Production Rules for General Database Users

Levent V. Orman
Cornell University

Production rules are suggested and developed as a standard notation for the general database users. They are shown to be appropriate for a variety of data models and a variety of database activities. A production based data language (PDL) is introduced and recommended as a standard notation for several data models and many database activities.

The expressiveness of the language is demonstrated through examples spanning a wide range of activities. The advantages of a single standard notation are argued to be the minimization of user effort in learning notations, the elimination of the need for conversion from one language to another whenever multiple models or activities are involved, the development of general principles cutting across data models and activities, and the transfer of knowledge and insight developed in one context to other environments. Experience with production rules as an end-user development tool is reported.

Data models are abstract structures that isolate the database users from the detailed physical data structures employed by database management systems. They hold the promise of an environment where general database users can access and maintain their data directly. A variety of general-user oriented languages have been designed and implemented, usually in the context of either the relational or the functional data models, with moderate success. Most of these languages have been directed to the problem of data retrieval (Boyce 1975; Chamberlin 1976; Orman 1984; Shapiro 1979) although some isolated attempts were made in the arena of simple application development (Jarke 1984; Schmidt 1977; Zloof 1977) transaction design (Mylopolos 1980) constraint specification (Buneman 1979; Hammer 1977) and derived data (Schmidt 1977; Shapiro 1979). However, almost invariably, the simplicity and the elegance of the retrieval languages are lost when they are extended to cover any of these areas important to the general users (Alban 1985; Schmidt 1977; Shapiro 1979) and consequently they are either ignored or covered by separate independent notations developed for each of those activities. This approach leads to a proliferation of languages each appropriate for a different task (Buneman 1984). Moreover, the general-user oriented languages have remained ad-hoc and specific to particular data models. A language that is appropriate for multiple data models and all areas of user activity is of great importance both in facilitating a general user interface, and in providing a
common framework to study and compare a variety of activities and a variety of approaches taken by different data models to those activities. The first step in developing such a common language requires developing a correspondence among the major data models. The following section briefly introduces three major data models, the functional, relational and the semantic data models, and reduces all three to the functional model to establish correspondence. The consequent section proposes production rules as a database language for the general user. A specific language based on production rules, extensive use of variables, and a two-dimensional layout of specifications is introduced. The rest of the paper is devoted to a demonstration of the generality of the language by extending it to specify not only queries, but constraints, transactions, procedures and derived data for all three data models, all within the same framework. Some experience with the language as an end-user development tool is also reported.

PDL differs from database query languages (Boyce 1975; Chamberlain 1976; Orman 1984; Zloof 1977) in its completeness. Query languages do not ordinarily provide an application programming environment except for a few ad-hoc functions such as SUM and AVERAGE to meet common statistical needs. Some languages such as SQL (Chamberlain 1976) and QBE (Zloof 1977) provide simple arithmetic operations but fail to integrate them with the data model through transaction design and derived data. Application programming languages on the other hand generally fail to support data models. The few that do such as PASCAL/R (Schmidt 1977) and GALILEO (Albano 1985) create two-tiered systems where the data models of database systems and the data structures of the programming languages remain separate, and the language merely provides an interface to move the data from relations of the data model to the arrays, records, lists or trees of the programming language. A notable exception is logic programming (Clocksin 1981) where the n-ary relation (the building block of the relational data model) is the major data structure. However, the unifying potential of this choice is not realized, since the n-ary relation proved inadequate as a general data structure and all logic programming languages resort to the employment of lists as an auxiliary data structure. Consequently, data have to be moved from relations to lists for manipulation, and back to relations for storage. The resulting environment has the same disadvantages as two-tiered systems. PDL is unique in its use of functions as the only data structure throughout the application development environment. A functional data model is employed, and all data manipulation is performed directly on functions which constitutes the only data structure. The resulting novel approach eliminates all data restructuring. Its completeness follows from the completeness of functional programming languages (Orman 1988) since arbitrary functions can be defined in terms of the given primitive functions, and its data structure ‘function’ is a generalization of lists. The implementation of PDL on top of a functional programming language is straightforward and a LISP implementation is underway.

**Data Models as Functions of PDL**

The functional model of data consists of functions defined on data sets (Shipman 1981). Each function is a set valued mapping from zero, one or more data sets (called arguments) to another set (called range) and characterized by a function name in addition to the names of its arguments and range. Each function identifies a logical relationship between the argument and the range data sets. A data set is a named set of objects. Real world entities in addition to the character strings and numbers used to describe those entities are treated as database objects and they are grouped into abstractions called data sets.

Example 2.1: A university environment can be modelled using the following functions where a function F from the arguments $A_1, ..., A_n$ to the range R is denoted by $F(A_1, ..., A_n) \rightarrow R$ or $F(A_1, ..., A_n)$ when the range is obvious:

- NAME(INSTRUCTOR)
- PHONE#(INSTRUCTOR)
- SALARY(INSTRUCTOR)
- DEPARTMENT(INSTRUCTOR)
- COURSE(INSTRUCTOR)
- STUDENT(INSTRUCTOR)
- COURSE#(COURSE)
- INSTRUCTOR(COURSE)
- TEXT(COURSE)
- STUDENT(COURSE)
- TEXT(COURSE, INSTRUCTOR)

Note that functions are allowed to have identical names as long as they are defined on different arguments and hence can be distinguished from the context.

The relational model of data is composed of relations. Each relation in turn can be viewed as a collection of data sets and functions defined on those data sets. A relation $R(A_1, ..., A_n)$ is viewed as a collection of $n+1$ data sets and a large number of functions representing all the domains and all the implied mappings among the domains of the relation. The $n+1$ data sets $A_1, ..., A_n, R$ contain one data set corresponding to each domain and an additional data set containing all the entities represented by the relation. The relation $R(A_1, ..., A_n)$ corresponds to $n$ single valued functions $A_i(R), ..., A_n(R)$ mapping the entity set to each one of the domains, plus a number of functions that are implied by these $n$ functions. This collection represents every possible mapping among the domains $A_1, ..., A_n$ implied by the relation $R$, and they are denoted by
Related Content

Effectively and Efficiently Designing and Querying Parallel Relational Data Warehouses on Heterogeneous Database Clusters: The F&A Approach
www.igi-global.com/article/effectively-efficiently-designing-querying-parallel/76665?camid=4v1a

Improving Sequence Diagram Modeling Performance: A Technique Based on Chunking, Ordering, and Patterning
www.igi-global.com/article/improving-sequence-diagram-modeling-performance/100404?camid=4v1a

Selecting a Database Partitioning Technique
www.igi-global.com/article/selecting-database-partitioning-technique/51123?camid=4v1a

A Database Service Discovery Model for Mobile Agents
www.igi-global.com/chapter/database-service-discovery-model-mobile/7926?camid=4v1a