AUTOMATIC DISCOVERY OF CANDIDATE IN THE RELATIONAL DATABASES KEYS BY USING ATTRIBUTES SETS CLOSURE

Yasmeen F. Al-ward
Department of Computer Science, College of Science, Al-Nahrain University.

Abstract

The automation of database design, design weak points' detection, re-engineering and schema modification, and normalization of the database systems became the crucial aspects in databases studies. In real applications databases the discovery of candidate keys is regarded as a challenge for the designers. This research proposes an algorithm to automate the discovery of the candidate keys in the databases depending on the attributes set closure and functional dependency, FD, rules. In this research, the functional dependency rules are regarded as production rules to product binary strings which represent the sets of attributes and the candidate keys. Representing the rules and the sets of attributes as binary string allows proposing novel string matching and ordered strings merging which linearly speed up the execution time according to the number of attributes and the number of functional dependency rules.

The proposed system was tested successfully by using many actual and synthesis schemas and dependencies some of these schemas and dependencies include one hundred attributes and one hundred FD rules respectively.

Keywords: Data Redundancy, Canonical synthesis, Candidate keys, Normalization, Functional Dependency, Attributes set closure, Production Rule System.

1. Introduction

The designer of relational database system should address four measures of quality for relation schema design [1]: Semantic of the attributes described by *functional dependency* (FD), Reducing the Redundant information in tuples, Reducing the null values, and Disallowing the possibility of generating spurious tuples. FD is used to Formalize some knowledge about the domain through some special integrity constraints that is *Functional dependencies, **FD.***

In this research, a concentration will be given for FD because it will be used in the proposed algorithm.

2. Functional Dependency

This section includes some important definitions as preliminaries to elucidate the proposed algorithm and its implementation. *The definitions are adopted from* [1, 2].

Definition (1):

Let R(A1,A2,....An) be a relational schema, and let X and Y be subsets of {A1,A2,....An}; we say that X *functionally determines* Y, denoted by X $\rightarrow$ Y, if for each state *r* (or extension) of R, it is not possible that two tuples exist in *r* having equal values for all attributes in X, and different values for the attributes in Y.

Definition (2):

Let *F* be a set of functional dependencies defined over a relational schema R, and let X à Y be a functional dependency, we say that F *logically implies* X à Y, denoted as F |= X à Y, if each relation *r* of schema R that verifies the functional dependencies in *F*, also verifies X à Y. As a example: {A à B, B à C} |= A à C. Such set of rules is known as *Armstrong's axioms* [3, 4].

Definition (3):

The notion of key will be re-formalized by using the functional dependencies as follows:

- Let R be a relational schema with attributes A1 A2.....A_n, and *F* be a set of functional dependencies. Let X be a subset of R. X is a *key* of R if:

1.X $\rightarrow$ A1 A2.....An is in F+

2. There is no Y, proper subset of X, such that *Y $\rightarrow$ A1 A2.....An is in* F^+ (minimality condition).

- We use the term *superkey* to denote a superset of the key.

Note that according to this definition, there are many X's which are subsets of R, i.e., there are many keys for a given table. These keys are called *candidate keys.*

Definition (4):

- Let F be a set of functional dependencies defined on a set D of attributes; let X be a subset of D. The ***closure of X*** with respect to F is denoted by X^+ and is defined as the set of attributes{A | F |= X à A} = {A | X à A follows from F by applying the Armstrong axioms}
- X à Y$\in$ F+ if and only if Y$\in$ X^+ with respect to F.

This research depends on the definitions above especially (4, and 5) to produce an algorithm to automatically generate all the candidate keys in a relational database.

