NoSQL
A Relational Database Management System

Carlo Strozzi

Italian Linux Society.

v0.9, 4 March 1998

NoSQL is a fast, portable, relational database management system without arbitrary limits, (other than memory and processor speed) that runs under, and interacts with, the UNIX Operating System. It uses the Operator/Stream DBMS paradigm described in "Unix Review", March, 1991, page 24, entitled "A 4GL Language". There are a number of "operators" that each perform a unique function on the data. The "stream" is suplied by the UNIX Input/Output redirection mechanism. Therefore each operator processes some data and then passes it along to the next operator via the UNIX pipe function. This is very efficient as UNIX pipes are implemented in memory. NoSQL is compliant with the "Relational Model".

Contents

Chapter 1

Foreword and Introduction

1.1 Copyright

NoSQL RDBMS, Copyright (C) 1998 Carlo Strozzi, with permission from the original RDB author, W.Hobbs.

This program comes with ABSOLUTELY NO WARRANTY; for details refer to the GNU General Public License.

A copy of the GNU General Public License is included in the appendix, at the end of this document.

1.2 Preface

This working draft describes, and provides instructions for the use of, NoSQL (I personally like to pronounce it *noseequel*), a close derivative of the RDB DataBase system. The original RDB system was (and still is) developed at RAND Organization by Walter V. Hobbs. Most of the NoSQL code, as well as the text of this document, have been taken directly from RDB, so most of the credit for it goes to the original author.

NoSQL uses exactly the same table format as RDB, and therefore tables are called 'rdbtables' also in the NoSQL context.

NoSQL's major differences over the original code are:

- The documentation has been re-written in SGML for the sake of portability.
- All the operators and utilities have been renamed to prevent conflicts with other system commands with the same name.
- Removed some old-fashioned and less portable components, namely the curses/cterm data entry interface and the old RDB terminal interface. A new RDB terminal interface 'rdbi' has now been developed by W.Hobbs and it has been included in NoSQL with the name of 'nsq'.
- NoSQL comes with tools that translate tables between /rdb and RDB/NoSQL formats. This makes it usable with **Dbedit**, a WWW form interface to rdbtables. More details at: http://admin.gnacademy.org:8001/HyperNews/get/tech/dbedit.html
- Added extra utilities and operators.

- NoSQL tends to be biased in favour of Linux. This means that, wherever it applies, NoSQL makes use of the GNU versions of the various UNIX commands, as those are the ones normally found on this UNIX workalike.

Other major contributors to the original RDB system, besides the author, were:

Chuck Bush

Don Emerson

Judy Lender

Roy Gates Rae Starr

1.3 Introduction

A good question one could ask is "With all the relational database management systems available today, why do we need another one ?" There are five reasons. They are:

1. NoSQL is easy to use by non-computer people. The concept is straight forward and logical. To select rows of data, the 'nsq-row' operator is used; to select columns of data, the 'nsq-col' operator is used.
2. The data is highly portable to and from other types of machines, like Macintoshes or MSDOS computers.
3. The system will run on any UNIX machine (that has the PERL Programming Language).
4. NoSQL essentially has no arbitrary limits, and can work where INGRES can't. For example there is no limit on data field size, the number of columns, or file size.

The data is contained in regular UNIX ASCII files, and so can be manipulated by regular UNIX utilities, e.g. ls, wc, mv, cp, cat, more, less, editors like 'vi', head, RCS, etc.

The form of each file of data is that of a relation, or table, with rows and columns of information.

To extract information, a file of data is fed to one or more "operators" via the UNIX Input/Output redirection mechanism.

There are also programs to generate reports, and to generate, modify, and validate the data. A more through discussion of why this type of relational database structure makes sense is found in the book, "UNIX Relational Database Management", Reference #2.

It is assumed that the reader has at least a minimum knowledge of the UNIX Operating System, including knowledge of Input/Outout redirection (e.g., STDIN, STDOUT, pipes).

This document presents information in the following order: The DATA section describes the structure of the data, with examples. There is a general discussion about operators in the section on OPERATORS, followed by several sub-sections, one for each operator in alphabetic order. Each has detailed instructions for use, and examples. There are sections describing selection of information using multiple operators, producing reports, and generating new rdbtables (data files in NoSQL format).

Chapter 2

Data formats

2.1 NoSQL table (rdbtable) structure.

Besides the regular UNIX editors and utilities, a good way to view the data of course, would be to use the NoSQL operator that prints such datafiles: 'nsq-pr' (named after the 'pr' UNIX utility).

The relation, or table structure is achieved by separating the columns with ASCII TAB characters, and terminating the rows with ASCII NEWLINE characters. That is, each row of data in a file contains the data values (a data field) separated by TAB characters and terminated with a NEWLINE character. Therefore a fundamental rule is that data values must NOT contain TAB characters.

The first section of the file, called the header, contains the file structure information used by the operators. The header also contains optional embedded documentation relating to the entire datafile (table documentation) and/or each data column (column documentation). The rest of the file, called the body, contains the actual data values. A file of data, so structured, is said to be an 'rdbtable'.

The header consists of two or more lines. There is an optional number (zero or more) of lines of table documentation followed by exactally two lines that contain the structure information: the column name row and the column definition row. The table documentation lines start with either a sharp sign (#) followed by a space character, or one or more space characters followed by a sharp sign (#). The rest of each line may contain any documentation desired. Note that the table documentation lines are the only lines in an rdbtable that are not required to conform to the table structure defined above. The fields in the column name row contain the names of each column. The fields in the column definition row contain the data definitions and optional column documentation for each column.

The column names are case sensitive, i.e. 'COUNT' is different from 'Count'. The guideline for characters that may be used in column names is that alphabetic, numeric, and non-alphanumeric characters that are not special to the UNIX shell are good choices. Column names must include at least one alphabetic character. It is highly recommended (but not required) that column names start with an alphabetic or numeric character.

Non-alphanumeric characters that are acceptable in column names are the percent sign (%) colon (:) at sign (@) equals (=) comma (,) and dot (.). The sharp sign (#) underscore (_) and dash (-) characters may also be used but they must not be the first character in a column name. The TAB character must never be used in column names, nor should internal spaces or UNIX I/O redirection characters (<,>,|) be used.

The data definitions include column width, data type, and justification. The column width must be explicitly specified; the others are optional and are frequently specified by default.

The data definitions are specified by adjacent characters in a single word. The width of each field is specified by a numeric count. The type of data is "string", "numeric", or "month". The types are specified by an 'S', 'N', or 'M' respectively, and the default is type string. Printout justification is 'left', or 'right', and is specified by an '<' or '>' character respectively. If not specified, data types string and month will be left justified and type numeric will be right justified.

Note that column width is used primarily by the operator 'nsq-pr' and in no way limits the actual data size. It is not an error if some actual data in a column is wider than the defined width; a listing produced with 'nsq-pr' may be out of alignment however.

The optional documentation for each column follows the data definition word in the field. There must be one or more space characters after the data definition word and before the column documentation; the column documentation may be as long as necessary. Note that the data definition and the optional column documentation are contained in a single field in the row.

If the column name and/or column definition rows contain much information and/or column documentation they can become long and confusing to read. However the operators 'nsq-valid' and 'nsq-headchg' have options to print the header contents as a 'template' file, an organized list of information about the header.

A sample rdbtable (named SAMPLE) that will be used in later examples is shown in Table 1. The picture in Table 1 is for illustrative purposes; what the file would actually look like is shown in Table 2, where a TAB character is represented by '<T>' and a NEWLINE character is represented by '<N>'.

```
                        Table 1

                   rdbtable (SAMPLE)

      # Table documentation lines. These describe and
      # identify the rdbtable contents.
      # They may be read by many normal UNIX utilities,
      # which is useful to easily identify a file.
      # May also contain RCS or SCCS control information.
      NAME    COUNT   TYP     AMT     OTHER   RIGHT
      6       5N      3       5N      8       8>
      Bush    44      A       133     Another This
      Hansen  44      A       23      One     Is
      Jones   77      X       77      Here    On
      Perry   77      B       244     And     The
      Hart    77      D       1111    So      Right
      Holmes  65      D       1111    On      Edge


                       Table 2

             rdbtable (SAMPLE) actual content

      # Table documentation lines. These describe and<N>
      # identify the rdbtable contents.<N>
      # They may be read by many normal UNIX utilities,<N>
      # which is useful to easily identify a file.<N>
      # May also contain RCS or SCCS control information.<N>
```

```
NAME<T>COUNT<T>TYP<T>AMT<T>OTHER<T>RIGHT<N>
6<T>5N<T>3<T>5N<T>8<T>8><N>
Bush<T>44<T>A<T>133<T>Another<T>This<N>
Hansen<T>44<T>A<T>23<T>One<T>Is<N>
Jones<T>77<T>X<T>77<T>Here<T>On<N>
Perry<T>77<T>B<T>244<T>And<T>The<N>
Hart<T>77<T>D<T>1111<T>So<T>Right<N>
Holmes<T>65<T>D<T>1111<T>On<T>Edge<N>
```

It is important to note that only actual data is stored in the data fields, with no leading or trailing space characters. This fact can (and usually does) have a major effect on the size of the resulting datafiles (rdbtables) compared to data stored in "fixed field width" systems. The datafiles in NoSQL are almost always smaller, sometimes dramatically smaller.

2.2 Notes on /rdb table format.

Besides NoSQL there are other UNIX DBMS's, both commercial and free, that are based on ASCII tables. A commercial implementation is /rdb, by Revolutionary Software, while among the free ones there are Starbase, developed at the Harvard Smithsonian Astrophysical Observatory, and Gunnar Stefansson's *reldb*, a collection of interesting tools available at sites that bring archives of the *comp.sources.unix* Usenet newsgroup.

