HORIZONTAL DECOMPOSITION TO IMPROVE A NON-BCNF SCHEME

A. L. Furtado
Pontificia Universidade Catolica do R. J.
Brasil

1. Introducing the problem

In general, converting a relation into Boyce-Codd normal form (BCNF) [Date;Fagin] is advantageous because, all dependencies being dependencies on keys, their enforcement can be done through the regular key-handling features, supported by most DBMSs.

However, a problem (to be explained shortly) arises in the situation where a non-key set of attributes determines part of the key. Let R be a relation and X, Y, Z sets of attributes of R, with XY being a key in R, and the dependencies:

XY -> Z

Z -> Y

Figure 1 illustrates this situation, letting the subscripted small letters denote distinct instances of the respective sets of attributes.

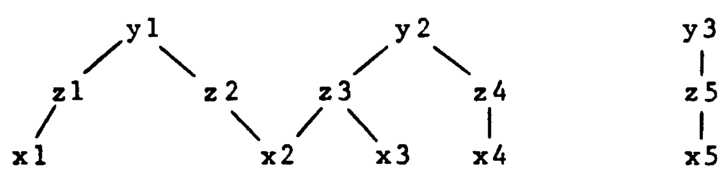

Fig. 1

The figure suggests a three-level structure that is "almost" a hierarchy. At the bottom level, each X-value can be linked to at most one Z-value in each Y-rooted "tree". For instance, the presence of the tuple (x1,y1,z1) excludes the presence of, say, (x1,y1,z2), because of the XY -> Z dependency. For reasons to be discussed in the sequel, it is useful to express this fact as follows:

Proposition 1: Let R be the relational scheme above. Then, if R is first partitioned by Y and then projected on XZ:

a. the blocks, even after the projection, constitute a partition (i.e., no two blocks have any (xi,zj) tuple in common);

b. in each block the dependency X -> Z holds.

Since R is not in BCNF, one would normally decompose it into R1(X,Z), which is an all-key relation, and R2(Z,Y), where Z -> Y. Notice that the XY -> Z dependency is somehow lost, the attributes involved being scattered between R1 and R2. This goes counter to the declared aim of BCNF decompositions, which should allow us to turn dependencies into key dependencies, while, of course, preserving all of them.

In the next section we propose a solution for alleviating this intrinsic difficulty, using proposition 1. The discussion will be entirely centered on the example [Date] of a relation R with attributes S (Student), C (Course) and T (Teacher), with the dependencies SC -> T, T -> C. Figure 2 shows a valid state of this relation.

R

S	C	T
Peter	Math	Euler
David	Computing	Turing
Mary	Computing	Von Neumann
Jane	Computing	Turing
Ariel	History	Durant
David	Math	Fermat

Fig. 2

2. The proposed decomposition

Whilst most of the discussion of decomposition strategies has been based on projection ("vertical" decomposition), some thought has also been given to decompositions by restriction (or some form of "horizontal" splitting) [Fagin; Smith and Smith]. In a similar vein, partitioning has been introduced as a relational algebra operation [Furtado and Kerschberg]. Extensions to the relational data model have been proposed [Chang; Codd], accomodating the idea that, as a consequence of horizontal decomposition, the same tokens could be taken alternatively as attribute-values or as names of the relations (or blocks) resulting from the decomposition; also, new operations are included to support this feature.

Our strategy uses horizontal decomposition and therefore assumes the availability of the attendant data definition and data manipulation features, in the style of one or another of the proposals above.

In terms of the academic data base example, we take R1 as a set of blocks (or relations) resulting from first partitioning R by C, and then projecting the blocks on the attributes ST. According to part b of proposition 1, the dependency S -> T holds in each block of R1. We let relation R2 be the projection of R on TC (as in the conventional decomposition), with T -> C.

Figure 3 illustrates the proposed decomposition.

R1

Math

S	T
Peter	Euler
David	Fermat

Computing

S	T
David	Turing
Jane	Turing
Mary	Von Neumann

History

S	T
Ariel	Durant

R2

T	C
Euler	Math
Fermat	Math
Turing	Computing
Von Neumann	Computing
Durant	History

Fig. 3

The reader may like to compare this to figure 1, which is a graphical version of the same data base state (the three levels in figure 1 corresponding, respectively, to courses, teachers and students).

The important novelty in this strategy is the key dependency S -> T, holding in each block of R1, induced by the partitioning by C. Thus the original dependency SC -> T in R now reads: S -> T within each C-block.

For reconstituting R from R1 and R2 we can take the union (U) of the blocks of R1 and then the natural join (*) of the result with R2. An interesting alternative is made possible by the following distributive property:

Proposition 2: Let S1, S2, ... , Sn be union-compatible relations and V be any relation. Then

(S1 U S2 U ... U Sn) * V = (S1 * V) U (S2 * V) U ... U (Sn * V)

Proposition 2 indicates that we can first perform the joins of each block of R1 with R2, which can be done in parallel. At the end the union is simply the collection of the results of the joins, because part a of proposition 1 ensures that we do not have to check for duplicate tuples (the results of the joins are pairwise disjoint).

