Teaching Students How to Find the Candidate Keys of a Relational Database Schema

Hossein Saiedian, Ph.D

ABSTRACT: *While most database textbooks provide formal and informal definitions for the candidate key of a relational database scheme, none provide a simple approach for finding the keys. In this paper, we present a simple algorithm that can be used by the students to correctly and efficiently identify the candidate keys of a relational database schema. The algorithm examines the position of attributes in the functional dependencies to determine which attributes are essential and useful for determining the keys and which attributes should be ignored. A key is found by computing the closure of essential attributes.*

KEYWORDS: *Relational Database, Candidate Keys, Functional Dependencies*

INTRODUCTION

Suppose you (or the textbook you have adopted for your database class) give the following problem to your students:

Given a universal relational database schema *R(A, B, C, D, E, F, G)* with a set of functional dependencies $\Gamma = \{A \rightarrow B, CD \rightarrow A, CB \rightarrow E, AF \rightarrow G, CF \rightarrow D\}$, find all candidate keys of *R*.

How are the students supposed to determine the candidate key(s) of *R*?

Most popular database textbooks provide only a definition for the key but do not describe how to determine one [3, 5]. Some authors, e.g., Ullman [8] indirectly provide an algorithm. The algorithm essentially involves computing the closure of all possible subsets of the attributes of *R*. One or more of such subset may exhibit candidate key properties. Computing the closure of all subsets of the attributes of a relation is no easy task.

In their new edition, Elmasri and Navathe [4] offer the following algorithm:

```
set K ← R;
  for each attribute A ∈ K
    compute (K - A)+ with respect to Γ
    if (K - A)+ contains all the attribut-
    es of R then
        set K ← (K - A)
```

The above algorithm suffers from two major deficiencies:

- First, it requires many computations. Suppose a relation has n attributes. This algorithm calls for computing the closure of *n - 1* attributes, then the closure of *n - 2* attributes, and so on until a key is found.
- Second, it returns only one key for *R*, and the returned key depends on the order in which attributes are removed. For example, for *R(ABCDEF)* with a set of FDs Γ, if we start removing attributes starting with *F*, followed by *E*, followed by *D*, we may end up with *ABC* as a key while not realizing that, for example, *E* by itself, or *F* by itself, or the combination of *EF*, or *E* or *F* combined with any of the attributes *A*, *B*, or C might have been keys of *R*.

Our experience with and observation of almost all of the academic problems of the above kind has shown that when determining the candidate keys of a relation, only a small subset of the attributes has to be considered, and that the candidate keys come from that small subset. The question is which subset and how to determine it? This is the objective of the rest of this paper.

OBJECTIVES

Our objective is to present a simple algorithm that can be used by the students in an introductory database course to determine the candidate keys of a universal relational database schema. We are presenting this algorithm because most of the popular database textbooks explain how to determine the candidate keys. Practically, most database textbooks give informal and formal definitions of a key, and some introduce rather complicated algorithms for computing the closures of a set of attributes or a set of functional dependencies, but the issues in determining the candidate keys are not covered. Thus our objective is to provide a simple algorithm that can be used by the students to find the keys of a universal relation, a point that is not handled with clarity in the textbooks. The algorithm that we provide may not necessarily have a broad theoretical appeal. But that is not our objective. We are familiar with theoretical contributions in [6] and [2] and the extensive discussion of the theory of relational databases and the functional dependencies in [7]. However, the concepts and algorithms presented in these and similar articles are too technical to teach to the introductory database students. We believe our key finding heuristic is simple and practical for such purpose.

A KEY FINDING ALGORITHM

Given a universal relation *R* and a set of functional dependencies Γ, we carefully examine the position of the attributes of *R* in Γ, and decide which attributes must be part of a key, which attributes will never be part of any key and which attribute may be part of some key. Let's refer to these three sets of attributes

as λ, ρ, and β respectively. When determining the keys, it becomes obvious that,

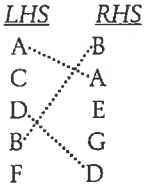
- attributes of ρ should not be considered as they will never be part of any candidate key,
- attributes of λ must be part of every candidate key of R, and
- attributes of β should be considered only when the set λ does not produce all the candidate keys.

In following paragraphs we discuss what λ, ρ, and β represent and how to construct these three sets.

