**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**DATABASE MANAGEMENT SYSTEM**

THEORY ASSIGNMENT#10

**Submitted by:**

**Shreesha Pokharel**

013BSCCSIT038

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav** |  |

Lecturer

Department of Computer Science

Date of submission: 8th October, 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. 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.

* 1. **Basic concepts**

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

* 1. **Closure of a 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 .

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.

**Pseudo transitivity rule:** if tex2html_wrap_inline1058 holds, and tex2html_wrap_inline1260 holds, then tex2html_wrap_inline1262 holds.

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*.

* 1. **Closure of attribute sets**

To test whether a set of attributes tex2html_wrap_inline958 is a super key, 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.

1. **Decomposition**

Let R be a relation schema. A set of relation schemas is a **decomposition** of R if. That is, every attribute in R appears in at least one for. Let r be a relation on R, and let. That is, is the **database** that results from **decomposing** R into.

* 1. **Lossless-join decomposition**

(R1, R2) is a lossless-join decomposition of R with respect to a set of FDs F if for every instance r of R that satisfies F:

πR1 (r) ✶ πR2 (r) = r.

A simple criterion for checking whether decomposition (R1, R2) is lossless-join:

R1 ∩ R2 → R1 ∈ F +, or

R1 ∩ R2 → R2 ∈ F +.

A sequence of decompositions of R into R1 and R2, R1 into R 0 1 and R 00 1 etc. may be viewed as a decomposition of R into more than two relation schemas.

* 1. **Dependency preservation**

Dependencies associated with relation schema R1 and R2 in decomposition (R1, R2):

FR1 = {X → Y |X → Y ∈ F + ∧ XY ⊆ R1}

FR2 = {X → Y |X → Y ∈ F + ∧ XY ⊆ R2}.

(R1, R2) preserves a dependency f iff f ∈ (FR1 ∪ FR2) +.