CUSTOMIZED SEGMENT SYNTHESIS

James J. Odell

ABSTRACT

Segment design is a process which must be exercised by every data-base designer. This paper presents an easily automated method of designing segments taking into account system-usage factors. The segments resulting from this algorithm are a subset of all possible synthesis solutions; but they are those results which may be of more importance to the D.B. designer.

INTRODUCTION

When a data base analyst designs a data base, he models a part of the real world. In his modeling, he groups varied but somehow related data elements. These groups of elements are commonly named segments, relations or data base record layouts.

Unfortunately, the current practice of segment design is usually by trial and error, and the segment contents are created on an ad hoc basis. For small data bases, a poor design may not seriously impact a computer system, but for large data base systems the results may impose stiff economic penalties.

Data base design tools are now being developed to aid the DB designer. However, the synthesis of logical segments has only been discussed by a few [1-4]; and the execution times of some of the "synthesis" algorithms are unfeasible for even a relatively small number of data elements. The algorithms, although very thorough, are cumbersome because they evaluate every possible segment design solution. In reality, the DB designer is interested in only a few "appropriate" solutions instead of all the theoretically possible results.

This paper presents an algorithm which provides an heuristic tool for designing logical DB segments. The algorithm serves as a guide because it uses "system-usage" or "importance" factors to derive one "best" design solution.

The factors can be massaged by the analyst and the algorithm re-executed to see how the design is affected by the various changes. In this way, the segment synthesis algorithm can "lead" the DB designer to more interesting results. The factors are weights which the DB analyst chooses to apply to each data element and/or the functional dependencies [1] between the elements (such as weight may be frequency of usage, size or cardinality of the fields). How the analyst chooses and applies these factors is not addressed in this paper, but it is an important topic for future development.

In summary, the algorithm presented in this paper produces logical segments for a data base using:

1) functional dependencies [1], and
2) system-usage or "importance" factors for each data element.

The functional relations and statistics will be analyzed using implication matrix technology [3] which can be carried out very efficiently by computer program. The results will be sets of system-tailored relations in the Third Normal Form [5]: simple, consistent, easy to understand, and logical structures. Other performance-oriented considerations, such as short response time and ease of maintenance, can be introduced separately after the logical design, but this subject is beyond the scope of this paper [2].

PRELIMINARY NOTIONS

Codd and Date [5,6] have discussed the validity of a relational view of data versus a network or hierarchial view of data. Therefore, the synthesis algorithm was developed based on the notion of the functional relations of Codd to derive the second and third normal forms.

A functional relation (FR) between two attributes A and B in a relation R (denoted A→B in R) exists if each value of A has exactly one value of B associated with it in relation R. An example is a personnel file in which an EMPLOYEE is assigned to exactly one DEPT. In this

case:

EMPLOYEE → DEPT.

Basically, EMPLOYEE is a unique key where it is accompanied by one DEPT data element. This concept also applies to groups of attributes; e.g., $A_1, A_2, \ldots, A_i \rightarrow B_1, B_2, \ldots, B_j$.

To represent the FRs, the implication matrix will be used. This method of FR depiction proves to be an especially useful vehicle in deriving the normal forms required for the segment design [3]. The implication matrix P is a set of m functional relations, $R_1, R_2, \ldots, R_m$ on n attributes, $A_1, A_2, \ldots, A_n$; where the matrix is mXn, the rows are labeled with the left side of the function (the key) $L_1, L_2, \ldots, L_m$ and the columns (attributes) $A_1, A_2, \ldots, A_n$ such that:

$$L_i A_j = \begin{cases} 1; & \text{if } A_j \in (L_i \cup R_i). \\ 0; & \text{otherwise.} \end{cases}$$

For example, consider the following FRs:

A→B, BC→D, AC→D.

Then, the implication matrix is:

P =

	A	B	C	D
A→B	1'	1		
BC→D		1'	1'	1
AC→D	1'		1'	1

(where 1' means this is a key attribute in the relation. The prime mark is for visual clarity only. The right side of the FR is also displayed for clarity)

The L_1, L_2 and L_3 are A, BC and AC respectively and A_1 thru A_4 are A thru D.

