IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 5, Ver. I (Sep.- Oct. 2017), PP 77-81
www.iosrjournals.org

Functional Dependency: Design and Implementation of a Minimal Cover Algorithm

*A.B Amure[1], M.A. Amosu[2], H.O.D. Longe[3]
[1]Research Student, University of Ilorin, Nigeria
[2]Network Administrator, ITD, University College Hospital, Ibadan, Nigeria
[3]Professor, University of Lagos, Nigeria
Corresponding Author: A.B Amure

Abstract: *Redundancy is the root of several problems in a relational schema. Redundancy simply means repetition of information. In minimizing the set of Functional Dependencies (FDs), database will be made consistent and correct. This paper focuses on solving redundancy problem of Functional Dependencies in a relational schema using the closure of Functional dependency and minimal cover algorithm. An algorithm was designed using the three Armstrong's axioms which are reflexivity, augmentation and transitivity and implemented to generate closure of functional dependencies (FDs) called F+. Other rules used are decomposition, union and pseudo-transitivity, which are products of the three basic axioms.The attribute closure algorithm will also generate the attribute closure X+ under any given set of FDs. Minimal cover (G+) algorithm is then designed and implemented to eliminate redundant FDs. This save storage and optimize the database by eliminating unneeded attributes in FDs*

Keywords: *Functional Dependency, Minimal Cover, Database redundancy, Relational schema, Closure*

Date of Submission: 30-08-2017 Date of acceptance: 13-09-2017

I. Introduction

Functional dependency (FD) is one of the main concepts associated with normalization which describes the relationship between attributes [3]. In the designing of databases, FD is important in relational model and database normalization. FDs and attribute domains are selected in order to generate constraints that will not have unnecessary data to the user domain from the system.

Functional dependency is a property of the meaning or semantics of the attributes in a relation. The semantics indicate how attributes relates to one another, and specify the functional dependencies between attributes. When a functional dependency is present, the dependency is specified as a constraint between the attributes.

Consider a relation with attributes A and B, where attribute B is functionally dependent on attribute A. It can be written as $A \rightarrow B$. If we know the value of A and we examine the relation that holds this dependency, we find only one value of B in all the tuples that have a given value of A, at any moment in time. Thus, when two tuples have the same value of A, they also have the same value of B. However, for a given value of B there may be several different values of A.

When a functional dependency exists, the attribute or group of attributes on the left hand side of the arrow is called the determinant and the right hand side is called dependent attribute.

II. The Three Level Ansi Sparc Architecture

An early proposal for a standard terminology and general architecture for database systems was produced in 1971 by the Data Base Task Group (DBTG) appointed by the Conference on Data Systems and Languages [6]. The DBTG recognized the need for a two-level approach with a system view called the schema and user views called subschemas. The American National Standards Institute (ANSI) Standards Planning and Requirements Committee (SPARC), ANSI/ X3/SPARC, produced a similar terminology and architecture in 1975, ANSI-SPARC recognized the need for a three-level approach with a system catalog. These proposals reflected those published by the IBM user organizations Guide and Share some years previously, and concentrated on the need for an implementation-independent layer to isolate programs from underlying representational issues (Guide/Share, 1970). Although the ANSI-SPARC model did not become a standard, it still provides a basis for understanding some of the functionality of a DBMS [4].

III. Relational Model: Definition

The relational model for databases assumes that the data are stored in tables, called relations. The columns of a table are labelled with names, called attributes. No two columns have the same name. Each attribute has an associated domain of values. The rows of a table usually referred to as records or tuples, can be viewed as mappings from the attributes to their domains. Let r be a tuple of a relation R and A be an attribute of R. Then r(A) is the A-component of r (i.e., the value of r for the attribute A). Some of the relational data structure terms are as follows:
Relation: A relation is a table with columns and rows.
Attribute: An attribute is a named column of a relation.
Domain: A domain is the set of allowable values for one or more attributes.
Tuple: A tuple is a row of a relation.
Degree: The degree of a relation is the number of attributes it contains.
Relational database: A collection of normalized relations with distinct relation names.

