To: MTB Distribution

From: Roger Lackey

Date: September 2, 1982

Subject: MRDS and DMS: vfile_ Relation Manager

Comments may be made:
Via forum: >udd>d>dmbt>con>mrdsdev

INTRODUCTION

This MTB will discuss the detailed design of the vfile Relation Manager. The draft of MTB-545 Relation Manager Functional Specification has been used as an interface specification. Each vfile_relmgr_$ entry described in this document has an equivalent page_file entry described in MTB-545.

The vfile_ Relation Manager will implement most of the operations described in MTB-545 using the existing vfile_ as the underlying file manager. Because the vfile_ Relation Manager must maintain compatibility with the existing MRDS databases, much of the descriptive information must be derived from the database db_model and file_models. The vfile_ Relation Manager will never write in to the database model but only read information when necessary. It is important to remember that the relation manager will only be dealing with one vfile_ at a time and that some of the entry points names do not accurately describe the function performed by the entry. An example is the create_relation entry point that for the vfile_ Relation Manager will only create the vfile as an msf. It will not manipulate the db_model or file_model.

To maintain compatibility with the pf_relation_manager the same error table should be used.

This MTB briefly describes the following vfile_relmgr_ entries.

close
create_cursor
create_index
create_relation
delete_tuples_by_id
destroy_cursor
destroy_index
destroy_relation_by_opening
destroy_relation_by_path
get_count
get_duplicate_key_count
get_tuple_id
get_tuple_by_id
get_tuples_by_id
get_tuples_by_spec
modify_tuples_by_id
open
put_tuple
set_scope

The following relation_manager_ entries will not be supported by the vfile_relmgr_. If any of these entries are called, an error code of error_table_$action_not_performed will be returned;

```
create_subset_index
delete_tuple_by_id
delete_tuples_by_spec
get_description
get_max_and_min_attributes
get_population
modify_tuple_by_id
modify_tuples_by_spec
put_tuples
```

DEFINITIONS

opening_id

the opening_id is a unique per-process 36 bit identifier that associates an opening with a vrm_rel_desc (see below). It is similar to the pf_opening_id described in MTB-545. Many of the entry points use an opening_id to specify which relation it is addressing.

vrm_rel_desc

is a structure created when a call is made to vfile_relmgr_$open and contains the relation specific information needed for the vfile_rel_mgr.

cursor_ptr

is a pointer to a cursor structure allocated in the used supplied area that maintains information about a position within a relations vfile_. Each vf_relation may have many cursors with each cursor being created with an opening_id and an index_id which can have one of three forms. A negative index_id would specify that vfile_scan_records is to be used for searching. If the index_id is zero then the primary key is to be used for searching, if the the index_id is positive it must equal one of the index_ids assigned to an attribute for the relation specified by the opening_id.

Each call to the vfile_relmgr_ which uses the cursor_ptr as a parameter will cause the cursor to be validated as having a valid rel_desc associated with it along with the proper "scope" setting.

ENTRY: vfile_relmgr_$close

The vfile_relmgr_$close function will cause the number of openings for the opening_id to be decremented. When the number of openings for the gets to zero, the relation description associated with the opening_id will be freed. Cursors associated with the opening_id will be invalid but will not find out they are invalid until first reference after opening_id was deleted.

ENTRY: vfile_relmgr_$create_cursor

If the opening_id is valid and the collection_id is valid a vrm_cursor will be allocated in the work_area supplied in the call and the pointer to vrm_cursor returned to the user.

A vfile_ attach and open will be made for each cursor, with the pathname and opening mode derived from the vrm_rel_desc.

ENTRY: vfile_relmgr_$create_index

If the relation is not populated the entry will only update the vrm_rel_desc with the attributes index_id obtained from the file_model (rel_info) and set the attribute index flag. If the relation is populated vfile_relmgr_$create_index will update the vrm_rel_desc and add the index keys to the vfile_ deriving the key values from the attribute values in each tuple. This will be done by creating two vfile attachments: one for use by scan_records to get all the attribute values and one for adding the key.

ENTRY: vfile_relmgr_$create_relation

The vfile_relmgr_$create_relation will only cause the relation data file (vfile_) to be created deriving all the necessary information from the database model and file_model (rel_info). The input parameters pf_creation_info_ptr and descriptor_ptr are not used by vfile_relmgr_$create_relation.

ENTRY: vfile_relmgr_$delete_tuples_by_id

All tuples in the vfile_ that match the tuple_ids provided in the input tuple_id array will be deleted along with all indexed keys associated with the tuples. The number of tuples deleted will be returned.

The position of the cursor may change as a result of this operation. The cursor is left positioned at the tuple following the last tuple deleted.

ENTRY: vfile_relmgr_$destroy_cursor

This entry point will close, detach and destroy the iocb associated with the cursor and free space used by the cursor (in users area). The cursor_ptr will be set to null before returning.

ENTRY: vfile_relmgr_$destroy_index

If the relation is not populated then only the vrm_rel_desc for that attribute is updated to reflect the fact that the attribute is no longer indexed and the index_id set to zero. If the relation is populated then the vfile_ is attached and opened and all indices with the index_id prefix are deleted.