Delobel and Casey [1] have presented the following properties of functional relations which are easily proved:

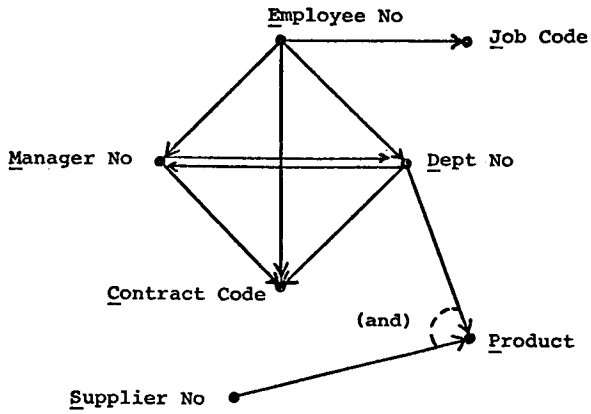
I) Transitivity: if E→F and F→G then E→G;
II) Reflexitivity: E→E;
III) Projectivity: if E is contained in F then F→E;
IV) Additivity: If E→F and E→G then E→F,G;
V) Psuedo-transitivity: if E→F and F,G→H then E,G→H;
VI) Augmentation: if E→G then E,F→G where F is any other attribute.

Using the properties of the implication matrix and the FRs, the following steps must be accomplished by the segment synthesis algorithm:

I) Execute additivity to combine all the attributes which have the same key attribute.
II) Create the set of all possible functional relations known as a transitive closure.
III) Reduce the closure to a quasi-closure based on attribute usage statistics to customize the synthesis results.
IV) Remove augmentations, from closure, to insure second normal form.
V) Remove transitivities from second normal matrix.
VI) Remove psuedo-transitivities giving the third normal form matrix.

The results will be a system-appropriate, non-redundant set of a functional relations from which all other function relations (i.e., the closure) can be derived: known as a non-redundant cover.

THE EXAMPLE



Each Employee has one Manager, Department, and Job. Each Manager has one Department and Contract. Each Department has one Manager and Contract. The Department may provide many Products; but a Dept-Supplier key identifies uniquely.

Based on this configuration a system-designer may have input the following FRs:

E→J,M,D D,S→P M→D
D→M,C D,S→C M→C

(Note- The FR D,S→C was entered by the designer through his misconception. The synthesis algorithm will detect and remove it as an Augmentation (Property VI). D→C by the D→M,C entry; therefore D,S→C is redundant.
Also, the 'usage' factors, mentioned

in the introduction, would be entered here by the analyst. The mechanics of their application to the matrix below is not presented in this paper. It is important the factors be entered and maintained for their use in Step III; but how the factors will be maintained is outside the scope.)

RESULTING IMPLICATION MATRIX

	E	M	D	C	J	S	P
E→J,M,D	1'	1	1		1		
D,S→P			1'			1'	1
M→D		1'	1				
D→M,C		1	1'	1			
D,S→C			1'	1		1'	
M→C		1'		1			

I. PERFORM ADDITIVITY

In order to reduce the number of rows to be processed in the implication matrix (and thereby the processing time) the property of functional additivity should be used. That is, every FR with the same key should be combined. This operation can be handled at function-load time to the algorithm; but can be accomplished with the initial matrix loaded.

For each functional row L_i in an m-relation, n-attribute matrix, compare the key attributes to every other L_j:

1) If every attribute A_k in L_i (i.e., *the key) is identical to every A_k in L_j, then for each non-keyed element of row L_i copy each "1" entry into the corresponding entries of row L_j. After all elements of row L_i have been moved to row L_j, delete the entire L_i row (as the L_i FR has been preserved in the L_j row.)
2) If condition 1) is not true then allow the L_i row into the matrix and go to row L_{i+1}.

