# ST.XAVIER’S COLLEGE

# MAITIGHAR, KATHMANDU

****

**ASSIGNMENT #10**

**Database Management System**

**Submitted By:**

Pratibha Panta

013BSCCSIT028

4th Sem, 2nd Year

**Submitted To:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Er. Sanjay Kumar Yadav

Department of Computer Science

Lecturer

1. **Functional Dependency**
   1. **Basic Concepts**

FD's are constraints on well-formed relations and represent formalism on the infrastructure of relation.

**Definition:** A *functional dependency* (FD) on a relation schema **R** is a constraint ***X → Y*,** where *X* and *Y* are subsets of attributes of **R.[1]**

**Definition**: an FD is a relationship between an attribute "Y" and a determinant (1 or more other attributes) "X" such that for a given value of a determinant the value of the attribute is uniquely defined.  [1]

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

**Definition**: An FD ***X → Y*** is *satisfied* in an instance **r** of **R** if for every pair of tuples, *t* and s: if *t* and *s* agree on all attributes in *X* then they must agree on all attributes. [1]

A key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:

* *SSN → SSN, Name, Address*
  1. **Closure of set of functional dependency**

We need to consider *all* functional dependencies that hold. Given a set *F* of functional dependencies, we can prove that certain other ones also hold. We say these ones are **logically implied** by *F*.

Suppose we are given a relation scheme *R*=(*A*,*B*,*C*,*G*,*H*,*I*), and the set of functional dependencies:

*A tex2html_wrap_inline1090 B*

*A tex2html_wrap_inline1090 C*

*CG tex2html_wrap_inline1090 H*

*CG tex2html_wrap_inline1090 I*

*B tex2html_wrap_inline1090 H*

Then the functional dependency tex2html_wrap_inline1194is logically implied.

To see why, let tex2html_wrap_inline940and tex2html_wrap_inline946be tuples such that

tex2html_wrap_inline1200

As we are given *A tex2html_wrap_inline1090B* , it follows that we must also have

tex2html_wrap_inline1204

Further, since we also have *B tex2html_wrap_inline1090H* , we must also have

tex2html_wrap_inline1208

Thus, whenever two tuples have the same value on *A*, they must also have the same value on *H*, and we can say that *A tex2html_wrap_inline1090H* .

The **closure** of a set *F* of functional dependencies is the set of all functional dependencies logically implied by *F*.

We denote the closure of *F* by tex2html_wrap_inline1222.

To compute tex2html_wrap_inline1222, we can use some rules of inference called **Armstrong's Axioms**:

**Reflexivity rule:** if tex2html_wrap_inline958is a set of attributes and tex2html_wrap_inline1158, then tex2html_wrap_inline1058holds.

**Augmentation rule:** if tex2html_wrap_inline1058holds, and tex2html_wrap_inline1234is a set of attributes, then tex2html_wrap_inline1236holds.

**Transitivity rule:** if tex2html_wrap_inline1058holds, and tex2html_wrap_inline1240holds, then tex2html_wrap_inline1242holds.

These rules are **sound** because they do not generate any incorrect functional dependencies. They are also **complete** as they generate all of tex2html_wrap_inline1222.

To make life easier we can use some additional rules, derivable from Armstrong's Axioms:

**Union rule:** if tex2html_wrap_inline1058and tex2html_wrap_inline1242, then tex2html_wrap_inline1250holds.

**Decomposition rule:** if tex2html_wrap_inline1250holds, then tex2html_wrap_inline1058and tex2html_wrap_inline1242both hold.

**Pseudotransitivity rule:** if tex2html_wrap_inline1058holds, and tex2html_wrap_inline1260holds, then tex2html_wrap_inline1262holds.

Applying these rules to the scheme and set *F* mentioned above, we can derive the following:

*A tex2html_wrap_inline1090H*, as we saw by the transitivity rule.

*CG tex2html_wrap_inline1090HI* by the union rule.

*AG tex2html_wrap_inline1090I* by several steps:

Note that *A tex2html_wrap_inline1090C* holds.

Then *AG tex2html_wrap_inline1090CG* , by the augmentation rule.

Now by transitivity, *AG tex2html_wrap_inline1090I* .

* 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+.[2]

**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 means replacing a relation with a collection of smaller relations. [5]

* 1. **Lossness Join Decomposition**

In computer science the concept of a Lossless-Join Decomposition is central in removing redundancy safely from databases while preserving the original data.[3]

It can also be called Nonadditive. If you decompose a relation R into relations R\_1 and R\_2 you will guarantee a Lossless-Join if R\_1⋈R\_2 = R.

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.Let R be a relation schema.

Let F be a set of functional dependencies on R.

Let R\_1 and R\_2 form a decomposition of R.

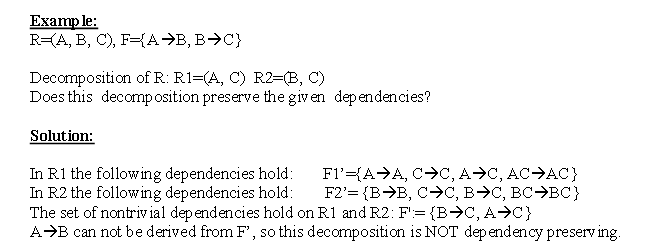
The decomposition is a lossless-join decomposition of R if at least one of the following functional dependencies are in F+ (where F+ stands for the closure for every attribute or attribute sets in F):

R\_1 ∩ R\_2 → R\_1

R\_1 ∩ R\_2 → R\_2

* 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. [4]



**Reference**

[1] <http://jcsites.juniata.edu/faculty/rhodes/dbms/funcdep.htm>

[2] <http://exploredatabase.blogspot.com/2014/08/how-to-find-closure-of-attributes-in-DBMS.html>

[3] <https://en.wikipedia.org/wiki/Lossless-Join_Decomposition>

[4] <http://www.sztaki.hu/~fodroczi/dbs/dep-pres-own.pdf>

[5] <http://www.answers.com/Q/What_is_decomposition_in_dbms>