To: Distribution

From: Chris Friesen

Subject: Multics Database Management Facility

Date: 01/10/75

This MTB proposes an overall design approach for a database management facility for Multics. The proposals set forth here shall be the topic of a Review Meeting to be held at Cambridge (CISL) during the week of February 10, 1975. Any questions or comments can be sent to me via Multics mail (Friesen Multics).

Multics Database Manager Preliminary Program Specifications

1. Introduction

This document proposes an overall design for the Multics Database Manager (DBM). It discusses the philosophy of the DBM, presents a description of the software included as part of the system and sets forth design goals. It is intended that this draft will be considered a focal point for discussion of possible revisions or enhancements to the DBM, out of which will emerge the final program specifications.

1.1 Purpose

The Multics DBM shall be designed

- to provide the user with a logical (rather than physical) view of the database.
- to allow dynamic reorganization of the database with as little effect as possible on the applications using the database.
- to provide the user with simple, versatile and secure access to the database.
- to allow definition of the database by means of a Data Description Language (DDL).
- to allow manipulation of the data within the database by means of a Data Manipulation Language (DML).
- to provide the user with some means of recovery and restart.
- to provide the user with the advantages of "I-D-S II-like capabilities."

The purpose of the Multics DBM is to provide an integrated set of functions to support the description and processing of large databases for the business, academic and governmental communities.

1.2 Software Design Philosophy

Several factors have influenced the overall design philosophy of the proposed DBM. The following factors are of special significance:

- environment

  The DBM shall operate as a Multics subsystem using the Multics file system to advantage and shall be written in the PL/1 programming language. This will provide ease of program maintenance as well as consistency with other Multics software.

-language interfaces

Rather than extend the syntax of any host language, the DML will be designed to interface with its host language via the CALL statement. This will provide a relatively simple interface mechanism and avoid the problem of modifying the host language. Initially this interface will be provided for PL/1, but it could easily be extended to include FORTRAN and COBOL.

-similarity to I-D-S II

A prime consideration is to offer the user "I-D-S II-like capabilities." This has been interpreted in a general sense. The user will be provided with the general functionality of I-D-S II which includes those functions "required to support the description, manipulation, restructuring, recovery, security, and analysis of a data base." (I-D-S II ERS, p. 6). Specifically, the database on which the DBM operates shall be capable of supporting singular, hierarchical and network data structures.

-data independence

Data independence is a key objective of the DBM. Consequently, the concepts of schema and sub-schemas have been utilized. Also contributing to data independence is the use of context variables to reference data elements logically from within the DML.

-database reorganization

The DBM shall be designed so that reorganization of the database(i.e., reclustering), as well as restructuring of the data shall not be a costly and time consuming process. Furthermore, database reorganization shall have a minimal effect on user application programs which use the database.

-recovery

Database recovery and database integrity are two more key objectives of the DBM. Consequently, the DBM shall include provisions for manually "cleanpointing" the database. This will allow for "clean" database dumps to tape and provide assurance that the integrity of a database is not impaired during recovery of the

database.

1.3 Multics and I-D-S II

The implementation of I-D-S II on the Multics system has been considered to be a desirable objective. This would seem to be true only if I-D-S II is defined in the most general of terms.

Some I-D-S II features seem to be at variance with the philosophy of Multics operating procedures.

-Chains

The concept of "chaining" records together forces a considerable degree of "hard to change" structure onto a database. For instance, it is quite difficult in I-D-S II to declare an existing record type to be a new member of an existing set. The imposition of this sort of permanent external structure seems to run counter to general Multics philosophy. In Multics the actual binding (i.e., structuring) of data is usually conceived of as a dynamic, rather than a static, process. The "binding" of data elements is usually postponed until the last possible moment (at execution time, for example). Admittedly data elements in a database must be structured to a higher degree than non-database elements because of the necessity, within a database, to carry along (or "remember") relationships. Nevertheless, it seems advisable to allow such relationships to be as dynamic as possible.

The "chaining" of records also tends to lose much of its usefulness as the chain pointers become more interpretive. That is, the most efficient chain pointers are those which are hardware oriented. But it is not realistic to think in terms of hardware oriented chains within the framework of the Multics virtual memory system or the Multics file system.