ENTRY: vfile_relmgr_$destroy_relation_by_opening

The vfile_relmgr_$destroy_relation_by_opening will destroy the vrm_rel_desc and the opening_id as well as deleting the vfile_ associated with the relation.

ENTRY: vfile_relmgr_$destroy_relation_by_path

The vfile_relmgr_$destroy_relation_by_path will destroy the vfile_ associated with the relation. Any opening_ids and vrm_rel_desc that exists for the relation will also be destroyed.

ENTRY: vfile_relmgr_$get_count

The vfile_relmgr_$get_count entry will return the number of items that satisfy the search specification using the cursor provided. The file will be positioned to BOF before the search starts and the cursor will be left at the EOF. A null specification pointer will cause this routine to return an exact number of tuples in the relation.

ENTRY: vfile_relmgr_$get_duplicate_key_count

The vfile_relmgr_$get_duplicate_key_count will return an approximate value for the duplicate_key_count. The duplicate count for an index will be calculated by getting the total duplicate key count form vfile_status, dividing by the number of indexed attribute plus one (for the primary key).

The input parameter number_of_duplicate_fields will not be used by the vfile_relmgr_$get_duplicate_key_count entry.

ENTRY: vfile_relmgr_$get_tuple_id

Returns the identifiers of the tuples that satisfy the search specification using the cursor specified. The cursor is left at the last tuple returned. If there is not enough room in the area provided by the user for all tuple_ids found, then as many as will fit will be returned, the cursor will be positioned to the last tuple returned and an error code will be set.

ENTRY: vfile_relmgr_$get_tuple_by_id

Returns the value for the single tuple that matches the input tuple_id. The cursor is left positioned at the tuple returned.

ENTRY: vfile_relmgr_$get_tuples_by_id

Returns the values for the tuples that match the tuple_ids in the tuple_id array parameter. The cursor is left position at the last tuple returned. If there is not enough room in the area provided by the user for all tuples found, then as many as will fit will be returned, the cursor will be positioned to the last tuple returned and an error code will be set.

ENTRY: vfile_relmgr_$get_tuples_by_spec

Returns the values of the tuples that satisfy the search specification using the cursor specified. The cursor is left at the last tuple returned. If there is not enough room in the area provided by the user for all tuples found, then as many as will fit will be returned, the cursor will be positioned to the last tuple returned and an error code will be set.

ENTRY: vfile_relmgr_$modify_tuples_by_id

All tuples in the vfile_ that match the tuple_ids provided in the input tuple_id array will be modified along with all indexed keys associated with the tuples. The number of tuples modified will be returned. If a tuple id that is in the input array and does not exist in the relation it will be left in the array. Thus upon return the tuple_id array will contain those tuple_ids that could not be modified.

The position of the cursor may change as a result of this operation. The cursor is left positioned at the last tuple modified.

ENTRY: vfile_relmgr_$open

The vfile_relmgr_$open module will generate a 36 bit string that is unique within that process for that relation. This is the relations opening_id.

With each opening_id a relation description will be created with much of its information be obtained from the MRDS db_model and file_models. The relation description will contain all the necessary information needed for the vfile_relmgr_ manipulation routines. In general it contains the number of attributes and attribute specific information such as the attribute descriptor, flags indicating if the attribute is part of the primary key, if it is indexed and the index id (8 bits).

ENTRY: vfile_relmgr_$put_tuple

The vfile_relmgr_$put_tuple entry adds a single new tuple to the vfile_ and updates all of the associated indices for the tuple. The tuple_ids of the tuples added will be returned. The cursor will be left at the last tuple added.

ENTRY: vfile_relmgr_$set_scope

This entry will cause the opening mode for the relation to be reset. The relation's open mode is checked by each cursor before is used and the vfile_ closed and reopened if the mode has changed.

Vfile Relation Manager Structure

This is a description of the relationships of the per-process tables used by the Vfile Relation Manager.

Opening_ids, vrm_rel_desc (relation_desciption) and cursors comprise the three major structures associated with each relation. Each opening_id will be related to a vrm_rel_desc. An opening_id and associated vrm_rel_desc may have many cursors associated with them. Each cursor will be related to only one opening_id and vrm_rel_desc.

The first call to the vfile_relmgr_$open for each process will create a temporary segment used to maintain vfile_relmgr_ control information. The format of the segment will consist of a header with the rest of the segment being an extensible area. The opening_id table will be established in this area along with all relation descriptions.

The procedure vrm_open_man will maintain an internal static pointer to the opening_id table. The module vrm_open_man will have an add_opening entry point, a set_rel_desc_ptr and a get_rel_desc_ptr entry point described later. Each entry in the opening_id table will contain the 36 bit identifier associated with the opening and a pointer to the vrm_rel_desc for that opening.

The vrm_rel_desc is a structure containing the information needed by the vrm_modules to manipulate the vfile_. It contains information about the number of attributes in the relation, which ones are part of the primary key, which are indexed and what their index_id are, information needed to build attach descriptions for the vfile. This information is acquired when the call to vfile_relmgr_$open is made and is obtained from the db_model and the relations file_model.