Derivation of Database Keys' Operations

Adio Akinwale, Olusegun Folorunso, and Adesina Sodiya
Department of Computer Science,
University of Agriculture, Abeokuta, Nigeria

aatakinwale@yahoo.com; folorunsolusegun@yahoo.com;
sinaronke@yahoo.co.uk

Abstract

Designing normalized relation is subject of great difficult, yet great importance. The paper presented automatic derivation of database keys as a step forward solution to designing normalized relations. It studied relational schema and functional dependencies to derive keys of primary, candidate, alternative and super key of relational database. The necessary algorithms to derive database keys were analyzed, modified and coded in Java Programming Language to display graphical key derivation input interface. The input interface accepts any length of schemas' attributes and functional dependencies to compute power set, closure, derived keys and specific group of keys which can aid database designers to control anomalies at initial stage of database design. The system also assists the students to find, "see", and learn database keys derivation. The effectiveness has a positive impact on the students' performance on normalization technique.

Keywords: database anomalies, inconsistency, database keys, functional dependencies, relational schemas, Armstrong's inference rules, prime and non-prime attributes, normal forms

Introduction

Designing database is an art process similar to building a house. Database designers always face the problems of designing a relational database that will be free of database anomalies. These anomalies bring repetition of tuples that delay processing time and occupy memory spaces.

Suppose that the value of the attribute BUILDER determines values of the attribute MODEL and PRICE, (BUILDER → MODEL, PRICE) and that the value for the attribute MODEL determines the value for PRICE, (MODEL → PRICE). Grouping these attributes in relation HOUSE(BUILDER, MODEL, PRICE) has several undesirable properties. First the relationship between MODEL and PRICE is repeated in the relation for each BUILDER who builds a particular MODEL of home. This repetition creates difficulties if a BUILDER who happens to be the last BUILDER of a certain MODEL home is deleted from the relation, then the relationship between the MODEL and its PRICE also disappears from the relation. This is called a deletion anomaly. Similarly, if a new builder who happens to be the first BUILDER of a certain MODEL home is added then the relationship between MODEL of a home and its PRICE will also be added. This is called an insertion anomaly. Suppose that the relationship between a MODEL and its PRICE is changed e.g. the price is increased; then the MODEL and PRICE relationship should be affected for every BUILDER of the

MODEL. This is called update anomaly. These anomalies are undesirable since the user is not likely to realize the consequence of the insertion, deletion or updating. The user may inadvertently affect a relationship that was not intended to be modified. Consistency, insertion, deletion and updating are not probe effecting all groupings of attributes. If the relation HOUSE(BUILDER, MODEL, PRICE) is normalized then the consistency and anomaly problems disappear.

Normalization is a step by step reversible process of replacing a given collection of relations by successive collection in which the relations have a progressively simpler and more regular structure (Date & Darwen, 2000). The reversibility guarantees that the original collection of relations can be recovered and therefore no information has been lost. Codd proposed three normal forms which he called first normal form (1NF), second normal form (2NF) and third normal form (3NF). A stronger definition of 3NF was proposed by Boyce and Codd and is known as Boyce-Codd Normal Form (BCNF). All these normal forms except 1NF are based on the functional dependencies among the attributes of a relation (Elmasri & Navathe, 1994).

First normal form relates to the structure of the relation. It requires that every attribute of a relation be based on a simple domain. The database designers have no problem to know if a relation violates first normal form. They can put the relation into first normal form algorithmically by replacing a non-simple domain by its constituent simple domains. In the second (2NF), third (3NF) and Boyce Codd normal form (BCNF), there is a need for the database designers to know the real meaning and application of database keys such as candidate key, primary key, super key, etc,.