Consequently, the entire concept of "chained" records seems to be of only limited value with reference to the Multics operating environment.

-Areas

The concept of "area", as used by I-D-S II, seems well suited to I-D-S where it is necessary to "reserve" a number of file pages prior to usage because the CALCing algorithm is dependent upon the total number of file pages assigned to the file (or area, or database). The concept is also a "natural" for GCOS users who have become accustomed to think in terms of reserving file space before actually using the file.

The "area" concept, however, seems alien to the philosophy of Multics wherein "file space" (or virtual memory) is allocated and used as needed.

There are some inherent shortcomings in I-D-S II which ought to be avoided if at all possible. These disadvantages are essentially the same as those listed in IBM's critique of DBTG in the "Position Paper Presented to the CODASYL Programming Language Committee" dated May, 1971. They may be grouped under three general headings of data independence, database reorganization and DML complexity.

-Data Independence
The required areas of correlation between the user's DML and the schema DDL are numerous. For example, the format of a FIND statement is dependent on the location mode with which the sought-after record was originally stored on the database. Such a close association between DDL and DML means that database reorganization will more than likely impact user application programs. This is an unattractive implication. The inter-relationship between DML and schema DDL contributes toward a tendency to restrain the natural evolution of the database structure.

-Database Reorganization
The reorganization of an I-D-S database generally strikes fear in the heart of a user. Not only is the process costly and time consuming, but it may also seriously affect various user programs. Nonetheless, it is only reasonable to expect the type of demands made upon a database to change as time passes. As these changes develop, the database will undergo some transformation in appearance. If the transformations are such as were not allowed for during the initial I-D-S database design period, then reorganization, regardless of cost, becomes a necessity. If instances of database reorganization are necessary, they ought at least to be made as painless as possible.

-DML Complexity
The complexity of the I-D-S II DML is immediately obvious to anyone who cares to examine the language syntax and rules. Such complexity is pardonable if the advantages gained thereby are sufficient to offset the associated disadvantages. It seems, however, that the only real advantage gained by this approach is that user application programs are allowed to capitalize on their knowledge of physical storage idiosyncracies (such as whether a record was stored via the CALC location mode). On the other hand, the disadvantages of such a complex DML include the loss of some data independence, an increase in costs of debugging and implementing user application programs and a forced commitment on the part of the user to be content with a relatively static and unchanging database structure.

Attempts should be made to overcome these obvious disadvantages of the I-D-S approach, while at the same time avoiding the introduction of other more serious shortcomings.

2. DBM Operating Environment

The Multics DBM is to run as a Multics subsystem and will use the standard Multics file system. All dataset I/O will be performed by the vfile_ I/O module.

3. Database Management System

The DBM will consist of essentially three distinct parts:

- those modules concerned with schema definition and processing.
- those modules concerned with sub-schema definition and processing.
- those modules concerned with run-time processing and data manipulation.

Additionally there will exist utility programs of various types.

All DBM activity will involve interaction between the Multics file system and user programs.

Design objectives:

- The DBM shall be designed to allow the user as much data independence as possible.
- Database reorganization shall be a function of a "database administrator" and shall not affect user application programs.
- Data redundancy shall be kept to a minimum, being required only to a very limited extent by the entity_keys (defined below).
- The CALL interface for the DML will make the DBM run-time routines easily available to programs written in languages other than PL/1.

3.1 Database Structure

The raw database will consist of a set of relations (files) in Third Normal Form as defined by E. F. Codd. The only non-data attributes contained within the data_relations shall be the unique identifier generated by the DBM for inter-file linkage

purposes within the Multics file system. All files created by the DdM shall be created by the Multics file system. Hence, they will be B*-tree files as defined by D. Knuth.

3.2 File Handling Procedures

Definitions:

File is a Multics Indexed Sequential File.

Relation is a collection of entities. It can be visualized as a two-dimensional table wherein the entities are the horizontal rows. Each relation is a Multics file.

Entity is a collection of attribute-value pairs established at schema definition time.

Attribute is analogous to "field name" and may be visualized as the columns of a relation, established at schema definition time.

Value is a quantity associated with each occurrence of an attribute.

Entity_key is the logical attribute (or collection of attributes) which uniquely defines each entity.

