**ST. XAVIER’S COLLEGE**

**MAITIGHAR, KATHMANDU**

****

**Database Management System**

**Theory Assignment #10**

**Submitted by:**

Anu Kadel

013BSCIT007

**Submitted to:**

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

Lecturer

Department of Computer Science

St. Xavier’s College, Maitighar

**[1]FUNCTIONAL DEPENDENCIES**

* 1. **BASIC CONCEPTS**

Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

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 (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.

* Functional dependencies are a constraint on the set of legal relations in a database.
* They allow us to express facts about the real world we are modeling.
* The notion generalizes the idea of a superkey.
* Let tex2html_wrap_inline1054and tex2html_wrap_inline1056.
* Then the functional dependency tex2html_wrap_inline1058holds on *R* if in any legal relation *r*(*R*), for all pairs of tuples tex2html_wrap_inline940and tex2html_wrap_inline946in *r* such that tex2html_wrap_inline1070, it is also the case that tex2html_wrap_inline1072.
* Using this notation, we say *K* is a superkey of *R* if tex2html_wrap_inline1078.
* In other words, *K* is a superkey of *R* if, whenever tex2html_wrap_inline1084, then tex2html_wrap_inline1086(and thus tex2html_wrap_inline1088).
  1. **CLOSURE OF A SET OF FUNCTIONAL DEPENDENCIES**

Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F .

* The set of all functional dependencies logically implied by F is the closure of F .
* We denote the closure of F by F +.

Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

* **Reflexive rule** − If alpha is a set of attributes and beta is\_subset\_of alpha, then alpha holds beta.
* **Augmentation rule** − If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
* **Transitivity rule** − Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.
* 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
  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 suppose 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**

**We use functional dependencies to:**

* test relations to see if they are legal under a given set of functional dependencies. If a relation r is legal under a set F of functional dependencies, we say that r satisfies F .
* specify constraints on the set of legal relations; we say that F holds on R if all legal relations on R satisfy the set of functional dependencies F .

**[2] DECOMPOSITION**

Decomposition means replacing a relation with a collection of smaller relations.

**2.1 LOSSLESS-JOB DECOMPOSITION**

If R is decomposed into R1 and R2, we require that for all possible relations r on schema R satisfies r = ΠR1(r ) ΠR2(r )

A decomposition of R into R1 and R2 is lossless joinif and only if at least one of the following dependencies is in F+–R1 ∩R2 →R1–R1 ∩R2 →R2

Example

*R = (A, B, C), F = {A* →*B, B* →*C)*

–Can be decomposed in two different ways

*R*1*= (A, B), R*2*= (B, C)*

–Lossless-join decomposition:*R*1 ∩*R*2*=* {*B*}and *B* →*BC*

–Dependency preserving: *R1 = (A, B), R*2*= (A, C)*

–Lossless-join decomposition:*R*1 ∩*R*2*=*{*A*}and *A* →A*B*

–Not dependency preserving (cannot check *B* →*C* without computing *R1 R*2)

**2.2 DEPENDENCY PRESERVATION**

A desirable property in database design is dependency preservation. We would like to check easily that updates to the database do not result in illegal relations being created. It would be nice if our design allowed us to check updates without having to compute natural joins

Let Fibe the set of dependencies F+that includes only attributes in Ri

* A decomposition is dependency preserving, if (F1∪F2∪…∪Fn)+= F+
* If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

**Testing Dependency Preservation**

* To check if a dependency α→β is preserved in a decomposition of *R*into *R*1, *R*2, …, *R*n

*result* = α

**repeat**

**for each** *Ri* in the decomposition *t*= (*result* ∩*Ri*)+ ∩*Ri, result = result* ∪*t*

**until** *result* does not change

* If *result* contains all attributes in β, then the functional dependency

α→β is preserved

* We apply the test on all dependencies in *F* to check if a decomposition is dependency preserving
* This procedure takes polynomial time

**Example**

* R = (A, B, C )

F = {A →B, B →C}

Key = {A}

* R is not in BCNF
* Decomposition R1 = (A, B), R2 = (B, C)