The ASCII table format of those database engines is very close to that of NoSQL, therefore data can easily be converted back and forth between them and NoSQL. To help with that, NoSQL provides a few simple convertion filters, namely *nsq-n2r, nsq-r2n, nsq-tabletolist* and *nsq-listtotable*.

Here is what the basic /rdb and Starbase table format look like :

```
                  Table 1a

             Starbase table (SAMPLE)

Table documentation lines. These describe and
identify the rdbtable contents.
They may be read by many normal UNIX utilities,
which is useful to easily identify a file.
May also contain RCS or SCCS control information.

NAME    COUNT   TYP     AMT
----    -----   ---     ---
Bush    44      A       133
Hansen  44      A       23
Jones   77      X       77
Perry   77      B       244
Hart    77      D       1111
Holmes  65      D       1111
```

As with the NoSQL format, the actual table contents are:

```
                   Table 2a

        Starbase table (SAMPLE) actual content

Table documentation lines. These describe and<N>
identify the rdbtable contents.<N>
They may be read by many normal UNIX utilities,<N>
which is useful to easily identify a file.<N>
May also contain RCS or SCCS control information.<N>
<N>
NAME<T>COUNT<T>TYP<T>AMT<N>
----<T>-----<T>---<T>---<N>
Bush<T>44<T>A<T>133<N>
Hansen<T>44<T>A<T>23<N>
Jones<T>77<T>X<T>77<N>
Perry<T>77<T>B<T>244<N>
Hart<T>77<T>D<T>1111<N>
Holmes<T>65<T>D<T>1111<N>
```

And here is its corresponding *list* format:

```
NAME      Bush
COUNT     44
TYP       A
AMT       133

NAME      Hansen
COUNT     44
TYP       A
AMT       23

NAME      Jones
COUNT     77
TYP       X
AMT       77

NAME      Perry
COUNT     77
TYP       B
AMT       244

NAME      Hart
COUNT     77
TYP       D
AMT       1111

NAME      Holmes
COUNT     65
TYP       D
```

```
AMT       1111
```

Chapter 3

NoSQL Operators

The NoSQL system comprises a set of programs called Operators.

Each operator is a separate program module that performs a unique function on the data. Operators can be grouped into data movers, report generators, and utilities.

The data movers are operators that extract or rearrange the data in some way. They each read an rdbtable via STDIN and write a rdbtable via STDOUT, and so are frequently connected using the UNIX pipe function to form a larger task. Each operator in such a "pipeline" style of operation gets its input from the output of the previous operator in the "pipeline". The data movers include:

nsq-col

Picks columns by name, outputs columns in listed order.

nsq-compute

Computes an arbitrary expression using column names.

nsq-headchg

Generates and replaces (or removes) the header of an rdbtable.

nsq-header

Extracts the header from an rdbtable.

nsq-join

Natural or "Master/Detail" join of two rdbtables.

nsq-listtotable

Converts files from /rdb *list* to /rdb *table* format.

nsq-merge

Merges two like rdbtables.

nsq-n2r

Converts tables from NoSQL to /rdb format.

nsq-r2n

Converts tables from /rdb to NoSQL format.

nsq-row

Selects rows based on arbitrary expressions.

nsq-search

Selects rows based on a multi-column key of a sorted or indexed rdbtable.

nsq-sort

Sorts a datafile by one or more columns.

nsq-subtot

Lists subtotals of specified columns.

nsq-tabletolist

Converts files from /rdb *table* to /rdb *list* format.

nsq-tee

Safely [over]writes an rdbtable.

nsq-uniq

Makes an rdbtable unique on specified columns.

The report generators each read an rdbtable via STDIN and produce a report on STDOUT, so when they are in a "pipeline" of operators they will be the operator at the end. The report generators are:

nsq-pr

Quick and easy printing of output formatted from information in the header.

nsq-report

Best form of output, with definable format.

nsq-summ

Summary/Statistical information about data values in an rdbtable.

nsq-valid

Verifies the structure of an rdbtable.

The utilities are used for manipulating the structure and content of rdbtables and are generally used as separate tasks,i.e. they do not read STDIN. The utilities are:

nsq

NoSQL interactive terminal interface.

nsq-ed

Uses an editor to allow modifications to an rdbtable.

nsq-index

Generate rdbtable index files to be used by 'nsq-search'.

nsq-lock

Trivial table locking program, for serializing access to rdbtables.

nsq-repair

Attempts to repair candidate NoSQL datafiles.

nsq-updseq

Updates/Creates a unique record ID on each row of an rdbtable. The field must be the first column in the table.

All operators take a '-h[elp]' option to show details of operation online. Following is a section for each operator, in alphabetic order.

3.1 nsq

Usage: **nsq [options] [rdbtable]**

Interactive analysis tool for NoSQL data tables (rdbtables). This is especially useful for rdbtables that were made from spreadsheet data.

Uses nine NoSQL modules: nsq-col, nsq-ed, nsq-pr, nsq-report, nsq-row, nsq-sort, nsq-t2l, nsq-summ, nsq-valid.

Specific module options are available using the '-help' option of individual modules. This utility uses the PAGER environment variable.

3.2 nsq-col

Usage: **nsq-col [options] list**

Selects ("*projects*") columns by name (and order) and outputs an rdbtable with these columns. Can effectively select, order, add, delete, or duplicate columns.

The value 'list' is normally a list of column names. If 'list' contains a triplicate of the form '-c NAME NEW' then column name 'NAME' will be changed to 'NEW'. If 'list' contains a triplicate of the form '-a NAME DEFN' then a new (null) column is added, at that point in the list of column names, with name 'NAME' and definition 'DEFN'.

This NoSQL operator reads an rdbtable from STDIN and writes an rdbtable to STDOUT. Options may be abbreviated.

As an example using the sample rdbtable from the DATA section (named sample), to select columns named 'NAME' and 'COUNT' the command would be:

```
nsq-col NAME COUNT < sample
```

To select all columns except column 'NAME' the command would be:

```
nsq-col -v NAME < sample
```

To add a new column named 'LENGTH' with a size of 10 the command would be:

```
nsq-col -v -a LENGTH 10 < sample
```

Note that to include documentation with the new column definition the command would be:

```
nsq-col  -v  -a  LENGTH  '10 length in meters'  <  sample
```

The '10 length in meters' must be quoted so that it will be treated as a single token.

3.3 nsq-compute

Usage: **nsq-compute [options] [statements]**

Computes values for data fields based on arbitrary statements using column names. Any characters that are special to the UNIX shell must be quoted.

Comparison operators may be of the form: gt, ge, lt, le, eq, ne. For example 'NAME eq Hobbs'. Logical constructors 'or' and 'and' may be used; as well as 'null' to indicate an empty data value. The supplied statements may be essentially any valid PERL statements.

All of the Comparison operators and Logical constructors are reserved and should not be used as column names (they are all lower case and four characters or less).

This operator reads a rdbtable via STDIN and writes a rdbtable via STDOUT. Options may be abbreviated.

If a file is used to contain the statements any line in the file that starts with a sharp sign (#) is treated as a comment and ignored. Also if there is a sharp sign preceded by a space character anywhere on the line the rest of the line is also treated as a comment.

Since column names and reserved words are parsed by the program, do not put the entire expression in a single pair of quotes as that will prevent the parsing. Also note that column names and reserved words need to be surrounded by blank spaces if they are not individually quoted. For example either form below is fine:

```
nsq-row   NAME    eq   "L Brown"  <  sample

nsq-row  "NAME"  "eq"  "L Brown"  <  sample
```

but do not use this form:

```
nsq-row  "NAME  eq  L Brown"  <  sample
```

Example rdbtable (named cfile):

```
name    count   type    amt
6       5N      4       5N
Bush    3       A       133
Hansen  39      A       23
Newton  8       E       8
Hobbs   42      B       144
Hart    2       C       55
Jones   4       B       244
Smith   5       D       1111
```

The command:

```
nsq-compute  count  +=  100  if  type  lt  D  <  cfile  |  nsq-pr
```

gives the output:

```
name    count  type    amt
------  -----  ----  -----
Bush      103  A       133
Hansen    139  A        23
Newton      8  E         8
Hobbs     142  B       144
Hart      102  C        55
Jones     104  B       244
Smith       5  D      1111
```

Example file of commands named 'XXX':

```
if( type eq A ){
    name = NEW ;
    amt = count * 2 ;
    type = 'AAA' ;
}
else{
    name = OLD ;
    amt = count + 1000 ;
    type = 'ZZZ' ;
}
```

Output from command:

```
nsq-compute  -fXXX  <  cfile  |  nsq-pr
```

would be:

```
name    count  type    amt
------  -----  ----  -----
NEW         3  AAA       6
NEW        39  AAA      78
OLD         8  ZZZ    1008
OLD        42  ZZZ    1042
OLD         2  ZZZ    1002
OLD         4  ZZZ    1004
OLD         5  ZZZ    1005
```

3.4 nsq-ed

Usage: **nsq-ed [options] rdbtable [col_spec] [line_spec] [pat_spec]**

This utility calls an editor to allow the editing of selected lines and/or columns of (or the entire) rdbtable. Options may be abbreviated.

A "col_spec" is a list of column names.

A "line_spec" is a list of line numbers, of increasing value, optionally separated by a dash to specify a range, e.g. "10-20". The form "N-" means from line N to end of file. The header is always included, so do not specify lines 1 or 2 (except as the first part of a larger group, e.g. "1-10").