Problem Statement

Database designers always find it difficult to determine these keys from relational database schemas. It has been difficult to motivate students and database designers to derive primary, candidate, alternative and super keys because they think this area is dry and theoretical. There are many algorithms to determine the database keys but they look abstract for students. Many database researchers indicated that relational database model to derive database keys tends to be complex for the average designers. Failure to determine the database keys at times leads to poor design that can generate database anomalies. The database key algorithms often require extensive relational algebraic backgrounds that database designers lack. Most database text books rely on the definition of the keys in their course areas. They simply give the definition of primary key, candidate key, alternative key and super key and hope the database designers would be able to apply the definition to determine these keys from relational schemas and functional dependencies. It is still difficult for students to understand which attributes from a given relation determines another attributes. These approaches may not be the best way to assist many database designers effectively understands the derivation of database keys. It has been noticed that normalization process, decomposition and loss less join happened when various database keys have been expressed.

Literature Review

There are many literature reviews on database normalization but none has singly or deeply involved on the derivation of database keys which are very fundamental in this area. Hsiang-Jui Kung and Hui-Lien Tung (2006) developed a web-based database normalization tool and its effectiveness in teaching and learning normalization. The aspect of database keys was not touched. Concepcion and Villafuerteq (1990) and Rosenthal and Reiner, (1994) proposed algorithms and tools to synthesize a normalized database using functional dependencies. The algorithms did not illustrate how to derive database keys of primary, candidate, alternative or secondary and super. Chilton, McHaney, and Chae (2006) developed a normalization tool to teach student on normalization process but the tool did not provide means of finding database keys. Diederich and Milton

developed new methods and fast algorithms for database normalization. These methods and algorithm did not include database keys derivation.

Most database authors provide a definition for the key but no algorithm for computing it. David Maier (1993) and Jeffrey Ullman (1988) provided algorithms for computing the closure of a set of attributes or a set of functional dependencies but the calculation of a key is left for the readers using the closure algorithms. Determining the candidate keys for a small relation schema with a small set of functional dependencies may be trivial but if a relation has a relatively large number of attributes and/or functional dependencie, then determining the keys may not be a trivial process. Ramez Elmasri and Shamkant Navathe (1994) offer a candidate key algorithm as shown in Figure 1.

$$
\begin{array}{l}
set\ K \leftarrow R \\
for\ each\ attribute\ A \in K \\
conpute\ (K - A)^{+}\ with\ respect\ to\ fds, \Gamma\ for\ R \\
if\ (K - A)^{+}\ contains\ all\ attributes\ of\ R\ then \\
set\ K \leftarrow K - (A)
\end{array}
$$

Figure 1: Elmasri and Navathe 's Candidate Key Algorithm

The algorithm in Figure 1 returns only one key for R and the returned key depends on the order in which attributes were removed. For example, for R(ABCDEF) with a set of functional dependencies, if we start removing attributes from the right side, that is F, followed by E, followed by D, we may conclude that ABC is a key. We may not realize that for example E by itself or F by itself or a combination of EF or E or F combined with any of the attributes A, B, or C are also keys of Ausiello, Datri, and Sacca (1983) who provide an approach for homogenous treatment of several related aspects of relational database schemas, namely, closure, minimalization and synthesizing a relational schema in 3NF. In their work, database key funding issues were not treated. Biskup, Dematrovics, Libkin, & Muchnik (1991) explored the relational database schemes having a unique minimal key but the emphasis is on their relationship with normal form relation schemes and lattice theoretic issues. Key funding issues were not emphasized. The problem of funding a primary key is much easier than funding all derived keys of all candidate keys, super keys and alternative keys.

The objective of the paper is to develop a graphical key derivation user interface that would allow students and database designers understand the step-by-step derivation of database keys. Claudio Lucchesi and Sylvia Osborn (1978) designed an algorithm which finds all the keys and whose running time is a polynomial of the size of the input and the size of output. The algorithm is modified to derive the keys of candidate, alternative, primary and super keys as shown in Figure 1.

Definitions

Functional Dependencies

The single most important concept in relational schema design is that of a functional dependency. 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 instance r of R. The constraint states that for any two tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$. We must also have $t_1[Y] = t_2[Y]$. This means that the values of the Y component of a tuple in r depend on or are de-

termined by the values of the X component, or 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 (Elmasri & Navathe, 1994).

