BILL'S RELATIONAL ALGEBRA DATABASE

BY

WILLIAM ERIC VOSS

B.A., University of Tennessee, 1988

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1994

Urbana, Illinois

Acknowledgements

I would like to thank my advisors; Ralph Johnson for his support through the years, and Marianne Winslett for making me see I did not need to earn the Turing Award to complete a Master's thesis.

Contents

Chapter 1

Introduction

I needed a free light weight database. I did not need multi-user support, transactions, or query optimization. However I did need the ability to efficiently store long strings, and to use those long strings as lookup keys. I needed a system which could easily interface with a Perl[WS91][Sch93] program. I also needed a system which supported the basic Relational Algebra [Cod70][EN89] operations.

I could not find such a system available for free, so I implemented one. The result was *Bill's Relational Algebra Database* [Vos93], a Perl library which uses Perl's associative array interface to gdbm[1] to store its data.

This thesis describes *Bill's Relational Algebra Database.*

[1] "GNU dbm is a set of database routines that use extendible hashing and works similar to the standard UNIX dbm routines."[gdb]

Chapter 2

Conceptual Interface

Bill's Relational Algebra Database is a library of functions based on Relational Algebra with a few additional support utilities thrown in.

The basic operations supported are:

(((Relation [Selection])[Projection]) Unary Operation) [Projection]

(((Relation [Selection])[Projection]) Binary Operation ((Relation [Selection]) [Projection])))[Projection]

This chapter contains discriptions of the terms Relation, Selection, Projection, Unary Operation, Binary Operation, and Key as they apply to *Bill's Relational Algebra Database.* The next chapter describes the actual function interfaces and includes example usage.

2.1 Relations

In *Bill's Relational Algebra Database* relations are *sets* of tuples. *Keys* specify which attributes are required to uniquely specify a tuple.

The system supports both persistent relations which are stored until explicitly removed, and temporary relations which are normally removed when an application terminates.

An application uses an opaque handle[1] to refer to relations. Permanent relations are explicitly created using **CreateTable** (see 3.2) which returns an opaque handle to the new relation. Perma-

[1] The opaque handle is currently simply the name of the table, but user code should not take advantage of that fact.

nent relations which were created during a previous session are accessed using **OpenTable** (see 3.3) which also returns a handle. Additionally the Relational Operations (see 2.4) normally create a temporary table, and return a handle to the temporary table. (The output can easily be placed in a permanent table instead if desired using **AddRow** (see 3.4) .)

2.2 Selections

All the underlying storage is done based on hashing, so only simple equality selections are supported. The user specifies (attribute,value) pairs for none, any, or all of the relation's attributes. Unspecified attributes match any value.

2.3 Projections

The user provides (old_name,new_name) pairs for all the attributes being projected. An unmentioned attribute is not projected.

2.4 Relational Operations

2.4.1 Unary Operations

Two unary operations are supported, **QueryByExample** (see 3.5) and **DeleteByExample** (see 3.6). **QueryByExample** is simply a way of applying the normal Relational Algebra operations Select and Project to a single table.

DeleteByExample is used to remove tuples from an existing table. Pragmatically, **DeleteByExample** just removes from a relation any tuples which match the selection condition. In Relational Algebra terms **DeleteByExample** takes a relation **A** and performs a Select to generate a relation **B**. Then it performs **A** difference **B** creating **C**. Finally it throws away the relations **A** and **B** and renames **C** as **A**.

2.4.2 Binary Operations

The standard Relational Algebra operations union, difference, and cartesian product are supported.

Union (see 3.7) as expected returns a relation containing all (subject to key restrictions, see 2.5) of the tuples from both relations.

Difference (see 3.8) returns a relation containing all the tuples in the first relation which are not in the second relation.

Cartesian Product is actually supported as a degenerate case of **EquiJoin** (see 3.9) . **EquiJoin** takes pairs of attribute names from the first and second relation, and performs a join operation using the Perl `eq` operator. If no attribute name pairs are given this degenerates to a cartesian product.

2.5 Keys

A key for a relation is a minimal set of attributes of the relation such that if two tuples agree on those attributes, then they agree on all the attributes of the relation. Because relations are *sets,* if two tuples agree on all the attributes of the key then they are in fact the exact same tuple.

The **CreateTable** (see 3.2) operation allows the user to specify which attributes form the key. In the worst case, the user may specify that all of the attributes together form a key. At least one attribute must be specified.

Bill's Relational Algebra Database enforces the key property. Any attempt to add a tuple for which all the key attributes agree with an existing tuple is silently converted to an update of the existing tuple.

Chapter 3

Actual Interface

The following sections describe each of the public functions in *Bill's Relational Algebra Database* which an application programmer might use.

3.1 OpenB_RA_Db

The **OpenB_RA_Db** function should be called before calling any of the other public functions. It initializes various internal database variables. It takes an optional directory prefix (/tmp by default) and will create all the temporary relations under that directory.

Example The following code fragment would initialize the database and set `/tmp` as the temporary directory.

```
&OpenB_RA_Db('/tmp');
```

3.2 CreateTable

This function creates a database table.

The user supplies references to

$table_str[1] The name of this table. To avoid any possible name conflicts, the program prepends

[1]In Perl the "$" character indicates a "scalar variable" which holds a single value. This value can be either a string or a number. Perl will transparently convert between string and numeric representations as needed. Each attribute stored in *Bill's Relational Algebra Database* is stored as such a "scalar variable."

"`T_`" to this name, thus if the user does not use "`_`" user and private table names cannot clash.