A "pat_spec" is a single pattern (of the form: /pat/) optionally followed by one or more column names, and may be preceded with the reserved word 'ne' to negate the meaning (e.g. the pattern should NOT match).

The order of "col_spec", "line_spec", and "pat_spec" is significant only to the extent that "col_spec" must precede "pat_spec" in the command line if both are given.

If none of "col_spec", "line_spec", or "pat_spec" are given then the entire rdbtable will be edited. If one or more of the three above options are given then the selected subset of the rdbtable will be edited. The option "col_spec" identifies which columns of the rdbtable are to be edited, and options "line_spec" and "pat_spec" determine which lines will be selected for editing, either by direct reference ("line_spec" given "col_spec" not given) or by pattern matching ("col_spec" given "line_spec" not given). If both "line_spec" and "pat_spec" are given then only lines within the bounds of "line_spec" will be considered for selection by pattern matching.

If "pat_spec" does not include column names then the pattern (any PERL regular expression) is matched against each entire row; a row is selected if there is a match anywhere in the row. If column names are included the pattern is matched against only the specified columns. In this case a row is selected if a match is found in any specified column. If the "ne" option precedes the "pat_spec" without column names then an entire row is selected if the pattern does not match anywhere in the row, and if column names are given then the row is selected if the pattern does not match in any specified column.

The form of the file to be edited is either "column" with visible column delimiters (the default) or "list" format where the column names are on the left and the data is on the right. The default editor is specified by the environment variable EDITOR if set, otherwise the editor 'vi' is used.

In either form of editing the delimiter is a "pipe" symbol (|). Care should be taken when editing not to use any "pipe" symbols in the data, or to delete any existing pipe symbols in the file. Also, in the case of "list" form, one or more blank lines must separate each record.

The rdbtable may be an existing file, or it may be automatically checked out from RCS. In the latter case it will be checked back into RCS after the editing is complete. The default action is that if the rdbtable does not exist an attempt will be made to find the rdbtable under RCS (the '-RCS' option may be used to force the use of an RCS file).

Afterward, except in the RCS case, the original contents of the rdbtable will be left in a file of the same name preceded with a comma, e.g. "sample" will be ",sample".

Uses NoSQL operators: nsq-col, nsq-pr, nsq-mktbl, nsq-t2l, nsq-l2t.

WARNING: If line_spec is given the number of columns must not be changed by editing, or if col_spec and/or "pat_spec" is given the number of lines must not be changed by editing, otherwise the results may be unpredictable.

An example command to edit the rdbtable (named sample) from the DATA section would be:

```
nsq-ed  sample
```

which would edit the entire rdbtable. The file as it is ready to edit is shown in Table 3. The pipe character '|' must not be removed during the editing process, although it may be moved left or right if necessary and the spaces around the pipe character may be deleted if desired.

This form of editing is fine if the rdbtable is not large. If it is large then editing only those parts that need changes is faster and less error prone. To edit only columns 'NAME', 'COUNT', and 'AMT', the command would be:

```
nsq-ed  sample  NAME  COUNT  AMT
```

To edit only lines five thru seven the command would be:

```
nsq-ed  sample  5-7
```

```
                      Table 3

     RDBTABLE (SAMPLE) READY TO EDIT, COLUMN FORM

NAME   | COUNT | TYP  |   AMT | OTHER     |    RIGHT
6      |    5N | 4    |    5N | 8         |       8>
Bush   |    44 | A    |   133 | Another   |     This
Hansen |    44 | A    |    23 | One       |       Is
Jones  |    77 | X    |    77 | Here      |       On
Perry  |    77 | B    |   244 | And       |      The
Hart   |    77 | D    |  1111 | So        |    Right
Holmes |    65 | D    |  1111 | On        |     Edge
```

To edit only lines five thru seven of only columns 'NAME', 'COUNT', and 'AMT' the command would be:

```
nsq-ed  sample  NAME  COUNT  AMT  5-7
```

and the file to edit would look like:

```
..>>> 1 2 CONTROL LINE, DO NOT TOUCH <<<
NAME   | COUNT |   AMT
6      |    5N |    5N
..>>> 5 3 CONTROL LINE, DO NOT TOUCH <<<
Jones  |    77 |    77
Perry  |    77 |   244
Hart   |    77 |  1111
```

Note that whenever a line_spec is given, control lines (starting with '..>>>' are inserted into the file to edit. They must not be modified during the editing process. They are used to reconstruct the rdbtable after editing.

If the rdbtable has data fields that are long, i.e. longer than convenient to edit in the column form shown above, the 'list' form is the preferred method. The usage of line_spec and col_spec are unchanged but the form of the file to edit is different. For example consider an rdbtable (named sample3) which is shown in Table 4, where the TAB characters are represented by '<T>and the newline characters are represented by <N>This small rdbtable looks very incoherent in raw form, and a file of any real size with long data fields is even more so. The command to edit the file sample3 in 'list' form would be:

```
                         Table 4

               RDBTABLE (SAMPLE3) ACTUAL CONTENT

name<T>datatype<T>agencysrc<T>dbms<T>contact<T>contents<T>notes<N>
46<T>15<T>60<T>15<T>21<T>530<T>600<N>
ACAS (Air Combat Assessment)<T>BDA<T>Bigplace AFB<T>File<T>Starr<T>Air
Combat Assessment BDA data.  Duplicates data under ACAS (Air Combat
Assessment) BDA Sorties, diskettes nr 1,2,3.<T>On two 3.5 inch
diskettes.<N>
ACAS (Air Combat Assessment) BDA Sorties<T>BDA<T>Sawyer AFB<T>File<T>
Hobbs/Emerson<T>85 files, 2 per day containing 12 and 24 hour reports.
This data is different from that under ACAS (Air Combat Assessment) BDA
Data, from diskettes 1,2,3.<T>Received 5/6/91.<N>
ATO (Air Tasking Orders) Original<T>ATO<T>HQ USAF, Universal AFB<T>
File<T>Marshall<T>Original ATO messages.  Both sets are incomplete.<T>
To be joined into single file and edited.  Missing sections not yet
ordered.  May be parsed completely, or only for key comments. Much data
to be processed.<N>
ABC Original<T>ABC<T>HQ USAF, Universal AFB<T>File<T>Marshall<T>Original
ATO messages.  Both sets are incomplete.<T>To be joined into single file
and edited.  Missing sections not yet ordered. May be parsed completely,
or only for key comments. Much data to be processed.<N>
```

The command to edit the file sample3 in 'list' form would be:

```
nsq-ed  -list  sample3
```

which would produce a file to edit as shown in Table 5. Note that each section holds information relating to one row in the rdbtable and that the first section holds information relating to the header of the rdbtable. Also note that each section is separated by a blank line (it could be any number of blank lines).

Each row in a section relates to a single data value. The pipe character '|' must not be removed during the editing process, although it may be moved left or right if necessary. Only one pipe character is to be in the information relating to one data value, although that information may be physically on more than one line in the section if the data value is long. The spaces on both sides of the pipe character as well as the spaces around the column names are only for readability; they may be moved or even deleted if desired.

```
                         Table 5

          RDBTABLE (SAMPLE3) READY TO EDIT, LIST FORM

     name | 46
 datatype | 15
agencysrc | 60
     dbms | 15
```

```
     contact | 21
    contents | 530
       notes | 600

        name | ACAS (Air Combat Assessment)
    datatype | BDA
   agencysrc | Bigplace AFB
        dbms | File
     contact | Starr
    contents | Air Combat Assessment BDA data.  Duplicates data under ACAS
               (Air Combat Assessment) BDA Sorties, diskettes nr 1,2,3.
       notes | On two 3.5 inch diskettes.

        name | ACAS (Air Combat Assessment) BDA Sorties
    datatype | BDA
   agencysrc | Sawyer AFB
        dbms | File
     contact | Hobbs/Emerson
    contents | 85 files, 2 per day containing 12 and 24 hour reports. This
               data is different from that under ACAS (Air Combat
               Assessment) BDA Data, from diskettes 1,2,3.
       notes | Received 5/6/91.

        name | ATO (Air Tasking Orders) Original
    datatype | ATO
   agencysrc | HQ USAF, Universal AFB
        dbms | File
     contact | Marshall
    contents | Original ATO messages.  Both sets are incomplete.
       notes | To be joined into single file and edited.  Missing sections
               not yet ordered.  May be parsed completely, or only for key
               comments. Much data to be processed.

        name | ABC Original
    datatype | ABC
   agencysrc | HQ USAF, Universal AFB
        dbms | File
     contact | Marshall
    contents | Original ATO messages.  Both sets are incomplete.
       notes | To be joined into single file and edited.  Missing sections
               not yet ordered.  May be parsed completely, or only for key
               comments.  Much data to be processed.
```

The advantage of this form of edit file is that even with very large data values most, if not all, of the information from each row of an rdbtable will be visible on the screen at once.

3.5 nsq-headchg

Usage: **nsq-headchg [options] file.tpl**

Replaces the header (first two rows) of an rdbtable with a header generated from information in the template file 'file.tpl'. Options are available to add, copy, or delete the header, or to generate a template file from an existing rdbtable.

Each line of the Template file contains info about a column, in order. The lines contain: (optional) index number (starting at 0 or 1), column name, definition, and (optional) comments or documentation, white space separated. If column name contains spaces it must be enclosed in double quotes. Names containing space characters are not recommended, however, as it is generally troublesome and error prone. A good substitute is the underscore character (_).

Lines that start with a sharp character '#' are skipped, as are blank lines. To start a column name with a sharp character '#' the name must be enclosed in double quotes. (but this is not recommended).