3. The Importance of The Candidate Keys Discovery

The discovery of candidate keys is time consuming process because it is massive computational process [2]. Note that if the relational table includes N attributes then the designer should check (2^N-1) sets of attributes to determine the set of candidate keys. The discovery of candidate keys in relational database is very important in many cases such as:

1) The conceptual phase of database design usually depends on entity-relation diagram (ERD) or Unified Modeling Language (UML). It is not uncommon that the designer performs many mistakes in this phase. The immorality of this phase will be transmitted to the mapping sub phase of logical phase which lead to imprecise table(s). The imprecise schemas frequently cause data redundancy. One solution to this problem is the normalization which depends mainly on the candidate keys such as 3NF, 4NF, and 5NF.
2) Database system re-engineering and database modification. Many enterprises use bad-designed databases which require re-engineering or special-care modification by adding or removing some attributes, data type conversion, table splitting, etc. All of these processes require re-determining of the candidate keys.
3) Candidate keys discovery easies the study of the relationships between the attributes of the tables of the database and then helps in the determining the foreign keys.
4) The candidate keys enable the best selection of the primary key. It is well-known that a table may contain many keys, candidate keys, and it may be complex process to compute all these keys and select a suitable one as a primary key.
5) All of the mentioned above, but the normalization, will enhance the querying of the database.

4. The Proposed Algorithm Of Candidate Keys Discovery

From the definitions presented in section two, it is obvious that the closure of set of attributes represents a key in the table if it contains all the attributes of the table because it represents direct or indirect dependency. Fig. (1) depicts the algorithm of attributes set closure, X^+. The input of this algorithm is the rules of functional dependency, the set of attributes of the table D, and a set of attributes, X, to computes its closure. The output of this algorithm is X^+. X is subset of D. This algorithm will be invoked frequently to compute the closures of all the possible set of attributes of the table's attributes, see Fig. (2). A sequence of sets will be generated until no more new set generated. Initially X(0)=X, then X(1)=X(0) + A if there is a dependency Y à Z is in F such as A is in Z and Y $\subseteq$ X(i). Then x(1) will be computed and so forth until X(i)=X(i+1).

```
Closure of a set of attributes algorithm
 ▎ Input: A finite set D of attributes, a set F of functional dependencies, and a set X ⊆
   D
 ▎ Output: X^+, closure of X with respect to F
 ▎ Approach: A sequence of attribute sets X(0), X(1),...,X(n) is computed:
   {
      X(0) is X; i=0;
     While(true)  {
       •   X(i+1) = X(i) ∪ A , where A is a set of attributes such that:
                  ▎  A dependency Y à Z  is in F
                  ▎  A is in Z
                  ▎  Y ⊆ X(i)
       •  If x(i)=x(i+1) return x(i+1);
       •  else i++; }
 }
```

Fig.(1) : Closure of a set of attributes algorithm.

```
Discovery of Candidate keys Algorithm
 ▎ Input: A finite set D of attributes, a set F of FDs, and a set X ⊆ D
 ▎ Output: X^+, closure of X with respect to F, for all X ⊆ D.
{
  1. N=length(D)
  2. for(i=1; i<=2^N-1; i++)
  4. {
  3.    Generate a set of attributes X;
  4.     Call Closure of a set of attributes algorithm (D,F, X); //see figure(1)
         X^+= Closure of a set of attributes algorithm (D,F, X);
   5.    If X^+ =D then add X to candidate keys table;
   6. }
}
```

Fig. (2) : Candidate keys Discovery Algorithm.

The higher order algorithm to compute the closures of the set is shown in Fig.(2). According to this algorithm the sets of length 1 is generated and sent to compute their closures, then the set of length 2, and so on. Actually the algorithm contains many general steps and many hidden steps which will be explained gradually to ease the understanding of the research.

5. The Proposed System Architecture

Fig.(3) shows the architecture of the proposed system. Some of the aspects of designing production rules systems, [3, 4] are adopted to design the proposed system. We regard the functional dependency rules, F, as production rules which produce the closures of attributes. The input to the system is F and the set of the attributes of a table. Indeed, the attributes of a given table is fetched from the data dictionary in a table of N rows, each row contains the *attribute name* and its *code*. Each attribute is assigned a code to avoid the complexity if the complete attribute name is used in the process. F is represented as a table of rules. Each row contains rule number, right hand side of the rule, and left hand side.