$table_dir The name of the directory containing files for this table.

@attributes[2] A list of all attributes for this table.

@indexed A list of those attributes which have secondary indexes. This list may be empty. *If key is a singleton, then key is removed from* **@indexed** *if it was present, because singleton keys are always specially indexed.*

@key A list of those attributes which together compose a key. These attributes do not need to be indexed, and this list may be identical to **@attributes.** However this list must not be empty, and most of the attributes which compose a key should normally also be declared as indexed. If none of the attributes which compose a key are indexed, then the database will need to check every tuple in the relation during updates and inserts, to find key conflicts. There is a performance advantage in having a singleton key.

The **CreateTable** function returns a reference to a relation (basically a user opaque handle). Currently we simply return the name of the table, but user code should not take advantage of that fact.

Example The following code fragment will create a new table called `WWWFile`. The new table will have five attributes, `inode`, `path`, `filename`, `size`, and `date`. The attribute `inode` will be the key. Because the key is a singleton, the table will have a primary index using the key. This table will also have secondary indexes for `path` and `filename`. The variable `$WWWFile_handle` can be used

[2] In Perl the "@" character indicates an array indexed by number. Such variables may also be treated as lists.

to refer to this table in the future. The function arguments `*WWWFile_attributes`,[3] `*File_index`, and `*WWWFile_key` are references to the corresponding arrays.

```
$WWWFile_name = "WWWFile";

$WWWFile_dir = '/usr/local/www/webmaster/TABLES/WWWFile';

@WWWFile_atttributes = ("inode","path","filename","size","date");

@WWWFile_indexed =  ("inode","path","filename");

@WWWFile_key = ("inode");


$WWWFile_handle = &CreateTable(

                              $WWWFile_name,

                              $WWWFile_dir,

                              *WWWFile_atttributes,

                              *WWWFile_indexed,

                              *WWWFile_key);
```

3.3 OpenTable

OpenTable opens an existing table created during a previous run by a call to **CreateTable** (see 3.2). It takes the name of the directory containing files for this table, and returns a opaque handle to the table.

Example The following code fragment will reopen the table created by a previous call to **CreateTable.** The `$WWWFile_handle` variable can be used in the future to access this table.

[3] In Perl the "*" character indicates a Perl **reference** which permits passing structures by reference instead of by value.

```
$WWWFile_dir = '/usr/local/www/webmaster/TABLES/WWWFile';

$WWWFile_handle = &OpenTable($WWWFile_dir);
```

3.4 AddRow

AddRow adds *or if the* key *attributes are unchanged, updates* a row of the table. It takes two arguments, a Perl reference to an associative array of *attribute* $\mapsto$ *value* pairs, and a table handle such as that previously returned by a call to **CreateTable** (see 3.2).

If a user wants one of the other functions such as **EquiJoin** (see 3.9) or **Difference** (see 3.8) to return their results in a permanent table, instead of a temporary table, then the user may simply create a permanent table, and pass to the function `"main\'AddRow",$table_handle` as the optional function reference with optional arguments.

Example This code fragment will update an existing row of the table, if there is a row with `inode` equal to `12345`. Otherwise it will add the new row contained in the variable `%newRow`[4] to the table.

```
%newRow = ();      # Clear this associative array

                   # then set the needed fields for one row of the table

$newRow{"inode"}    = '12345';

$newRow{"path"}     =  '/bogus/path/';

$newRow{"filename"} = 'aName';

$newRow{"size"}     = '987';
```

[4]In Perl the "%" character indicates an associative array, which can be thought of as an array indexed by string instead of by number.

```
$newRow{"date"}     = '54283420';

&AddRow(*newRow,$WWWFile_handle);  # Add (or update) the row.
```

3.5 QueryByExample

QueryByExample takes a relation handle, and associative arrays specifying the desired selection, and projection. **QueryByExample** also optionally takes a function reference with function arguments. If given a function reference, it invokes `&theFunction(*theRow,@_)` for each selected and projected row, where `*theRow` is a reference to an associative array for this row, and `@_` is the list of optional function arguments the user provided. Otherwise it creates a temporary relation, and returns a handle to the temporary relation.

Example In the following example, because no function reference is given, **QueryByExample** will create a new temporary table. Based on the values of `%projection`, the new table will have two attributes, `number` and `filename`. **QueryByExample** will select all attributes which match the `%selection` criteria. In this case that will be all rows with `path` equal to the `/bogus/path/` string. It will store the `inode` attribute value under the name `number` in the new table, and the `filename` attribute value again under the name `filename` in the new table. Finally it will return a handle to the new table.

```
%selection = ();  # Clear this associative array

$selection{"path"} = '/bogus/path/';
```

```
                  # Initialize this associative array, equivalent to
                  # %projection = (); $projection{"inode"} = "number"; ...
%projection = ("inode", "number",
               "filename","filename",
              );

$NewHandle = &QueryByExample($WWWFile_handle,*selection,*projection);
```

3.6 DeleteByExample

The function **DeleteByExample** takes a relation handle, and associative arrays specifying the desired selection, and projection. It ignores projection (which is actually optional), and removes from the table any rows which match the selection criteria.