Key of Relation

We say a set of one or more attributes { A_1, A_2, A_3,…,A_n } is a key for a relation if

(1) those attributes functionally determine all other attributes of the relation. That is, it is impossible for two distinct tuples of R to agree on all of A_1, A_2, A_3,…,A_n.

(2) there is no proper subset of { A_1, A_2, A_3,…,A_n } functionally determine all other attributes of R. Naturally, there are many keys that already identified in database schemas. Among them are candidate keys, primary keys, alternative keys, secondary keys, super keys and foreign keys.

Candidate key is unique identifier enforcing that no tuple will be duplicated. If a relation schema has more than one minimal key, each is called a candidate key. One of the candidate keys is arbitrarily designated to be the primary key of the relation schema and the other are called secondary keys. Each relation schema must have a primary key. A primary key is a set of attributes that uniquely determined an instance of an entity. Suppose we have the following schemas:

Customer(**customer_number**, customer_name, customer_address, customer_phone…,)

Order(**order_number**, customer_number, invoice_number,…,)

Invoice(**invoice_number**, customer_number, order_number,…,)

Order_Line(**order_number**, **order_line_number**, produce_code,…,)

In the above example, we have **four** relational schemas, namely, **customer**, **order**, **invoice** and **order_line**. The bold underlined attributes are candidate keys. The non-bold underlined attributes are foreign keys. Usually one candidate key is the primary key and used in preference over the other candidate keys which are alternative keys. For example, in order_Line(**order_number**, **order_line_number**,…,); order_number can be chosen as primary key while order_line_number can now be alternative key. But both are candidate keys.

A super key of a relation schemas R= { A_1, A_2, A_3,…,A_n } is a set of attributes $S \subseteq R$ with the property that no two tuples t_1 and t_2 in any relation instance r of R will have $t_1[s] = t_2[s]$. A key K is a super key with the additional property that removal of any attribute from K will result in K not being a super key any more. Every key is a super key. Some keys are not super keys. If a relation R is considered from a relationship then

(1) if the relationship is many to many (n: m) then the keys of both connected entity sets are the key attributes of R many to many.

(2) if the relationship is from entity sets many (E1) to entity set one (E2) (n:1) then the key attributes of E1 are the key attributes of R but those E2 are not.

(3) if the relationship is from entity set one to one (1:1) then the key attributes for either of the connected entity sets are the key attributes of R.

Database Key Constraints

The work identifies four database key constraints as follows:

i The difference between a key and a super key is that a key has to be minimal, that is, if we have a key K= $(A_1, A_2, A_3,\ldots,A_n)$ then K – A is not a key for $1 \le i < k$.

ii Consider the car relation: CAR(label, reg#, serialNo, make, model, year) which has two keys : K1 = {label, reg#} and K2 = {serialNo}. These keys {K1, K2}are super key (SK). {serialNo, make} is a super key but not a key. If a relation has several candidate keys, one is chosen arbitrarily to be the primary key. The primary key attributes are underlined. Any of the candidate keys that is not part of the primary key is called an alternative key.

iii Super key of R: a set of attributes SK of R such that no two tuples in any valid relation instance r(R) will have the same value for SK. That is, for any distinct tuples t_1 and t_2 in r(R), $t_1[SK] \neq t_2[SK]$.

iv In a BCNF relation R, every functional dependency in R must be of the form K→A where K is a key and A is any attribute. The following can be asserted:

- all non-prime attributes must be fully dependent on each key
- all prime attributes must be fully dependent on all keys of which they are not a part.
- No attribute (prime or not prime) can be fully dependent on any set of attributes that is not key.