Given the universal relational database schema *R(A, B, C, D, E, F, G)* with a set of functional dependencies $\Gamma = \{A \rightarrow B, CD \rightarrow A, CB \rightarrow E, AF \rightarrow G, CF \rightarrow D\}$, we divide the attributes of *R* into two not-necessarily-distinct sets. One set represents all attributes that occur on the left-hand side *(LHS)* of at least one functional dependency (FD) while the other set represents those attributes that occur on the right-hand side *(RHS)* of at least one FD. For relation *R*, *LHS* and *RHS* are as follows:

LHS	RHS
A	B
C	A
D	E
B	G
F	D

Next, we identify those attributes that appear on both sides by drawing a dotted line between them:



Now we can construct the three sets mentioned earlier. Set λ refers to those attributes that appear on the *LHS* only, namely C and *F*. Set ρ refers to those attributes that appear on the *RHS* only, namely *E* and G. Finally, set β refers to those attributes that appear on both *LHS* and *RHS*, namely *A*, *B*, and *D*. (Note that the above "tables" need not have been drawn. The sets λ, ρ, and β can be generated simply by inspecting the functional dependencies. However, we use the above approach in the classroom because it makes construction of the three sets more visual, simpler, and less prone to errors.) The key finding algorithm continues as follows:

- Ignore attributes of ρ. Using Armstrong inference rules [8], it can easily be shown that attributes appearing in ρ will never be a part of any candidate key of *R*.
- All attributes of λ must be part of every candidate key of *R*. Compute the closure of λ. If λ by itself forms a key of *R*, then it is the only key of *R*. (Both of these two assertions are provable by using the Armstrong rules.)
- If λ does not produce a key, add attributes, one at a time, from β to λ and compute their closure until all keys are found. Note that all attributes of β must be considered in order to find all candidate keys. Furthermore, note that computing each new closure is simple because the closure of attributes of λ has already been produced. Thus all that is needed is to determine if the addition of a new attribute from β to λ produces a key.

We apply the above algorithm to our problem. Here is the problem description repeated from Page 1:

Given a universal relational database schema *R (A, B, C, D, E, F, G)* with a set of functional dependencies $\Gamma = \{A \rightarrow B, CD \rightarrow A, CB \rightarrow E, AF \rightarrow G, CF \rightarrow D\}$, find all candidate keys of *R*.

We have already determined λ,β, and ρ: λ = {*CF*} ; β = {*ABD*} ; and ρ = {*EG*}. According to the algorithm, C and *F* must be part of any key while *E* and *G* will not participate in any key. Note that according to the algorithm, (a) we need not consider computing the closure of any single attribute because λ contains two attributes, (b) nor do we need to consider attributes *E* and *G* because they appear in ρ, and (c) we should consider attributes of λ before considering any other combination of attributes.

Let's consider λ : $\lambda^+ = \{CF\}^+ = CFDABGE$. Thus {*CF*} is a key of *R*. According to the algorithm, {*CF*} is the only key of *R*. We need not consider any other combination of attributes.

Brief Discussion

Note how the key finding process has been simplified by the algorithm. Computing the closures of every possible combination of attributes to find all potential keys of even a small relation like the above example would have been too time consuming.

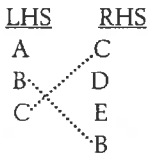
For the above example, we considered a schema for which the set λ was non-empty and formed the only key for the schema. In other words, we only had to consider λ. For the next example, we will consider a schema for which λ will also be non-empty but it will not form a key.

ANOTHER EXAMPLE

Consider the following problem description:

Given schema *S (ABCDE)* and the set of functional dependencies $\Gamma = \{AB \rightarrow CDE, AC \rightarrow BDE, B \rightarrow C, C \rightarrow D, B \rightarrow E\}$, find all candidate keys of *S*.

We have



Thus, we end up with the following attributes for λ, β, ρ: λ = {*A*}; β = {*BC*}; ρ = {*DE*} . According to the algorithm, *A* must be part of every key of *S* while *D* and *E* may never participate in any key.

Let's consider attributes of λ only: $\lambda^+ = A^+ = A$. Thus *A* is not a key of *S*. We now consider adding attributes from β to λ and compute their closure to find all keys:

$\{AB\}^+ = ABCDE$. *AB* is a key of *S*.
$\{AC\}^+ = ABCDE$. *AC* is a key of *S*.

AB and *AC* are the only keys of *S*.

Brief Discussion

According to the algorithm, *AB* and *AC* are the only keys of *S*. Note that:

- we did not need to consider any single attribute of *S* except *A*.
- we did not need to consider any combination of *D* or *E* with *A* because these two attributes may never be in any key of *S*,
- we did not need to consider *BC* because *A* must be a part of every key of *S*, and
- we did not need to consider other combinations, e.g., *ABC*, since they will yield a superkey.