Example This example will remove any rows from the table `$WWWFile_handle` that have `path` equal to the `/bogus/path/` string.

```
%selection = ();  # Clear this associative array

$selection{"path"} = '/bogus/path/';

&DeleteByExample($WWWFile_handle,*selection);
```

3.7 Union

The **Union** function takes two relation handles, and two pairs of associative arrays specifying the desired selection, and projection. It optionally also accepts a function reference. If given a function reference, it invokes `&theFunction(*theRow,@_)` for each selected and projected row. Otherwise it creates a temporary relation, and returns a handle to the temporary relation. If the same row is listed in both relations, `theFunction` will be called twice for that row. However by the key property, the row will only appear once in a temporary relation. Thus if having `theFunction` called twice is a potential problem, the user should have **Union** return a temporary relation, then pass `theFunction` to **QueryByExample** (see 3.5) with an empty selection condition.

Example Using our `$WWWFile_handle` from the **DeleteByExample** (see 3.6) example, and our `$NewHandle` table from the **QueryByExample** (see 3.5) example, the following code fragment will return a temporary table containing only the attributes `inode` and `filename`. The resulting table will contain all the rows from both `$WWWFile_handle` and `$NewHandle`.

```
%wwwSelection = ();
%wwwProjection = ("inode", "inode",
                  "filename","filename",
                 );


%newSelection = ();
%newProjection = ("number", "inode",
                  "filename","filename",
                 );
```

```
$anotherHandle = &Union($WWWFile_handle,*wwwSelection,*wwwProjection,
                        $NewHandle,*newSelection,*newProjection);
```

3.8 Difference

Difference takes two relation handles, and two pairs of associative arrays specifying the desired selection, and projection. It also optionally takes a function reference. It returns anything which is in the first relation but not in the second relation. Again if given a function reference, it invokes `&theFunction(*theRow,@_)` for each selected and projected row. Otherwise it creates a temporary relation, and returns a handle to the temporary relation.

Example The following code fragment starts by creating a table of only those files which are named `core` using **QueryByExample** (see 3.5) as we have seen before. Then we use the **Difference** function to create a table consisting of all files not named `core`.

```
%projection = ("inode","inode",
               "path","path",
               "filename","filename",
               "size","size",
               "date","date");
%selection = ();  # Clear this associative array
$selection{"filename"} = 'core';


$justCores = &QueryByExample($WWWFile_handle,*selection,*projection);
```

```
%selection = ();  # Clear this associative array
%projection2 = ("inode","inode",
               "path","path",
               "filename","filename",
               "size","size",
               "date","date");
%selection2 = ();  # Clear this associative array

$noCores = &Difference($WWWFile_handle,*selection,*projection,
                        $justCores,*selection2,*projection2);
```

3.9 EquiJoin

EquiJoin takes two relation handles, and two pairs of associative arrays specifying the desired selection, and projection. **EquiJoin** also requires a (possibly empty) map from attribute to attribute to join on, final projections for each relation (the values should not intersect), and optionally a function reference. *If the join map is empty,* **EquiJoin** *degenerates into a simple cross product.* Again if given a function reference, **EquiJoin** invokes `&theFunction(*theRow,@_)` for each selected and projected row. Otherwise it creates a temporary relation, and returns a handle to the temporary relation.

Example This example starts by opening two existing tables. It selects everything from both tables. It arbitrarily projects the original attributes to use shorter names. It then joins the two tables on the dept attribute. Next it projects the attributes again to arbitrary names, but this time it does not project the dept attribute. Finally we provide a print function which is called for each joined row.

```
$empHandle = &OpenTable('/database/tables/EmpDept');
%empSel = ();
%empProj = ("employee","name",
            "department","dept");
%empFinalProj = ("name","person");

$locHandle = &OpenTable('/database/tables/LocDept');
%locSel = ();
%locProj = ("location","loc",
            "department","dept");
%locFinalProj = ("loc","place");

%crossMap = ("dept","dept");

&EquiJoin($empHandle,*empSel,*empProj,$locHandle,*locSel,*locProj,
          *crossMap,*empFinalProj,*locFinalProj,"main\'printRow");
exit;

#
# Subroutine printRow prints the person and place attributes of
```

```
# the associative array aRow
#
sub printRow {

  (*aRow) = @_;   # Name the function argument.

  print aRow{"person"}, "\t", aRow{"place"}, "\n";
}
```

3.10 CloseB_RA_Db

CloseB_RA_Db closes all open gdbm files, and removes any temporary relations created since the call to **OpenB_RA_Db** (see 3.1). It does not take any arguments.

Example This trivial example closes all open tables, and removes any temporary storage created since calling **OpenB_RA_Db** (see 3.1) at the beginning of this program.

```
&CloseB_RA_Db();
```

Chapter 4

Implementation Limits

Bill's Relational Algebra Database is a Perl library which uses Perl's associative array interface to gdbm to store its data. Perl is normally compiled to support gdbm on systems which have gdbm installed. "The size in gdbm is not restricted like dbm or ndbm. Your data can be as large as you want."[Fre93] You can literally store and retrieve /vmunix in an attribute, even in an INDEXED attribute! **ShowOff.pl** (see appendix A) has an example which does just that!

Table names, and attribute names should be strictly alpha-numeric. If you use the "_" character there is a theoretical possibility of a name clash with the library's internal variables.

Enforcing the key property can be extremely expensive. A key consists of some user specified set of a table's attributes. However the user is not required to declare any of these attributes as indexed. Thus it is possible for none of the key's attributes to be indexed, which makes updates and insertions extremely expensive. However the database uses intersection of all applicable indexes to narrow all its searches, so if some of a key's attributes are indexed, enforcing the key property is not excessively expensive.

The database uses many open files, at least two per table, plus two per attribute, plus two per index for every open table (including temporary relations). This is primarily an artifact of using a separate gdbm associative array for each attribute. Under Perl's current implementation, each such array requires two open files.