The number of columns in the header is normally reported on STDERR. This operator reads an rdbtable via STDIN and writes an rdbtable via STDOUT. Options may be abbreviated. This operator uses the NoSQL operator: nsq-valid.

As an example, to generate a template file named 'new.tpl' from the rdbtable (named sample) from the DATA section, the command would be:

```
nsq-headchg  -templ  <  sample  >  new.tpl
```

The contents of file 'new.tpl' would then be:

```
0                NAME  6
1               COUNT  5N
2                 TYP  4
3                 AMT  5N
4               OTHER  8
5               RIGHT  8>
```

To change the header of rdbtable 'sample', the procedure is to edit the file 'new.tpl', and then run 'nsq-headchg' using the modified file. For example, to change the names so that only the first letters are upper case and to make column 'OTHER' numeric, edit file 'new.tpl' so it looks like the following:

```
0                Name  6    All names are first letter upper case.
1               Count  5N
2                 Typ  4
3                 Amt  5N
4               Other  8N   Now numeric.
5               Right  8>
```

Note the index in the zeroth column and the documentation in the fourth column, both of which are optional, but recommended. The command to change the header of rdbtable 'sample' and make a new rdbtable called 'new.sample' would be:

```
nsq-headchg  new.tpl  <  sample  >  new.sample
```

3.6 nsq-header

Usage: **nsq-header [options]**

Section yet to be written. See 'nsq-header -help' in the meantime.

3.7 nsq-index

Usage:

nsq-index [options] rdbtbl column ...

or

nsq-index [options] -update [index_file ...]

The first form of usage of this utility generates an index file for the column(s) given, that refers to the specified rdbtbl. An index file is actually another (smaller) rdbtable containing only the column(s) given plus a column for index information. An index file can be used by the operator 'nsq-search' to quickly locate rows of data in the referenced rdbtable.

Index files are named by appending an 'x' and the column name(s) (seperated by a dots) to the base name of the rdbtable it refers to. For example an index file for the rdbtable "area.rdb" on column "strip" would be "area.x.strip". An index file that was also on column "depth" would be "area.x.strip.depth".

The second form of usage of this utility updates the index file(s) given. If no files are given all index files in the current directory are updated. An update of an index file is necessary when the rdbtable it refers to has been modified.

This utility writes or rewrites rdbtables with defined names in the current directory. Options may be abbreviated.

3.8 nsq-join

Usage: **nsq-join [options] col.name[=col.name_2] rdbtable_2 < rdbtable_1**

where rdbtable_2 is the "secondary" rdbtable, while rdbtable_1 is the the "primary" one.

Does a join of two rdbtables on the column(s) specified. The default is a "natural" join, with optional "Master/Detail" or cartesian (cross-product) type joins. Options may be abbreviated.

The Table from STDIN is the master. A natural join produces a new rdbtable that contains only rows from the input rdbtables that match on the specified columns (key columns). A master-detail join produces a new rdbtable that contains all rows from the master rdbtable and those rows from the secondary rdbtable that match. A cartesian join produces an rdbtable that contains all rows of both input rdbtables.

Each item in the list of column(s) may specify column names that are different in the two rdbtables, i.e. '=column_2', if given, refers to a name in rdbtable_2 that corresponds to 'column' in rdbtable_1. If '=column_2' is not given it means that the corresponding column name in both rdbtables is the same.

If different column names are specified, the name of the join columns in the output rdbtable will be from rdbtable_1.

Note that the two rdbtables must be sorted on the columns specified in order for a join operation to function correctly.

The order of columns in the output rdbtable will be: first the join columns, then the other columns from rdbtable_1, then the other columns from rdbtable_2.

This operator reads an rdbtable via STDIN and writes an rdbtable via STDOUT.

If we have the rdbtable (named samplej) here:

name	nr	typ	amt
6	2	4	4
Bush	1	A	133
Bush	2	A	134
Hansen	3	A	143
Hobbs	4	B	144
Hobbs	5	B	144
Jones	6	C	155
Perry	7	D	244
Perry	8	D	311

and the rdbtable (named samplej2) here:

name	cnt	typ	amt
6	5N	4	5N
Hobbs	41	A	141
Hobbs	42	BB	142
Hobbs	51	BB	144
Hobbs	43	CC	143

then the command to do a natural join of samplej and samplej2 on column name is:

```
nsq-join  name  samplej2  <  samplej
```

and the result is shown in Table 6.

Table 6

NATURAL JOIN OF RDBTABLES SAMPLEJ AND SAMPLEJ2

name	nr	typ	amt	cnt	typ	amt
6	2	4	4	5N	4	5N
Hobbs	4	B	144	41	A	141
Hobbs	4	B	144	42	BB	142
Hobbs	4	B	144	51	BB	144
Hobbs	4	B	144	43	CC	143
Hobbs	5	B	144	41	A	141
Hobbs	5	B	144	42	BB	142
Hobbs	5	B	144	51	BB	144
Hobbs	5	B	144	43	CC	143

The command to do a "masterdetail" join of the same two rdbtables on column name is:

```
nsq-join  -md  name  samplej2  <  samplej
```

and the result is shown in Table 7.

Table 7

MASTER-DETAIL JOIN OF RDBTABLES SAMPLEJ AND SAMPLEJ2

name	nr	typ	amt	cnt	typ	amt
6	2	4	4	5N	4	5N
Bush	1	A	133			
Bush	2	A	134			
Hansen	3	A	143			
Hobbs	4	B	144	41	A	141
Hobbs	4	B	144	42	BB	142
Hobbs	4	B	144	51	BB	144
Hobbs	4	B	144	43	CC	143
Hobbs	5	B	144	41	A	141
Hobbs	5	B	144	42	BB	142
Hobbs	5	B	144	51	BB	144
Hobbs	5	B	144	43	CC	143
Jones	6	C	155			
Perry	7	D	244			
Perry	8	D	311			

3.9 nsq-l2t

Usage: **nsq-l2t [options]**

Converts a file in ”list” format to an rdbtable. Long data fields may be folded. This operator is mainly used by other operators. Options may be abbreviated.

3.10 nsq-listtotable

Usage: **nsq-listtotable [options]**

Section yet to be written. See 'nsq-listtotable -help' in the meantime.

3.11 nsq-lock

Usage: **nsq-lock [options]**

Section yet to be written. See 'nsq-lock -help' in the meantime.

3.12 nsq-merge

Usage: **nsq-merge [options] < old_table column ... merge_table**

This operator merges and/or deletes rows of 'old_table' based on data values in 'merge_table' in the specified column(s). Both tables should be sorted on the specified column(s).

In the normal case, one or more rows in 'merge_table' either replace one or more existing rows in 'old_table' if the key column(s) match, or are inserted in order if the key column(s) do NOT match.

If the delete option is specified on the command line, one or more existing rows in ’old_table’ will be deleted if there is a key column(s) match and the data in the delete column is equal to the delete string, ”>>DEL<<” (without the quotes) by default. The delete column is the first non-key column in ’merge_table’.

Both tables should have similar data structures. The header for the new rdbtable is taken from ’merge_table’, thus allowing a change of header information to be made.

This operator writes an rdbtable via STDOUT. Options may be abbreviated.

3.13 nsq-mktbl

Usage: **nsq-mktbl [options]**

Makes a file of data in columns (with visible column delimiters) into an rdbtable. The column delimiter is the pipe symbol (|). This operator is mainly used by other operators.

This operator reads a file via STDIN and writes an rdbtable via STDOUT. Options may be abbreviated.

3.14 nsq-n2r

Usage: **nsq-n2r [options]**

Section yet to be written. See ’nsq-n2r -help’ in the meantime.

3.15 nsq-pr

Usage: **nsq-pr [options]**

This operator used for quick and easy printing of an rdbtable, in a simple but useful form. It prints an rdbtable using formatting information from the header.

The printing of each row of data will be on one line if possible, but when multiple lines are necessary the second and later lines are indented for readability. Also when multiple lines are necessary a simple space availability algorithm is used to minimize the number of lines printed for each row of data. This may result in the order of some data values being rearranged from their order in the rdbtable. The ’-b0’ option can override this algorithm and force the same printing order as in the rdbtable.

This NoSQL operator reads an rdbtable from STDIN and writes a formatted report on STDOUT. Options may be abbreviated.

