"Data Base Systems Helps Management Help Itself."

Computer Decisions
October, 1979, page 86.

"The problem of database representation is complicated by the fact that no single model of reality may be appropriate for all users and problem domains. The properties which are considered relevant and the mechanism by which they are most naturally referenced vary across differing world views."

David D. Shipman

The Functional Data Model and the Data Language

DAPLEX [SHI81]
5.1 Introduction

In the previous chapter, we have discussed a proposed Access Control Mechanism for a DHDBMS. This access control mechanism could have been tested by implementing it on a real life case or on a prototype but it was not feasible because of lack of availability of resources (both in terms of time and manpower), tools and environment required for its implementation. The first and foremost requirement was to have an integrated heterogeneous database environment. This environment was not available. It could have been created but to create such an environment, we needed a Programming and Data Manipulation Language like DAPLEX [SHI81]. As the data modelling capabilities of DAPLEX incorporates the data modelling capabilities of hierarchical, relational, and network models, it could have been used to define Global Conceptual Schema (Superview) and then mappings could have been defined between global data objects and local data objects. We did not have access to such type of facility.

We also did not have access to any prototype DHDBMS, otherwise the proposed access control mechanism could have been tested by incorporating this feature in a prototype DHDBMS. We have already discussed in Section 2.3 that there are many important issues in designing a DHDBMS such as Query Language, Query Processing Algorithm, Data Structure, Data Model, Concurrency Control, Directory
Management and Recovery Mechanism etc. It was humanly not possible for an individual in a limited time to develop a worthwhile prototype DHDBMS otherwise the access control mechanism could have been implemented and tested on this prototype DHDBMS. Such type of systems are developed with team efforts. If resources would have been available, a small prototype DHDBMS based on a General DHDBMS Architecture (discussed in Section 2.5) could have been developed. Also it should be kept in mind that each design issue mentioned above is itself a full fledged research area as of today. Therefore the proposed access control mechanism could not be implemented and tested in a practical manner. However, in this chapter, we briefly discuss the implementation aspects.

5.2 Implementation Methodology

We have discussed the functional structure of a DHDBMS in Section 2.6. This functional structure conforms to the General DHDBMS Architecture discussed in Section 2.5. The functional system is divided into four functional components: (1) Global Functional System, (2) Communication Network System, (3) Local Database Interface and (4) Local Functional System. The functions of all these components have been discussed in Section 2.6. To implement a DHDBMS, the first three functional components need to be developed. Then these components will have to be integrated with the Local Functional Systems which are already operational. The
various modules which need to be developed to perform different functions are discussed below.

5.2.1 Global Transaction Parser

The Global Transaction Parser does lexical and syntax analysis of a global query. A standard parsing technique can be used for parsing the query. The Global Transaction Parser calls the Access Checker right after the successful completion of parsing to perform appropriate authorization checks.

5.2.2 Access Checker

The Access Checker does authorization checks and helps in allowing only authorized users to access the data. It receives the parsed query from the Global Transaction Parser and compares the access rights required to process the parsed query with the access rights available to the user (given by the DBA). If the access rights required to process the parsed query are available to the user then it calls the Global Transaction Decomposer to decompose the query; otherwise it rejects the query and sends an appropriate message of authorization failure to the user through the Global User Interface.

5.2.3 Global Transaction Decomposer

The Global Transaction Decomposer replaces view
names in the query by global data object names, adds required qualification clauses (if any) from the view definition; and then breaks the query into a set of subtransactions which will be processed at different sites. For each data object referenced in the query, the Global Transaction Decomposer finds the site where the data resides. The Global Transaction Decomposer uses the global catalogue to retrieve the location of data objects that appear in the parsed query. The substransactions are sent to the Global Transaction Optimizer.

5.2.4 Global Transaction Optimizer

The Global Transaction Optimizer accepts substransactions and generates a set of precendance graphs and selects feasible graphs with minimum cost. It passes the selected feasible graph to the Global Execution Plan Generator.

5.2.5 Global Execution Plan Generator

The Global Execution Plan Generator uses the information from the Global Transaction Optimizer to produce the global execution plan. The global execution plan contains the following information:

i) database objects to be accessed by each subtransaction,

ii) commands and operations to be performed by each subtransaction,

iii) Site for execution for each subtransaction,
iv) destination for results of each subtransaction, and
v) exceptions and exception handling algorithms for the subtransactions.

