To: MTB Distribution

From: Noah S. Davids and Paul W. Benjamin

Date: May 14, 1982

Subject: MRDS Restructuring for MR10.1

Send comments by one of the following means:

By Multics mail (on System M):
Benjamin.Multics

By Telephone:
HVN 341-7302 or 602-249-7302

By Forum (method of choice)
Link to transaction 137 (subject restructuring mtb) in the >udd>Demo>dbmt>mrds_rst meeting.

INTRODUCTION:

The purpose of this document is to present the subsystem that will be implemented for MR10.1 to allow a DBA to restructure a MRDS database.

RESTRUCTURING CAPABILITIES:

The restructuring capabilities to be provided in MR10.1 are:

- adding new relations
- adding a new populated relation based on a selection expression
- adding new secondary indices to relations
- deleting relations
- deleting secondary indices

These capabilities were selected based on providing the most useful capabilities given the time allowed for the task. While an earlier version of this document proposed changing the data model, a decision was made to use the existing structure of the data model. When the larger task is re-visited, it is proposed that changes to the model should take place at that time.

DBA INTERFACE:

The interface that has been chosen is that of an ssu_-based subsystem. The ssu_ (subsystem utility) package is a powerful tool for subsystem design. The use of the subsystem approach will allow the requests to have shorter names than the corresponding commands would, without being ambiguous, as well as reducing the number of arguments that each request needs and allowing the per database checks (existence and DBAness) to be done only once instead of for each request.

CONCURRENCY:

While it would be possible to design a system that allowed all the various levels of concurrency needed for restructuring I do not feel that it is advisable.

INTERRUPTION and INCONSISTENCY:

Although rmdb will be an interactive subsystem, there are operations that it performs that can potentially take long periods of time. Adding secondary indices to a large relation would be measured more easily in minutes than it would in milliseconds. Because of this, the issue of interruption and release is of greater import than in most interactive subsystems. If the user quits after 45 minutes of uncompleted work, what does one do? Deleting the indices that had been created could well take another 45 minutes. Releasing leaves the database in an inconsistent state. The decision was made to allow the user an option. In the cleanup handler the user will be queried and given 2 choices. Either the operation will be continued (as if she had typed start rather than release) or the release will continue. In the latter case the database is flagged as inconsistent and two things are stored: a text string containing the reason for the inconsistency, and an 'undo' operation. An undo operation is a an rmdb request line that will make the database consistent. The undo operation for the create_index request is the corresponding delete_index request, for example. Subsequent attempts to open the database will be refused (displaying the fact that the database is inconsistent and why). Changes to various MRDS modules that open the database or display the status of the database will be necessary. Upon attempting to use the database in rmdb the DBA will be informed that the database is inconsistent, that undo operation will make it consisten once again, and does she wish to have that request line executed on her behalf? It is stated in the Standards SDN that cleanup handlers should never print anything. We have, however, been in touch with the developer primarily concerned with such matters and been assured that, if approached properly, it can be done in this situation.

TIME ESTIMATES:

The time estimates for implementing these restructuring capabilities may be broken down into three parts, implementing the subsystem, implementing the individual requests, and modifing the ancillary MRDS code. These estimates are based on the existing MRDS project staff and currently perceived difficulty of changing the ancillary MRDS code.

```
implementing the subsystem:
     design: .5 person week (pw)
     documentation: .5 pw
     test design: .5 pw
     implementation
        coding: 1 pw
        test execution and bug fix: 1.5 pw

     total: 1 person month (pm)


implementing the restructuring commands:
     design: 2 pm
     documentation: 0.5 pm
     test design: 1 pm
     implementation:
        coding: 2 pm
        test execution and bug fix: 2 pm

     total: 7.5 pm


modification of ancillary code:
     design: 1 pw
     documentation: 1 pw
     test design: 1 pw
     implementation
        coding: 2 pw
        test execution and bug fix: 1 pw

     total: 1.5 pm
```

SYNTAX:

```
     restructure_mrds_db {db_path}
     rmdb {db_path}
```

FUNCTION:

This command causes the invoking process to enter the MRDS restructuring subsystem. If the optional database argument is given, that database is quiesced.

ARGUMENTS:

db_path
: A relative or absolute path to the database to be restructured.

NOTES:

1) This command may only be used against a version 4 or later database.

2) This command may only be used by the (one of the) database's DBA.

3) This command may not be used against a database that is already open by any process. The database may be opened (only by the process invoking this subsystem) after the subsystem has been entered by invoking linus or mrc via the ".." request.

LIST OF REQUESTS[1]

create_index relation_name attr_name
cri relation_name attr_name
Makes the indicated attribute a secondary index into the relation. This operation is not allowed on any attribute that may already be treated like an index. An attribute may be treated like an index if it is an index or if it is the first attribute of the relations primary key.

create_relation relation_name {(attr1 attr2 ...)} {-index attri attrk...} {-se STR}
crr relation_name {(attr1 attr2 ...)} {-index attri attrk...} {-se STR}
Creates a new relation. An unpopulated relation may be specified by listing the attributes that will make up the relation, each attribute must already be defined. Attributes that are to make up the relation's primary key are followed by an "*". A relation may also be specified by a selection expression. In this case the relation will take the format of the attributes in the select clause of the expression, again attributes followed by an "*" (in the select clause) indicate that the attribute is part of the primary key. The relation will be populated by those tuples selected by the selection expression. Only one of these specifications formats may be used. The index control argument indicates which attributes are to be secondary indices into the relation.

delete_index relation_name attr_name {-force}
dli relation_name attr_name {-force}
Deletes the secondary index over the indicated attribute in the relation. The -force control argument will cause partially created indices (or indices that appear to be inconsistent with the model) to be deleted without comment. The default in such situations is to report an error and abort processing.

delete_relation relation_name {-force}
dlr {-force} relation_name
Deletes the indicated relation from the database. The -force control argument will cause partially a created relation (or a relation that appear to be inconsistent with the model) to

[1] The standard ssu_ requests, including ?, ., .., abbrev, answer, debug_mode, do, exec_com, execute, help, if, list_help, ready, ready_off, ready_on, subsystem_name and subsystem_version are supplied. They (with the exception of debug_mode and the ready requests) will be fully documented, but for the purposes of this MTB are not documented here.

be deleted without comment. The default in such situations is to report an error and abort processing.

free_db, fdb
: Unquiesces the database.

quit, q
: Unquiesces the current database and leaves the mrds_restructuring subsystem.

ready_db db_path
rdb db_path
: Quiesces the indicated database and makes it available for restructuring. Note that only one database can be restructured at one time.