Entry_path is a linkage between the entities of two relations. A one-way entry_path allows passage from an entity in data_relation-A to an associated entity in data_relation-B. A two-way entry_path allows passage as well from the entity in data_relation-B to the associated entity in data_relation-A.

Record refers to the sub-schema definition of one or more entities. It is the logical representation of a group of attributes as seen by the user application program.

Field refers to the sub-schema definition of an attribute. It is the logical representation of an attribute as seen by the user application program.

As a general rule there will exist

- one Multics file for each normalized relation. This file shall contain all the attribute values of each entity in the specified relation. Each entity shall be a logical record within the Multics file. Each entity shall contain one (or a number of)

attribute(s) constituting an entity_key. The entity_key shall be visible to the user only if it has been explicitly defined by the user. Each user visible entity key shall provide the user with a "handle" for that entity. Each entity shall also contain one attribute (invisible to the user) entitled a unique_id. For those entities without a user visible entity_key, the unique_id attribute shall play the role of an entity_key recognizable as such by the DBM. The unique_id shall contain a unique bit representation (generated by the DBM) which will serve as a system identification tag for each entity.

For example, a name_address entity with an entity_key of person_name might be represented as follows:

```
attributes: person_name str_addr city st unique_id
entity:     Smith Joan  101 Elm  Ajo  AZ f(t)
```

where f(t) is a unique function of t and t represents the time of entity creation.

This relation shall be termed a data_relation.

Properties of data_relations

<> there will exist one data_relation for each normalized relation.

<> each entity within a data_relation will be assigned a unique_id.

<> each entity_key within a data_relation must be unique within that data_relation.

<> the data_relation will be an indexed sequential file indexed on ascending entity_keys.

<> the only user-visible entry points into a data_relation are the entity_keys explicitly defined as such at schema creation time.

<> if a data_relation is not to be entered directly by a user (i.e., if it is to act as a secondary record-type in I-D-S terminology), then the unique_id attribute for each entity shall also be the entity_key for that entity; such a data_relation shall require no associated key_relation(defined below).

<> the entities within a data_relation will contain a fixed number of attributes (at least in the initial implementation).

-one Multics file for each data_relation containing a user visible entity_key. This file shall contain only the unique_id and the entity_key of the specified data_relation. This file shall be indexed, in the Multics file system, according to the attribute_value of the unique_id for each entity. This provides a system-internal linkage mechanism among the various relations. For those data_relations where the unique_id and entity_key attributes are identical, there shall exist no key_relation.

This file shall be termed a key_relation.

Properties of key_relations

<> the only two attributes of an entity within a key_relation will be the entity_key and its associated unique_id.

<> the key_relation will be an indexed sequential file indexed on the unique_id of the associated entity.

<> there will exist one key_relation for each data_relation, with a user visible entity_key, to be logically associated with another data_relation. That is, if an entity within a data_relation is to be retrieved when its entity_key is unknown, then there must exist a key_relation for that data_relation.

<> there will be as many entities within a key_relation as there are entities within the associated data_relation.

-one Multics file for each one-way entry path into a relation. Each of these files shall contain two classes of attributes, consisting of the unique_id for the source entity and a variable number of unique_ids for the target entities. This file shall be indexed on the unique_id of the source entities. In addition to the entity_key attribute, there will exist one attribute for each target entity associated with the source entity (identified by the entity_key). This will enable the treatment of hierarchical and network relationships between different data_relations.

This file shall be called a cross_relation.

Properties of cross_relations

<> there will exist one cross_relation for each entry_path between two data_relations. For example, if an employee data_relation is such that it will always be entered from a department data_relation and the department data_relation will never be entered from the employee data_relation (e.g., given an employee's name, there will never be an attempt to retrieve that employee's department name), then we may say there exists a one-way entry_path between the employee and department data_relations.

<> there will exist as many entities in a cross_relation as the number of entities in the associated source data_relation.

<> each entity within a cross_relation will consist of two classes of attributes, as follows:

- the unique_id of the entity in the source data_relation shall constitute the indexed sequential key (or index)

- the unique_ids of those entities in the target data_relation shall constitute the non-prime (i.e., non entity_key) attributes for each cross_relation entity. There will exist as many target unique_id attributes as there are entities associated with the specified entity in the source data_relation. If the source data_relation has no user visible entity_keys, then the target unique_id will also be an entity_key value.

