# ST.XAVIER’S COLLEGE

# MAITIGHAR, KATHMANDU

****

**ASSIGNMENT ON**

**Database Management System**

**Submitted By:**

Shaurabh Chapagain

013BSCCSIT037

**Submitted To:**

Er. Sanjay Kumar Yadav

Lecturer

Department of Computer Science

St. Xavier’s College

**Functional dependency**

**Basic concept:**

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.

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.

Examples:

*Bear Number determines Student Name:*

*BearNum ---> StuName*

*Department Number and Job Rank determine Security Clearance:*

*(DeptNum, JRank) --->SecClear*

*Social Security Number determines Employee Name and Salary:*

*SSN ---> (EmpName, Salary)*

*Additionally, the above can be read as:*

*SSN --->EmpName and SSN Salary*

**Closure of a Set of Functional Dependencies**

1. Let us 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.
2. 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   tex2html_wrap_inline940 and tex2html_wrap_inline946  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.
4. 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 .
5. 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.
6. 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 .

## 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*

*while (changes to* result*) do*

*for each functional dependency tex2html_wrap_inline1240*

*in F do*

*begin*

*if tex2html_wrap_inline1302* result

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

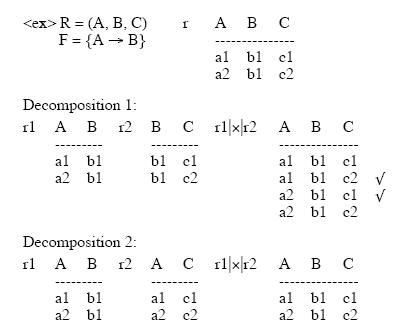
*end*

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

**Decomposition**

Decomposition means replacing a relation with a collection of smaller relations. Selecting all data without any grouping and aggregate functions is called Decomposition. The data is selected, as it is present in the table.

**Example:**



**Lossless Join Decomposition:**

Definition:

Let { R1, R2 } be a decomposition of R (R1 U R2 = R); the decomposition is lossless if for every legal instance r of R:

r = ΠR1(r) |X| ΠR2(r)

**Testing Lossless Join:**

* Lossless join property is necessary if the decomposed relation is to be recovered from its decomposition.
* Let R be a schema and F be a set of FD’s on R, and α = (R1, R2) be a decomposition of R. Then α has a lossless join with respect to F iff

R1 ∩ R2 -> R1 (or R1 - R2 ) or R2 ∩ R1 -> R2 (or R2 - R1 ) where such FD exist in Closure of F.

PS This is a sufficient condition, but not a necessary condition.

**Example:**

From the previous example : R = (ABC) F = {A -> B}

R1 = (AB), R2 = (AC)

R1∩ R2 = A, R1- R2 = B

check A -> B in F ? Yes. Therefore lossless

R1 = (AB), R2 = (BC)

R1∩ R2 = B, R1 - R2 = A , R2 - R1= C

check B -> A in F ? NO

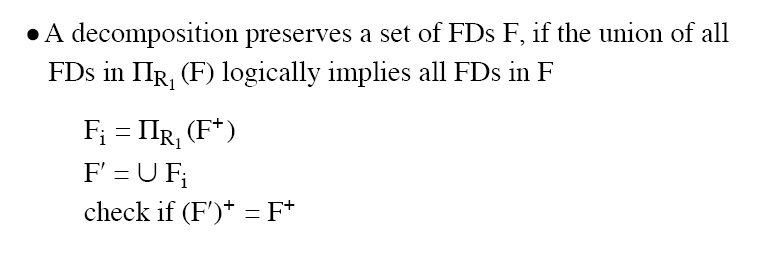
check B -> C in F ? NO

So, this is lossy join.

**Dependency Preservation:**

Definition:

* Each FD specified in F either appears directly in one of the relations in the decomposition, or be inferred from FDs that appear in some relation.



**Test of Dependency Preservation:**

* If a decomposition is not dependency-preserving, some dependency is lost in the decomposition.
* One way to verify that a dependency is not lost is to take joins of two or more relations in the decomposition to get a relation that contains all of the attributes in the dependency under consideration and then check that the dependency holds on the result of the joins.
* Find *F* - *F*', the functional dependencies not checkable in one relation.
* See whether this set is obtainable from *F*' by using Armstrong's Axioms.
* This should take a great deal less work, as we have (usually) just a few functional dependencies to work on.

**Example:**

* Consider relation ABCD, with FD’s :

A ->B, B ->C, C ->D

* Decompose into two relations: ABC and CD.
* ABC supports the FD’s A->B, B->C.
* CD supports the FD C->D.
* All the original dependencies are preserved.