Bill's Relational Algebra Database was written as a light weight single user database. It does not support multiple users, does not perform any locking, does not keep a log, and does not understand the idea of a transaction.

Chapter 5

Usage Experience

The original project which motivated writing *Bill's Relational Algebra Database* was an application to automatically collect information from the University of Illinois Smalltalk ftp archive[STf], place that information in a database, generate reports based on that data, and make those reports available via an http server[STw]. For a more detailed description of the entire application, including a complete schema description, see [Vos]. The project was a success, and *Bill's Relational Algebra Database* fulfilled most of my expectations, though an in-memory version would probably have been both sufficent and more efficient.

A cron job updates the database, and generates a new report nightly. Regretably neither *Bill's Relational Algebra Database* nor the project application code[Vos] is bug free yet. For some as yet unknown reason, three times in the past five months, the system has gotten into a state where no database report is generated. Because of time constraints, the brute force workaround has been to simply totally reset the system.

I have also found a case where a database index file kept growing to unreasonable sizes. Examination of the file indicated that it contained only the expected values, so I currently believe this was a hash collision problem within Perl or the gdbm library. The workaround was simply to stop indexing the problem attribute. Other than those two problems, the system has worked beautifully; though it is rather slow, and requires considerable disk space.

The ftp archive currently contains 1,927 files. Many of those files are described by a **CATALOG-CARD** file, or by having an embedded comment within the file itself. The information stored in a **CATALOG-CARD** normally includes an application's name, author, function, a list of compatible Smalltalk versions, any prerequisites, a version number, and a version release date. This catalog information is collected and stored in the database. Some of the files in the database are not yet cataloged, and many applications consist of multiple files; however, the ftp archive currently contains 290 cataloged applications,

Traversing the entire ftp archive and updating the database currently takes roughly thirty minutes each night, largely because some brute force algorithms were used which delete then regenerate a large amount of data.

After updating the database, the **report_AppName.pl** program (presented in appendix B) generates a roughly 100,000 byte html report listing every application for which catalog information is available. Doing this involves first an **EquiJoin** (see 3.9) between a table of application descriptions and a table of Smalltalk versions, so that all descriptions will eventually use canonical version strings. For example "Smalltalk-80 Release 4" might be the cannonical version string for "ST80 R4," "R4," "Release Four," etc.[1] After joining the descriptions with the Smalltalk versions, **report_AppName.pl** invokes **EquiJoin** on the resulting temporary table and a permanent table of all files in the archive to create a second temporary table. Again using the original table of descriptions, **report_AppName.pl** performs a **QueryByExample** (see 3.5) to generate a set of just the application names. The program sorts this set, then for each element in the set performs a **QueryByExample** against the second temporary table. This second **QueryByExample** is

[1] Until a human enters the cannonical mapping for a version string, the original string is used. No cannonical mappings have been entered yet, so the Smalltalk version table currently consists of one entry for every raw version string.

passed a function argument. The function prints out an html fragment which includes a hypertext link to the full description file, descriptions of the application's function, application version, application date, author, and the cannonical Smalltalk versions it supports. Finally the fragment also contains a hypertext link to the actual application code. Generating this report requires between two and three minutes on a Sun SparcStation 10. These times are slow but acceptable because this application was always intended for a batch processing environment.

Chapter 6

Future Work

If *Bill's Relational Algebra Database* is enhanced in the future the enhancements might include the following.

6.1 Adaptive Indexing

For many tables, a given column of the table contains many discrete values, and a few values which are very very common. An index based on that column is a big win for the uncommon values, but can be very expensive for the common values.

For example an index based on the column SEX might be a great win for the value hermaphrodite, but worse than useless for the values male and female.

An adaptive indexing scheme would start by indexing all values, but would then stop indexing any values which proved to be very common in actual use. This would provide nearly the best of both worlds.

6.2 More Support Functions

It would be nice if the system provided more tiny programmer support functions to handle common operations the programmer must currently handle manually.

When calling a function such as **QueryByExample** (see 3.5), **DeleteByExample** (see 3.6), or **Difference** (see 3.8) the programmer usually simply wants to project all attributes. We saw this in the **Difference** example. A function which created the required associative array given a table handle would be convenient.

6.3 In-Memory Version

An in-memory version of *Bill's Relational Algebra Database* which loaded entire tables into memory on open, and wrote them back out to disk on close would certainly run faster, and require less disk space. For the http server[STw] application[Vos] described in chapter 5 such an in-memory version would almost certainly be sufficient, and more efficient.