Since each key is fully dependent on every other key, the keys are functionally equivalent. That is, for every pair of keys there is a bijection, i.e., functions in both directions connecting them

proof:

$$H(A) \to X(A) \;//\; characteristic\ function\ in\ set\ A$$

$$H:\ P(N) \to \{0,1\}^N$$

$$A = \{1,3,6,7,8,12,\ldots,\} \xrightarrow{H} \begin{pmatrix} 0,1,0,1,0,0,1,1,1,\ldots, \\ 0,1,2,3,4,5,6,7,8,\ldots, \end{pmatrix}$$

$$H\ to\ bijection$$

$$H: P(N) \xrightarrow[on]{1\to 1} \{0,1\}^N$$

$$1 \to 1 because\ A \neq B \to X(A) \neq X(B)$$

$$H(A \wedge B) = H(A) \circ H(B)$$

$$X(A \wedge B) = X(A) \circ X(B)$$

$$X(A \wedge B) = \min imal(X(A), X(B))$$

$$X(A \wedge B), X(A), X(B) : N \to \{0,1\}$$

$$X(A \wedge B) \qquad X(A) \ \circ\ X(B)$$

$$\xleftarrow[-,-,k,-]{-,-,0,-} \qquad \xleftarrow[-,-,k,-]{-,-,1,-}$$

The function is one to one in every position

$$X(A)_{(k)} = \begin{Bmatrix} 0 & k \notin A \\ 1 & k \in A \end{Bmatrix}$$

$$\forall_k \in N$$

$$X(A \wedge B)_{(k)} \quad \min imal(X(A)_{(k)}, X(B)_{(k)})$$

$$X(A \wedge B)_{(k)} = 0 \Leftrightarrow k \notin A \wedge B$$

$$k \notin A \quad or \quad k \notin B$$

$$X(A)_{(k)} = 0 \quad X(B)_{(k)} = 0$$

$$hence \quad X(A \wedge B)_{(k)} = \min imal(X(A)_{(k)}, X(B)_{(k)})$$

$$\Rightarrow X(A \wedge B) = X(A) \circ (B)$$

$$k \in A \wedge B \Leftrightarrow k \in A \quad or \quad k \in B$$

$$k \in A \wedge B \Leftrightarrow k \notin A \quad or \quad k \notin B$$

These four constraints were embedded into database keys derivation to derive candidate leys, primary keys, alternative keys and super keys.

Methodology

The following algorithms are needed to generate database keys of primary, candidate, alternative, secondary and super keys from relational schemas R and a set of functional dependency (fd)

- Power set of relational schemas
- Closure T^+ Algorithm
- Keys Algorithm

Power Set of Relational Schemas

This power set is to determine the power set of relation T(A_1, A_2, A_3,…A_n) except the empty set. The **n** attributes will be combined into $2^n - 1$. For example a relation R(A,B,C) will generate a power set of $2^3 - 1$ (ABC permutation) as follows: (A, B, C, AB, AC, BC, ABC) minus empty set (ϕ).

Closure X^+ Algorithm

The schema designer will specify the relation and functional dependencies that are semantically obvious. There are usually numerous other functional dependencies that will also hold in all legal relation instances that satisfy the dependencies in T. The set of all such functional dependencies is called the closure of T and is denoted by T^+. A systematic way to determine these additional functional dependencies is first to determine each set of attributes X that appears as a left-hand side of some functional dependencies in T and then use Armstrong's inference rules of reflexive, augmentation, transitive, decomposition union and pseudotransitive to determine the set of all attributes that dependent on X as shown in figure 2. Thus, for each set of attributes X, we determine the set X^+ of attributes that are functional determined by X; X^+ is called the closure of X under T.

1. Let X be a set of attributes that eventually will become the closure

   -we first initialize X to be $\{A_1, A_2, A_3, ..., A_n\}$

2. we repeatedly search for some functional dependency

   $B_1, B_2, B_3, ..., B_m \rightarrow C$

   such that all $B_1, B_2, B_3, ..., B_m$ are in the set of attribute X

   but **C** is not.

   we add C to the set X

3. repeat 2 as many times as necessary until no more attributes can be added to X

4. the set X after no more attributes can be added to it is the correct value of

   $\{A_1, A_2, A_3, ..., A_n\}^+$

Figure 2: Closure Algorithm

Key Algorithms

Key algorithm used the algorithm of candidate key (cd) which was modified to determine those attributes that must be part of a key and those attributes may not be part of the keys. It considered attributes that would lead to primary key (pk), alternative key (ak) and supper key (sp). Figure 3 describes the algorithm of derived keys.

