**ST. XAVIER’S COLLEGE**

**Maitighar, Kathmandu**



**Data Base Management System**

**Theory Assignment #10**

**Submitted by**

Urendra Khakurel

013BScCSIT045 (4th Semester)

**Submitted to**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  (Lecturer, St. Xavier’s College ) |  |

**1. Functional dependencies**

FD's are constraints on well-formed relations and represent a formalism on the infrastructure of relation.

**Definition:** A *functional dependency* (FD) on a relation schema **R** is a constraint ***X → Y*,** where *X* and *Y* are subsets of attributes of **R.**

**Definition**: an FD is a relationship between an attribute "Y" and a determinant (1 or more other attributes) "X" such that for a given value of a determinant the value of the attribute is uniquely defined.

* X is a determinant
* X determines Y
* Y is functionally dependent on X
* X → Y
* X →Y is trivial if Y ⊆ X

## Basic Concepts

* + 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).

Functional dependencies allow us to express constraints that cannot be expressed with superkeys.

Consider the scheme

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

if a loan may be made jointly to several people (e.g. husband and wife) then we would not expect loan# to be a superkey. That is, there is no such dependency

loan# *tex2html_wrap_inline1090* cname

We do however expect the functional dependency

loan# *tex2html_wrap_inline1090* amount

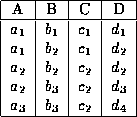
loan# *tex2html_wrap_inline1090* bname

to hold, as a loan number can only be associated with one amount and one branch.

A set *F* of functional dependencies can be used in two ways:

* + To specify constraints on the set of legal relations. (Does *F* hold on *R*?)
  + To test relations to see if they are legal under a given set of functional dependencies. (Does *r* satisfy *F*?)

Figure shows a relation *r* that we can examine.

    
**Figure**   Sample relation *r*.

We can see that tex2html_wrap_inline1150is satisfied (in this particular relation), but tex2html_wrap_inline1152is not. tex2html_wrap_inline1154is also satisfied.

Functional dependencies are called **trivial** if they are satisfied by all relations.

In general, a functional dependency tex2html_wrap_inline1058is trivial if tex2html_wrap_inline1158.

In the customer relation of figure 5.4, we see that tex2html_wrap_inline1160is satisfied by this relation. However, as in the real world two cities can have streets with the same names (e.g. Main, Broadway, etc.), we would not include this functional dependency in our list meant to hold on Customer-scheme.

The list of functional dependencies for the example database is:

On Branch-scheme:

bname *tex2html_wrap_inline1090* bcity

bname *tex2html_wrap_inline1090* assets

On Customer-scheme:

cname *tex2html_wrap_inline1090* ccity

cname *tex2html_wrap_inline1090* street

On Loan-scheme:

loan# *tex2html_wrap_inline1090* amount

loan# *tex2html_wrap_inline1090* bname

On Account-scheme:

account# *tex2html_wrap_inline1090* balance

account# *tex2html_wrap_inline1090* bname

There are no functional dependencies for Borrower-schema, nor for Depositor-schema.

## 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 functionaldependencies:

*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_inline1194is logically implied.

To see why, let tex2html_wrap_inline940and tex2html_wrap_inline946be tuples such that

tex2html_wrap_inline1200

As we are given *A tex2html_wrap_inline1090B* , it follows that we must also have

tex2html_wrap_inline1204

Further, since we also have *B tex2html_wrap_inline1090H* , 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_inline1090H* .

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_inline958is a set of attributes and tex2html_wrap_inline1158, then tex2html_wrap_inline1058holds.
  + **Augmentation rule:** if tex2html_wrap_inline1058holds, and tex2html_wrap_inline1234is a set of attributes, then tex2html_wrap_inline1236holds.
  + **Transitivity rule:** if tex2html_wrap_inline1058holds, and tex2html_wrap_inline1240holds, then tex2html_wrap_inline1242holds.

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_inline1058and tex2html_wrap_inline1242, then tex2html_wrap_inline1250holds.
  + **Decomposition rule:** if tex2html_wrap_inline1250holds, then tex2html_wrap_inline1058and tex2html_wrap_inline1242both hold.
  + **Pseudotransitivity rule:** if tex2html_wrap_inline1058holds, and tex2html_wrap_inline1260holds, then tex2html_wrap_inline1262holds.

Applying these rules to the scheme and set *F* mentioned above, we can derive the following:

* + *A tex2html_wrap_inline1090H*, as we saw by the transitivity rule.
  + *CG tex2html_wrap_inline1090HI* by the union rule.
  + *AG tex2html_wrap_inline1090I* by several steps:
    - Note that *A tex2html_wrap_inline1090C* holds.
    - Then *AG tex2html_wrap_inline1090CG* , by the augmentation rule.
    - Now by transitivity, *AG tex2html_wrap_inline1090I* .

(You might notice that this is actually pseudotransivity if done in one step.)

## Closure of Attribute Sets

To test whether a set of attributes tex2html_wrap_inline958is a superkey, we need to find the set of attributes functionally determined by tex2html_wrap_inline958.

Let tex2html_wrap_inline958be a set of attributes. We call the set of attributes determined by tex2html_wrap_inline958under a set *F* of functional dependencies the **closure** of tex2html_wrap_inline958under *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_inline1306then we find:

* + We start with *result* = AG.
  + *A tex2html_wrap_inline1090B* causes us to include B in *result*.
  + *A tex2html_wrap_inline1090C* causes *result* to become ABCG.
  + *CG tex2html_wrap_inline1090H* causes *result* to become ABCGH.
  + *CG tex2html_wrap_inline1090I* 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.

**2 .Decomposition**

**Lossless-Join Decomposition**

1. We claim the above decomposition is lossless. How can we decide whether a decomposition is lossless?
   * Let *R* be a relation schema.
   * Let *F* be a set of functional dependencies on *R*.
   * Let tex2html_wrap_inline1620and tex2html_wrap_inline1622form 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.

**Dependency Preservation**

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_inline1550be a decomposition of *R*.
  + The **restriction** of *F* to tex2html_wrap_inline1556is the set of all functional dependencies in tex2html_wrap_inline1628that 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_inline1660is 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_inline1670is a **dependency-preserving** decomposition.

The algorithm for testing dependency preservation follows this method:

compute tex2html_wrap_inline1628

**for each** schema tex2html_wrap_inline1556 in *D* **do**

**begin**

tex2html_wrap_inline1688:= the restriction of tex2html_wrap_inline1628 to tex2html_wrap_inline1556 ;

**end**

tex2html_wrap_inline1694

**for each** restriction tex2html_wrap_inline1688 **do**

**begin**

tex2html_wrap_inline1698

**end**

compute tex2html_wrap_inline1700 ;

**if** ( tex2html_wrap_inline1670 ) **then** return (true)

**else** return (false);

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*.

As the above example shows, it is often easier not to apply the algorithm shown to test dependency preservation, as computing tex2html_wrap_inline1628takes exponential time.

**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_inline1628and 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.

Use this simpler method on exams and assignments (unless you have exponential time available to you).