A Taxonomy of Data Definition Languages

by: T. William Olle

June 1969

This topic of file definition techniques or data description languages (whichever you prefer to call it) is one very recently recognized. This is why we have this Special Interest Committee in the ACM and why USSAI has formed an ad hoc sub-committee to investigate the problem to see what there is to investigate. At RCA we have been rather worried about data definition and have been doing some thinking about it. Basically, the problem is not "what is a data definition language?" but "what are data definition languages?" They fell into a number of classes and one has to try to establish some sort of taxonomy, so that there is a framework for talking about them.

I would like to equate the study of data description languages with the study of statics in classical mathematics. We are really talking about something which is static. Data is stored in a memory. Admittedly it is moved about, but we are trying to study the problem in a static state. Some people may question the validity of this, and maintain that it is a dynamic problem. However, you can freeze the time element and study it in that way which is what we have got to do, because otherwise we have a tiger by the tail and we shall not have any chance at all. Let me give you my classes of language.

A.	CONCEPTUAL
B.	LOGICAL
C.	PHYSICAL
	C1 OWN
	C2 OTHER
D.	STORAGE REPRESENTATION OF C

Figure 1: Five classes of Data Definition Languages

Class A, the top level, I call the conceptual level. The second class, class B, I call logical - the outside world of the data. The third one is physical, and I found it convenient to split this into two--own and other--and I will try to explain those later. Finally, class D, a storage representation of C -- namely something which is encoded in machine form. Whatever C is, C is something that people look at. D is the storage representation of C.

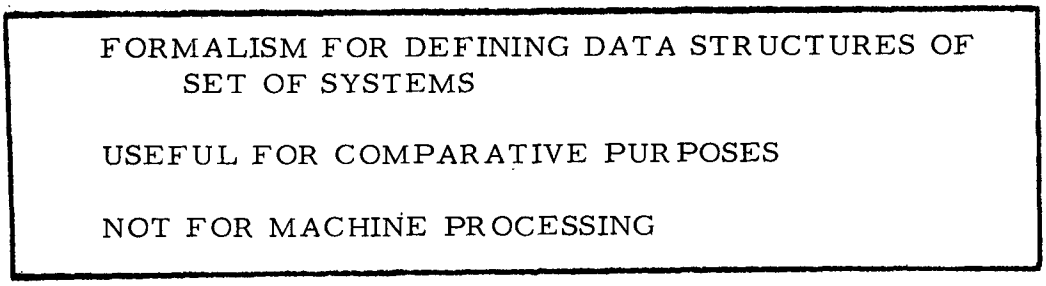

Figure 2: Class A. Conceptual

The first class - conceptual--is one that we should have had in the recent work on the CODASYL Systems Committee. We have been studying a set of generalized data base management systems, and one of the important sections of the survey recently released was data structure. We looked at the data structure of nine different systems as they are described in nine fairly radically different sets of terminology and tried to reduce them to a common whole. This was very difficult, and it would have been easier if we had had some formalism that we could have used for defining the data structure of each system. One could define the data structure capability--the degrees of freedom--of a system in a standard formal way. It would be preferable if it was mathematical but at least it should have some flavor of mathematical exactness. This would be useful for comparative purposes. It is very hard to compare the data structure capabilities of system A with data structure capabilities of System B.

There is one final point on the conceptual class. It is not necessarily for machine processing and we should not have a class A data definition language and feel that this will have to be entered into the computer for any particular reason.

