Automatic Determining Of Candidate Keys Sets Depending On Functional Dependency

Nada Adnan

Al-Rafidein University College

Abstract

This paper presents a step in automatic database design, re-engineering and schema modification, design weak points detection, and normalization. This step represents determining the candidate keys of the tables automatically via proposing and implementing an algorithm which depends on the functional dependency and attribute set closure. Also, some techniques of *Apriori* algorithm of association rule mining are used to generate keys and minimize the keys space. The implementation is done by using Oracle PL/SQL to utilize the facilities provided by ORACLE DBMS and to give the features of embedded system to the proposed one.

التحديد الآلي لمجموعة المفاتيح المرشحة في قواعد البيانات باستخدام الاعتماديةالدالية

ندى عدنان

كلية الرافدين الجامعة

الخلاصة

هذا البحث يمثل خطوة في التصميم الآلي لقواعد البيانات واعادة هندسة وتعديل اطر الجداول العلائقية، والكشف عن نقاط ضعف التصميم، وكذلك عملية التطبيع. هذه الخطوة تتمثل في تحديد المفاتيح المرشحة للجداول بشكل آلي بواسطة تقديم وتطبيق خوارزمية تعتمد على الاعتمادية الدالية وخاصية انغلاق المجاميع، وخوارزمية تعدين قواعد الارتباط. خوارزمية التعدين استخدمت لتوليد مجموعة المفاتيح وترشيق فضاءها. البحث انجز باستخدام (لغة برمجة/لغة الاستفسار المهيكلة في أوراكل) للاستفادة من الخواص والتسهيلات المتوفرة في انظمة ادارة قواعد البيانات وكذلك لاعطاء ميزات الانظمة المنضوية للنظام المقترح.

1. Introduction

1.1 **Redundancy**

In non database systems each application has its own private files that fact can often lead to considerable redundancy in stored data, with resultant waste in storage space. For example, a personnel application and an education-records application may each own a file that includes department information for employees, those two files can be integrated, and the redundancy eliminated, if the DBA is aware of the data requirements for both applications, i.e., if the DBA has the necessary overall control.

Incidentally, we do not mean to suggest that all redundancy should necessarily be eliminated. Sometimes there are sound businesses or technical reasons for maintaining multiple copies of the same stored data. However, we do mean to suggest that any such redundancy should be carefully controlled. That is, the DBMS should be aware of it, if it exist, and should assume responsibility for "propagating dates".

1.2 **Definition: Functional Dependency**

We can define what we called Functional Dependencies by the following:

Given a relation R, attribute Y, of R is functionally dependent on attribute X of R- in symbols: R.X ◊ R.Y (read "R.X functionally determines R.Y"). If and only if each X-value in R has associated with it precisely one Y-value in R (at any one time). Attributes X and Y may be composite.

ﻧ Note that if attribute X is a candidate key of relation R in particular, if it is the primary key- then all attributes Y of relation R must necessarily be functionally dependent on X.

- Also note, however, that there is no requirement in the definition of functional dependence (hence forth abbreviated FD) that X in fact be candidate key of R; in another words, there is no requirement that a given X-value appear in only one tupelo of R .We give an alternative definition of FD that makes this point more explicit:
- Given a relation R, attribute Y of R is functionally dependent on attribute X of R if and only if, whenever two tupelos of R agree on their X-value, they must necessarily agree on their Y-value.

We also define the concept of (full) functional dependence.

Attribute Y of relation R is fully functionally dependent on attribute X of relation R if it is functionally dependent on X and not functionally dependent on any proper subset of X (that is, there does not exist any proper subset Z of the attributes constituting X such that Y is functionally dependent on Z).

1.3 **Logical Implications of dependencies:**

Suppose R is a relation scheme and A, B and C are some of its attributes. Suppose also that FD A◊ and B◊ C are known to hold in R. In proof, suppose r is a relation that satisfies A◊B and B◊ C, but there are two tupelos t and u in r such that t and u agree in the component for A but disagree in C, then we must ask whether t and u agree on attribute B, if not, then r would violate A◊B, if they do agree on B then since they disagree on C, r would violate B◊C hence r must satisfy A◊C.