Appendix A

Program ShowOff.pl

The following code stores and retrieves the /vmunix binary in the database.

```
#!/usr/bin/perl

#
# This little program is just a show off.
# I read /vmunix into a string
# I write the string to /tmp/vmunix
# and I compare /tmp/vmunix with /vmunix
#
# Then I store the same string in my database (under an INDEXED attribute!)
# I find it in the database and write it to /tmp/big2
# and then I compare /tmp/big2 with /vmunix
#

open(VMUNIX,"</vmunix") || die "Can't open /vmunix";

($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,
 $atime,$mtime,$ctime,$blksize,$blocks) = stat(VMUNIX);

print "stat says the file is $size\n";

$readBytes = read(VMUNIX,$big,$size);

print "Actually read in $readBytes\n";

close(VMUNIX);

open(OUTPUT,">/tmp/vmunix") || die "Can't open /tmp/vmunix";

print OUTPUT $big;

close(OUTPUT);

print "cmp says: ",`cmp /vmunix /tmp/vmunix`,"\n";

`rm /tmp/vmunix`;
```

```
print "Now to store it in the database!\n";

require 'B_RA_Db.pl';

&OpenB_RA_Db('/tmp');

$Test_dir = "/tmp/vmunix_test";
`rm -rf $Test_dir`;
mkdir($Test_dir,16877) unless (-e $Test_dir);

$V_name = "Vtable";
$V_dir = "$Test_dir/$V_name";
@V_columns = ("name","file");
@V_indexed = ("name","file");   # Want to be able to lookup fast by binary :-)
@V_key = ("name","file"); # There may be multiple binaries of same name.

$V_table = &CreateTable($V_name,$V_dir,*V_columns,*V_indexed,*V_key);

print "Table is created, now to build the row\n";

%newRow = ();
$newRow{"name"} = "/vmunix";
$newRow{"file"} = $big;

&AddRow(*newRow,$V_table);

print "The new row is now stored.\n";

$Big2 = '';

# Setup for query by example
%selection = ();
$selection{"name"} = "/vmunix";

%projection = ("name","name",
       "file","file");

print "Performing the query\n";

&QueryByExample($V_table,*selection,*projection,"main\'storeInBig2");

print "Back from the query\n";

`rm -f /tmp/big2`;
```

```
open(TMP_BIG2,">/tmp/big2");

print TMP_BIG2 $Big2;

print "And this time cmp reports: ",`cmp /vmunix /tmp/big2`,"\n";

print "Now I'll perform a query using the binary as our key!\n";

# Setup for query by example
%Bselection = ();
$Bselection{"file"} = $big;
%Bprojection = ("name","name");   # We don't need to return file this time

&QueryByExample($V_table,*Bselection,*projection,"main\'printName");

print "Back from the query using the binary as our key!\n";


print "Don't forget to remove /tmp/big2 and $Test_dir/ after you are done!\n";

&CloseB_RA_Db();

exit;

sub storeInBig2 {

    local(*aRow) = @_;

    $Big2 = $aRow{"file"};
    local($name) = $aRow{"name"};
    print "storeInBig2 just stored the file for $name\n";
}

sub printName {

    local(*aRow) = @_;

    local($name) = $aRow{"name"};
    print "printName was given the name of $name\n";
}
```

Appendix B

Sample Report Generator

The program report_AppName.pl generates the html file listing all applications in the Smalltalk ftp archive[STf][STw] for which catalog information is available.

This appendix contains the source for report_AppName.pl and the source for those files it directly includes using the "require" statement. For complete source listings of the files they include, and an overview of the database schema see [Vos].