5.2.6 Global User Interface

The Global User Interface accepts requests from the user and passes back to the user. User here is used in the general sense. It may be a request from the communication network, for example. The Global User Interface accepts a computed result from the system and passes the result to the user. It may accept data, message, command, query etc. and passes these types of information to the Global Transaction Execution Handler. A parsed query or subqueries may be passed among different sites of the network. Hence, the Global User Interface may accept a parsed query or a subquery.

5.2.7 Global Transaction Execution Handler

The Global Transaction Execution Handler accepts the global execution plan generated by the Global Execution Plan Generator and passes the subtransactions to their execution sites with the help of the Communication Network System. It also accepts data, result, message and command which are from the Communication Network System or the Global Control Information Module. Each subtransaction contains a unique transaction descriptor for identification. The Global Transaction Execution Handler also replaces,
wherever necessary, as per the access control mechanism (discussed in section 4.7.1.3), the User Identification Number and other identification information of a user by the User identification Number and corresponding information of the Data Base Administrator (DBA) or the authorizer in all the subtransactions. Any subtransaction which must be processed at the query origination site is sent directly to the Global Subtransaction Handler for the site. The Global Transaction Execution Handler of the query origination site is responsible for the query and will coordinate processing of the query by integrating the results of subtransactions and issuing appropriate responses to users through the Global User Interface. To accomplish this, it accepts results of subtransactions from other sites and passes them to the origination site for further processing to produce final result.

The Global Transaction Handler also handles global concurrency control. Any standard algorithm used for a distributed database management system can be used. It is also responsible for Recovery Operations.

5.2.8 Global Control Information Module

This module contains all the information needed for the Global Functional System to operate. The contents of the Global Conceptual Schema are included in this module. This includes such things as global transaction management
information, schema to schema object mappings, global catalogue information, data distribution information, data authorization requirements, and data integrity requirements. We have already discussed about storing of access rules and other related information in Section 4.7.1.1 and 4.7.1.2.

5.2.9 Global Subtransaction Handler

The Global Subtransaction Handler accepts the results (data or message) from the Local Database Interface at this site and passes it to the Global Transaction Execution Handler of the destination site for additional processing, to complete the query.

It also accepts a subtransaction from the Global Transaction Execution Handler and passes it to the Local Database Interface at this site to translate the subtransaction from the global query language into the local query language for the site.

5.2.10 Communication Network System

The Communication Network System handles the data communication among the sites of the Network. It may accept data, messages, commands and results from the Local Functional System through a Local Database Interface. It sends result to the Global User Interface through the Global Transaction Execution Handler and other information to the Global Subtransaction Handler at appropriate sites. The
Communication Network System provides the interface between the Global Functional System and the Local Functional System of each site. It also monitors and controls network communication among the sites of the network.

5.2.11 Local Database Interface

Since we assume that each Local Functional System is a complete DBMS, operating independently from the Global Functional System, an interface between both these systems is required. The Local Database Interface serves this purpose. The functions of the Local Database Interface are discussed below:

1) It accepts a subtransaction from the Global Subtransaction Handler.

2) It translates the subtransaction from the global query language into the local query language for the site.

3) It creates a transaction or a site query (equivalent to the subtransaction received) for the Local Functional System to process it.

4) It accepts data/result from the Local Functional System and passes it to the Global Subtransaction Handler for transmission to the destination site. It may perform, wherever necessary, data conversion before its transmission to the destination site.
5.2.12 Concurrency Control Module

There are some algorithms available for handling the synchronization for a DDBMS [MOH84]. Any standard algorithm used for a DDBMS can be used for the concurrency control in a DHDBMS.

The Global Transaction Execution Handler is responsible for data from different sites of the network. It may have to wait for data or a message which may be from either the current Global Subtransaction Handler or a Global Transaction Execution Handler from another site. Hence, concurrency control has to be invoked in the Global Transaction Execution Handler module.

For the Global Subtransaction Handler of a site to complete processing a subtransaction, it may have to wait for data which will be transferred from the other sites. To handle such situations, concurrency control has to be called. This may be important when Content Dependent Access Control mechanism is used (See Section 4.7.1.3).

5.2.13 Recovery Module

Several solutions have been proposed for recovery facilities for preventing the loss of the data from system crashes in a DDBMS. We feel that any standard recovery mechanism used for a DDBMS can be applied to the DHDBMS model.
5.3 Integration of Local Conceptual Schemas

The first and foremost requirement for implementing a DHDBMS is to integrate the local conceptual schemas (of the databases to be integrated) in a global conceptual schema. This global conceptual schema is a superview of all databases taken together in a distributed heterogeneous database environment. We will explain here a process of database integration with the help of an example.