As an example using the sample rdbtable from the DATA section (named sample), the command to view this rdbtable would be:

```
nsq-pr < sample
```

which would produce the output shown in Table 8. The same command with a page heading for printing:

```
nsq-pr  -PP  <  sample
```

produces the output as shown in Table 9. Using an rdbtable (named sample4) that has long data values, shown in Table 10, the command to print the rdbtable using the truncate option is:

```
nsq-pr  -t  <  sample4
```

```
                   Table 8

    PRINTING RDBTABLE (SAMPLE) USING NSQ-PR

NAME    COUNT  TYP     AMT  OTHER        RIGHT
------  -----  ----  -----  --------  --------
Bush       44  A       133  Another       This
Hansen     44  A        23  One             Is
Jones      77  X        77  Here            On
Perry      77  B       244  And            The
Hart       77  D      1111  So           Right
Holmes     65  D      1111  On            Edge
```

```
                   Table 9

PRINTING RDBTABLE (SAMPLE) WITH PAGE HEADING USING NSQ-PR

Page   1   Mon Dec  2 16:56:43 PST 1991

NAME    COUNT  TYP     AMT  OTHER        RIGHT
------  -----  ----  -----  --------  --------
Bush       44  A       133  Another       This
Hansen     44  A        23  One             Is
Jones      77  X        77  Here            On
Perry      77  B       244  And            The
Hart       77  D      1111  So           Right
Holmes     65  D      1111  On            Edge
```

```
                    Table 10

  RDBTABLE WITH LONG DATA VALUES (SAMPLE4) ACTUAL CONTENT

name<T>type<T>contact<T>contents<N>
10<T>4<T>21<T>20<N>
Hansen<T>AAA<T>R. Starr at the UCLA & USC<T>Duplicate data under
processing order number 55-7.<N>
Hart<T>CCC<T>Hobbs/Emerson at RAND Corporation<T>85 files, 2 per
day containing 12 and 24 hour reports.<N>
Hobbs<T>EEE<T>Marshall at Universal AFB<T>Original PAF messages.
Both sets are incomplete.<N>
Bush<T>KKK<T>General USAF personnel<T>Duplicate ATO messages,
incomplete.<N>
Lender<T>RRR<T>Army base in Nevada<T>Nothing.<N>
Emerson<T>UUU<T>Navy at Washington DC<T>More than we thought at
first.<N>
```

which will produce output with the data values truncated to the defined column width as in Table 11. Using the same rdbtable with the fold option:

```
nsq-pr  -fold  <  sample4
```

produces output with the long data values 'folded' within their defined column widths as shown in Table 12. Note that each line is repeated until the entire data value for each column is completely shown. This makes this type of output variable length.

If you need a quick and easy way to look at the data in an rdbtable use the -win option. This option will cause nsq-pr to list as many columns as possible in single line records that will fit in the current window or terminal width. Note that you do not have to type the column names (or even know them) to use this option.

```
                         Table 11

     PRINTING RDBTABLE (SAMPLE4) WITH NSQ-PR -TRUNC OPTION

name        type  contact                contents
----------  ----  ---------------------  --------------------
Hansen      AAA   R. Starr at the UCLA   Duplicate data under
Hart        CCC   Hobbs/Emerson at RAND  85 files, 2 per day
Hobbs       EEE   Marshall at Universal  Original PAF message
Bush        KKK   General USAF personne  Duplicate ATO messag
Lender      RRR   Army base in Nevada    Nothing.
Emerson     UUU   Navy at Washington DC  More than we thought
```

```
                         Table 12

      PRINTING RDBTABLE (SAMPLE4) WITH NSQ-PR -FOLD OPTION

name        type  contact                contents
----------  ----  ---------------------  --------------------
Hansen      AAA   R. Starr at the UCLA   Duplicate data under
                  & USC                  processing order
                                         number 55-7.
Hart        CCC   Hobbs/Emerson at RAND  85 files, 2 per day
                  Corporation            containing 12 and 24
                                         hour reports.
Hobbs       EEE   Marshall at Universal  Original PAF
                  AFB                    messages.  Both sets
                                         are incomplete.
Bush        KKK   General USAF           Duplicate ATO
                  personnel              messages,
                                         incomplete.
Lender      RRR   Army base in Nevada    Nothing.
Emerson     UUU   Navy at Washington DC  More than we thought
                                         at first.
```

It may be combined with the -t option to increase the number of columns of data shown on each line at the expense of some column width.

For example the command 'nsq-pr < d11c' on an 80 character wide window or terminal produces the following:

```
name    count            type  amt    n1                       n3
------  ---------------  ----  -----  -----------------------  ------------
    n2                       n4                       n5
    -----------------------  -----------------------  ------------
    n6           n7
    ------------  ------------
Bush    3                A     133    alpha22.307              117722
 baker                     DDBBx17                    other
    124567       8GGXXH17
Hansen  39               A     23     beta222.307              117723
charlie                   DDBBx18                    data
    1239870      GGXXH17
Newton  8                E     8      gama22.333               117724
dog                       DDBBx19                    exists
    1239870      GGXXH17
Hobbs   42               B     144    delta3.3.118             117725
echo                      DDBBx20                    here
    1239870      GGXXH17
Hart    2                C     55     epslion33.118            117726
foxtrot                   DDBBx21                    also
    1239870      GGXXH17
```

This is readable, but not very nice to look at, and even worse if there are more columns. The command

```
nsq-pr -win < d11c
```

produces:

```
name   count           type amt   n1                      n3
------ --------------- ---- ----- ----------------------- ------------
Bush   3               A    133   alpha22.307             117722
Hansen 39              A    23    beta222.307             117723
Newton 8               E    8     gama22.333              117724
Hobbs  42              B    144   delta3.3.118            117725
Hart   2               C    55    epslion33.118           117726
```

Not all the data is listed, but the first few columns (sometimes the most important) are easier to view. The command

```
nsq-pr -win -t6 < d11c
```

shows even more of the data, at the expense of some data width:

```
name   count  type amt   n1     n2     n3     n4     n5     n6     n7
------ ------ ---- ----- ------ ------ ------ ------ ------ ------ ------
Bush   3      A    133   alpha2 baker  117722 DDBBx1 other  124567 8GGXXH
Hansen 39     A    23    beta22 charli 117723 DDBBx1 data   123987 GGXXH1
Newton 8      E    8     gama22 dog    117724 DDBBx1 exists 123987 GGXXH1
Hobbs  42     B    144   delta3 echo   117725 DDBBx2 here   123987 GGXXH1
Hart   2      C    55    epslio foxtro 117726 DDBBx2 also   123987 GGXXH1
```

3.16 nsq-r2n

Usage: **nsq-r2n [options]**

Section yet to be written. See 'nsq-r2n -help' in the meantime.

3.17 nsq-repair

Usage: **nsq-repair [options] file ...**

Attempts to repair candidate NoSQL datafiles, e.g. files that have been ported from a MacIntosh or PC (MSDOS computer) in spreadsheet form but that do not yet have valid rdbtable structure. Generates definition lines (second line of header). The width of all data values is checked and the maximum width for a column is used as the column width in the definition line for that Table.

It also works with existing rdbtables ('-exist' option) and is convenient for removing leading and trailing space characters from data values (-blank option).

Adds fields as necessary to rows (null), or to header (DUM1, DUM2, ...) to make the Table structure valid.

The new rdbtables will be in the current directory (even if the input files are not) and will have the suffix changed (or added) to '.rdb' by default.

Options may be abbreviated.

3.18 nsq-report

Usage: **nsq-report [options] file.frm**

Formats and prints an arbitrary style report, with the format specified in the file "file.frm". A page header may be specified.

This NoSQL operator reads an rdbtable from STDIN and writes a formatted report on STDOUT. Options may be abbreviated.

The "file.frm" file (or form file) shows pictorially one 'record' of output, which will contain data from one row of an rdbtable. An optional page header may be defined as well.

The form file contains regular text, picture fields, and associated column names. Regular text prints just as given. Picture fields define the width and justification for printing a data value from a column. The names of the associated columns are listed on the line following the picture fields and in the same order. Note that this file should not contain any TAB characters; space characters should be used instead.

Picture fields start with either '@' or '^' and are followed by one of three primary characters to define the width of the field. The three characters are '<', '>', or '|' to specify left, right, or center justification respectively. There is also an alternate right justification character for printing numeric data, with optional decimal point. The character is the sharp sign '#', and a period specifies the decimal point placement, as in '@########.##'.

A numeric picture field has the following features:

- Data is lined up on the decimal point (if any).
- Automatic rounding of data.

- Automatic conversion of data in scientific notation.

Numeric Data may be in the form of integers, fixed point, or scientific notation' e.g. 12345, 4567.345, or 1.678E17.

Normally picture fields start with the '@' character. That means to put the referenced data value into the defined picture field, or as much of the data as will fit into the field, if the data is larger than the field. If the field starts with the '^' character it means to repeat the field on as many lines as necessary in order to print the entire data value. This is useful for large data fields, such as comments or free text.

Instead of a column name there are some special names that can be used to have other information inserted. This are especially useful if there is a page header. The special names and what they mean are:

- _pgnr_ - current page number
- _date_ - current date
- _rcnr_ - current record number (row number)
- _`cmd arg1 ... argN`_ - the UNIX command is executed once, and its output is put into the associated picture field. Note that they are BACKTICKS (grave accents) not single quotes.
- _COLNAME_cd_ - the column documentation for column name 'COLNAME'.
- _tbld_ - the table documentation, all lines.
- _tbld_3.7_ - the table documentation, lines 3 thru 7. If either first or second number is missing it means line 1 or the last line of the header, respectively.

An example of a form file for use with rdbtable 'sample' is shown in Table 13.