B.1 report_AppName.pl

```
#!/usr/bin/perl -w

require 'upcaseCompress.pl';
require 'find_CAT.pl';
require 'constants.pl';
require 'report_lib.pl';
require 'mv_lib.pl';

&report_AppName;
exit;

exit if ($Default_DescInode);
&StoreITIn if (0); # To keep "perl -w" happy
&reportThisApp if (0); # To keep "perl -w" happy
############################################################################
############################################################################
##
## TODO
##
## The next report should fully support multiple clean versions.
## (This one knows that the join with clean version gives a single match.)
##
## The next version should perhaps list all files, and be aware of
## possible ../dir.tar.Z files.
############################################################################
############################################################################
#
# report_AppName
#
```

```
# This is my very first attempt at writing reports.
#
sub report_AppName {

    die "Hey where are the tables?\n"
unless (($FtpFile_dir) &&
($Description_dir) &&
($RawVersionToClean_dir));

    &OpenB_RA_Db('/tmp'); # Don't really care where.

    local($FtpFile) = &OpenTable($FtpFile_dir);
    local($Description) = &OpenTable($Description_dir);
    local($RawVersionToClean) = &OpenTable($RawVersionToClean_dir);

    # Get a version of Description with clean version strings

    local(%RawSel) = ();
    local(%RawProj) = ("VersionUpcased", "VersionUpcased",
       "CleanVersion", "CleanVersion");
    local(%RawFinal) = ("CleanVersion", "CleanVersion");

    local(%DescSel) = ();
    local(%DescProj) = ("DescInode","DescInode",
"APPNAMEUpcased","APPNAMEUpcased",
"AppName","AppName",
"VersionUpcased","VersionUpcased",
"Author","Author",
"VersionNum","VersionNum",
"Date","Date",
"Function","Function",
);
    local(%DescFinal) = ("DescInode","DescInode",
 "APPNAMEUpcased","APPNAMEUpcased",
 "AppName","AppName",
 "Author","Author",
 "VersionNum","VersionNum",
 "Date","Date",
 "Function","Function",
 );
    local(%RawDescCross) = ("VersionUpcased","VersionUpcased");

    local($clean_intermediate_desc) = &EquiJoin(
   $RawVersionToClean,
   *RawSel,*RawProj,
```

```
   $Description,
   *DescSel,*DescProj,
   *RawDescCross,
   *RawFinal,*DescFinal);

    #
    # Now our descriptions use the "Clean Version" strings.
    # Next we join them with FtpFile
    #

    local(%cid_sel) = ();
    local(%cid_proj,%cid_final_proj);
    foreach $val (values(%RawFinal)) {
$cid_proj{$val} = $val;
$cid_final_proj{$val} = $val;
    }
    foreach $val (values(%DescFinal)) {
$cid_proj{$val} = $val;
$cid_final_proj{$val} = $val;
    }

    local(%ftp_sel) = ();
    local(%ftp_proj) = (
"inode","DescInode",
"path","path",
"filename","filename",
);
    local(%ftp_final_proj) = (
"path","path",
"filename","filename",
);

    local(%join_map) = (
    "DescInode","DescInode",
    );

    local($clean_desc) = &EquiJoin(
   $clean_intermediate_desc,
   *cid_sel,*cid_proj,
   $FtpFile,
   *ftp_sel,*ftp_proj,
   *join_map,
   *cid_final_proj,*ftp_final_proj);

    #
```

```
    # Lets fetch the upcased app names, sort them, then gen a report
    #

    local(%AllApps);
    die "Where do the temporary NewDesc files go?" unless ($tmp_new_desc);
    dbmopen(%AllApps,$tmp_new_desc,0666);
    %AllApps = ();

    local(%AppSel) = ();
    local(%ITProj) = ("APPNAMEUpcased","IT");

    &QueryByExample($Description,
   *AppSel,
   *ITProj,
   "main\'StoreITIn",
   *AllApps);

    #
    # Now generate the report
    #
    local(%CleanSel) = ();
    local(%CleanProj) = ("CleanVersion", "CleanVersion",
"DescInode","DescInode",
"AppName","AppName",
"Author","Author",
"VersionNum","VersionNum",
"Date","Date",
"Function","Function",
"path","path",
"filename","filename",
);

    die "Where is \$report_one_header" unless ($report_one_header);

    `rm -f $report_one_file.tmp`;
    `cp $report_one_header $report_one_file.tmp`;

    # Used to keep track of <A NAME=> tags in a page.
    local(%html_NAMES);
    die "Where do the temporary html NAMES files go?" unless ($tmp_html_NAMES);
    dbmopen(%html_NAMES,$tmp_html_NAMES,0666);
    %html_NAMES = ();

    open(REPORT_FILE,">>$report_one_file.tmp") || die "Can not open report file";
```

```
    foreach $app (sort(keys(%AllApps))) {  # Hear the memory groan!

$CleanSel{"APPNAMEUpcased"} = $app;

&QueryByExample($clean_desc,
*CleanSel,
*CleanProj,
"main\'reportThisApp",
$FtpFile,
*REPORT_FILE,
*html_NAMES
);
    }

    dbmclose(%AllApps);
    `rm -f $tmp_new_desc.pag $tmp_new_desc.dir`;

    dbmclose(%html_NAMES);
    `rm -f $tmp_html_NAMES.pag $tmp_html_NAMES.dir`;

    &standardTail(REPORT_FILE,
    "<A HREF=\"http://st-www.cs.uiuc.edu:/webmaster/LC/report_AppName.pl.html\">
    report_AppName.pl</A>");

    close(REPORT_FILE);

    &mv_or_error("$report_one_file","$report_one_file.tmp");
}
############################################################################
sub StoreITIn {

    local(*aRow,*Result) = @_;

    $Result{$aRow{"IT"}} = 1 if ($aRow{"IT"});
}
############################################################################
die "Hey, where is \$FtpFile_url" unless (defined $FtpFile_url);

sub reportThisApp {

    local(*aRow,$FtpFile,*FILE,*NAMES) = @_;

    return unless (($aRow{"AppName"}) && ($aRow{"AppName"} ne $null));

    return if ($aRow{"DescInode"} eq $Default_DescInode);
```

```
    print FILE "<HR><H2><A ";
    unless (defined $NAMES{$aRow{"AppName"}}) {
print FILE "NAME=\"",$aRow{"AppName"},"\" ";
$NAMES{$aRow{"AppName"}} = 1;
    }
    print FILE "HREF=\"$FtpFile_url",$aRow{"path"},$aRow{"filename"};
    print FILE "\">",$aRow{"AppName"},"</A></H2>\n<UL>\n";

    if (($aRow{"Function"}) && ($aRow{"Function"} ne $null)) {
print FILE "<LI> ",$aRow{"Function"},"\n";
    }
    if (($aRow{"VersionNum"}) && ($aRow{"Date"} ne $null) &&
($aRow{"VersionNum"} ne $null) && ($aRow{"Date"} ne $null)) {

print FILE "<LI> Version ",$aRow{"VersionNum"},
" Dated ",$aRow{"Date"},"\n";
    }
    if (($aRow{"Author"}) && ($aRow{"Author"} ne $null)) {
print FILE "<LI> By ",$aRow{"Author"},"\n";
    }
    if (($aRow{"CleanVersion"}) && ($aRow{"CleanVersion"} ne $null)) {
print FILE "<LI> For ",$aRow{"CleanVersion"},"\n";
    }
    print FILE "</UL>\n";

    local($filename) = $aRow{"filename"};
    local($matchFile);

    if ($filename =~ m/^(.+).CAT$/o) {
$matchFile = $aRow{"path"} . $1;
    } elsif  ($filename =~ m/^(.+).st$/o) {
$matchFile = $aRow{"path"} . $1 . ".st";
    } elsif  ($filename =~ m/^CATALOG-CARD$/o) {
$matchFile = $aRow{"path"};
    } else {
# Couldn't understand the match, oh well hope.

$matchFile = $aRow{"path"} . $filename;
    }
    if ($matchFile) {
print FILE "<A HREF=\"$FtpFile_url",$matchFile,"\">$matchFile</A>\n\n";
    }

}
```

```
###############################################################################
1;
```