Note that it is possible from the user's point of view, for all entities within a given data_relation to be associated with two or more inter-dependent entity_keys from other data_relations. This is roughly analogous to the concept of a secondary detail record with two master record-types in I-D-S. Of course there is no need for the user application program to be cognizant of the different data_relations -- that is the proper concern of the database administrator. Nevertheless, an example of the above mentioned condition would be an entity which indicates the quantity of different parts committed to various projects. (This corresponds to a secondary record with two masters of different record-types, in I-D-S jargon.) Since such an entity would possess no entity_key visible to or knowable by the user, its index key would be generated by the DBM as a unique string.

3.3 Schema Processor

The schema itself shall describe a database physically arranged in a relational (i.e., tabular) manner. The essential functions performed by the schema processor will be two:

- it will create, modify and delete all files necessary to represent the data as defined by declarations within the Schema Processor.

- it will store information about the schema structure, to be used later by the Sub-schema Processors

The specific functions performed by the Schema Processor shall be to:

- create a directory file identifying the database with a given schema name. When sub-schemas are subsequently created, their names shall be recorded in this directory.

- maintain a list of all attributes and entity_keys along with the (Multics) file names of the data_relations in which they participate. The physical characteristics of each attribute (e.g., size, type, etc.) shall be described as a part of this list. The locks to be assigned to each attribute shall also be a part of this list.

- maintain a list of all data_relation names along with the attributes, locks and entity_keys associated with each of them.

- maintain a list of all the entity_paths existing among all data_relations. An indication whether a data_relation is dependent upon another data_relation will appear in this list. (When an entity is added to or deleted from a dependent data_relation, then the cross_relation files between the associated data_relations must be updated. Otherwise, the cross_relation files would be updated only if the associated data_relations were referenced by the sub-schema in use).

A schema may be as complex or as simple as desired. Care should be taken, however, to insure that it is normalized. Else, considerable inefficiencies could be introduced into a user application program which attempts to modify data. To draw a rough analogy with I-D-S, normalization of a relational database might be compared to defining a database in terms of data structure diagrams.

3.4 Sub-schema Processor

The Sub-schema shall provide the user application program with its view of a database. It shall be a subset of a schema. The user application program shall be allowed to access more than one sub-schema and/or schemas.

In interfacing with the schema, the Sub-schema Processor need only be aware of the name of the schema(i.e., the database) and the names of those attributes (and entity_keys) with which this sub-schema is concerned. In general it shall be unnecessary for the user of the Sub-schema Processor to be concerned with the names of the data_relations , key_relations or cross_relations. A general knowledge of the logical relationships among the attributes will usually suffice. The Sub-schema Processor will update the list of sub-schema names associated with this schema.

A Sub-schema DDL shall be defined, initially, for the PL/1 programming language.

Specific functions performed by the Sub-schema Processor shall include the following:

- the associated schema directory file shall be updated to contain the entry name of the specified sub-schema.

- a list shall be maintained which shall associate all the logical database records defined in the sub-schema with the specified entities defined by the Schema Processor (there need not be a one-to-one correspondence). A list of sub-schema specified locks shall also be maintained.

- a list shall be maintained which associates field names defined in the sub-schema with specified attribute names previously defined in the schema. This list shall also contain a description of the physical characteristics of each field defined by the sub-schema. These characteristics may differ from the characteristics of the corresponding attribute defined in the schema. The DBM will make the necessary data conversions at run-time. A list of locks to be applied against each field shall also be maintained.

- a report showing any inconsistencies (such as missing entity_key definitions within the sub-schema) will be generated. Also, if an entity is referenced by the sub-schema which is dependent to an entity in another data_relation, then whenever a dependent entity is added to or deleted from the

database, a "hidden" cross_relation file must be updated. All possibilities of such "hidden" side effects shall be reported out.

3.5 Data Manipulation

The DML shall be implemented through the use of CALL statements. The following functions will be supported:

-OPEN
make a sub-schema available for update or retrieval-only (all necessary files, such as data_relations, key_relations and cross_relations, shall be attached and opened by the DBM run-time routines -- provided the user has access to all the files).

-STORE
a new record with all its fields (which may be null valued) will be placed on the database -- the user must provide a non-null entity_key and its value as well as the field names and values to be associated with the specified record.