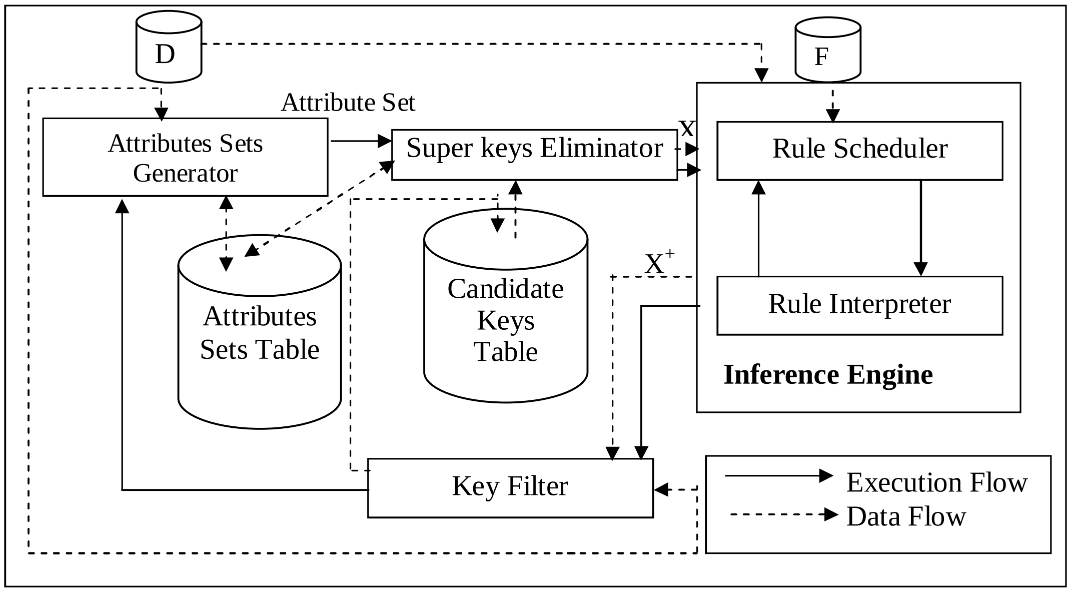

Fig.(3) : The Proposed System Architecture.

The rules representation depends on the code stored in D for each attribute name. The structure of F will be elucidated in section (5.3.1).

5.1. Attributes Sets Generator

The attributes sets D, the LHS and RHS of F, and the generated sets of attributes are stored and manipulated alphabetically, this process easies the generation process of the sets of attributes which we need to compute their closures. This module is designed according to the algorithm presented in Fig. (4). For more explanation Consider example (1):

<u>Example (1):</u>

Let R=(C,G,H, R, S,T) is the attributes set of a table where C, G,H, R, S, AND T are the code of the following attributes: C=**C**ourse, T=**T**eacher, H=**H**our, S=**S**tudent, G=**G**rade, and R=**R**oom. F consists of the following dependencies: C **à** T; *each course has only one teacher,* HR **à** C; *only one course can be given in a given classroom and hour,* HT **à** R; *a teacher cannot be in two classrooms at the same hour*, CS **à** G; *each student receives a single grade for each exam*, HS **à** R; *a student cannot be in two different classrooms at the same time.*

```
1.1st _level=D;
2. Insert 1st _level in Attributes_Sets Table
3. 2nd _level={All alphabetically ordered sets of length 2 generated from D}
4. Insert 2nd _level in Attributes_Sets Table
5. for (i=3; i<= N; i++){
7.   for all s1 belongs to ith_level-1 and s2 belongs to ith level-1 do the following
8.      if s1[1]=s2[1] && s1[2]=s2[2]&&.....s1[i-2]=s2[i-2]&& s1[i-1]<s2[i-1]
9.      {  s=concat(s1,s2[i-1]);  ith_level += s; }
10.  insert ith_level in Attributes_Sets Table;
11 }
```

Fig. (4) : Attributes Sets Generator Algorithm.

Now according to step (1), 1st_leve1={C,G,H,R, S,T}. The second step will generate 2nd _level={CG, CH, CR, CS, CT, GH, GR, GS, GT, HR,HS, HT, RS, RT,ST}. The iterative steps will generate the following levels:

3rd_level={CGH,CGR, CGS, CGT, CHR, CHS, CHT, CRS, CRT, CST, GHR, GHS, GHT, GRS, GRT, GST, HRS, HRT, HST, RST}
4th_level={CGHR,CGHS, CGHT, CGRS, CGRT, CGHT, CHRS, GHRT, CHST, CRST, GHRS, GHRT,GHST, GRST, HRST}
5th_level={CGHRS, CGHRT, CGHST, CGRST, CHRST,...}
6th_level={CGHRST}.