B.2 constants.pl

```
#!/usr/bin/perl -w
#############################################################################
#
# For me this should be ~ftp/pub
#
$base_of_archive = "/speedy/nfs/root/archive/ftp/pub";
$base_of_WWW = "/usr/local/www";
#############################################################################
$null = "-null-";
#############################################################################
$report_one_file = "$base_of_WWW/REPORTS/report_by_AppName.html";
$report_one_header = "$base_of_WWW/REPORTS/report_by_AppName.header";
#############################################################################
$Default_DescInode = "1";   # Historically bad blocks were linked to inode 1
$Default_DescInode_FullName = "/default.CATALOG-CARD";
$Default_DescInode_Size = "1";
$Default_DescInode_Date = "1";
#############################################################################
#     I assume all the permanent data files will be under the same
#     directory.  What directory is that?
#
# See: CreateTables.pl
#

$directory_prefix = '/usr/local/www/webmaster/TABLES';

$FtpFile_dir = "$directory_prefix/FtpFile";
$FtpFile_url = "ftp://st.cs.uiuc.edu:/pub";
$WWWFile_dir = "$directory_prefix/WWWFile";
$Description_dir = "$directory_prefix/Description";
$RawVersionToClean_dir = "$directory_prefix/RawVersionToClean";
$CannonicalVersions_dir = "$directory_prefix/CannonicalVersions";
#############################################################################
#
# These variables specify where the find output goes.
#
# See: file_find.pl and CreateTables.pl
#
$flat_directory_prefix = "$directory_prefix/flat_data";
```

```
$flat_fname_old = "$flat_directory_prefix/fname.old";
$flat_fname_new = "$flat_directory_prefix/fname.new";
$flat_fsize_old = "$flat_directory_prefix/fsize.old";
$flat_fsize_new = "$flat_directory_prefix/fsize.new";
$flat_fmtime_old = "$flat_directory_prefix/fmtime.old";
$flat_fmtime_new = "$flat_directory_prefix/fmtime.new";

$flat_delete = "$flat_directory_prefix/delete.assoc";
$flat_update = "$flat_directory_prefix/update.assoc";
$flat_add = "$flat_directory_prefix/add.assoc";
###########################################################################
$tmp_new_desc = "$flat_directory_prefix/tmp_new_desc";
$tmp_html_NAMES = "$flat_directory_prefix/tmp_html_NAMES";
###########################################################################
$mgr_error_file = "$flat_directory_prefix/mgr_errors";
###########################################################################
$bogus_inode = $null;

1;
```

B.3 find_CAT.pl

```
#!/usr/bin/perl -w

require 'add_delete_lib.pl';
require 'parse_CAT.pl';
require 'constants.pl';

&update_DescInode if (0); # To keep "perl -w" happy
##########################################################################
#
# update_DescInode
#
# I figure out which Description to use, and start the process
# of adding the Description information if necessary.
#
sub update_DescInode {

    local(*ThisFile,$FtpFile,*NewDescriptions) = @_;

    die "Hey, where is the inode?" unless ($ThisFile{"inode"});

    local(%Project_FtpFile) = (
       "inode","inode",
       "path","path",
```

```
        "filename","filename",
        "size","size",
        "date","date",
        "DescInode","DescInode",
        );

    local(%DescFile) = ();

    &find_CAT($FtpFile,*ThisFile,*Project_Ftp_File,*DescFile);

    $ThisFile{"DescInode"} = $DescFile{"inode"};

    &AddRow(*ThisFile,$FtpFile);  # Update the entry for this file.

    # Make a note of this file, it may need to be added to Descriptions

    $NewDescriptions{$DescFile{"inode"}} = 1;
}

############################################################################
#
# find_CAT
#
# We find the appropriate file in $FtpFile, and by
# side-affect return the row in %Result.
#
sub find_CAT {

    local($FtpFile,*ThisFile,*Project_Ftp_File,*Result) = @_;

    # Is there a filename.CAT file?

    local(%CAT_Match) = ("path", $ThisFile{"path"},
 "filename" , ($ThisFile{"filename"} . '.CAT'));

    &QueryByExample($FtpFile,
    *CAT_Match,
    *Project_FtpFile,
    "main\'returnRow",
    *Result);

    return if ($Result{"inode"});

    # Is there a CATALOG-CARD file?
```

```
    local(%CARD_Match) = ("path", $ThisFile{"path"},
  "filename", "CATALOG-CARD");

    &QueryByExample($FtpFile,
    *CARD_Match,
    *Project_FtpFile,
    "main\'returnRow",
    *Result);

    return if ($Result{"inode"});

    # Ok, does the file contain its own descriptive information?

    die "Where is the archive?" unless ($base_of_archive);

    local($fullName) = $base_of_archive .
               $ThisFile{"path"} .
       $ThisFile{"filename"};

    local(%CatInfo) = ();

    &parse_CAT'parseFile($fullName,*CatInfo);

    if (($CatInfo{"AppName"}) &&
        ($CatInfo{"Author"}) &&
        ($CatInfo{"Function"}) &&
        ($CatInfo{"Version"})) {  # The essentials

foreach $key (keys(%ThisFile)) {

    $Result{$key} = $ThisFile{$key};
}

return;
    }

    # I guess not, must return the default

    die "What is the \$Default_DescInode?" unless ($Default_DescInode);

    local(%Default_Match) = ("inode", $Default_DescInode);

    &QueryByExample($FtpFile,
    *Default_Match,
    *Project_FtpFile,
```

```
    "main\'returnRow",
    *Result);

    return if ($Result{"inode"});

    die "This should never happen, we could not even find Default!";
}
############################################################################
1;
```