In general, horizontal decompositions tend to be useful in practice when the following requirements are met:

a. The set of blocks is relatively stable. Referring to the example, it is not uncommon in the academic world that the same courses be offered over a number of semesters.

b. The cardinality of the set of blocks is relatively small. If the function underlying the dependency T -> C is surjective, i.e. if there is at least one teacher per course, then the set of courses cannot be larger than the set of teachers; also, in practice, there are usually more students than courses.

c. The cardinalities of the blocks themselves are approximately of the same order of magnitude, thus providing a balanced way to segment the information. Again, sizes of classes under each teacher tend to fall between close lower and upper limits.

Other examples of the same situation appear to be amenable to the present strategy, one of which is a relation involving Cities, Streets and Zip_codes, where:

City Street -> Zip_Code

Zip_Code -> City

Through this second example, one sees that the horizontal decomposition (in this case, partitioning the relation by city) may, in a sense, mimic the usual "manual" procedures. For a large city, it is customary to print booklets giving the Zip codes of streets within the city.

3. Views and operations

Another measure of the adequateness of schemes is their use in the most frequent or more important (according to some criterion) operations that can be anticipated. Such operations do not have to be confined to any specific relation or block, but will involve an arbitrary function of the data base, which is called a view in the data base terminology [Date].

For our academic data base example we shall concentrate on the update operations given below, where the small letters s, c, t appearing in the argument lists denote the attributes from which the arguments are taken:

enroll(s,c,t) - enrolls s in c under t

drop(s,c) - s drops c

transfer(s,c,t) - a combined drop/enroll: transfers s, already enrolled in c, to a different teacher t of c

appoint(t,c) - assigns t to c

cancel(t) - cancels the present appointment of t

We shall examine enroll in more detail, making a quick reference to the other operations. The operation can be executed along the following stages:

a. Check in R2 if t really teaches c. This is easy, since t is a key in R2.

b. Find in R1 the block named by c.

c. Check if some tuple (s,-), where "-" stands for an arbitrary value, already exists in the block. This is also easy, because s is a key in the block.

d. Insert (s,t) in the block.

The drop operation affects the appropriate block in R1, and the T-value does not have to be indicated. It seems reasonable to require that an appointment cannot be cancelled until one has decided what to do (drop or transfer) with each student taking the course under the teacher involved; thus a precondition for the cancel operation is that there be no tuple (-,t) in the respective block of R1 - which is a costlier search, perhaps requiring a secondary index on teachers teaching each course.

In some cases, a slightly different decomposition may be convenient, introducing a "level of indirection", which, as such devices usually do, makes some information (the enrollment of students in classes, in the example) "relocatable". In R1 we could have as attributes S and O (Offering or section, i.e. a specific instance of a course), instead of S and T. In turn, R2 would have O, T and C, noting that both O and T are keys. Now an operation like

replace(o,t) - replace the teacher in charge of o by t

will only affect R2.

After any decomposition of the original relation R we may want to update the resulting relations in ways that will no longer permit the expected reconstitution of R. For instance, executing

appoint(Codd,Human Relations)

and trying to reconstitute R immediately after that will result in the same contents that we had prior to the update. This is a consequence of the conventional definition of natural join; the information of the new appointment will be lost unless at least one student is enrolled in Human Relations (in a newly created block of R1). The outer join proposed in [Codd] would preserve the appointment information even without any previous enrollments.

An operation that would seem to be desirable is reappoint(t,c), which would be a handy combination of cancel/appoint, moving t from whatever course he presently teaches to another course c. At first glance, this operation seems (syntactically) admissible, since even the reconstitution of R by natural join can still be done without losing any tuple from R1 or R2. However, its meaning would be that students will follow their teacher, rather than keep their enrollment in the course initially chosen. But this would be very unusual, since the relationship between students and courses is normally stronger than that between students and teachers. We regard this as a case of semantical connection trap. So, instead of defining a new operation reappoint, it makes sense to use cancel followed by appoint, because, as said, the preconditions for cancel require the previous execution of drop or transfer for each student affected.

4. Conclusion

Although other solutions, catering to other relevant objectives , may be devised, we claim that the proposed strategy has the advantages of relying on key dependencies to a large extent and of incorporating only a minimum of redundancy (what is redundant is the double appearance of Courses as an attribute in R2 and as a set of block names in R1).

Also, there are cases where horizontal decomposition may be useful even at the physical storage level, as a criterion for a balanced segmentation of large files, and as a distribution strategy in a distributed data base.

We left purposefully vague the specific way to implement "blocks", merely pointing to different references in the literature. Matters of data model, operations in the DBMS chosen and physical level resources will determine solutions ranging from a conventional inverted file (on courses, in the example) to dynamically created relations named after the attribute values.

Acknowledgements

The author is grateful to R. Fagin and M. A. Casanova for useful suggestions.

References

C. L. Chang - A hyper-relational model of data bases - IBM S. Jose T.R. RJ-1634 (1975).

E. F. Codd - Extending the database relational model - ACM/TODS, vol. 4, n. 4 (1979) 357-434.

C. J. Date - An introduction to database systems - Addison-Wesley (1977).

R. Fagin - Normal forms and relational database operators - Proc. ACM/SIGMOD (1979) 153-160.

A. L. Furtado and L. K. Kerschberg - An algebra of quotient relations - Proc. ACM/SIGMOD (1977).

J. M. Smith and D. C. P. Smith - Data abstractions: aggregation and generalization - ACM/TODS vol. 2, n. 2 (1977) 105-133.