IV. Methodology

A functional dependency, denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they also have t1[Y] = T2[Y]. This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. We also say that there is a functional dependency from X to Y, or that Y is functionally dependent on X. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side. The Left-hand side is the determinant while the right-hand side is the dependent attribute. Thus, X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value.
The following are to be noted:
If a constraint on R states that there cannot be more than one tuple with a given X- value in any relation instance r(R)-that is, X is a candidate key of R-this implies that X → Y for any subset of attributes Y of R (because the key constraint implies that no two tuples in any legal state r(R) will have the same value of X).
If X → Y in R, this does not say whether or not Y → X in R.
Assume that a relational schema has attributes (A, B, C,..., Z) and that the database is described by a single universal relation called R = (A, B, C,..., Z). This assumption means that every attribute in the database has a unique name.
Functional dependency describes the relationship between attributes in a relation. For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A → B), if each value of A is associated with exactly one value of B. (A and B may each consist of one or more attributes.)
Consider a relation with attributes A and B, where attribute B is functionally dependent on attribute A. If the value of A is known and examining the relation that holds this dependency, we find only one value of B in all the tuples that have a given value of A, at any moment in time. Thus, when two tuples have the same value of A, they also have the same value of B. However, for a given value of B there may be several different values of A. The dependency between attributes A and B can be represented diagrammatically, as shown below;

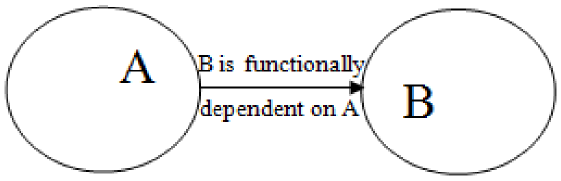

Figure 1: Functional Dependency

An alternative way to describe the relationship between attributes A and B is to say that "A functionally determines B"
A set of Inference Rules (IR), called Armstrong's axioms, specifies how new functional dependencies can be inferred from given ones [5]. For this paper, let X, Y, and Z be subsets of the attributes of the relation R. Armstrong's axioms are as follows:
The following six axioms IR1 through IR6 are well- known inference rules for functional dependencies:
1. IR1 (Reflexivity rule): If X ⊇ Y then X → Y.
The Reflexive rule can also be stated as X → Y; that is, any set of attributes functionally determines itself.
2. IR2 (Augmentation rule): {X → Y} then XZ → YZ.

The augmentation rule can also be stated as {X → Y} then XZ → Y; that is, augmenting the left-hand side attributes of an FD produces another valid FD.

3. IR3 (Transitivity rule): {X → Y, Y → Z} then X → Z.

Each of these three rules stated above can be directly proved from the definition of functional dependency. The rules are complete in that given a set X of functional dependencies, all functional dependencies implied by X can be derived from X using these rules. The rules are also sound in that no additional functional dependencies can be derived that are not implied by X. In other words, the rules can be used to derive the closure of X.

Several further rules can be derived from the three given above that simplify the practical task of computing X+. In the following rules, let D be another subset of the attributes of relation R, then:
4. IR4 (Decomposition or Projective rule): {X → YZ} then X → Y and X→Z.
5. IR5 (Union or Additive rule): {X → Y, X→ Z} then X → YZ.
6. IR6 (Pseudo-transitivity rule): {X → Y, WY→ Z} then WX → Z.

Proof of IR1 (Reflexivity rule)
Suppose that X ⊇ Y and that two tuples t1 and t2 exist in some relation instance r of R such that t1 [Xl = t2 [X]. Then t1 [Y] = t2 [Y] because X ⊇ Y; hence, X → Y must hold in r.