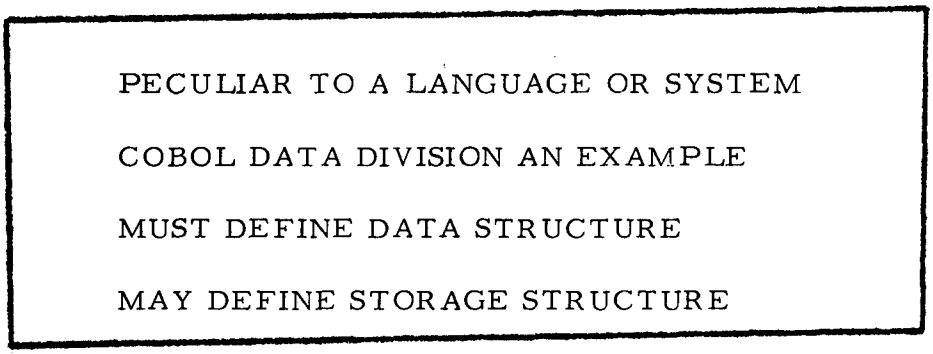

Figure 3: Class B. Logical

This level, class B, is the class where all the work has been going on in the past 20 years. A data definition language on a logical level is peculiar to a given language or system. A very good and certainly the most widely known and widely used example in this class is the Data Division of the COBOL language. It has certain constraints and certain degrees of freedom. The COBOL Data Division statements, as written, I would place in class B.

Next we get into the fuzzy area between B and C. The logical language must define the data structure, the data structure being the outside world of the data. It defines the data as the user sees it. It may define the storage structure. Don Hatfield used this term storage structure, which is a very useful term to have. In the type of system evolving nowadays, there is a difference between data structure in the outside world and storage structure, which is how the data is stored in memory and which the user of the system does not necessarily have to comprehend. It need not be apparent to the user on any of the levels within the structural hierarchy. In some systems--FORTRAN for example-- you have a FORMAT statement, in which you are defining the data structure and at the same time defining the storage structure because there is no difference.

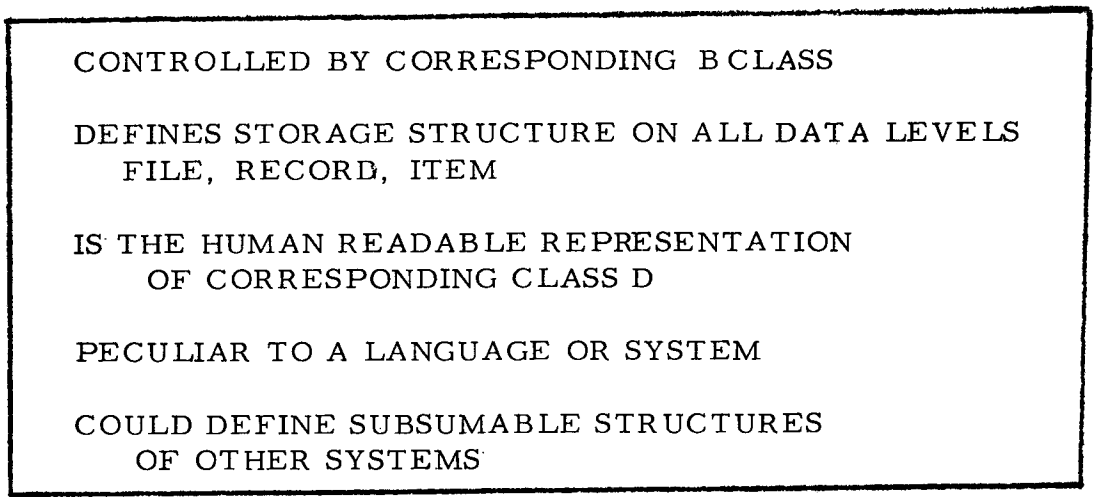

Figure 4: Class C. Physical

Class C should be controlled by a corresponding B class. A physical class data definition language cannot exist in abstracto--it has to be under some corresponding class B language. As I have already pointed out, some of these are the same; sometimes it is very hard to see what the difference is and why you need the difference. A storage structure definition language or a language of class C, physic 1, should define the storage structure on all levels. The good old terms--file, record, and data item, as used in COBOL--are good terms to use. They are particularly of value in communicating with people--the wide mass of people who tend to be confused if you start babbling about attributes and properties and using all sorts of weird terminology that you have invented.

These are terms that one should use, because they communicate effectively. We need other terms as well, for instance we need a term above the file level, which it is very useful to call the data base. We need a term below the record level and above the item level which it is very useful to call a group, a group item for instance, or a repeating group. A storage structure definition should define the data on all these levels.

There is such a thing as a file-level storage structure. File-level storage structure definition would imply that we would be able to define IBM's indexed sequential, RCA's inexed sequential, etc. Their storage structures are rather different, even though they have the same name. This is really a goal for our class C physical data definition language.