In general, let F be a set of FD's for relation scheme R, and let X◊Y be a FD. We say F logically implies X◊Y, written F |= X◊Y, if every relation r for R that satisfies the dependencies in F also satisfies X◊Y. We say above that if F contains A◊B and B◊C, then A◊C is logically implied by F. That is {A◊B, B◊C}|= A◊C.

Definition 2:

Let F+, the closure of F, be the set of FD's that are logically implies by F, F+={X◊Y| F|=X◊Y}.

Example 1:

Let R=ABC and F= {A◊B, B◊C}. Then F+ consists of all those dependencies X◊Y such that either:

1. X contains A, e.g., ABC◊AB, AB◊BC, or A◊C,
2. X contains B but not A, and Y does not contain A, e.g. , BC◊B, B◊C, or B◊0 and
3. X◊Y is one of the two dependencies C◊C or C◊0.

2. **Key Finding**

When we talking about entity sets, we assumed that there was a key, set of attributes that uniquely determined any entity. There is an analogous concept for relations with FD's.

Definition 3

If R is a relation scheme with attributes A1A2……An and FD's F, and X is a subset of A1A2…..An, we say X is a key of R if:

1. X◊A1A2….An is in F+, that is the depending of all attributes on the set of attributes X is given or follows logically from what is given, and

2. For no proper subset Y is X is Y◊A1A2…An in F+.

We should be a ware that the term "key" does imply minimalism. Thus, the given key for an entity set will only be a key for the relation representing that entity set if the given key was minimal. Otherwise, one or more subsets of the key for the entity set will serve as a key for the relation.

As there may be more than one key for a relation, we sometimes designate one as the "Primary Key". The primary key might serve as the

file key when the relation is implemented. For example, however, any key could be the primary key if we desired.

The term "**Candidate Key**" is sometimes used in the literature to denote any minimal set of attributes that functionally determine all attributes,

We also use the term super key for any super set of a key.

Example2:

Let F be the following set and suppose we would like to compute the (BD)+

AB◊C C◊A BC◊D ACD◊B

D◊EG BE◊C CG◊BD CE◊AG

. X (0) =BD

. X (1): the dependency D◊EG is used

X (1) =BDEG

. X (2): the dependency BE◊C is used

X (2) =BCDEG

. X (3): we consider C◊A BC◊D CG◊BD CE◊AG

X (3) =ABCDEG

. (BD)+=X (3)

3.The Aim of the Research

In this research, we propose and implement an algorithm to automatic determining of the set of candidate keys in a relational database. This algorithm depends on the functional dependency rules and (set of attributes)$^{+}$, set of attributes closure, i.e., F^{+}. This algorithm and its implementation system; Automatic candidate keys determiner, ACKD, are very useful in many cases such as:

1) Database system design and database system re-engineering: The designers of actual applications database systems frequently

perpetrate many mistakes during the design phases. The lack of design may occur in the conceptual phase which leads to bad Entity-relationship or UML diagrams. Bad designed ERD or UML means an imprecise mapping in the logical phase and hence imprecise schema. Also, the inaccurate logical phase design leads to the imprecise schema. The weakness of the design in these phases causes the data redundancy which makes many problems in the case of updating, deleting, and inserting the data in the database.

2) Database system modification and re-engineering: Many tables and attributes may be added to the database; therefore the dependency rules will be changed and hence the keys of the tables.

3) ACKD represents a step in the automatic database normalization. The 3NF, 4NF, and 5NF depend on determining the set of candidate keys and the dependencies. Therefore, ACKD produces the keys on the golden trail to the manual or automatic normalization system.
4) Selection of an alternative primary key due to a particular reason.

5) Analyzing the relationships between the keys to determine the foreign keys of the tables of the database to achieve best form operators between the tables.

