**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Lab Assignment #10**

**Submitted by:**

Hemanchal Joshi  
013BSCCSIT021

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav**  **Lecturer**  **St. Xavier’s College** |  |

# Date of Submission: Thursday, October 08, 2015

1. **Functional Dependencies**

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.

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
  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. 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.

Functional Dependency is the starting point for the process of normalization. Functional dependency exists when a relationship between two attributes allows you to uniquely determine the corresponding attribute’s value. If ‘X’ is known, and as a result you are able to uniquely identify ‘Y’, there is functional dependency. Combined with keys, normal forms are defined for relations.

**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

* 1. **Closure of set of functional dependencies**

 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_inline1194 is logically implied.

 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*.

 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_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.

 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 .

* 1. **Closure of attribute sets]**
* 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 .
* 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 .
* 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

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.

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

**Proofs through Closure of Attributes**

* The closure *X*+ of *X*under *F*is the set { attributes inferred from *X*}.
* Observation: *X*→ *Y*if and only if *Y*⊂ *X*+
* Algorithm:

*X*+ := *X*  
while *Y*→ *Z*in *F*  
      such that *Y*⊂ *X*+   
      and not *Z*⊂ *X*+   
do  
           *X*+ := *X*+ ∪ *Z*  
end

1. **Decomposition**

 The previous example might seem to suggest that we should decompose schema as much as possible.

Careless decomposition, however, may lead to another form of bad design.

 Consider a design where *Lending-schema* is decomposed into two schemas

Branch-customer-schema = (bname, bcity, assets, cname)

Customer-loan-schema = (cname, loan#, amount)

 We construct our new relations from *lending* by:

branch-customer = tex2html_wrap_inline1540

customer-loan = tex2html_wrap_inline1542

* 1. **Lossless-Join Dependencies**

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.[[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.

* 1. **Dependency preservation**

Dependency Preservation A FD X → Y is preserved in a relation R if R contains all the attributes of X and Y . A FD can therefore be checked by accessing only R. Example. In the previous slide: A → B is preserved in R1. B → C is not preserved in any relation. Dependency Preservation Let us revisit the scenario of decomposing R(A, B, C, D) under F = {A → B, B → C}. Consider the following decomposed tables: R1(AB), R2(BC), and R3(AD) all of which are in BCNF. This decomposition is better than the previous one because: Both A → B and B → C are preserved. Hence, each can be checked in one table (thus avoiding joins, which are typically slow).

Let: S be the set of tables in our final design. F be the set of FDs we have collected from the underlying application. F 0 be the set of FDs each of which is preserved in at least one table in S. Definition Our design S is dependency preserving if F 0+ = F +. In other words, by checking only the FDs in F 0 , we effectively have checked the entire F +. Dependency Preservation Example 1 If we decompose R(A, B, C, D) under F = {A → B, B → C}. into R1(AB), R2(AC), and R3(AD), then: S = {R1, R2, R3}. F 0 = {A → B, A → C, (omitting trivial FDs)} F 0+ 6= F + Therefore, S is not dependency preserving.

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.

**Example: R=(A, B, C), F={AÆB, BÆC} Decomposition of R: R1=(A, C) R2=(B, C) Does this decomposition preserve the given dependencies?**

Solution: In R1 the following dependencies hold: F1’={AÆA, CÆC, AÆC, ACÆAC} In R2 the following dependencies hold: F2’= {BÆB, CÆC, BÆC, BCÆBC} The set of nontrivial dependencies hold on R1 and R2: F':= {BÆC, AÆC} AÆB can not be derived from F’, so this decomposition is NOT dependency preserving.