If λ = ∅ and ρ = ∅ (that is, β contains all attributes), then it implies the worse case scenario, i.e., every attribute might be a potential component of a key. One solution is to use the "traditional" approaches, i.e., considering all potential combination of attributes but such scenarios rarely happen, at least in academic problems.

A MORE TYPICAL ACADEMIC EXAMPLE

In this section, we consider another academic example. The example is adopted from the exercises at the end of Chapter 7 of Ullman [8].

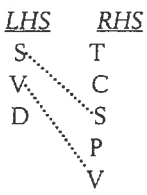
Consider a database of ship voyages with the following attributes:

- ship name represented by S,
- type of ship, represent by T,
- voyage identifier, represented by V,
- cargo carried by one ship on one voyage, represented by C,
- port name, represented by P, and
- voyage day, represented by D.

It is assumed that a voyage consists of a sequence of events where one ship picks up a single cargo, and delivers it to a sequence of ports. A ship can visit only one port in a single day. The following functional dependencies may be assumed: $\Gamma = \{S \rightarrow T, V \rightarrow SC, SD \rightarrow PV\}$.

The above represents a typical example of an academic problem. The readers are challenged to determine the key(s) before preceding further.

We have



The attributes of λ, β, and ρ are as follows: $\lambda = \{D\}$, $\beta = \{SV\}$, and $\rho = \{TCP\}$. According to the algorithm, *D* must be a part of every candidate key while *T*, *C*, and *P* will not participate in any key. Let us consider the closure of λ: $\lambda^+ = D^+ = D$. Thus *D* is not a key of the database.

Now, we consider adding attributes one at a time from β to λ, and compute its closure:

{DS} + = DSPVCT. DS is a key.
{DV} + = DV S... DV is also a key.

Thus the ship database has two keys, namely *DS* and *DV*. We do not need to consider any other combination of attributes.

CONCLUDING REMARKS

We believe the algorithm presented in this article is simple and practical for academic purposes. We have applied it to many small and large academic problems. The results have been correct and the process has been quite efficient.

ACKNOWLEDGMENTS

The author wishes to acknowledge anonymous reviewers for their helpful comments. This research was partially supported by a 1995 UCR grant, University Committee on Research, University of Nebraska at Omaha.

REFERENCES

[1] C. J. Date. *An Introduction to Database Systems*, volume I. Addison-Wesley, Reading, MA, 6th edition, 1995.

[2] D. Kroenke. *Database Processing*. Prentice-Hall, Englewood Cliffs, NJ, 5th edition, 1995.

[3] J. Ullman. *Principles of Database and Knowledge-Based Systems*, volume I. Computer Science Press, Rockville, MD, 1988.

[4] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin-Cummings, Menlo Park, CA, 2nd edition, 1994.

[5] C. Lucchesi and S. Osborn. Candidate keys for relations. *Journal of Computer and Systems Sciences*, 17(2):270-279, 1978.

[6] G. Ausiello, A. D'atri, and D. Sacca. Graph algorithms for functional dependency manipulation. *Journal of the ACM*, 30(4):752-766, 1983.

[7] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, MD, 1983.

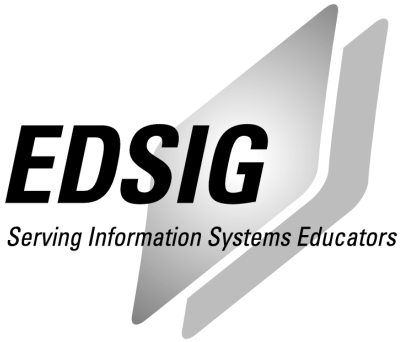
[8] W. Armstrong. Dependency structures of database relationships. In Proc. 1974 IFIP Congress, pages 580-583, Geneva, 1974.

Hossein Saiedian, Ph.D.
Department of Computer Science
University of Nebraska at Omaha
Omaha, Nebraska 68182
hossein@unomaha.edu

Hossein Saiedian is an associate professor in the Department of Computer Science at the University of Nebraska at Omaha, USA. He is a member of the IEEE Computer Society, Sigma Xi, the ACM, and currently serves as the Chair of the ACM SIGICE (Special Interest Group in Individual Computing Environments).

Dr. Saiedian has published over 40 technical articles including articles in recent issues of IEEE Computer, International Journal of Computing & Information Technology, Computer Networks & ISDN Systems, Journal of Systems and Software, and Journal of Information & Software Technology. His pedagogical articles have appeared in Computer Science Education, Journal of Information Systems Education, and SIGCSE Bulletin. Dr. Saiedian was ranked as the third leading software systems scholar in the October 1994 issue of Journal of Systems and Software.

Information Systems & Computing
Academic Professionals



STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096