Implementation

The algorithms of power set, closure (Figure 2) and database keys (Figure 3) were developed using JAVA programming language. The tool has input interface to key in and display relational schema and functional dependencies. The tool is in position to accept any length of relational attributes and functional dependencies. It will derive the database keys and group them into primary key, candidate key, alternative key and super key. As illustrated in the introduction section, Figure 4 depicts sample of three attributes of relation House(Builder = B, Model = M, Price = P) and two functional dependencies: B→M, M→P. It is possible for the user to add more attributes and functional dependencies. As shown in figure 4 the user has keyed in three attributes and two functional dependencies. The next step is for the user to press power set button to generate the power set of R as still shown in Figure 4.

Given a header R(A_1, A_2, A_3,…, A_n) and a set of FD that only contain subset of R. It holds that X→Y is in S^+ if only if $Y \cup X^+$.

Input: R(A_1, A_2, A_3,…, A_n) and a set of FDs

Output: a set of derived keys that hold as primary, candidate, alternative and super keys in all relation universes over R in which all FDs in S hold

begin

1. set F' to a minimal cover of F
2. partition all attributes in R into necessary, useless and middle-ground attribute sets

   according to F'

Let $X = \{C_1, C_2, C_3, ..., C_n\}$ be the necessary attribute set

Let $Y = \{B_1, B_2, B_3, ..., B_k\}$ be the useless attribute set

Let $M = \{A_1, A_2, A_3, ..., A_n\} - (X \cup Y)$ be the middle-ground attribute set

If $X = \{\}$ then go to step 4

3. compute X^+

if $X^+ = R$ then set $K = \{X\}$ terminate

4. let $L = < Z_1, Z_2, Z_3, ..., Z_m >$ be the list of all non-empty subsets of M,