Resulting matrix example after additivity step:

	E	M	D	C	J	S	P
E→J,M,D	1'	1	1		1		
M→C,D		1'	1	1			
D→M,C		1	1'	1			
D,S→P,C			1'	1		1'	1

*be careful to note the distinction between the references of "L_i" (the key elements) and "row L_i" (all elements of the relation in row i).

II. THE CLOSURE

Fadous and Forsyth [3] have suggested, with supporting proofs, a method of using the implication matrix to create a transitive closure. The process end-product is the original relations with all associated transitivities present.

1) For every two distinct rows L_i and L_j in the matrix, if every attri-A_k in L_j, the entry L_iA_k=1, then copy all "1" entries in row L_j into the corresponding 0 entries in row L_i.
2) Repeat process "1" until the matrix cannot be changed further.

Resulting matrix example after Closure Step:

	E	M	D	C	J	S	P
E→J,M,D,C	1'	1	1	1	1		
M→C,D		1'	1	1			
D→M,C		1	1'	1			
D,S→P,C,M		1	1'	1		1'	1

(where 1 indicates the attributes added as per the algorithm)

III. THE QUASI-CLOSURE

As a result of the previous Closure step, there may be identical rows; that is, $A_kL_i=A_kL_j$ (for all A_k, k=1,2,...,n), where L_i and L_j are distinct rows (whether the attribute is key or non-key is not considered in the above equality). With identical attributes, there are three conditions which the keys, of the respective rows, may assume:

1) The keys are identical - this condition cannot occur if the first (Additivity) step combined these.
2) The key attributes of L_i are a subset of the key attributes of L_j - the L_i is an augmented FR and should be removed to assure second normal form.
3) The key attributes do not relate as 1 and 2 above - these FRs are relations which have alternate keys. This is the turning point of the segment synthesis method: choosing only one of the keys of the FRs, in this class, as the major or primary key. The remaining keys should be used as possible secondary keys, if importance dictates. The primary key is chosen on the basis of "usage" by the DB system in question. Logically, there may be more than

one key for a relation; why not pick the one the system is most likely to use? The remaining possible keys will be of lesser (or maybe no) importance to a DB designer. It is true different keys will produce different non-redundant covers (or segment designs); but this kind of all-possibilities operation requires much computing time and may not be desired by the designer.

The result of this step is a reduced and specialized closure. It results from deleting all L_j rows which $L_iA_k=L_jA_k$ (for all k where k=1,2...n).

Matrix and system-usage value before quasi-closure:

	E	M	D	C	J	S	P	Usage Value
E→J,M,D,C	1'	1	1	1	1			25
M→C,D		1'	1	1				66
D→M,C		1	1'	1				43
D,S→P,C,M		1	1'	1		1'	1	78

After the quasi-closure:

	E	M	D	C	J	S	P
E→J,M,D,C	1'	1	1	1	1		
M→C,D		1'	1"	1			
D,S→P,C,M		1	1'	1		1'	1

(the 1" indicates a possible secondary key)

The M→C,D was chosen over D→M,C because M was "more used" as a key than D.

IV. AUGMENTATION REMOVAL

The second normal form is defined as a normalized relation in which all non-key attributes are fully dependent on the entire key of that relation. So, if R_1:E→G and R_2:E,F→G are functional relations, then R_2 is not in second normal form as G is not fully dependent on E and F, (because R_1 states that G is fully dependent on only the attribute E). In other words, functional augmentations are not allowed. After this step, the closure will become a matrix in second normal form.

The matrix process is very straight forward:

For each function row L_i in an m-relation, n-attribute matrix, compare the key attributes to every other L_j:

1) If every attribute A_k in L_i are contained in the key attribute of any L_j then delete all non-keyed attributes, the 1 entries, which are contained in the L_i. If the L_j becomes a relation with keys only, delete the L_j entirely.
2) If condition 1 does not hold, go on to next comparison.

After augmentation removal, the matrix is:

	E	M	D	C	J	S	P
E→J,M,D,C	1'	1	1	1	1		
M→C,D		1'	1	1			
D,S,→P			1'			1'	1

The M and C were removed from the the last FR because they were augmentations. The matrix now represents relations in second normal form.