The generated sets of attributes above are inserted in the *attributes sets table*. Each row in this table consists of the set and its length. Then Superkeys Eliminator will fetch one set by one set from the table according to its length from the first level to nth level. The design and the important duty of this module are explained in the next section.

5.2.Superkeys Eliminator

This module depends on smart fact derived from definition (4) that is "*All the super sets, (superkeys) of a key are not keys*". The key should satisfy the minimality property. Therefore, this module fetches the sets of length one, and then of length two, and so on from *attributes set generator*. When this module fetches a set K of length L, it degenerates K to its subsets of length L-1. If anyone of these subsets is stored in ***Candidate Keys Table***, then K will be removed because it is not a key but it is a superkey of K. For instance, suppose that HS is a key in the schema presented in example (1), hence this module will ignore {CHS, GHS, HRS, HST, CGHS, CHRS, CHST, GHRS, GHST, HRST, CGHRS, CGHST, CHRST, CGHRST} because these sets of attributes are super keys of HS. The eliminator is accomplished according to the algorithm presented in Fig.(5). This module takes its inputs from *Attributes Sets Table* and *Candidates Keys Table* as shown in Fig. (3). It retrieves a set S with its length L from the sets of attributes table in step (4) and degenerates S to its subsets of length L-1 by using a function called subset. This function takes S, its length L as input and produce all its subsets of length L-1, these operations are done in step (5).

```
Input: Attributes Sets Table and Candidates Keys Table
Output: S a set of attributes
1. SuitableS=false;
2. while(!SuitableS || end_of_table(Attributes Sets Table) {
4.     Fetch a set S and its length L from Attributes Sets Table.
5.     Degenerate S to its set subsets of length L-1;
           generate_subset(S,L,SUB);
6.     For all sl belongs to SUB do {
8.         if sl is stored in Candidates Keys Table do not send S to IE Module;
9.         exit for; }
11.    if (!SuitableS) {
13.        Return  S to Inference Engine Module;
14.        SuitableS=!Suitable; }
15. }
```

Fig.(5) : Super Keys Eliminator Algorithm.

Step (6) keeps a loop according to the cardinality of SUB to check if one of the elements of SUB is a key and stored in *candidate keys Table*. If so, S will be ignored and the outer loop will be continued to fetch new S from *Sets of Attributes Table*.

5.3. Inference Engine

Inference Engine, IE is responsible for producing X^+, i.e., attributes set closure one closure at a time and sends the closure to *Key filter module* to check the validity of the closure as a key.

IE consists of two modules the *rules scheduler* and *interpreter*. The skeleton of IE is shown in Fig. (6). The outer loop is to manipulate all the sets of attributes after generating these sets by the *attributes sets generator*.

5.3.1 Rules scheduler

This module is a part of the *inference engine*, *IE*. To explain the design of this module, we'll explain the structure of F table. The structure of this table is represented as binary table which consists of three fields rule number, left hand side LHS, and right hand side, RHS. Indeed, LHS and RHS are compound fields each of one consists of many fields. There exists one field for each attributes of a table under processing.

```
Inference Engine Algorithm
l  Input: indices of finite set D of attributes,
      A set F of functional dependencies, each functional dependency represented
      as (binary string)→(binary string), and the Attributes Sets Table. The table
      consists of Xs; X ⊆ D; X is binary string.
●     Output a sequence of X+'s.
   {
      Make fresh copy of F; F_Fresh_Copy;
      Order F_Fresh_Copy  from longest to shortest RHS length.
       while (Attributes Sets Table is not empty) do{
           fetch a new set of attributes X;
          X+=interpreter(D, F, X);
       //Note that the interpreter will call the scheduler, see figure(8).
          Key_filter(X+);}
   }
```

Fig.(6) : The Skeleton of IE.

The values of the sub-fields are 0 or 1. For instance, the rules of example (1) are presented in Table (1). The first raw represents the first rule C à T, the second rule is presented in the second raw HR à C, and so on. This representation is sparse-matrix like which wastes some extra storage but there are many benefits which are gained in string matching and substitution as will be explained.