the middle-ground attributes such that L is arranged in according

order of the size Z_i

add all attributes in X, necessary attribute to each Z_i in L

set $K = \{\}$; $i = 0$

while $L \neq$ empty do begin

$i = i + 1$

remove the first element Z from L; compute Z^+

if $X^+ = R$ then

begin

set $K \leftarrow K \cup \{Z\}$

for any $Z_j \in L$

if $Z \subset Z_j$ then $L \leftarrow L - \{Z_j\}$

end

Figure 3: Derived Keys Algorithm

Figure 4: Key Interface to generate Power Set

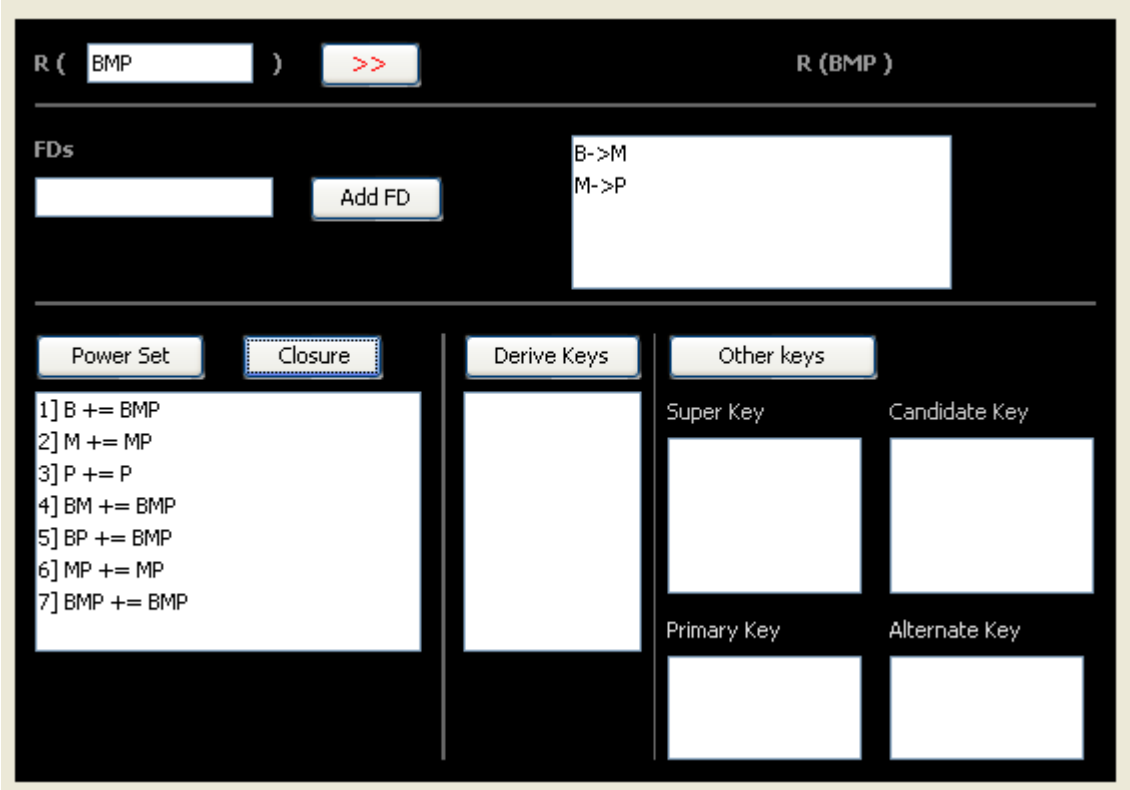

Figure 5: Key Interface to generate Closure of the Attributes

The next step is for the user to press **closure button** that will display all the attribute values of each element as shown in Figure 5. Pressing **derive keys button** will display all attributes in closure that contain relational attributes as shown in Figure 6. Figure 6 has derived key of B, {BM} and {BP} and they contain {BMP} in closure box. The user will press **other keys button** to see how attributes in derive key are distributed into primary, candidate, alternative and super keys as shown in Figure 7.

Figure 6: Key Interface to generate Derived Keys

The specific group of keys can aid database designers to design good schema and check database anomalies. For example, primary keys can be used to sort and translate schema. It can also be used to map entity relations and relational model. The candidate keys can be used to list instance object for loaded models. Super keys would assist to check if relations violate 3NF or BCNF. For example, as shown in Figure 7, the only primary key is B. The super keys are {BM} and {BP}. It is clearly seen that the left hand side of M→P is not a super key. The schema is not in 3NF because the right hand side is not part of the key.

Figure 7: Key Interface to generate Other Keys

Looking at Figure 8 where we have the same schema: House(Builder = B, Model = M, Price = P) and FDs were changed into BM→P, and P→B, the whole process has changed. The derived keys are {BM} and {MP} as illustrated in figure 8. It is easy for database designer to check if the relation violate either 3NF or BCNF. The left hand side of BM→P is a key so it is okay. The right hand side of P→B is part of the key so it is okay. Therefore, the schema is in 3NF.

It is also easy to ask if the relation is in BCNF. Based on the definition of BCNF and derivation of keys as shown in Figure 8, this schema is not in BCNF because the only keys are {BM} and {MP} and the FD, P→B has a left hand side which is not a super key. It is the right hand side which was part of the key that made it in 3NF. This process is easy to follow without extensive background in relational algebra and database theories. The user needs to know the definition of third or boyce-codd normal form in order to determine the keys.

Figure 8: Key Interface with Changes in Functional Dependencies

Detailed Analysis

Students offer database design (CSC422) at 400 level in the Department of Computer Science, University of Agriculture, Abeokuta, Nigeria. Out of 15-week course, Lecturers and Tutors normally spend 3 weeks on normalization topic. In their final examination paper, one question on normalization technique always appears. Table 1 depicts the students' performance between 2000 and 2006.

The detailed comments on student's performance question by question by external examiner indicated that students found it difficult to determine the closure R^+ when the attributes of the relation were more than 5. More so, they found it so difficult to derive candidate and super keys. The students' performance on normalization question compared with other questions was very poor from year 2000 to 2006.