V. TRANSITIVITY REMOVAL

After the augmentations have been removed from the closure, the transitivities are next to be removed. Since all relations now contain every possible transitive "chain" (as per the other FRs given in the matrix) as is possible for that relation, transitivity removal becomes fairly straight forward:

1) Start with the smallest function(s); that is, where the cardinality of L_iA_k is least. These FRs are, by the property of closure, the most primitive and elmented. They do not contain transitivities as this implies a smaller FR must exist (because an FR must be larger than the transitive-causing function in order to contain it).
2) Compare each of the chosen-cardinality of FRs where one is say row L_i, to all other rows L_j.
   a) If L_iA_k is contained by L_jA_k (for all A_k, where k=1,2,...,n) the row L_j contains a transitive FR (in this case row L_i) and transitivity-causing function L_i must be removed from the relation in row L_j. This is accomplished by resetting (setting to 0) all attributes in L_iA_k where $L_iA_k=L_jA_k$ (for all non-keyed A_k of L_j or L_i, where k=1,2,...,n. The "non-keyed" specification is important because if both rows L_i and L_j have the same A_k as a key, then the situation is psuedo-transitive and will be handled in step VI. If one or

b) If condition A is not true, proceed to examine the next chosen small FR and go back to 2.a.

3) Go to next highest cardinality of L_iA_k.
   a) If this is the highest cardinality, we are done and the matrix is transitivity free.
   b) If condition 3a is not true go back to process 2.

After transitivity removal the matrix is:

	E	M	D	C	J	S	P
E→J,M	1'	1			1		
M→C,D		1'	1	1			
D,S→P			1'			1'	1

The C and D were removed from the first relation to avoid transitivity because of the second FR.

VI. PSEUDO-TRANSITIVITY REMOVAL

The last step, to finally achieve third normal form, is to remove the pseudo-transitivities (P-T). The process is similar to the previous step because it also removes subset information from larger functions. However, the process is more involved structurally:

1) There may be a row L_i which was a pseudo-transitive FR even before the closure. This row(s) was removed in the Pseudo-Closure step. It can be proved that one of the L_i rows which is a basis for the P-T becomes identical to the P-T relation, row L_j, in the closure step; that is $L_iA_k=L_jA_k$ (for all A_k, k=1,2,...,n where n is the total number of attributes). The only difference is that the key attributes of L_i are a subset of the key of L_j -- an augmentation which is removed in Step III.

   Example (after Closure step and not from main example):

Orig. FRs	E	F	G	H
E→G	1'		1	1
E,F→H	1'	1'	1	1
E,G→H	1'		1'	1

   (Note - by step III, E,G→H will be removed as augmentation.)

2) Although the original P-T was removed, its "effect" was captured in the Closure step: in the example above, the H is turned-on in E→G. And the P-T effect of the smaller FR was passed onto the larger FR in the Closure step, as it should: this is the G in E,F→H.

3) Enough of examples; to assure that the P-T property is removed: given any two distinct functions, rows L_i and L_j, if the (key) attributes of L_i are a subset of the key of L_j, the non-key elements of rows L_i and L_j cannot intersect (i.e., L_iA_k cannot = L_jA_k for all non-keyed A_k in row L_i and L_j). Any equality would promote P-T redundancy.

4) To remove the redundancy:
   a) If, for any non-key A_k, $L_iA_k=L_jA_k$ (where the L_i key is the subset key) and attribute L_iA_k was originally provided by the designer (i.e., L_iA_k=1; not =1), then the attribute L_jA_k should be removed.
   b) If 4a is true except the subset function attribute has L_iA_k=1 (i.e., is not "original"), then the attribute L_iA_k should be set to zero.

Results of P-T removal on example above:

	E	F	G	H
E→G	1'		1	
E,F→H	1'	1'		1

(the P-T redundant H and G are removed.)