4. ACKD Architecture

Figure (1) depicts the architecture of ACKD. It consists of many modules each of which accomplishes specified duty. This architecture are built according to the process view and structural view. The dashed arrows represent data flow while continued arrows represent execution flow. These modules are Database's Tables Names Retriever, Table's Schema Extractor, X^i Generator; K-attributes set Generator, Key Checker. The dependency rules are stored as a table and will be read as input to be used to compute the closure. One or set of databases can be manipulated by at a time. The output of ACKD is a table of candidate

keys. The design of the modules will be explained in next sections in detail. Figure (2) depicts the general algorithm of ACKD.

The name of the database is fetched from the database catalog in step#1. Then, step#2 fetches the names of the tables in the tables_name cursor. The Oracle PL/SQL cursor provides high flexibility in the implementation of ACKD.

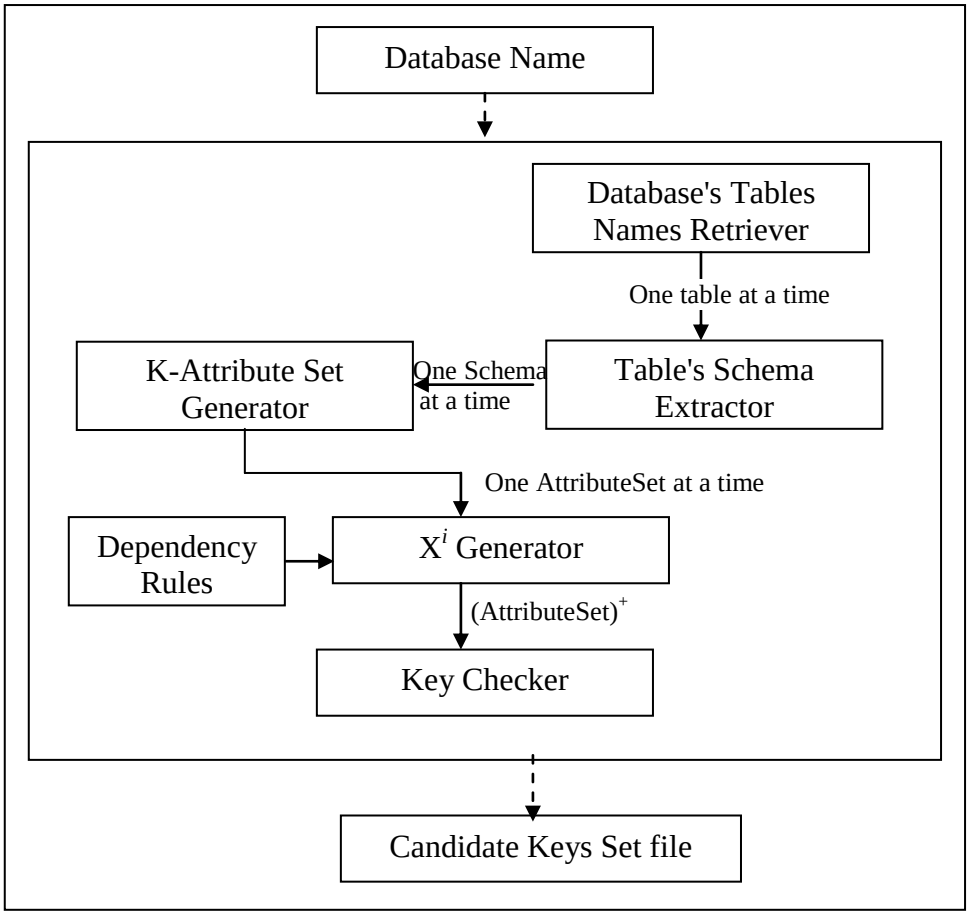

Figure (1) the Architecture of ACKD

The next steps extract the structure of each table to specify its attributes and all subsets of these attributes set will be generated and checked according to the dependency rules table. The dependency rules are stored as a table of two fields; RHS and LHS. RHS is varchar2 (30).