Before the beginning of 2007 second semester database design lectures, the graphical key derivation interface as shown in above Figures 3-8 has been developed as part of teaching and learning normalization topic. In 2007 database design tutorials of normalization topic, students were given exercises on relations with three to four attributes and various functional dependencies. It was very easy for average students to get the keys. The results were compared with the text books' answers and graphical key derivation interface which showed no discrepancies. Two in-class exercises were given to students as follows:

Exercise A: R(A, B, C, D, E, F, G, H, I, J, K, M, N, P)

FDs: A, G → B, C, D, E, F, H, J, K, L, M, N, P

J → K ,L, M, N, P

M → N, P

G → H

Exercise B: CAR_SALE(invoice#, date, line#, quantity, selling_price, coupon _deduction, saving, sub_total, product#, product_name, product_price, vendor#, vendor_name, vendor_city)

FDs: invoice#, line# → date, quantity, selling_price, coupon_deduction, saving, sub_total, product#, product_name, product_price, vendor#, vendor_city

product# → product_name, product_price, vendor#, vendor_name, vendor_city

vendor → vendor_name, vendor_city

invoice# → date

None of the 114 students offered database design course for 2007 session was able to derive the closure of exercise A. Later, they were exposed to use the graphical key derivation interface to get the closure of R^+. After deriving the closure, 102 out of 114 students derived the keys. They used the tool to do more exercises from the text book. Exercise B was given as continuous assignment test and the result showed that 105 out of 114 students got the closure, non-trivial functional dependencies and the keys after the use of graphical key derivation interface. The tool was used for both teaching and learning of normalization topic for 2007 and 2008 sessions. Table 2 illustrates the students' performance on normalization questions for 2007 and 2008 second semester examination on database design.

The external examiner remarks on the students' performance in normalization questions for both 2007 and 2008 sessions were very impressive compared with other questions. Looking at Tables 1 and 2, before and after introduction of graphical key derivation interface, students knew and answered normalization questions better in respect of grade-wise.

Table 1: Students' Performance on Normalization question

Year offered the course	Normalization question	No. of Student offered the course	No of student attempted the question	Overall performance (%)
2000	**1**	**65**	**5**	**30%**
2001	**2**	**82**	**7**	**35%**
2002	**1**	**67**	**11**	**22%**
2003	**3**	**49**	**15**	**32%**
2004	**1**	**55**	**10**	**35%**
2005	**1**	**63**	**12**	**28%**
2006	**6**	**42**	**5**	**39%**

Table 2: Students' performance on normalization question after introduction of graphical key derivation interface

Year offered the course	Normalization question	No. of Student offered the course	No of student attempted the question	Overall performance
2007	**1**	**114**	**113**	**85%**
2008	**6**	**105**	**96**	**92%**

Conclusion and Future Research Direction

The paper identified definitions of each component of relational schema that are necessary to determine various database keys. It enumerated the problems of database designer to identify specific keys for designing relational schemas that would free database anomalies. The work implemented necessary algorithms to derive database keys in their specific groups. These groups can provide piece of information for schema translation, separation of multiple mapped keys and determination of relations that violate normal forms. It was justified that the developed graphical key derivation interface guided students to learn normalization topic effectively. This tool can be expanded to display non-trivial functional dependencies from derived keys and if any functional dependency followed specified functional dependencies. The incorporation of decomposition and loss-less join dependency is proposed if a relational schema violates either 3NF or BCNF as further studies for proper perfection of relation from database anomalies.

References

Atamturk, E. L., Johnson, E. L., Linderoth, J. T., & Savelsbergh, W. P. (1999). A relational modeling system for linear and integer programming. *Information and Management*, *48*(5) 846-857.

Ausiello, G., Datri, A., & Sacca, D. (1983). Graph algorithms for functional dependency manipulation. *Journal of the ACM, 30*(4), 752-766.

Biskup, J., Dematrovics, J., Libkin, L., & Muchnik, I. (1991). On relational database schemes having unique minimal key. *Journal of Information Processing Cybernetics, 27*, 217-225.