Table (1)
Functional Dependency Rules Representation.

Rule #	**LHS**						**RHS**					
	C	**G**	**H**	**R**	**S**	**T**	**C**	**G**	**H**	**R**	**S**	**T**
1	**1**	0	0	0	0	0	0	0	0	0	0	**1**
2	0	0	**1**	**1**	0	0	**1**	0	0	0	0	0
3	0	0	**1**	0	0	**1**	0	0	0	**1**	0	0
4	**1**	0	0	0	**1**	0	0	**1**	0	0	0	0
5	0	0	**1**	0	**1**	0	0	0	0	**1**	0	0

This representation is used for the generated sets of attributes which are produced by *Attributes Sets Generators*. For example, the set of attributes {G, H, R}, i.e., GHR is represented as "011100". 1's represent the presence of G, H, and R attributes, while 0's represent the absence of C, S, and T attributes. Recall that the rule scheduler presents a rule for firing by the *interpreter*. The scheduler determines the required rule by checking the condition: LHS(R) ⊆ X(*i*). This checking requires implementation of one of string matching algorithms [5,6]. To avoid this execution-time-consumption implementation

we convert it to ANDing operation only. For example, suppose $(HS)^+$ is required to be computed, and the candidate rule is HS à R. The scheduler will (and) HS with $(HS)^+$, i.e., 001010 & 001010= 001010. The result means that
LHS(R) $\subseteq$ X(*i*) and the rule is suitable for firing. The firing is done by the interpreter module. The scheduler algorithm is elucidated in Fig.(7). The scheduler uses the fresh copy of the dependencies which are ordered from the RHS-highest length to lowest one. The scheduler selects the top list rule. This heuristic rapidly constructs X^+ by minimum number of steps because maximum number of attributes will be added to X(i) in each iteration. The LHS of the selected rule will be added to the closure if LHS is included in the X(i).

5.3.2 Rule Interpreter

The interpreter receives a rule number of the rule to be fired. The firing process is generating X(*i+1*) by joining two ordered sets of attributes. To accomplish the joining process, the algorithm of merging two ordered lists [5, 6] is required to be implemented, which is a part of merge sort. This algorithm is time consumption process. Therefore, a novel operation is suggested, to avoid merging operation, that is ORing the X(i) and RHS of a rule. For example, HS(0)=HS and the candidate rule is HS à R, HS(1)=HRS which obtained by (001010 | 000100)=001110= HRS. Accordingly, the algorithm of computing attributes set closure, X^+, becomes as presented in Fig.(8).

Rule Scheduler Algorithm

```
▌ Input: indices of finite set D of attributes,
    A set F of functional dependencies, each FD represented as (binary
    string)→(binary string), and a set X ⊆ D; X is binary string.
▌ Output: A rule of the form (binary string)→(binary string).
▌ Approach: A sequence of attribute sets X(0), X(1),..., X(n) is computed:
  {
    no_suitable_rule=false;
    While(!no_suitable_rule)  {
          • Fetch a highest order rule R from F_Fresh_Copy;
          • if (X & lhs(R)== X)  // x anding lhs(R)
          • {  store rule number in the explanation_list;
          •    return(R); //return R and exit}
          • If (!no_suitable_rule) return (null);}
  }
```

Fig.(7) : The Rules scheduler Algorithm.

```
Closure of a set of attributes algorithm
▎ Input: indices of finite set D of attributes,
    A set F of functional dependencies, each FD represented as binary
    string→binary string,& a set X ⊆ D; X is binary string.
▎ Output: X+, closure of X with respect to F represented as binary string.
▎ Approach: A sequence of attribute sets X(0), X(1),...,X(n) is computed :
  {
     X(0) is X;  i=0;
    While (true)  {
       //invoke the scheduler to fetch asuitable rule where lhs(R) is subset of X(i).
          • If Rule_scheduler(R) {
          •     X(i+1) = X(i) | RHS(R);   //ORING
          •     If x(i)=x(i+1) return x(i+1);}
          • else return X(i);  //null rule
          • i++;}
 }
```

Fig.(8) : Closure of a set of attributes algorithm.