Proof of IR2 (Augmentation rule)
Assume that X → Y holds in a relation instance r of R but that XZ → YZ does not hold. Then there must exist two tuples t1 and t2 in r such that
(1) t1 [X] = T2 [X]
(2) t1 [Y] = t2 [Y]
(3) t1 [XZ] = t2 [XZ]
(4) t1 [YZ] ≠ t2 [YZ]
This is not possible because from (1) and (3) above we deduce
(5) t1 [Z] = t2 [Z]
And from (2) and (5) above we deduce
(6) t1 [YZ] = t2 [YZ], contradicting (4)

Proof of IR3 (Transitivity rule)
Assume that
(1) X → Y and
(2) Y→ Z both hold in a relation r.
Then for any two tuples t1 and t2 in r such that t1 [X] = t2 [X] we must have
(3) t1 [Y] = t2 [Y], from assumption (1); hence we must also have
(4) t1 [Zl = t2 [Z]
From (3) and assumption (2); hence X → Z must hold in r.

Proof of IR4 (Decomposition rule) Using IR1 through IR3
1. X → YZ (given).
2. YZ → Y (using IR1 and knowing that Y is a subset of YZ).
3. X → Y (using IR3 on 1 and 2).

Proof of IR5 (Union rule) using IR1 through IR3
1. X → Y (given).
2. X → Z (given).
3. X → XY (using IR2 on 1 by augmenting with X; notice that XX = X).
4. XY→ YZ (using IR2 on 2 by augmenting with Y).
5. X → YZ (using lR3 on 3 and 4).

Proof of IR6 (Pseudo-transitivity rule) using IR1 through IR3
1. X → Y (given).
2. WY → Z (given).
3. WX → WY (using IR2 on 1 by augmenting with W).
4. WX → Z (using IR3 on 3 and 2).
It has been shown by Armstrong that inference rules IR1 through IR3 are sound and complete. By sound, we mean that given a set of functional dependencies F specified on a relation schema R [5], any dependency that we can infer from F by using IRI through IR3 holds in every relation state r of R that satisfies the dependencies in

F. By complete, we mean that using IRI through IR3 repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F. In other words, the set of dependencies F+, which we called the closure of F, can be determined from F by using only inference rules IR1 through IR3. Inference rules IR1 through IR3 are known as Armstrong's inference rules. For each such set of attributes X, we determine the set X+ of attributes that are functionally determined by X based on F; X+ is called the closure of X under F.

Closure of Functional Dependency

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

Algorithm 1: Computing F+ (Closure of Functional Dependency)

To compute the closure of a set of functional dependencies F:

```
F+ = F
Repeat
For each functional dependency f in F+
Apply reflexivity and augmentation rules on f
Add the resulting functional dependencies to F+
For each pair of functional dependencies f1 and f2 in F
If f1 and f2 can be combined using transitivity then add the resulting functional dependency to F+
Until F+ does not change any further
```

Attribute Closure of Attribute set (X+)

Given a set of attributes α, define the closure of X under F (denoted by X+) as the set of attributes that are functionally determined by X under F

Algorithm 2: Computing the Closure of X under F (X+)

```
X+ := X;
Repeat
oldX+ := X+;
For each functional dependency Y → Z in F do
If X ⊇ Y then X+ := X+ U Z
Until (X+ = oldX+)
```

The Algorithm starts by setting X+ to all the attributes in X. By IR1, we know that all these attributes are functionally dependent on X. Using inference rules IR3 and IR4; we add attributes to X+, using each functional dependency in F. We keep going through all the dependencies in F (the repeat loop) until no more attributes are added to X+ during a complete cycle (of the "FOR" loop) through the dependencies in F.

Breakdown of Closure Algorithm

Given

A set of attributes {A1, A2,...,An} and

A set FDs S,

Compute X = {A1, A2, ..., An}+

Use the splitting rule so that each FD in S has one attribute on the right.

Set X ← {A1, A2, ..., An}+

Find an FD B1B2, ..., Bk → C in S such that {B1, B2, ..., Bk} ⊆ X but C ∉ X

Add C to X

Repeat the last two steps until you cannot find such an attribute C.

The final value of X is the desired closure.