Cattel, R. G. G., & Barry, D. K. (1997). *The object database standard*. Morgan Kaufmann Publishers.

Ceri, S., Navathe, S. B., & Wielderhold, G. (1983). Distribution design of logical database schemas, *IEEE Transactions on Software Engineering*, 487-503.

Chilton, M. A., McHaney, R. & Chae, B. (2006). Data modeling education. *The Journal of Information Systems Education, 17*(1), 17-20.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), 377-387.

Codd, E. F. (1982). Relational database: A practical foundation for productivity. *Communications of the ACM, 25*(2), 109-117.

Codd, E. F. (1990). *The relational model for database management: Version 2*. Addison-Wesley.

Concepcion, A. A., & Villafuerte, R. M. (1990). Expert database: An assistant database design system. *Proceedings for the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems,* Vol 1.

Date, C. J. (1986). *An introduction to database systems* (Vol. 1). Addison-Wesley.

Date, C. J. (2003). *An Introduction to database systems* (8th ed.). Addison-Wesley.

Date C. J., & Darwen, H. (2000). *Foundation for future database systems* (2nd ed.). Addison-Wesley.

Dionysis, C., Tsichritzis, F., & Lochorski, H. (1982). *Data models.* Prentice-Hall.

Diederich, J., & Milton, J. (1988). New method and fast algorithm for database normalization. *ACM Transaction on Database Systems*, *13*(3), 339-365

Elmasri. R., & Navathe, S. B. (1994), *Fundamentals of Database Systems*, Benjamin/Cummings

Kung, H-J., & Tung, H-L. (2006). A Web-based tool to enhance teaching/learning database normalization. *Proceedings of the Southern Association for Information Systems Conference*, pp. 251-258.

Lucchesi, C., & Osborn, S. (1978). Candidate keys for relations. *Journal of Computer and System Sciences, 17*(2), 270-279.

Maier, D. (1983). *The theory of relational databases*. Rockville, MD: Computer Science Press.

Rosenthal, A., & Reiner, D. (1994). Tools and transformation for practical database design. *ACM Transaction on Database System, 19*(2), 167-211.

Saiedian, H., & Spencer, T. (1996). An efficient algorithm to compute the candidate keys of a relational database schema. *The Computer Journal, 39*(2), 124-132.

Teorey, T. J. (1994). *Database modeling and design* (2nd ed.). Morgan Kaufmann.

Thompson, C. B., & Sward, K. (2005). modelling and teaching technique for conceptual and logical relational database design. *Journal of Medical Systems, 29*(5).

Ullman, J. (1988). *Principles of database and knowledge based systems* (Vol. 1). Rockville, MD: Computer Science Press

Biographies

Akinwale Adio Taofiki received his Magister in Informatics from Warsaw University, Warsaw, Poland, M.Sc./PhD in Economic Cybernetics and Computer Science from Oskar Lange University, Wroclaw, Poland. His area of research interest is knowledge database management system and optimization.

Olusegun Folorunso is a Senior Lecturer in the Department of Computer Science, University of Agriculture, Abeokuta. He obtained a B.Sc degree in Mathematical Sciences from the University of Agriculture, Abeokuta in 1992, M.Sc in Computer Science from University of Lagos in 1997 and a PhD in Computer Science in 2003 from the University of Agriculture, Abeokuta. His research interest includes Adoption of Information Systems strategies, Human Computer Interaction (HCI), Knowledge Management, Image Processing and Computational Intelligence. He is a member of Nigeria Computer Society and Computer Professional Registration Council of Nigeria. He has published in reputable international and local Journals.

Dr. **A. S. Sodiya** is presently a Senior Lecturer in the department of Computer Science, University of Agriculture, Abeokuta, Nigeria. He completed his PhD in the same university in 2004. His areas of research interest are Computer Security, Computer Network, Artificial Intelligence and Software Engineering. He has published both in local and international Journals. He has also attended and presented papers in several conferences.