B.4 mv_lib.pl

```
#!/usr/bin/perl -w

###############################################################################
#
# mv_or_error
#
# We take an old filename, and a new filename.
# If the new file is about the same size, or larger than
# the old file, we remove the old file, and replace
# it with the new file.  Otherwise we send the webmaster
# an error message.

sub mv_or_error {

    local($oldFile,$newFile) = @_;

    local($oldSize) = -s $oldFile;

    local($newSize) = -s $newFile;

    print "$oldFile is $oldSize\n$newFile is $newSize\n";

    if ($newSize > ($oldSize * 0.9)) {
`rm -f $oldFile`;
`mv $newFile $oldFile`;
    } else {
`rm -f $oldFile.mv_lib.pl_error`;
`mv $newFile $oldFile.mv_lib.pl_error`;

open(MAIL,"| /usr/lib/sendmail -Fmv_lib.pl webmaster@st-www.cs.uiuc.edu");
print MAIL "To: webmaster@st-www.cs.uiuc.edu\n";
print MAIL "Subject: mv_or_error problem ($0) \n";
print MAIL "\n";
```

```
print MAIL "\$oldFile == $oldFile\n";
print MAIL "\$oldSize == $oldSize\n";
print MAIL "\$newFile == $newFile\n";
print MAIL "\$newSize == $newSize\n";
print MAIL "So $oldFile was not replaced by $newFile\n";
print MAIL "Instead $newFile was moved to $oldFile.mv_lib.pl_error\n";
print MAIL "\n";
close(MAIL);
    }
}
###############################################################################
1;
```

B.5 report_lib.pl

```
#!/usr/bin/perl -w

###############################################################################
#
# today
#
# Simply return a pretty string representing today's date.
#
sub today {
    local(@local) = localtime(time);
    local($day) = ("Sunday","Monday","Tuesday","Wednesday","Thursday",
   "Friday","Saturday")[$local[6]];
    local($month) = ("January",
     "February",
     "March",
     "April",
     "May",
     "June",
     "July",
     "August",
     "September",
     "October",
     "November",
     "December")[$local[4]];
    local($mday) = $local[3] .
("th","st","nd","rd","th","th","th","th","th","th",
 "th","th","th","th","th","th","th","th","th","th",
 "th","st","nd","rd","th","th","th","th","th","th",
 "th","st","nd","rd","th","th","th","th","th","th",
 "th","st","nd","rd","th","th","th","th","th","th",
```

```
 "th","st","nd","rd","th","th","th","th","th","th",
 "th","st","nd","rd","th","th","th","th","th","th",
 "th","st","nd","rd","th","th","th","th","th","th",
 "th","st","nd","rd","th","th","th","th","th","th")[$local[3]];
    local($year) = $local[5] + 1900;

    local($today) = "$day, $month $mday, $year";

    return($today);
}
###############################################################################
#
# standardTail
#
# Write a standard format tail for an html file.
#
# $creator is hopefully something like
#             <A HREF="blabla/prog#function>prog function</A>
#
sub standardTail {

    local(*FILE,$creator) = @_;

    local($today) = &today;
    print FILE "\n<P><HR>This page created by $creator on $today\n";
    print FILE "<address>The <A HREF=\"http://st-www.cs.uiuc.edu:/\">";
    print FILE "Smalltalk Archive</A> is maintained by ";
    print FILE "<A HREF=\"http://st-www.cs.uiuc.edu:/users/voss/BillVoss.html\">";
    print FILE "Bill Voss &ltwebmaster@st-www.cs.uiuc.edu&gt</A></address>\n";
    print FILE "\n</BODY>\n";
}
###############################################################################
1;
```

B.6 upcaseCompress.pl

```
#!/usr/bin/perl -w

###############################################################################
#
# We take a string, chop white space down to single spaces,
# and convert any letters to upper case
#
sub UpcaseCompress {
```

```
    local($theString) = @_;

    $theString =~ tr/ \t\n,/,,,,/s; # Squeeze whitespace to comma
    $theString =~ tr/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/;

    return($theString);
}
###############################################################################
1;
```

Bibliography

[Cod70] E. Codd. A relational model for large shared data banks. *CACM*, 13(6), June 1970.

[EN89] Ramez Elmasri and Shamkant B. Navathe. *Funadamentals of Database Systems*, pages 135–173. The Benjamin/Cummings Publishing Company, Inc., 390 Bridge Parkway Redwood City, California 94065, 1989.

[Fre93] Free Software Foundation, Inc. *Unix man page GDBM(3)*, 16 November 1993.

[gdb] Readme file from gdbm-1.7.1. URL file://prep.ai.mit.edu/pub/gnu/gdbm-1.7.1.tar.gz.

[Sch93] Randal L. Schwartz. *Learning Perl.* O'Reilly & Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, first edition, November 1993.

[STf] The University of Illinois Anonymous Ftp Smalltalk Archive. URL file://st.cs.uiuc.edu/.

[STw] The University of Illinois World Wide Web HTTP Server. URL http://st-www.cs.uiuc.edu/.

[Vos] William Eric Voss. *Smalltalk Archive Manager's Base Page.* URL http://st-www.cs.uiuc.edu/webmaster/webmaster.html.

[Vos93] William Eric Voss. *Bill's Relational Algebra Database (base page).* URL http://st-www.cs.uiuc.edu/webmaster/B_RA_Db/B_RA_Db.html, 1993.

[WS91] Larry Wall and Randal L. Schwartz. *Programming perl*. O'Reilly & Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, first edition, January 1991.