ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Theory Assignment #10**

**Submitted by:**

Pankaj Niroula  
013BSCCSIT025

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay K. Yadav  Lecturer Department of Computer Science |  |

**Date of Submission: Oct 08, 2015**

1. **Functional Dependencies**
   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.

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**

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*

tex2html_wrap_inline1194Then the functional dependency  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.

**Pseudotransitivity 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.

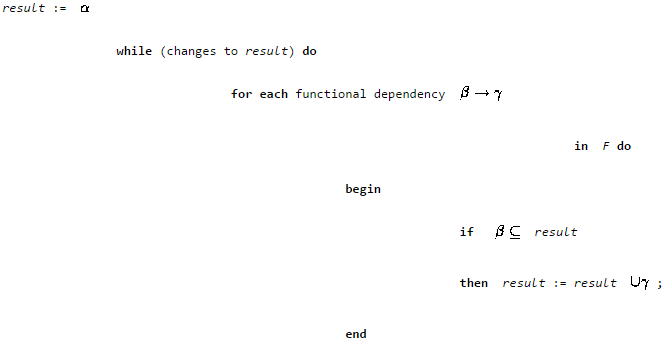
Now by transitivity, *AG tex2html_wrap_inline1090 I*.

* 1. **Closure of Attribute Sets**

To test whether a set of attributes tex2html_wrap_inline958 is a 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 :



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**
   1. **Lossless Join Decomposition**

We claim the above decomposition is lossless. How can we decide whether decomposition is lossless?

Let *R* be a relation schema.

Let *F* be a set of functional dependencies on *R*.

Let tex2html_wrap_inline1620 and tex2html_wrap_inline1622 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 tex2html_wrap_inline1628 :

* 1. tex2html_wrap_inline1630
  2. tex2html_wrap_inline1632

Why is this true? Simply put, it ensures that the attributes involved in the natural join ( tex2html_wrap_inline1634 ) are a candidate key for at least one of the two relations.

This ensures that we can never get the situation where spurious tuples are generated, as for any value on the join attributes there will be a unique tuple in **one** of the relations.

We'll now show our decomposition is lossless-join by showing a set of steps that generate the decomposition:

First we decompose *Lending-schema* into

*Branch-schema = (bname, bcity, assets)*

*Loan-info-schema = (bname, cname, loan#, amount)*

Since *bname* tex2html_wrap_inline1526 *assets bcity*, the augmentation rule for functional dependencies implies that

*bname* tex2html_wrap_inline1526 *bname assets bcity*

Since *Branch-schema* tex2html_wrap_inline1640 *Borrow-schema* = *bname*, our decomposition is lossless join.

Next we decompose *Borrow-schema* into

*Loan-schema = (bname, loan#, amount)*

*Borrow-schema = (cname, loan#)*

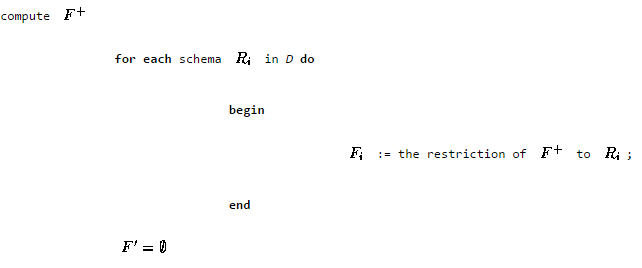
As *loan#* is the common attribute, and

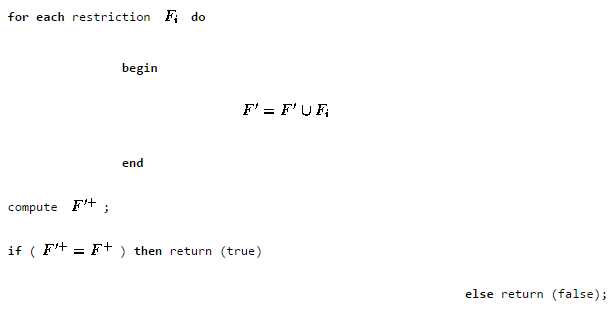
*loan#* tex2html_wrap_inline1526 *amount bname*

This is also a lossless-join decomposition.

* 1. **Dependency Preservation**

1. Another 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.
   * To know whether joins must be computed, we need to determine what functional dependencies may be tested by checking each relation individually.
   * Let *F* be a set of functional dependencies on schema *R*.
   * Let tex2html_wrap_inline1550 be a decomposition of *R*.
   * The **restriction** of *F* to tex2html_wrap_inline1556 is the set of all functional dependencies in tex2html_wrap_inline1628 that include only attributes of tex2html_wrap_inline1556 .
   * Functional dependencies in a restriction can be tested in one relation, as they involve attributes in one relation schema.
   * The set of restrictions tex2html_wrap_inline1660 is the set of dependencies that can be checked efficiently.
   * We need to know whether testing only the restrictions is sufficient.
   * Let tex2html_wrap_inline1662 .
   * *F*' is a set of functional dependencies on schema *R*, but in general, tex2html_wrap_inline1668 .
   * However, it may be that tex2html_wrap_inline1670 .
   * If this is so, then every functional dependency in *F* is implied by *F*', and if *F*' is satisfied, then *F* must also be satisfied.
   * A decomposition having the property that tex2html_wrap_inline1670 is a **dependency-preserving** decomposition.
2. The algorithm for testing dependency preservation follows this method:





1. We can now show that our decomposition of *Lending-schema* is dependency preserving.
   * The functional dependency
   * *bname* tex2html_wrap_inline1526 *assets bcity*

can be tested in one relation on *Branch-schema*.

* + The functional dependency
  + *loan#* tex2html_wrap_inline1526 *amount bname*

can be tested in *Loan-schema*.

1. As the above example shows, it is often easier not to apply the algorithm shown to test dependency preservation, as computing tex2html_wrap_inline1628 takes exponential time.
2. **An Easier Way To Test For Dependency Preservation**

Really we only need to know whether the functional dependencies in *F* and not in *F*' are implied by those in *F*'.

In other words, are the functional dependencies not easily checkable logically implied by those that are?

Rather than compute tex2html_wrap_inline1628 and tex2html_wrap_inline1700 , and see whether they are equal, we can do this:

* + 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.