LHS is defined as large data type to hold the set of attributes which are supported by the attribute of the RHS and separated by ',' comma to

LHS	RHS
A	B, C

easies the checking operation. For example the rule A◊BC is stored as follows:

The candidate key table is one attribute table it holds all the candidate keys. Also the attributes of the key are separated by comma.

1. Fetch the name of the database; db.
2. Store the name of the tables of db in a cursor;TABLES_NAMES CURSOR
3. While TABLES_ NAMES is not empty
4. Begin
5. Store the attributes of a table t in *t_attributes array*
6. Call k-attribute-generator (*t-attribute, k_attributes_set*)
7. i=0;
8. While (*k_attributes_set* is not empty)
9. Begin
10. Call X^i_generator (*k_attributes_set*[*i*],X^+);
11. If (*key_checker*(X^+))
Write *k_attributes_set* in the candidate keys table
as a candidate key.
12 End
13. End

Figure (2) The general algorithm of ACKD.

4.1 K-attributes Generator

This module with Xi generator represents the core of the system. It designed according to the algorithm presented in Figure(3) and Figure(4).

K-AttributeSet Generator()
1) L_1= {Table Attributes};
2) For (k=2; L_{k-1} =ϕ; k++) do
3) L_k =K_Attribute Set_gen (L_{k-1})// see next figure
4) Return =$\cup_k L_k$;

Figure (3) the general Algorithm of K-attribute Generator

Step#1 stores the attributes set of a table in L1 list, then frequently calls an algorithm named K-Attribute_Set_Gen, see Figure (4). This algorithm generates from the elements of L1 all the sets of attributes of length 2 and

stores these sets in L2, and so on. The finding of next length attributes sets depends on join operation which join the sets which have only one different attribute of length k to generate a set of length k+1.

K-AttributeSet_Gen (L_{k-1});
3.1) $C_k = \phi$
3.2) for all attribute set $l_1 \in L_{k-1}$ and $L_2 \in L_{k-1}$ do
3.3) if ($L_1[1] = L_2[1] \wedge ... \wedge (L_1[k-2] = L_2[k-2]) \wedge (L_1[k-1] < L_2[k-1])$ then
3.4) begin
3.5) $C = L_1 \infty L_2$; // join step
3.6) add C to C_k
3.7) end
3.8) Return C_k

Figure (4) K-AttributeSet_Gen Algorithm

Table (1) the generated sets of attribute by K-attributes set generator

L1	L2	L3	L4	L5	L6
A	AB	ABC	ABCD	ABCDE	ABCDEG
B	AC	ABD	ABCE	ABCDG	
C	AD	ABE	ABCG	ABCEG	
D	AE	ABG	ABDE	ABDEG	
E	AG	ACD	ABDG	ACDEG	
G	BC	ACE	ABEG	BCDEG	
	BD	ACG	ACDE		
	BE	ADE	ACDG		
	BG	ADG	ACEG		
	CD	AEG	ADEG		
	CE	BCD	BCDE		
	CG	BCE	BCDG		
	DE	BCG	BCEG		
	DG	BDE	BDEG		
	EG	BDG	CDEG		
		BEG			
		CDE			
		CDG			
		CEG			

		DEG			

For example it joins {A, B, C} and {A, B, D} which are element in L3 to generate {A, B, C, D}. The algorithm ignores the joining of for example {A,C,D} and {A,D,G} because there are two different attribute at the end of each set, see step 3.1 and 3.2 of figure (4).

According to example (2), the K-attributes set generator will generate L1, L2…, L6. These sets are shown in table (1), these sets compose a lattice see figure (5). The complexity of this lattice elucidates the complexity of generating the sets of attributes. Easily, a reader can consider the hugeness of the number of the generated keys. Therefore, we adopt an efficient heuristic to avoid this weak point. This heuristic is a priori fact [3], that is "*the sub set of large item set are large*". We confirm this fact to be suitable for the research problem, according to definition #3, this fact becomes "*the super sets of any key are not keys*". Therefore, any super set of any key should be removed before computing its closure. The removes done by using subsystem called ***supersets remover***. The removed supersets are faded to show the hugeness of eliminated sets depending on the mentioned heuristic.

