ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**TheoryAssignment #10**

**SUBMITTED BY:**

**Apil Neupane**

**013BSCCSIT008**

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **Lecturer** |  |
| **Department of Computer Science** | |

Submission Date: 8th October 2015

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

Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X->Y, which specifies Y is functionally dependent on X. Here X is a determinant set and Y is a dependent attribute. Each value of X is associated precisely with one Y value. i.e.

* X is a determinant
* X determines Y
* Y is functionally dependent on X
* X → Y
* X →Y is trivial if Y ⊆ X

Functional dependency in a database serves as a constraint between two sets of attributes. Defining functional dependency is an important part of relational database design and contributes to aspect normalization.

Let’s consider the relation:

* + **Movie**(title, year, length, filmType, studioName, starName).
  + There are several functional dependencies that we can reasonably assert.
    - title year →length
    - title year →filmType
    - title year →studioName

**shorthand:** title year →length filmType studioName

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.
* 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.

* 1. **Closure of Attribute Sets:**

After finding a set of functional dependencies that are hold on a relation, the next step is to find the Super key for that relation (table). The set of identified functional dependencies play a vital role in finding the key for the relation. We can decide whether an attribute (or set of attributes) of any table is a key for that table or not by identifying the attribute or set of attributes’ closure. If A is an attribute, (or set of attributes) then its attribute closure is denoted as A+.

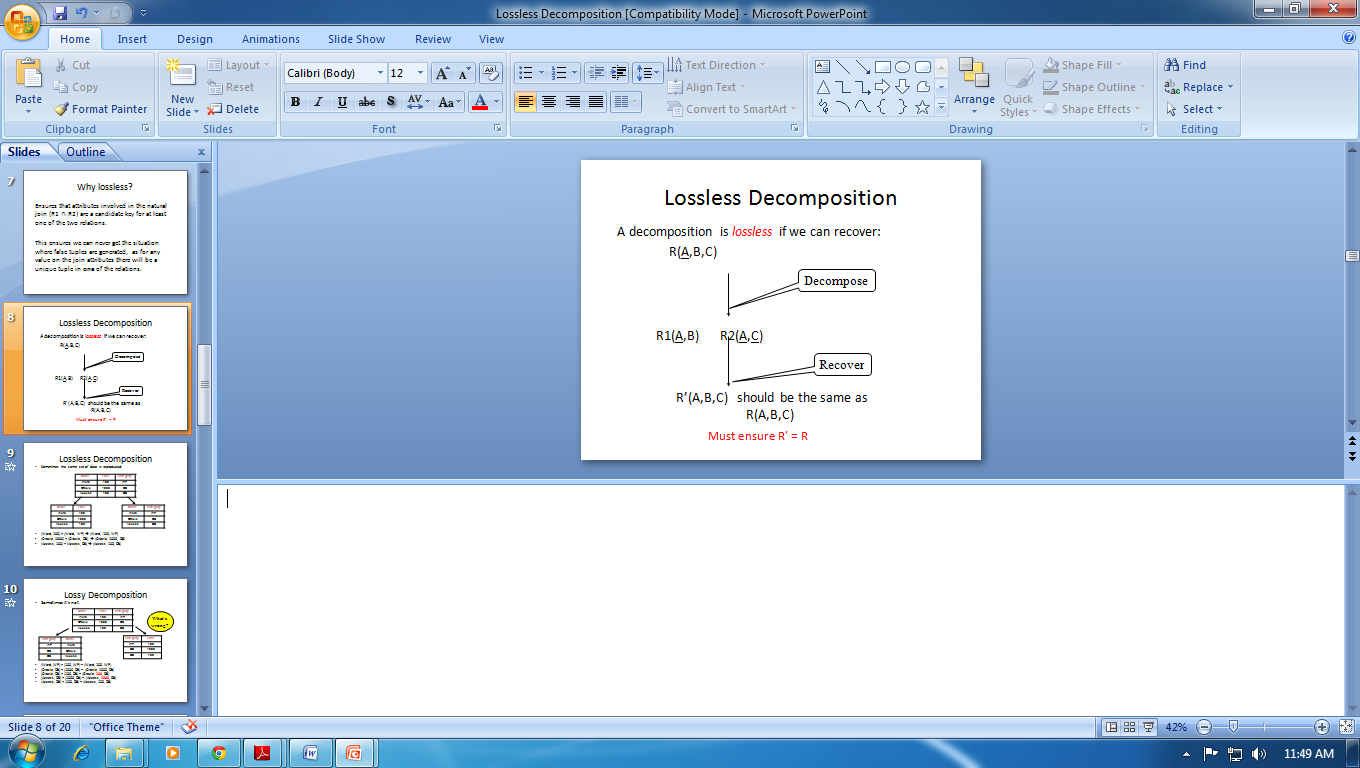
**Algorithm:**

The following algorithm will help us in finding the closure of an attribute;