The first and last lines (that start with 'format' or a single period) define the pictorial records and must be as shown. The first record defines the header and is optional. If this form file (named sample.frm) were used in the command:

```
nsq-report  sample.frm  <  sample
```

it would produce the one page report as in Table 14.

```
                         Table 13

                 FORM FILE FOR RDBTABLE (SAMPLE)

format top =
Page @>,            The Page Header     @<<<<<<<<<<<<<<<<<<<<<<<<<<<
  _pgnr_                                        _date_

.
format =
 Name:   @<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<          Type:  @>>>>
             NAME                          TYP
                Total: @<<<<<<<<     Other:  @<<<<<<<<<<<<<<<
                       AMT                          OTHER
```

```
.
```

```
                         Table 14

            PRINTING RDBTABLE (SAMPLE) WITH NSQ-REPORT

Page  1,           The Page Header     Mon Dec  2 16:21:18 PST 1991

Name:   Bush                                       Type:      A
                  Total: 133          Other:  Another
Name:   Hansen                                     Type:      A
                  Total: 23           Other:  One
Name:   Jones                                      Type:      X
                  Total: 77           Other:  Here
Name:   Perry                                      Type:      B
                  Total: 244          Other:  And
Name:   Hart                                       Type:      D
                  Total: 1111         Other:  So
Name:   Holmes                                     Type:      D
                  Total: 1111         Other:  On
```

For another example, one might want to have a date on a report in other than the standard date output format, and an idea of who executed the program, and have a reference count of the records being produced. The form file might be as in Table 15. which could produce:

```
                          Table 15

                      ANOTHER FORM FILE

format top =
Run By: @<<<<<<<        The Date/Time is    @<<<<<<<<<<<<<<<<<<<<<<<<<
        _`whoami`_                           _`date "+%m/%d/%y %H:%M"`_
.
format =
    RecordNr:  @>>      @<<<<<<<<<<<<<<<<<<<<<
             _rcnr_    OTHER
.
```

```
Run By: hobbs           The Date/Time is    10/15/91 09:43

RecordNr:        1      Other data here
RecordNr:        2      and here,
RecordNr:        3      and so on.
...             ...     ...
```

Another example shows how longer data values can be handled. If the form file (named sample2.frm) looks like Table 16, and if the following command is used:

```
nsq-report  sample2.frm  <  sample2
```

then the output would be as shown in Table 17.

Table 16

FORM FILE FOR RDBTABLE (SAMPLE2)

```
format top =
Page @>,               Page Header here                @<<<<<<<<<<<<<
_pgnr_                                     _`date "+%m/%d/%y %H:%M"`_
                Executed by @<<<<<<< on: @<<<<<<<<<
                            `whoami`     _`hostname`_

format =
    Name:   @<<<<<<<<<<<<<<<  Other: @<<<<<<<<<<        Type: @<<<<<<<<
            NAME              OTHER                              TYP
 Comment:   ^<<<<<<<<<<<<<<    Long: ^<<<<<<<<<<<<<<    Right: @>>>>>>>
            COMMENT                   LONG                       RIGHT
=======================================================================
```

Table 17

PRINTING RDBTABLE (SAMPLE2) WITH NSQ-REPORT

```
Page  1,               Page Header here                12/02/91 16:32
                Executed by hobbs    on: id

    Name:   Bush              Other: Another            Type: A
 Comment:   A comment          Long: This a long        Right:     This
            here.                    message for
                                     test.
=======================================================================
    Name:   Hansen            Other: One                Type: A
 Comment:   A longer           Long: This a long        Right:       Is
            comment here.            message for
                                     test.
=======================================================================
    Name:   Jones             Other: Here               Type: X
 Comment:   A longer,          Long: Short test.        Right:       On
            longer comment
            here.
=======================================================================
    Name:   Perry             Other: And                Type: B
 Comment:   A short comment    Long: This a long        Right:      The
            here.                    message for
                                     test.
=======================================================================
    Name:   Hart              Other: So                 Type: D
 Comment:   Little here.       Long: Here too.          Right:    Right
=======================================================================
    Name:   Holmes            Other: On                 Type: D
```

```
 Comment:   A comment here     Long: A short            Right:     Edge
            that is a                message.
            little ongoing,
            so to speak.
=====================================================================
```

Note that since there were two picture fields that started with the '^' character on one line the length of output records varies according to the length of the two associated data values.

3.19 nsq-row

Usage: **nsq-row [options] expression**

Selects rows from the input rdbtable based on an arbitrary expression using column names. Characters that are special to the UNIX shell must be quoted.

Logical constructors 'or' and 'and' may be used; as well as 'null' to indicate empty data values. Comparison operators may be of the form: gt, ge, lt, le, eq, ne, mat, nmat. The first six are the usual operators, E.g 'name eq Hobbs' or 'COUNT gt 100'. The last two stand for 'match' and 'non-match' and are used for pattern matching. They are exactally the same as using the PERL operators '= ' or '! ' respectively, except that pattern matching can be specified easier in expressions, as in:

```
NAME  mat   /[Hh]obbs/  <<< First letter either case
NAME  mat   /hobbs/i    <<< any combination of case
NAME  nmat  /[aeiou]/i  <<< names without vowels
```

where 'NAME' and 'COUNT' are column names, of course. A warning message is produced on STDERR if either of 'mat' or 'nmat' is used with a numeric type column, but the execution continues. It does not check the '= ' or '! ' forms.

All of the Comparison operators and Logical constructors are reserved and should not be used as column names (they are all lower case and four characters or less).

Since column names and reserved words are parsed by the program, do not put the entire expression in a single pair of quotes as that will prevent the parsing. Also note that column names and reserved words need to be surrounded by blank spaces if they are not individually quoted. For example either form below is fine:

```
nsq-row   NAME    eq   "L Brown"  <  sample
nsq-row  "NAME"  "eq"  "L Brown"  <  sample
```

but do not use this form:

```
nsq-row  "NAME  eq  L Brown"  <  sample
```

This operator reads an rdbtable via STDIN and writes an rdbtable via STDOUT. Options may be abbreviated.

As an example using the sample rdbtable from the DATA section (named sample), to select rows that have the NAME column equal to 'Hansen' the command would be:

```
nsq-row  NAME  eq  Hansen  <  sample
```

which would produce:

```
NAME    COUNT   TYP     AMT     OTHER   RIGHT
6       5N      4       5N      8       8>
Hansen  44      A       23      One     Is
```

to select rows that have the TYP column equal to 'A' or that have the AMT column greater than 1000 the command would be:

```
nsq-row  TYP  eq  A  or  AMT  gt  1000  <  sample
```

producing:

```
NAME    COUNT   TYP     AMT     OTHER   RIGHT
6       5N      4       5N      8       8>
Bush    44      A       133     Another This
Hansen  44      A       23      One     Is
Hart    77      D       1111    So      Right
Holmes  65      D       1111    On      Edge
```

Note that in some rare cases there could be a column name that is identical to a data value specified in an expression using another column name that might cause a problem (this actually happened). For example if two column names are 'N' and 'T', and column 'N' has a data value of 'T', to select all rows where column 'N' is equal to 'T' the normal command would be:

```
nsq-row  <  table  N  eq  T
```

Unfortunately the 'T' in the expression gets translated to 'column name T', not used as 'data value T'. That is, the expression askes for all rows where the data value of column N equals the data value of column T, legal, but not what was wanted. There is a simple workaround however. The 'T' in the expression can be escaped with a backslash to prevent the translation to a column name, as in the revised command:

```
nsq-row  <  table  N  eq  '\T'
```

Thus either meaning can be specified, as desired.

3.20 nsq-search

Usage:

nsq-search [options] rdbtbl < keytbl

or:

nsq-search [options] -ind index_file [rdbtbl] < keytbl

This operator does a fast search of 'rdbtbl' (or index_file) using a binary search on a key of of one or more columns. The 'rdbtbl' (or index_file) must be sorted on the key columns. Each column in the key may be of type string or type numeric (but be carefull with numeric data and exact matches). In the second form of usage for this operator if 'rdbtbl' is not given its name will be inferred from the name of index_file. For example if index_file is 'skb.x.typ' then the rdbtbl name inferred will be 'skb.rdb'.

The column(s) in the file 'keytbl' specify both the key column name(s) and the argument values to search for. File 'Keytbl' is in rdbtable format.

Normally an argument value and a data field must compare exactally for a match to occur (exact match). If the paritial match otpion (-part) is selected, and if the argument value compares with the initial part of the data field it is considered a match. This applies to string type data only. Note that for numeric type data an exact match is always necessary.

Normally all rows that match will be written to the new rdbtable, in the same order as in the old rdbtable. If only a single row key match is appropriate some execution time can be saved by specifing the '-sgl' option.

This operator writes an rdbtable via STDOUT. Options may be abbreviated. Returns the number of non-finds at exit.

3.21 nsq-sort

Usage: **nsq-sort [options] [-r] column [[-r] column] ...**

Sorts an rdbtable on one or more columns. Each column may be sorted in normal (ascending) or reverse (descending) order. Also a column of monthnames (Jan, Apr, ...) in any case letters, may be sorted.

This operator reads an rdbtable via STDIN and writes an rdbtable via STDOUT. Options may be abbreviated. Uses the UNIX 'sort' routine.

For example, using the sample data file from the DATA section (named sample) in the following command:

```
nsq-sort  COUNT  TYP  <  sample
```

would produce:

```
NAME    COUNT   TYP     AMT     OTHER   RIGHT
6       5N      4       5N      8       8>
Bush    44      A       133     Another This
Hansen  44      A       23      One     Is
Holmes  65      D       1111    On      Edge
Perry   77      B       244     And     The
Hart    77      D       1111    So      Right
Jones   77      X       77      Here    On
```

Of course it would look better if it was piped through 'nsq-pr'. The command:

```
nsq-sort  COUNT  -r  AMT  <  sample
```

would produce:

```
NAME    COUNT   TYP     AMT     OTHER   RIGHT
6       5N      4       5N      8       8>
Bush    44      A       133     Another This
Hansen  44      A       23      One     Is
Holmes  65      D       1111    On      Edge
Hart    77      D       1111    So      Right
Perry   77      B       244     And     The
Jones   77      X       77      Here    On
```

3.22 nsq-subtot

Usage: **nsq-subtot [options] B_column ... -s column ...**

This operator lists subtotals of specified column(s) whenever the value of specified break columns(s) (B_column(s)) changes. A single break column will produce a sub-total of all specified columns on each line. If there is more than one break column given then in addition whenever the value of the first break column changes an additional line will be printed showing the sub-total for that group.

If no break column is given the first column is used; if no subtotal column is given then all columns of type numeric are sub-totaled.