For example when this module generates (AB), it will be sent to the X^i generator, which generates $(AB)^+$ as follows:

X(0)= AB

X(1)= ABC

X(2)=ABCD

X(3)=ABCDE

X(4)=ABCDEG

X(5)=ABCDEG

$(AB)^+$= ABCDEG which includes all the attributes of the table, therefore AB is a key. Hence AB will be stored in the candidate key file,

and X^i generator will ignore the following sets of attributes because all of these are super set of AB; {ABC, ABD, ABE, ABG, ABCD, ABEG, ABDB, ABCG, ABCE, ABEG, ABCEG, ABDEG, ABCDEG}

Table (2) shows the manipulated sets of attributes after applying the mentioned heuristic , this table shows the effect of this heuristic in minimizing the key space.

Table (2) Minimized sets space

L1	L2	L3	L4
A	AB		
	AC		
B	AD	ACG	ACDG
C	AE	ADE	ACEG
D	AG	ADG	ADEG
E	BG	AEG	BCDE
G	DE	BCG	BCDG
	DG	BDG	CDEG
	EG	BEG	
	CG	CDG	
		DEG	

X^i computing Algorithm: Input (F,X)
Output X^+

```
Begin
    X+ =X;
    Repeat
        OldX+= X+ ;
        While the set of Functional Dependency F is not empty do
         Begin
            Fetch the rule R of the form Y◊Z;
            If  X+ belongs to Y then X+=concat(X+,Z);
        End;
    Until (X+= OldX+);
End;
```

Figure (6) Xi generator Algorithm

4.2 X^i Generator Module

This module receives a set of attributes from the K-Attributes Generator to compute the set closure. This computation is done according to the algorithm shown in figure (6).

4.3 Key Checker

Key Checker is a simple module to check the result of Xi generator, i.e., the closure of a set of attributes. If it includes all the attributes of a table, the set will be added to candidate keys table as a key. This operation is done by the following algorithm:

```
While the set of attributes of X+ is not empty
Begin
        Fetch an attribute A from   X+
        If A does not belong to the set of table's attributes
                Begin
                        Ignore X; it is not a key;
                        Exit;
                End;
        //else fetch next attribute
End
```

Figure (7) Key Checker Algorithm

This algorithm stores the attributes of X^+ in ORACLE variable size array or cursor. Then it frequently fetches an attribute and checks it's belong ness to the set of the attributes of the table under processing.

5. Conclusions and Future works

The proposed system is tested by using many schemas of databases. The candidate keys sets are detected correctly and efficiently. From the implementation view point most of the execution time is consumed in generating and degenerating the sets of attributes. Also, the execution time is increased according to the number of the dependency

rules, because each rule should be checked against each set of attributes which passed the pruning step. Another factor which influences the execution time is the number of attributes in the schema because the number of the combination of the attribute sets will be increased.

There are many developments that can be done as future works:

1) Defining and implementing the criteria that can be used to select the primary key from the set of candidate keys.
2) Modify the proposed system to perform automatic normalization.
3) Defining and implementing the criteria to determine the foreign keys of database tables.
4) It is very important future work that is extracting the functional dependency rules depending data mining and Armstrong's theorem. In this way we avoid the manual detection of these rules which consumes a huge amount of time and efforts.

6. Reference

1. C. J. Date, "An Introduction to Database Systems", Addison-Wesley Publishing Company.
2. Ramiz Elmasri, "*Fundamentals of Database Systems*", Pearson International Edition, 5ed, 2007.
3. Rakesh Agrawal, Ramakrishman Srikant, "*Fast Algorithm for mining Association Rules*", IBM Almaden Research Center, 1997
4. Jeffrey D. Ullman, "Principles of DATABASE SYSTEMS",Stanford University,Galgotia Publications Ltd. ,1997