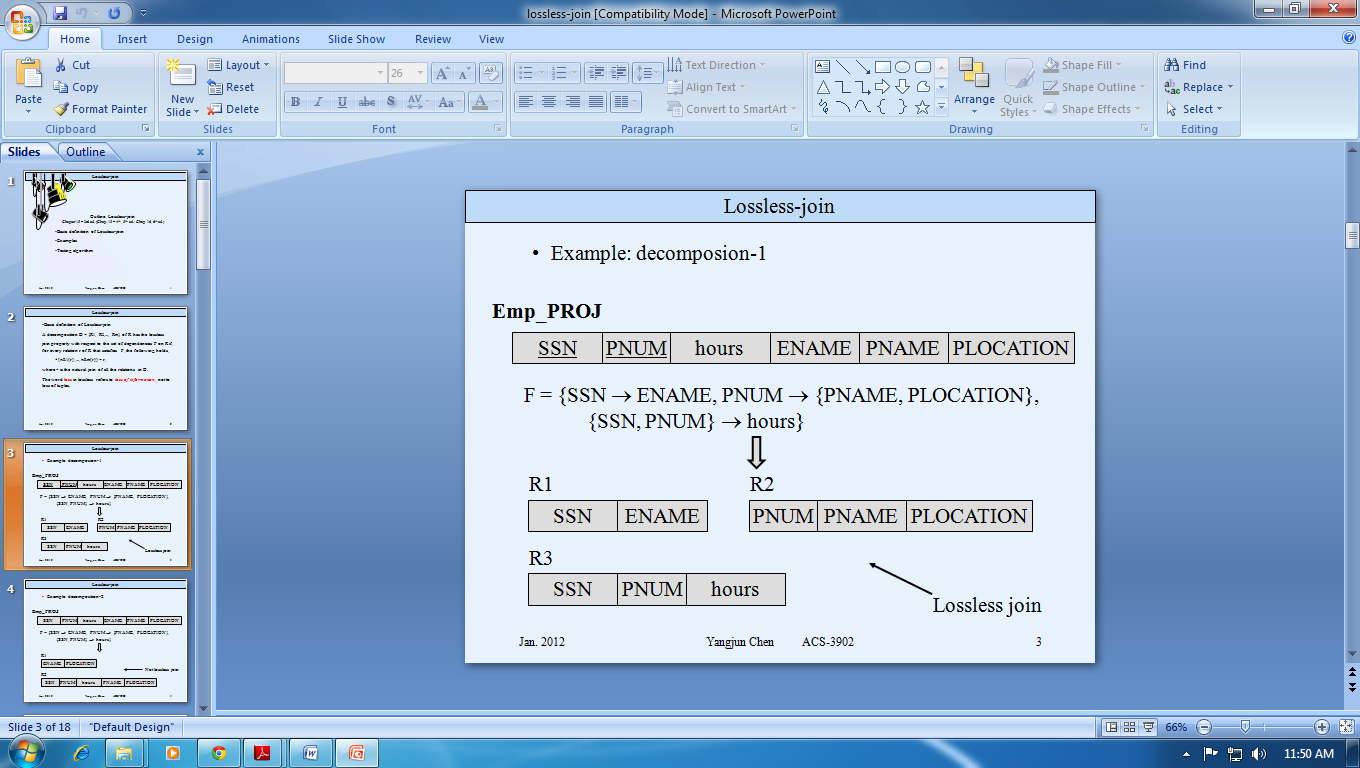
|  |
| --- |
| *result*:= *A*;  **while**(changes to *result*) **do**  **for each**functional dependency B → C**in***F***do**  **begin**  **if**B ⊆ result **then**result := result ∪ C;  **end** |

1. **Decomposition:**

* Decomposition – 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
  1. **Lossless-Join Decomposition:**

****

* Lossless means functioning without a loss.In other words, retain everything.Important for databases to have this feature.
* 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+*
  + 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 we started with.

****

* 1. **Dependency Preservation:**

Dependency preserving Decomposition mean that decomposition should be such that every dependency of relation R must be implied in sub-relations or in combination of dependencies implied in sub relation.

Let R be any relation with FD set F,decomposed into sub-relations R1 and R2 with FD sets F1 and F2respectively.  
Then every dependency of R must be implied in F1 or F2 or in combination of these two.

F1∪F2 = F (dependency must be preserved in decomposition)  
If F1∪F2⊂ F (not dependency preserving decomposition)  
F1∪F2⊃ F (this is not possible)

**Example:** Given relation R(ABCDE) with following FD’s

A→B, B→C, C→D, D→E, D→B  
decomposed into AB, BC, CD, DE  
check whether the decomposition is dependency preserving or not

**Solution:**  In these type of questions dependencies of sub-relations will not be given we have to find them

Find the closure of all attributes of a particular sub-relation from original FD set and then dependencies containing the attributes of that sub-relation will be implied.  
For 1st sub-relation  
A+ = ABCDE  
B+ = BCDE  
so A→B and B→C are dependencies for 1st sub-relation  
for 2nd sub-relation  
B+ = BCDE  
C+ = BCDE  
so B→C and C→B(Redundant) are dependencies for 2nd sub-relation  
For 3rd sub-relation  
C→D  
D→C (redundant)  
For 4th sub-relation  
D→Eall subrelations contain all dependencies of original relation except D→B which can be derive easily from the dependencies of sub relation  
so F1∪F2∪F3∪F4 = F  
hence dependency preservation is satisfied  
so **dependency preserving decomposition.**

**Example:** R(ABCD) has following FD’s

F = {AB→CD, D→A}  
decomposed into D = {ABC, AD,BCD}  
check whether dependency preservation is satisfied or not

**Solution:**

For R1  
AB+ = ABCD so AB→C  
For R2  
A+ = A, D+ = AD  
D→A  
For R3  
BD→C (AB→ABCD D→A so replace A by D so BD→ABCD)But we can’t derive AB→D from dependencies of subrelations  
so **Dependency preservation is not satisfied.**