ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System Assignment**

**Theory assignment #10**

**Submitted by:**

Arjun Neupane  
013BSCCSIT009

**Submitted to:**

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

**1.Functional Dependencies:**

**1.1.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.

Example:

Let R be   
**NewStudent**(***stuId, lastName, major, credits, status, socSecNo***)

FDs in R include

* ***{stuId}→{lastName}***, but not the reverse
* ***{stuId} →{lastName, major, credits, status, socSecNo, stuId}***
* ***{socSecNo} →{stuId, lastName, major, credits, status, socSecNo}***
* ***{credits}→{status}*,** but not ***{status}→{credits}***

*ZipCode****→****AddressCity*

* 16652 is Huntingdon’s ZIP

*ArtistName→BirthYear*

* Picasso was born in 1881

*Autobrand→Manufacturer*, *Engine type*

* Pontiac is built by General Motors with gasoline engine

*Author, Title→PublDate*

* Shakespeare’s Hamlet was published in 1600

**1.2. 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.
  + **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*.

**1.3. 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.

**2. Decomposition:**

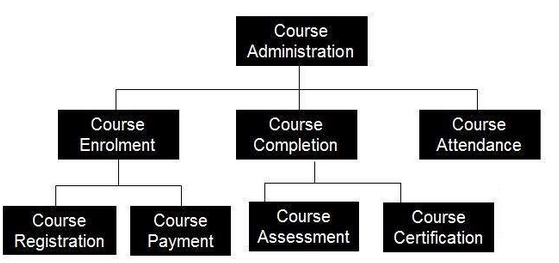
Functional decomposition corresponds to the various functional relationships as how the original complex business function was developed. It mainly focuses on how the overall functionality is developed and its interaction between various components.

Large or complex functionalities are more easily understood when broken down into pieces using functional decomposition.

**How and why?**

* Functional decomposition is mostly used during the project analysis phase in order to produce functional decomposition diagrams as part of the functional requirements document.
* Functional Decomposition is done after meeting with business analysts and subject matter expertise.
* Decompose the first level components with their functions and continue to decompose to lower levels until sufficient level of detail is achieved
* Perform an end-to-end walk-through of the business operation and check each function to confirm that it is correct.

Examples: The below example, best describes the Functional Decomposition:



**2.1. Lossless-Join Dependencies**

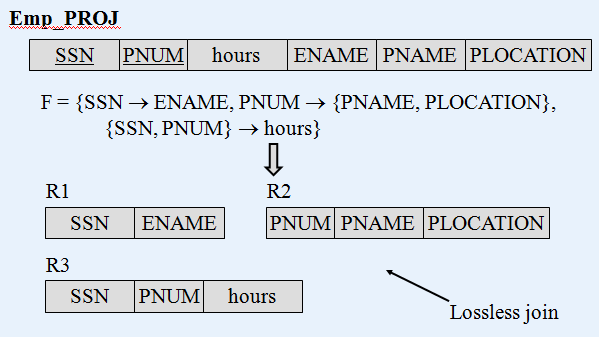
A decomposition D = {R1, R2,..., Rm} of R has the lossless join property with respect to the set of dependencies F on R if, for every relation r of R that satisfies F, the following holds,

\*(πR1(r), ..., πRm(r)) = r,

where \* is the natural join of all the relations in D.

The word loss in lossless refers to *loss of information*, not to loss of tuples.

Example:

****

**2.2. 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_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.
* 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_inline1628 takes 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_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.

**Reference:**

<http://jcsites.juniata.edu/faculty/rhodes/dbms/funcdep.htm>

<http://databasemanagement.wikia.com/wiki/Functional_Dependencies>