-FIND
retrieve a specified record and its associated field values as defined by the sub-schema.

-MODIFY
modify a specified record or field value(s) to equal a given set of values.

-DELETE
remove from the database a specified record or field value(s) satisfying a given set of conditions; also remove all relevant linkages.

-CLEANPOINT
the segments within the sub-schema which have been altered since the last CLEANPOINT command was issued shall be made eligible to be dumped to tape with the Multics Backup facility.

-CLOSE
the sub-schema shall be released from the user application program (all associated files shall be closed and detached).

Additional functionality, such as algebraic operations, may be added at a later time. However, it may be advisable for such capabilities to be provided by separate

end-user-facilities, rather than through PL/1. A primary feature of the DML shall be that all references to attributes will be on the logical level. The user need know little about the physical organization of the database. All references to data will be through the field names as defined by the user's sub-schema. The concept of "current logical record" shall be implemented. It shall refer to the last logical record of each type (as defined in the sub-schema) to have been operated on by a STORE or FIND function. These records, however, shall not be explicitly referred to as "current" records. They shall be retrieved by use of the FIND function with an argument to indicate "first," "next" or "all" records satisfying some condition.

3.6 Operational Considerations

-Recovery/Integrity

A means of insuring database integrity shall be an essential part of the system. The DBM shall take care to update all linkages whenever a datum is modified by a user application program. If desired, "before" and/or "after" images of all altered segments could be journalized by being written to auxiliary files and dumped to tape periodically using the Multics BACKUP facility. The overhead for such protection may be high. It is not clear at this time how much should be provided in this area.

Another possibility is that all updates be written to a reserve area before actually being "posted" to the database. Again, the overhead price for such a procedure would be considerable.

Long term recovery procedures must also be provided for those instances when part of the database is inadvertantly destroyed. The backup tapes can be used to bring such a destroyed database back up to its current state of integrity. This requires, however, that the backup tapes contain "clean" data. This becomes especially relevant when dealing with multi-segment files, since DBM linkages in one segment may reflect a different state than those in another segment. To this end, the DBM must establish some intelligent communication interface with the BACKUP facility if it is to be used to advantage. User-initiated CLEANPOINTs will contribute toward such a goal, as will DBM initiated CLEANPOINTs. The DBM will dump updated segments only after it has assured itself that all segments to be dumped are in a CLEANPOINT state.

-Privacy/Security

The DBM shall rely primarily on Access Control Lists to insure privacy on the schema, sub-schema and relation levels. This shall fall under the responsibility of the Database Administrator. Provisions will be made to provide the future capability to LOCK and UNLOCK items at the attribute level.

-Concurrent Update

Concurrent update of a sub-schema shall not be protected. This feature may be provided in the future.

4. DBM Support Software

The support software to be provided shall include utilities to load, recover and analyze a database as well as an end-user-facility.

4.1 Load and Unload Utility

The Load Utility, written in PL/1, shall be designed to run as an absentee process, which will accept parameters defining the raw data to be loaded. The Load Utility will generate all necessary unique_ids and will load all relevant linkage files (which will previously have been created by the Schema Processor). The database referenced by the Load Utility shall be that database defined by th schema (rather than a sub-schema). The Load Utility shall normally be the responsibility of the database administrator.

The Unload Utility shall be designed to dump to tape all or selected portions of a database, including all relevant system data such as pointers and directories.

4.2 Recovery Utility

The Recovery Utility shall interface with the Multics BACKUP facility to provide for "clean" database recovery. The recovery shall be as automatic as possible.

4.3 Analysis Utility

A Utility to analyze the size of data_relations, key_relations and cross_relations shall be provided. The Analysis Utility shall also serve as a verification aid in

determining the existence, or lack thereof, of linkages between the entities of various data_relations.

4.4 End-User-Facility

Some type of end-user-facility shall be provided at some time. This facility shall include but not be limited to the following capabilities:

- Boolean retrievals
- computations
- sorting
- creation of aggregate datasets
- report generation

4.5 Instrumentation

Some provision should be made for measuring DBM performance. Some obvious areas to examine would be the time required to perform overhead labor, such as pointer updating, the time required to retrieve entities participating in different structures and the time required to respond to a request in an interactive environment.