CONCLUSION

The resulting segment design from the above algorithm and main example is:

S_1 (Employee No, Job Code, Manager No)

S_2 (Manager No, Dept No, Contract Code)

S_3 (Dept., Supplier No, Product)

where the primary key is underlined once and the possible secondary key is underlined twice.

Although there are four possible synthesis solutions, the three resulting segments (or relations) were chosen as per system-usage statistics.

General timing consideration yields the following "worst possible" case (each of the algorithm steps are separate factors):

$$n\ m \log m + n\ m^3 + n\ m^2 + n\ m^2 + n\ m^2 \leq n\ m^3 + 4\ n\ m^2 .$$

(where n is the number of attributes; and m is the number of FRs)

BIBLIOGRAPHY

[1] Delobel, C. and R.G. Casey; "Decomposition of a Data Base and the Theory of Boolean Switching Functions"; IBM J. of Res and Dev.; Vol. 17 #5 (Sept 1973).

[2] Wang, C.P. and H.H. Wedekind; "Segment Synthesis in Logical Data Base Design"; IBM J. of Res. and Dev.; Vol. 19 #1 (Jan. 1975).

[3] Fadous, R. and J. Forsyth; "Finding Candidate Keys for Relational Data Bases"; Proc 1975 ACM-SIGMOD.

[4] Bernstein, P.A., S.R. Swanson, and D.C. Tsichritzis; "A Unified Approach to Functional Dependencies and Relations"; Proc 1975 ACM-SIGMOD.

[5] Codd, E.F., "Further Normalization of the Data Base Relational Model"; Courant Computer Science Symposium, in Data Base Systems; Prentice-Hall, 1972.

[6] Date, C.J.; Introduction to Data Base Systems; Addison-Wesley; 1975.

ACM SEEKS NOMINATIONS FOR ANNUAL GRACE MURRAY HOPPER AWARD

The Association for Computing Machinery is seeking nominations for its Grace Murray Hopper Award, given each year to the outstanding young computer professional selected on the basis of a single recent major technical or service contribution to the computer industry. In order to qual ify, candidates must have been 30 years of age or less at the time the qualifying contribution was made.

The Award will be presented at the opening session of the Association's Annual Conference on December 4, 1978, in Washington, D.C. The Award is in the amount of $1,000 donated by the Univac Division of Sperry Rand, and is accompanied by a certificate.

"Clearly, the award that had the biggest effect on my life," said Grace Murray Hopper recently, "was the first one I received, when I was young It told me that someone was looking at my work and evaluating it. It encouraged me tremendously; it was like a pat on the back. That is what I hope this ACM award will do. I hope it will lead people to look at and appreciate all of the fine work the young people in the computer field are doing -- in business data processing, in personal computing, in medical applications of computers, and so on. I hope it will say to the young people, 'You have done a good job, you are in the right field, keep going!' Further, I hope this will encourage our senior people to look at the work of their juniors and recommend that they be rewarded for their achievements. If the Award does that, I will be very happy."

While the Award is given to the outstanding young "computer" professional, emphasis for the 1978 award will be placed on contributions in the fields of business data processing and personal computing. The Committee felt that these fields have not been adequately rewarded for outstanding contributions in the past.

The last three winners of the Grace Murray Hopper Award are: Edward A. Shortliffe, for his development of a program that consults with physicians about diagnosis and treatment of infections; Allen L. Scherr, for his pioneering study in quantitative computer performance nalysis; and George N. Baird for the development and implementation of the U.S. Navy's COBOL compiler evaluation system.

Nominations, which may be made by the nominees themselves, should be sent to:

Richard G. Canning
Chairman, ACM Grace Murray Hopper Award Committee
925 Anza Avenue
Vista, California 92083

In order to be considered for the 1978 Award, nominations should be received by Mr. Canning no later than June 30, 1978.

Please include the following information:

1. Name, address, and phone number of the person making the nomination.
2. Name, address, and phone number of the nominee.
3. A statement (200 to 500 words) on why the candidate deserves the Award describing the contribution.
4. The date of birth of the nominee and the date on which the qualifying work was completed.