5.4 Key Filter Module

The input of the *Key Filter module, KF*, is the set of the attribute of the table under processing and a closure of set of attributes, X^+, generated by the IE. KF is frequently invoked by IE to check the elements of X^+, it should be includes all the members of D. To avoid the iterative sequential search, this module depends on the binary ORing operation to check the validity of X^+ as a candidate key. KF ors X^+ and D, if the result consists of stream of 1's with length equals to the |D|, then X^+ is a candidate key. KF writes the candidate keys to the candidate keys table.

Discussion, Conclusions, and Future Works

Many actual and synthetic schemas had been used to test the candidate keys discovery system, some of which contain up to one hundred rules and one hundred attributes. The considerable amount of execution time is consumed in the generating of the sets of attributes which is accomplished by the *attributes sets generator* module. Another time consumer module is the *superkeys eliminator* module, which trims the super keys of a valid candidate key because they are not keys. This implementation was developed depending on the binary representation of the functional dependency table F and the ANDing and ORing operators. This implementation reduced the execution time of "one hundred dependencies and one hundred attributes" test to % 65.33 of the execution time of the first implementation which depends merging two sorted lists. Also, the tests show that the execution time is increased according to the number of dependencies and the number of attributes.

This research emits many future works and researches such as:

1) Add a complementary module to automatically determine the functional dependencies depending on Armstrong's axiom. This module will be responsible for building F table.
2) This research can be used for automatic normalization, especially 2NF, 3NF, BCNF, and 4NF.
3) The research can be used for automatic analyzing the schemas of the tables of a database to determine the foreign keys.
4) One can define the criteria to determine the primary key of the table under processing, and then these criteria can be applied on the candidate keys to select a suitable one as a primary key.

References

[1] Molina, Jeffrey D., Ullman, Jennifer Widom, "*Database systems: The complete Book*", Prentice Hall, 2008.

[2] Joseph M. Hellerstein, Michael Stonebraker, "*Readings In Database Systems*", MIT Press, 2005.

[3] Joseph C. Giarratano, "*Expert Systems: Principle and Programming*", 4ed, Course Technology, 2004.
[4] Jones, G., "*Production Systems and Rule-based Inference*", Encyclopedia of Cognitive Science, Vol.3, 2003.
[5] ROBERT SEDGEWICK, "*ALGORITHMS*", Addison-Wesley, 2003.
[6] Thomas H. Cormen, Ronald L. Revest, Charles E. Leierson, "*Introduction to Algorithms*", MIT Press, 2001.

الخلاصة

تأليلُ تصميمِ قواعدِ البيانات، كشفِ نقاطِ الضعفِ في مراحل التصميم، إعادةِ هندسة وتعديلِ وصوفاتها، واجراء عملية التطبيع عليها اصبحت السماتَ الحاسمةَ في مجال الدراسات الحديثة في حقل قواعد البيانات. في قواعدِ بيانات التطبيقاتِ الحقيقيةِ، إكتشاف المفاتيحِ المرشّحِة يُعتَبرُ تحدّ لمصممي قواعد البيانات. هذا البحثُ يَقترحُ ويُنجّز خوارزميةِ لإكتِشاف المفاتيح المرشّحِة في قواعدِ البيانات آلياً بالاعتماد على إنغلاق مجموعاتِ الخواصَ و التبعيةِ الوظيفيةِ. في هذا البحث، قواعد التبعيةِ الوظيفيةِ اعتبرت على انها قواعدُ إنتاجِ لسلاسل نصية ثنائيةِ والتي تُمثّلُ المفاتيحَ المرشّحَة المُولدة. تَمْثيل القواعدِ ومجموعاتِ الخواصِ بصيغة ثنائيات نصية َسمحت بإقتِراح طريقة مبتكرة لمقارنة النصوص الثنائية ودمجها لتوليد مجاميع الخواص مما جعل زمن التنفيذ يتغير بشكل خطي مع عدد الخواص في الجدول وعدد قواعد التبعية الوظيفية. لقد أُختبرت الخوارزمية والنظام المُصَمم باستخدام العديد من قواعد البيانات التي وصل عدد خواص البعض منها الى مائة خاصية وعدد قواعد التبعية الوظيفية فيها الى مائة قاعدة وقد اكتشفت مفاتيحها المرشحة بنجاح.