Minimal sets of Functional Dependencies

A minimal cover of a set of functional dependencies G is a set of functional dependencies F that satisfies the property that every dependency in G is in the closure of F. In addition, this property is lost if any dependency from the set F is removed; F must have no redundancies in it, and the dependencies in G are in a standard form. To satisfy these properties, we can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions;

Every dependency in F has a single attribute for its right-hand side.

We cannot replace any dependency X → A in F with a dependency Y→ A, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to G

We cannot remove any dependency from F and still have a set of dependencies that is equivalent to G

Algorithm 3: Computing Minimal Cover G+ for a Set of Functional Dependencies F

Set G=F and make the Right hand side (RHS) atomic (by decomposition)

```
Replace each FD X → {A1, A2,...An} in G by n functional dependencies X → A1, X → A2, ..., X → An
Remove any redundant Left hand side (LHS).
For each FD X → A in G that has multiple attributes in X
For each attribute B ε X
{
Compute (X-B)+ with respect to (G – (X → A)) U ((X-B) → A)
If (X-B)+ contains B, then replace X → A in G with (X-B) → A
}
Remove any redundant FDs
For each FD X → A in G
{
Compute X+ with respect to the set of dependencies (G- (X → A))
If X+ contains A, then G = (G – (X → A))
}
```

Sets of functional dependencies may have redundant dependencies that can be inferred from the others
For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C\}$
Parts of a functional dependency may be redundant
E.g.: on RHS: $\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow CD\}$ can be simplified to
$\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\}$
E.g.: on LHS: $\{A \rightarrow B,\ B \rightarrow C,\ AC \rightarrow D\}$ can be simplified to
$\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\}$
A canonical cover of F is a "minimal" set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies.
Closure allows us to prove correctness of rules for manipulating FDs
If A1A2...An → B1B2...Bm and B1B2...Bm → C1C2...Cn
Then A1A2...An → C1C2...Cn
Closure allows us to procedurally define keys. A set of attributes X is a key for a relation R if and only if {X}+ is the set of all attributes of R and for no attribute A ε X is {X-{A}}+ the set of all attributes of R.

V. Conclusions

The database is now an underlying framework of information system and has fundamentally changed the way many organisations operate. The development of this technology over the years has produce systems that are more powerful and more intuitive to use. Hence, there is need to make such system consistent and void of redundancy so as to maintain its integrity.

The Armstrong's axioms are set of axioms used to infer all the functional dependencies on a relational database. The axioms are sound in that they generate only functional dependencies in the closure of a set of functional dependencies (denoted as F+) when applied to that set (denoted as F). They are also complete in that repeated application of these rules will generate all functional dependencies in the closure (F+). Attribute closure algorithm can be used to test for superkey.

Minimal cover helps to eliminate redundancy functional dependencies in the database. This will help to save storage. It also helps in optimization of database by eliminating unwanted rules and unneeded attributes.

References

[1]. Anupama A, Vijak K "Mining Functional Dependency in Relational Databases using FUN and Dep-Miner: A comparative study" in Proceedings of IJCA, September 2013.

[2]. Jalal A, Dojanah B, Arafat A "Mining Functional Dependency from Relational Databases Using Equivalent Classes and Minimal Cover" in Proceedings of Journal of Computer Science, 2008.

[3]. Maier D. (1983). "The Theory of Relational Databases". New York, NY: Computer Science Press Management Systems. Interim Report, FDT. ACM SIGMOD Bulletin, 7(2)

[4]. American National Standards Institute (1975). "ANSI/X3/SPARC Study Group on Database"

[5]. Armstrong, W. W. (1974). "Dependence Structures of Database Relationships". Pages 580-583 of: IFIP World Congress 1974. North-Holland.

[6]. CODASYL Database Task Group Report (1971). ACM, New York, April 1971

A.B Amure. "Functional Dependency: Design and Implementation of a Minimal Cover Algorithm." IOSR Journal of Computer Engineering (IOSR-JCE) , vol. 19, no. 5, 2017, pp. 77–81.