ABCGH (CG *→* H *and* CG *⊆* AGBC)

*4.* result = ABCGHI (CG *→* I *and* CG *⊆* AGBCH)

Is *AG* a candidate key?

*Is AG a super key?*

Does *AG* → *R? ==* Is (AG)+ ⊇ R

*Is any subset of AG a superkey?*

Does *A* → *R*? *==* Is (A)+ ⊇ R

Does *G* → *R*? == Is (G)+ ⊇ R

**USES OF CLOSURE OF ATTRIBUTE SETS**

There are several uses of the attribute closure algorithm:

**Testing for superkey:**

*To test if α is a superkey, we compute α+, and check if α+ contains all attributes of* R*.*

**Testing functional dependencies**

To check if a functional dependency α → β holds (or, in other words, is in F+), just check if β ⊆ α+.

That is, we compute α+ by using attribute closure, and then check if it contains β.

Is a simple and cheap test, and very useful

**Computing closure of F**

*For each γ ⊆* R, *we find the closure γ+, and for each* S *⊆ γ+, we output a functional dependency γ →* S.

1. **DECOMPOSITION**

Decomposition is the process of breaking down in parts or elements.

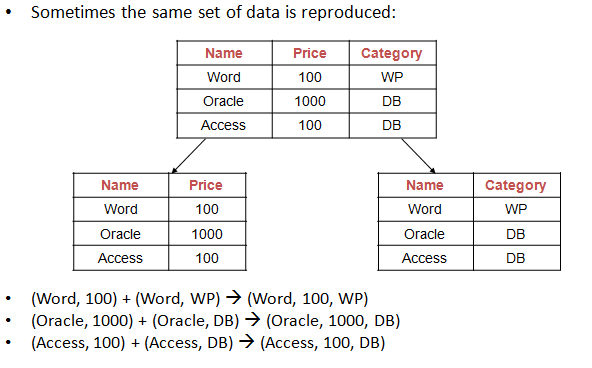
Decomposition in database means breaking tables down into multiple tables.

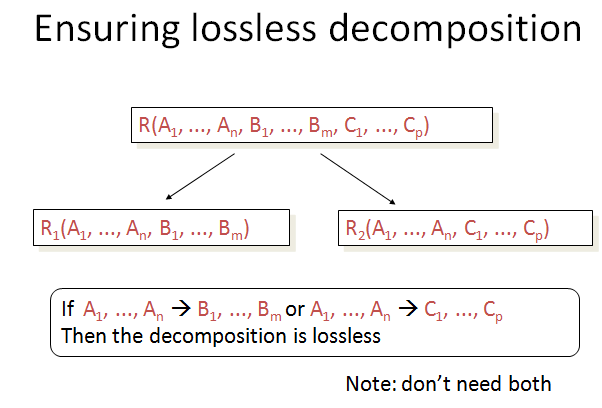
From Database perspective means going to a higher normal form.

**LOSSLESS-JOIN DECOMPOSITIONS**

* Lossless means functioning without a loss.
* In other words, retain everything.
* Important for databases to have this feature.
  + Let { R1 , R2 } be a decomposition of R (meaning that R1 È R2 = R); the decomposition is lossless if for every legal instance r of R:

r = PR1(r) wv PR2(r)

****

****

**DEPENDENCY PRESERVATION**

A decomposition D = {R1, R2, ..., Rn} of R is dependency-preserving with respect to F if the union of the projections of F on each Ri in D is equivalent to F; that is  
 if (*F*1∪ *F*2 ∪ *…* ∪ *F*n )+ = *F +*

**For example:**

If we decompose

R(A;B; C;D) under F = fA ! B;B ! Cg.

into

R1(AB), R2(BC), and R3(AD),

then:

S = fR1; R2; R3g.

F0 = fA ! B, B ! C, (omitting trivial FDs)g

F0+ = F+

Therefore, S is dependency preserving.

If we decompose

R(A;B; C;D) under F = fA ! B;B ! Cg.

into

R1(AB), R2(AC), and R3(AD),

then:

S = fR1; R2; R3g.

F0 = fA ! B, A ! C, (omitting trivial FDs)g

F0+ 6= F+

Therefore, S is not dependency preserving.