We can assume that a distributed heterogeneous database is a collection of data that logically belong to the same system but are spread over the sites of a computer network according to the generalized DHDBMS architecture discussed in Chapter 2.

The above contexts require that an integrated global conceptual schema be designed from the local schemas, which refer to existing databases. This can be considered a database design activity. We can use any of the semantic database models discussed in Chapter 2 to facilitate the integration. We have already discussed that a DHDBMS will map the request of users from such a semantic data model into the actual databases.

The database integration activity is described in a general way in Figure 5.1. It shows that this activity has an input, the local conceptual schemas and the output is a global conceptual schema.
Figure 5.1: Input and Output of Database Integration
To illustrate the main features and problems of schema integration, let us consider an example. In Figure 5.2, we show two descriptions of requirements and corresponding possible local conceptual schemas that model them. The local conceptual schemas are shown in the E-R model form. We have already discussed in Section 2.4.1 that the E-R model may be viewed as a generalization or extension of the relational, hierarchical or network models. It means that from the E-R model, the basic models i.e. the relational, hierarchical or network models may be derived or vice versa. Therefore, we can assume that the two descriptions of requirements shown in Figure 5.2, may have been implemented in any of the basic models (i.e. relational, hierarchical or network model), but they can be transformed in the E-R model form. Because of this reason, we have shown both the local conceptual schemas.

The following additional information applies to this example:

1. The meaning of "Topics" in the first schema is same as that of "Keyword" in the second schema.

2. "Publication" in the second schema is a more abstract concept than "Book" in the first schema. That is, "Publication" includes additional things such as proceedings, journals, monographs, etc.
The data of interest is about Books. Books have titles. They are published by Publishers with names and addresses. Books are adopted by Universities having a name and belonging to a State. Books refer to certain topics.

Figure 5.2: Examples of requirements and Corresponding Schemas
Figures 5.3 to 5.8 show a set of activities that may be performed to integrate the schemas.

Let us look at the two schemas in Figure 5.3. Topics and keywords correspond to the same concept. Since we have to merge the schemas, the names should be unified into a single name. Let us choose the name Topics. Observe the corresponding change in the schema as we go from Figure 5.3 to 5.4. When we look at the new schemas (Figure 5.4), another difference we notice is that Publisher is present in the two schemas with different types: It is an entity in the first schema and an attribute in the second. The reason for choosing different types (attribute vs. entity) comes from the different relevance that Publisher has in the two schemas. However, we have to conform the two representations if we want to merge them. Therefore we transform the attribute Publisher into an entity in the second schema and add a new attribute, Name, to it (See Figure 5.5).

We now can superimpose the two schemas, producing the representation in Figure 5.6. We have not finished merging yet, since we have to look for properties that relate concepts belonging to different schemas, which were "hidden" previously. This is the case with the subset relationship between the concepts Book and Publications. We can add such a subset relationship to the merged schema, producing the result shown in Figure 5.7. Now, to simplify the representation, we can restructure the schema, dropping the properties (relationships and attributes) of Book that
Figure 5.3: Original schemas
Figure 5.4: Choose "Topics" for "Keyword" (In schema 2)
Figure 5.5: Make Publisher into an entity (in Schema 2)
Figure 5.6: Superimposition of Schemas
Figure 5.7: Creation of a Subset relationship
Figure 5.8: Drop the properties of Book Common to Publication.
are common to Publications. This is allowable since the subset relationship implies that all the properties of publications are implicitly inherited by book. The final schema is shown in Figure 5.8. The example of schema integration used above is obviously a "toy example" that highlights some of the basic problems involved. That the integration of realistic sized component schemas can be complex endeavour is amply evident from the example. Schema integration also includes the resolution of naming the conflicts and scale differences etc. [DAY84].

This is one method which can be used for schema integration. There are other methods available to do the same thing [DAY84], and [CLA85].

5.4 Data Definition and Manipulation Language

We have already discussed in the previous section, a methodology of integrating heterogenous databases. In this section we will discuss some important features of a Data Definition and Manipulation Language required for a DHDBMS. We will also provide definitions of these features. We can assume that this language will provide a "conceptually natural" database interface language.

We can use DAPLEX [SHI81] as a base for defining a data definition and manipulation language. First of all let us go through some of the relevant features of DAPLEX which can be used as it is. Than we can add some more features to
this as per our requirements.

