# ST.XAVIER’S COLLEGE

# MAITIGHAR, KATHMANDU

****

**ASSIGNMENT #10**

**Database Management System**

**Submitted By:**

Prashraya Hada

013BSCCSIT027

4th Sem, 2nd Year

**Submitted To:**

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

Er. Sanjay Kumar Yadav

Department of Computer Science

Lecturer

1. **Functional Dependencies:**
   1. **Basic Concepts**

In relational database theory, a functional dependency is a constraint between two sets of attributes in a relation from a database.

Given a relation R, a set of attributes X in R is said to functionally determine another set of attributes Y, also in R, (written X → Y) if, and only if, each X value is associated with precisely one Y value; R is then said to satisfy the functional dependency X → Y. Equivalently, the projection \pi\_{X,Y}R is a function, i.e. Y is a function of X.[1][2] In simple words, if the values for the X attributes are known (say they are x), then the values for the Y attributes corresponding to x can be determined by looking them up in any tuple of R containing x. Customarily X is called the determinant set and Y the dependent set. A functional dependency FD: X → Y is called trivial if Y is a subset of X.

In other words, a dependency FD: X → Y means that the values of Y are determined by the values of X. Two tuples sharing the same values of X will necessarily have the same values of Y.

A functional dependency is trivial if Y is a subset of X. In a table with attributes of employee name and Social Security number (SSN), employee name is functionally dependent on SSN because the SSN is unique for individual names. An SSN identifies the employee specifically, but an employee name cannot distinguish the SSN because more than one employee could have the same name.

Functional dependency defines Boyce-Codd normal form and third normal form. This preserves dependency between attributes, eliminating the repetition of information. Functional dependency is related to a candidate key, which uniquely identifies a tuple and determines the value of all other attributes in the relation. In some cases, functionally dependent sets are irreducible if:

* The right-hand set of functional dependency holds only one attribute
* The left-hand set of functional dependency cannot be reduced, since this may change the entire content of the set
* Reducing any of the existing functional dependency might change the content of the set

An important property of a functional dependency is Armstrong’s axiom, which is used in database normalization. In a relation, R, with three attributes (X, Y, Z) Armstrong’s axiom holds strong if the following conditions are satisfied:

* Axiom of Transivity: If X->Y and Y->Z, then X->Z
* Axiom of Reflexivity (Subset Property): If Y is a subset of X, then X->Y
* Axiom of Augmentation: If X->Y, then XZ->YZ
  1. **Closure of a set of functional dependencies**
  2. 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*.
  3. 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_inline1194 is logically implied.

* 1. To see why, let **t1** and **t2** be tuples such that

tex2html_wrap_inline1200

As we are given *A tex2html_wrap_inline1090 B*, it follows that we must also have

tex2html_wrap_inline1204

Further, since we also have *B tex2html_wrap_inline1090 H*, 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_inline1090 H*.

* 1. The **closure** of a set *F* of functional dependencies is the set of all functional dependencies logically implied by *F*.
  2. We denote the closure of *F* by tex2html_wrap_inline1222 .
  3. To compute tex2html_wrap_inline1222 , we can use some rules of inference called **Armstrong's**

**Axioms**:

* + - **Reflexivity rule:** if tex2html_wrap_inline958 is a set of attributes and tex2html_wrap_inline1158 , then tex2html_wrap_inline1058 holds.
    - **Augmentation rule:** if tex2html_wrap_inline1058 holds, and tex2html_wrap_inline1234 is a set of attributes, then tex2html_wrap_inline1236 holds.
    - **Transitivity rule:** if tex2html_wrap_inline1058 holds, and tex2html_wrap_inline1240 holds, then tex2html_wrap_inline1242 holds.
  1. 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 .
  2. To make life easier we can use some additional rules, derivable from Armstrong's Axioms:
     + **Union rule:** if tex2html_wrap_inline1058 and tex2html_wrap_inline1242 , then tex2html_wrap_inline1250 holds.
     + **Decomposition rule:** if tex2html_wrap_inline1250 holds, then tex2html_wrap_inline1058 and tex2html_wrap_inline1242 both hold.
     + **Pseudotransitivity rule:** if tex2html_wrap_inline1058 holds, and tex2html_wrap_inline1260 holds, then tex2html_wrap_inline1262 holds.
  3. Applying these rules to the scheme and set *F* mentioned above, we can derive the following:
     + *A tex2html_wrap_inline1090 H*, as we saw by the transitivity rule.
     + *CG tex2html_wrap_inline1090 HI*by the union rule.
     + *AG tex2html_wrap_inline1090 I*by several steps:
       - * Note that *A tex2html_wrap_inline1090 C*holds.
         * Then *AG tex2html_wrap_inline1090 CG*, by the augmentation rule.
         * Now by transitivity, *AG tex2html_wrap_inline1090 I*.
  4. **Closure of attribute sets**

1. To test whether a set of attributes tex2html_wrap_inline958 is a superkey, we need to find the set of attributes functionally determined by tex2html_wrap_inline958 .
2. Let tex2html_wrap_inline958 be a set of attributes. We call the set of attributes determined by tex2html_wrap_inline958 under a set *F* of functional dependencies the **closure** of tex2html_wrap_inline958 under *F*, denoted tex2html_wrap_inline1292 .
3. The following algorithm computes tex2html_wrap_inline1292 :

*result* := tex2html_wrap_inline958

tex2html_wrap_inline1240 **while** (changes to *result*) **do**

**for each** functional dependency

**in**  *F* **do**

tex2html_wrap_inline1302 **begin**

**if**  *result*

**then** result := *result* tex2html_wrap_inline1304 ;

**end**

1. If we use this algorithm on our example to calculate tex2html_wrap_inline1306 then we find:

* We start with *result* = AG.
* *A tex2html_wrap_inline1090 B*causes us to include B in *result*.
* *A tex2html_wrap_inline1090 C*causes *result* to become ABCG.
* *CG tex2html_wrap_inline1090 H*causes *result* to become ABCGH.
* *CG tex2html_wrap_inline1090 I*causes *result* to become ABCGHI.
* The next time we execute the while loop, no new attributes are added, and the algorithm terminates.

e) This algorithm has worst case behavior quadratic in the size of *F*. There is a linear algorithm that is more complicated.

1. **Decomposition**
   1. **Lossless – Join Dependencies**

Can also be called Non additive. 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.[[1]](https://en.wikipedia.org/wiki/Lossless-Join_Decomposition#cite_note-1) Let R be a relation schema.

Let F be a set of [functional dependencies](https://en.wikipedia.org/wiki/Functional_dependency) 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):[[2]](https://en.wikipedia.org/wiki/Lossless-Join_Decomposition#cite_note-2)

* R_1 ∩ R_2 → R_1
* R_1 ∩ R_2 → R_2

Example

* Let R = (A, B, C, D) be the relation schema, with A, B, C and D attributes.
* Let F = \{ A \rightarrow BC \} be the set of functional dependencies.
* Decomposition into R_1 = (A, B, C) and R_2 = (A, D) is lossless under F because R_1 \cap R_2 = (A), A is a super key in R_1 ( A \rightarrow BC ) so R_1 \cap R_2 \rightarrow R_1.
  1. **Dependency Preservation:**

A decomposition D = {R1, …, Rm} of R is dependency-preserving with respect to 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