This operator reads an rdbtable via STDIN and writes an rdbtable via STDOUT. Options may be abbreviated.

Example rdbtable (named small):

```
name      amt    typ    count   n
6         5N     4      5N      2
Hansen    39     A      23      3
Hansen    9      A      3       3
Hansen    9      B      3       4
Jones     42     B      144     5
Jones     4      B      14      5
Hart      9      C      3       5
Hart      2      C      55      6
Hart      2      D      55      6
Hobbs     57     X      7       4
Hobbs     5      X      57      4
```

The output from the command:

```
nsq-subtot  name  -s  amt  <  small  |  nsq-pr
```

would be:

```
name      amt
------  -----
Hansen     57
Jones      46
Hart       13
Hobbs      62
```

The output from the command:

```
nsq-subtot name typ -s amt count < small | nsq-pr
```

is shown in Table 18.

```
                    Table 18

        OUTPUT FROM THE NSQ-SUBTOT OPERATOR

name    typ     amt  count
------  ----  -----  -----
Hansen  A        48     26
Hansen  B         9      3
                 57     29

Jones   B        46    158
                 46    158

Hart    C        11     58
Hart    D         2     55
                 13    113

Hobbs   X        62     64
                 62     64
```

3.23 nsq-summ

Usage: **nsq-summ [options] [column ...]**

Produces "summary" information about the rdbtable. If no columns are given then information about all columns is produced. A Count of the data rows is always shown.

This operator reads an rdbtable via STDIN and writes a summary report via STDOUT. Options may be abbreviated.

3.24 nsq-t2l

Usage: **nsq-t2l [options]**

Converts an rdbtable to "list" format. Long data fields are folded. This operator is mainly used by other operators.

This NoSQL operator reads an rdbtable from STDIN and writes an rdbtable to STDOUT. Options may be abbreviated.

3.25 nsq-tabletolist

Usage: **nsq-tabletolist [options]**

Section yet to be written. See 'nsq-tabletolist -help' in the meantime.

3.26 nsq-tee

Usage: **nsq-tee [options] rdbtable**

Reads an rdbtable via STDIN and [over]writes the rdbtable given as a command line argument, abiding by the NoSQL table locking scheme. This operator is named after yhe UNIX utility *tee*.

This operator can be safely used in constructs like :

```
nsq-compute ... < table.rdb | nsq-tee table.rdb | ...
```

without worring that the original input file table.rdb be destroyed before the first pipeline has completed.

If no output stream is desired on STDOUT then the latter can be directed to /dev/null, as usual:

```
nsq-compute ... < table.rdb | nsq-tee table.rdb > /dev/null
```

This operator is especially meant to be used in programs that need to modify rdbtables 'in place'. It has been designed for the GNU Bourne Again Shell (Bash), but it should work just as well with other Bourne compatible shells.

Note: the output table name given on the command line may not start with the carachter "," , to prevent possible clobbering of the backup table.

3.27 nsq-uniq

Usage: **nsq-uniq [options] column ...**

Reads the input rdbtable and compares adjacent rows. The second and succeeding copies of repeated rows, considering only the selected columns, are removed. That is, adjacent rows are considered equal if the data values in all of the selected columns are equal. The remaining rows are written to the output rdbtable.

Note that repeated rows must be adjacent in order to be found. Normally this means that the input rdbtable should be sorted on the selected columns for this capability to work properly.

This NoSQL operator reads an rdbtable from STDIN and writes an rdbtable to STDOUT. Options may be abbreviated.

3.28 nsq-updseq

Usage: **nsq-updseq [options]**

Section yet to be written. See 'nsq-updseq -help' in the meantime.

3.29 nsq-valid

Usage: **nsq-valid [options] [file ...]**

Validates the structure of one or more rdbtables. Checks number of data fields per line, max width of column names and data values, and checks numeric data type values. Reports errors by line number and column name.

Reads from STDIN if filenames are not given. Writes diagnostic information on STDOUT. Options may be abbreviated.

If there is more than one file given each file will be identified on the output.

The '-size' option has proven very useful as it shows the actual size of the largest data value for each column, in addition to the template information. The command:

```
nsq-valid  -size  sample
```

shows the following output:

```
0              NAME  6     6
1             COUNT  5N    2
2               TYP  4     1
3               AMT  5N    4
4             OTHER  8     7
5             RIGHT  8>    5
Columns: 6, Rows: 6, File format valid  sample
```

The last two columns above show the defined size of each column in the rdbtable, and the actual maximum size of the data values for each column in the rdbtable.

Chapter 4

Extracting information from rdbtables

4.1 General

The following shows some examples of how the system is usually used, which involves a combinations of operators. Using the rdbtable named 'sample' the command:

```
nsq-col NAME OTHER TYP AMT < sample | nsq-sort TYP AMT | nsq-pr
```

gives the output:

```
NAME    OTHER     TYP    AMT
------  --------  ----  -----
Hansen  One       A        23
Bush    Another   A       133
Perry   And       B       244
Hart    So        D      1111
Holmes  On        D      1111
Jones   Here      X        77
```

Note that columns COUNT and RIGHT were excluded by the 'nsq-col' operator, and that the order of the selected columns was changed from that in the rdbtable. Of course to save the output in a file, (redirection of STDOUT into a file) something like the following is used:

```
nsq-col ... < sample | nsq-sort ... | nsq-pr > file.out
```

An example using the operator 'nsq-row' on the rdbtable sample is:

```
nsq-row AMT lt 900 < sample | nsq-col NAME AMT TYP RIGHT |\
      nsq-sort l NAME | nsq-pr
```

Note that the '\' character at the end of the first line of the above command is the signal to the UNIX shell that the command is continued on the next line. Here we select some rows using 'nsq-row', select some columns using 'nsq-col', sort what we have with 'nsq-sort', and print with 'nsq-pr'. The output is:

```
NAME      AMT  TYP      RIGHT
------  -----  ----  --------
Bush      133  A         This
Hansen     23  A           Is
Jones      77  X           On
Perry     244  B          The
```

4.2 A real world problem

The following shell script shows how the NoSQL operators and other UNIX utilities can be fitted together to solve a real world problem. The problem was to find out if the rows in a large rdbtable were unique over four columns. Since 'nsq-summ' will tell us whether the rows of an rdbtable are unique on a single column, we need to construct a temporary tdbtable.

To illustrate the solution on a small rdbtable, the script below works on an rdbtable like 'sample' but with some rows repeated. First the script selects four columns and adds a dummy column named 'uniq' (using 'nsq-col'). It then puts the combined values of the four columns into the dummy column (using 'nsq-compute'). Next it examines the value of the dummy column 'uniq' for uniquness (using 'nsq-summ') and then uses the UNIX command 'egrep' to show only the lines of interest, e.g. those lines that start with something other than a '1'.

```
nsq-col < table.rdb  Unit  Day  Time  MSN  -a uniq 12 |\
    nsq-compute uniq = Unit . Day . Time . MSN |\
    nsq-summ  -cuu  uniq  |\
    egrep -v  '^  *1'
```

The output was like the following:

```
Rows: 9
Unique values for uniq: 6
       2 Jones77X77
       3 Perry77B244
```

meaning that there were (in this example) two rows that had duplicates over the four columns with one set of values, and three rows that had duplicate with another set of values.

4.3 Another real world problem

This next one is a bit more complicated although very useful, and it does demonstrate the use of 'nsq-compute' using a newly created column. The idea is to make a summary of the data in rdbtable sample3 (from the section on 'nsq-ed'). The command is:

```
nsq-col name datatype -a nr 2 < sample3 |\
      nsq-compute nr = '++$x' |\
      nsq-sort datatype name | nsq-report sum.frm
```

Note that 'nsq-col' selects the two columns on which to make the summary and adds a new (null) column 'nr'. Then 'nsq-compute' puts data into the new column by using the PERL expression shown. The expression '++$x' merely increments itself by one each time it is evaluated, an easy way to get an increasing number. Finally 'nsq-sort' sorts the newly created rdbtable and it is then printed with 'nsq-report' using the form file 'sum.frm'.

```
                           Table 19

                      FORM FILE (SUM.FRM)

format top =
                                               @<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                 _date_
            Sample Three Database Summary by Datatype

       Datatype  Nr  Name
       --------  --  -----------------------------------------
.
format =
       @>>>>>>>  @>  @<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
       datatype  nr  name .
```

```
                           Table 20

                    DATABASE SUMMARY EXAMPLE

                                 Wed Dec  4 21:23:20 PST 1991
Sample Three Database Summary by Datatype

Datatype  Nr  Name
--------  --  -----------------------------------------

ABC    5  ABC Duplicate
ABC    4  ABC Original
ABC    8  LAF-S Air Floating Model
ATO    3  ATO (Air Tasking Orders) Original
ATO    6  ATO Number 2222-11
BDA    1  ACAS (Air Combat Assessment)
```

```
BDA   2  ACAS (Air Combat Assessment) BDA Sorties
BDA   7  BDA Supplement number 17
```

The form file is shown in Table 19 and the output in Table 20.

Chapter 5

Generating or modifing rdbtables

5.1 Generating new rdbtables

Any editor may be used to construct or modify an rdbtable, since it is a regular UNIX file, and this 'direct editing' method is occasionally used, especially for small amounts of data. However, avoid using an editor that destroys TAB characters.

To generate a new rdbtable the best plan (and usually the safest one) is to first generate a template file, then convert it to rdbtable format and add the rows of data. Any convenient editor may be used to generate a template file. To convert it to an rdbtable the command 'nsq-headchg -gen' may be used, which will produce an empty rdbtable. Next use the operator 'nsq-ed' to edit in rows of data.