The next point is the human readable representation of corresponding class D. I am now trying to tie in across the boundary. I will come to this when I discuss class D. Right now a storage structure definition is peculiar to the language or system. COBOL has its storage structure definition, some of which is implied in the implementation of the COBOL compiler, some of which is implied in the Data Division. It would be an advantage if we could develop a language that could define subsumed structures of other systems. Some systems have less degrees of freedom in their data structure and hence in their corresponding storage structures than others.

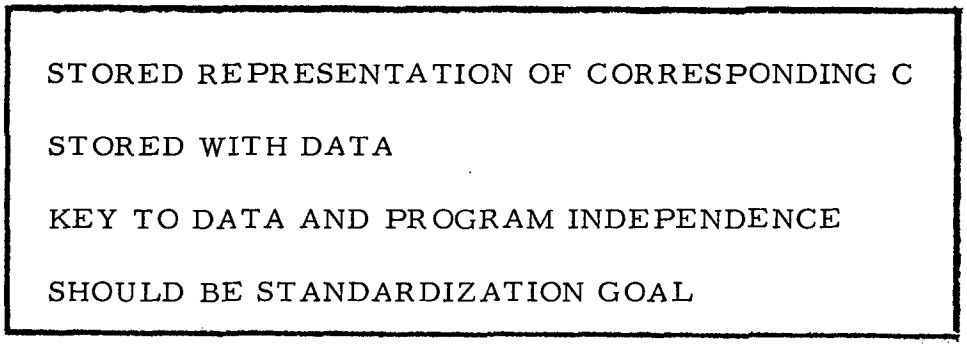

Figure 5: Class D. Storage Representation

Finally, we come to class D - storage representation. Class D is the stored representation of the corresponding class C. It is something that is not meant for human digestion, but rather for machine digestion. It is an ironic fact the way languages have evolved over the last fifteen years. For some reason or other we have always stored the data definition with the program. This has typically been something on level B and C combined. In the systems that are evolving presently, GIS, Mark IV, TDMS, and our own system UL/1, the definition of the data is stored with the data. It is not stored in the program, or in what is the equivalent of the program.

This means, of course, it does not have to be input every time a function is performed on that set of data. It would be an advantage if we could always store that representation of the data definition

with the data and this has to be the holy grail we are seeking. This class C is what we would really like. We would like to have some standard way of storing data structures so that when you take the movable storage medium, either a disc pack or a tape, and plug it in, the first block read is the class D Data Definition. This is the key to data and progamming independence. We know quite a bit about this particular matter and would like program and data to be independent. They are independent, but we have to have some way of having a class D storage representation of the data and, as I have already said, this should be the goal for standardization efforts.

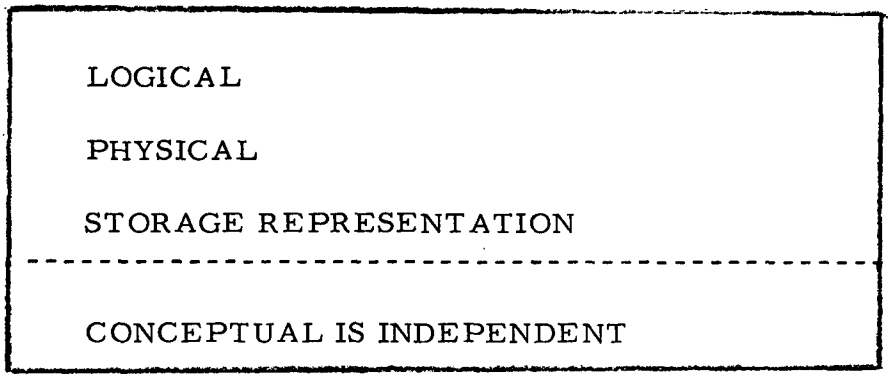

Figure 6: Proposed Path

How do these all fit together? The conceptual one is a fairly independent thing. It is an interesting thing to develop such a formalism for comparing data structures. Attempts have been made, and I am not sure how successful they are. On the Systems Committee we did not find anything particularly suitable for our purposes. The other three--logical, physical, and storage representation--hang together. I think you have to go down that path, in that order--logical, physical, and storage representation. You cannot go up and you cannot start in the middle. You must go down.

Thank you for your attention.