The basic constructs of DAPLEX are the entity and the function. Entities are intended to represent real-world objects, and functions to represent properties of objects or relationship among objects. A database schema in this model is represented by a labeled directed multigraph G, whose nodes are labeled with entity types and edges with function names. Functions may be single valued or multivalued.

In every state of G, there is a finite set E of entities. Each element of E is of a specified entity type. The extension in E of entity type X is the set of all entities in E of type X. (We sometimes call this the entity set X.) For each single valued functions name f:X -> Y, there is a function f from entity set X into entity set Y; for each multivalued functions name f:X -> -> Y, there is a function f from entity set X into the power set of entity set Y.

5.4.1 Data Definition

Two important features to define data are discussed below.

5.4.1.1 The Declare Statement

The Declare statements establish functions in the system. Functions are used to express both entity types and
properties of an entity. Let us consider following statement.

DECLARE Title (Publication) -> string

This states that "Title" is function which maps entities of type "Publication" to entities of type STRING. STRING is one of a number of entity types provided by the system along with such other types as INTEGER and BOOLEAN. The statement

DECLARE Publisher (Publication) -> Publisher

states that "Publisher" is a function which applied to a "Publication" entity returns an entity of type "Publisher". These two functions are called single valued as they always return a single entity. Single valued functions are indicated by use of a single arrow (->) in their definition. An example of a multivalued function, indicated by double arrows, is

DECLARE University (Publication) -> -> University

Here the "University" function, applied to a "Publication" entity, returns a set of entities of type "University". A multivalued function may return the empty set.

The functions may be defined with multiple arguments also.

5.4.1.2 Function Inversion

One may be concerned that these functions map in
only one direction. Thus, given a "Publication" entity, we may apply the function "University" to obtain the Universities which have adopted this Publication. But, given an "University" entity, how do we determine the Publications which it has adopted? This problem is solved through the use of function inversion, as illustrated by

\[
\text{DEFINE Publication (University) } \rightarrow \rightarrow \\
\text{INVERSE OF University (Publication)}
\]

We now have a function Publication which can be applied to "University" entities. In so doing, we have entered in the domain of derived functions (notice the keyword DEFINE has replaced DECLARE). A later section is devoted to the definition and use of derived data.

It should be clear here that this facility can be used to map global data(s) set in to local data set(s) and vice versa.

We have discussed about subset relationship between the concept "Book" and "Publication". DAPLEX also supports subtype and supertype relationship and this concept can be implemented by using this feature of the language.

5.4.2 Data Manipulation

The basic elements of the DAPLEX syntax are statements and expressions. Statements direct the system to
perform some action and include the data definition statements and FOR loops. Expressions, which always appear within statements, evaluate to a set of entities. Expressions may involve qualification, quantification, Boolean operators, and comparisons. Details can be seen in [SHI81]. Queries can be formulated using entity variables, which can be declared in range statements.

\[ \text{RANGE OF } <\text{entity-var}> \text{ IS } <\text{entity-type}> \]

Queries are retrieval statements of the form

\[ \text{RETRIEVE INTO } <\text{result-entity-type}> (\langle\text{target-list}\rangle) \]
\[ \text{WHERE } <\text{qualification}> \]

\(<\text{target-list}\rangle\text{ is a list of assignments; each assignment is a singlevalued/multivalued function name.} \]

This feature can be added to DAPLEX for retrieval in a DHDBMS. Similarly other statements for Insertion, Updation and Deletion can be added.

5.4.3. Derived Data

The use of derived data dramatically extends the naturalness and usability of a database system. The concept of "derived data" here is interpreted to mean "derived function definitions". Essentially we are defining new properties of objects based on the values of other properties. Derived functions are specified by means of DEFINE statements.
To define a function "XYZ" over "Topic" which returns Universities which adopt Publications on a topic, use:

\[
\text{DEFINE XYZ (Topic) -> University (Publication (Topic))}
\]

The function "XYZ" may now be used in queries exactly as if it had been a primitive function. The user need not be aware that it is derived data. The same concept can be used to define views. Using these concepts superview as well as user views can also be defined.

5.5 Summary

In this chapter, we have discussed the implementation aspects of a DHDBMS. We have briefly explained the implementation methodology and role of various modules in a DHDBMS. We have also illustrated integration of local databases by working through an example. We have also talked about important features required in a Data definition and manipulation language for a DHDBMS.