A typical template file is shown below:

```
# These are lines of table documentation. They can be of any length,
# and any number of such lines may exist.
# Each line must start correctly, e.g with "# " or " #". Any number of
# space characters may preceed the sharp sign in the second case above.
0       Name  24   Name of item
1       Type   1   Type: 1,2,3,7,9,A,B,X
2      Count   3N  Number of items
3          K   1   Constant modifier
4        SS7   2   Special status for type 7
5       Size  12N  In Kilobytes
```

It makes sense to have all significant or critical documentation about an rdbtable embedded in the rdbtable, rather than in some other place. The above template file contains the usual elements to describe a table of six columns: table documentation (the comment lines that each start with a sharp sign '#'), index number (the first number on each of the column lines), column name ("Name", "Type", "Count", ...), column definition ("24", "1", "3N", ...), and column documentation for each column (the text at the end of each column line).

Note that the index number, column name, and column definition all consist of contiguous characters, each forming a word separated by whitespace. Also note that there is one or more space characters after the column definition and before the column documentation. That is, the column documentation starts with the fourth word on the line.

When the template file is converted into an rdbtable, all documentation will remain in the header (although the column documentation may be hard to read if there are many columns). At any time the entire header, including documentation, can be viewed by using the command 'nsq-valid -templ < rdbtable' (or 'nsq-headchg -templ < rdbtable). The output from either command will be essentially like the above example.

5.2 Modifing existing rdbtables

Basically there are two ways to modify an existing rdbtable: Use either 'nsq-ed', or 'nsq-merge'.

The operator 'nsq-ed' can be used to add new rows, change existing rows, or delete existing rows of data in an rdbtable. To modify an rdbtable 'nsq-ed' can be used in either column or list form. The choice of form to use depends somewhat on the structure of the rdbtable. If the rdbtable has several columns of relatively narrow data (that will all fit in the width of the current window or terminal) and also several very wide columns (none of which will fit) and changes need to be made to some of the narrow columns, then it makes sense to use 'nsq-ed' on the desired narrow columns in 'column' form, as in:

```
table  narrow_cola  narrow_colb  ...
```

If changes need to be made to some of the wide columns then use 'nsq-ed' in 'list' form on the wide columns, plus any key columns necessary, as in:

```
nsq-ed  -list  table  control_col  ...  wide_cola  wide_colb  ...
```

After editing an rdbtable it is always recommended that the structure of the rdbtable be checked with the operator 'nsq-valid'. If there are data values that are longer than the defined column width, use the command 'nsq-valid -w' to cause a more verbose output.

The 'nsq-merge' process actually involves other operators, like 'nsq-search' and 'nsq-ed', and works only when the existing rdbtable is sorted on one or more columns (which is a fairly common case). The process includes selecting rows from an existing sorted rdbtable (using 'nsq-search') into a small rdbtable which is easy to edit (using 'nsq-ed') and then combining the two rdbtables again (using 'nsq-merge'). Since 'nsq-ed' is used modifications may include changes, additions, or delitions of rows. Also note that 'nsq-merge' keeps the final table in sort order.

The difference is that 'nsq-search' is much faster than 'nsq-row' or 'nsq-ed', the editing is done on a table of conveniently small size, and that the 'nsq-merge' operation can be done in the background. Remember that whether one uses 'nsq-merge' or 'nsq-ed', putting the data back together after editing requires the entire original table to be passed, which can take some time if the original rdbtable is large.

5.3 Concatenating rdbtables

The need to concatenate rdbtables comes up every so often and although it is simple to do it may not be obvious. The UNIX 'cat' command can not be used as it would result in duplicating the header and thus make an invalid rdbtable. And of course, only rdbtables with the same header should be concatenated, otherwise an invalid rdbtable would result (in this case it could be a gross inconsistency if the number of columns were different). If we have two rdbtables, TABA and TABB, then to concatenate TABB onto the end of TABA we use the command:

```
nsq-headchg -del < TABB >> TABA
```

Note that this avoids duplicating the header. Note also that in this case the operator 'nsq-headchg' does not use a template file.

Note also that the operator 'nsq-merge' may be used to merge two like rdbtables based on a key of one or more columns. In this case however the two rdbtables must be sorted on the key.

Chapter 6

Conversion of external data into rdbtables

The easiest and probably most efficient way to convert data in files that were received from an external source is to use the AWK Programming Language. If the data file to be imported has a fixed field structure then GAWK (GNU-AWK) may be preferable over standard AWK, as the former can cope easily with fixed-field records. Note that the rdbtable header must be inferred manually from the structure of the input file.

The best way to accomplish the above is to first construct a template file from known or observed information in the external data file, then generate the header using 'nsq-headchg -gen'. Then run the AWK script appending the output to the header file. Then run 'nsq-valid' to make sure everything went allright.

Another possibility is using the PERL Programming Language. A typical PERL script for such a conversion from a file of fixed column data is shown in Table 21. The last two lines do most of the work. Note that this process converts the data into an rdbtable body only; the rdbtable header must still be generated manually.

Table 21

PERL SCRIPT FOR DATA CONVERSION

```
#! /usr/local/bin/perl
$templ = "A6 A12 x A5 x6 A18 A17 x A38 x4 A2 x A x2 A15 x3 A6 A4 x2" .
         " A2 x4 A5 x A3 A3 A4 x26 A12 A12 A12 A29 x7 A3 A3 x6 A6 A6 A8" .
         " A4 A3 x3 A3 x3 A8 x10 A2 x4 A12 x138 A6 x66 A24 A36 A12 x6" .
         " A6 A6 x6 A6 A24 A12 x50 A42 A42 A42 x34 A12 A12 x186 A6" ;
$0 =~ s-.*/-- ;
$HelpInfo = <<EOH ;

    Strip out and reform an 'external' data file into an rdbtable.

Usage:  $0  [options]  file
```

```
Options:

    -help    Print this help information.

Strips out the first 46 fields from an 'external' data file and
reformats it into 'rdbtable' format (TAB delimited with NEWLINE at end).
Output is on STDOUT.
EOH
while ( $ARGV[0] =~ /^-/ ) {                            # Get args
    $_ = shift ;
    if( /-h.*/ ){ die $HelpInfo ; }
    die  "Bad arg: $_\n", $HelpInfo ;
}
while(<>){
    @a = unpack( $templ, $_ );
    print join( "\t", @a), "\n" ;
}
```

Chapter 7

Data access control

7.1 General

Since rdbtables are regular UNIX files, we have so far found no need to implement, and have the associated overhead of, general data access controls in NoSQL. Setting the UNIX permissions on files or directory has proven very useful and effective. This is another example of how the NoSQL system works "with" UNIX, not in addition to it, e.g. not duplicating UNIX functions.

The Revision Control System (RCS) is one of the best configuration management tools available, it can be used for version control of many types of files, including rdbtables. The operator 'nsq-ed' will automatically check out an rdbtable for editing, and then check the new version back into RCS. Other operators can utilize rdbtables that are under RCS control by using commands like:

```
co -p table | nsq-row ... | nsq-col ... | nsq-pr
```

Note that this checks out an rdbtable, sends it to 'nsq-row', then to 'nsq-col', and finally prints the data with 'nsq-pr'. In general, any series of commands necessary can be constructed to do a given task even if the rdbtables are checked into RCS.

7.2 Write concurrency control

When either of the two utility operators 'nsq-ed' or 'nsq-tee' is used (which modify an rdbtable in place) there could be a possibility of silmultaneous wrting of an rdbtable by multiple users. That is, if two or more users, on the same computer or perhaps on different computers on a network, attempted to modify a given rdbtable with either 'nsq-ed' or 'nsq-tee' at the same time, the rdbtable could become corrupted. To prevent this, write concurrency control is provided by the use of a lockfile, and is in effect when either of the two utility operators is used.

The name of the lockfile is the name of the rdbtable being modified with a suffix of ".LCK". For example an rdbtable named "main.rdb" would have a lockfile named "main.rdb.LCK". The lockfile is placed in the same directory as the rdbtable and is normally removed after the modification process is complete, even if the operation is aborted with an INTERRUPT signal (CONTROL-C or <DEL>). However in the event of an emergency such as a computer system crash the lockfile could be left in place, preventing the use of 'nsq-ed' or 'nsq-tee' when the computer system is again operable. When an attempt to use either utility operator is made and there is an existing lockfile associated with the referenced rdbtable an online message is produced and the operator dies. In this case simply remove the lockfile with the UNIX command 'rm' and proceed. If an emergency has not occurred appropriate caution should be exercised before removing a lockfile, due to the possibility of data corruption. To overcome this problem, NoSQL honours the environment variable NSQLOCKS. If set, this variable instructs NoSQL to use a centralized directory for creating the lock files. It is then quite easy to set up an automated procedure that cleans up that directory at machine boot time.

A related environment variable is NSQLOCKER; it designates an external locking program to be used in place of the *nsq-lock* shell script provided by NoSQL. A very good locking program is the *lockfile* utlility, normally distributed with *procmail* (a mail filtering system), which usage is recommended over the trivial nsq-lock utility. NSQLOCKER, if set, must contain any extra arguments needed by the locking program, e.g. NSQLOCKER="lockfile -r1". Currently only *nsq-tee* honours NSQLOCKER, while the other NoSQL utilities that need file locking handle it internally (i.e. they do not even use nsq-lock).

Chapter 8

Fast access methods

The NoSQL operator 'nsq-search' may be used to execute one of two fast access methods, binary or hashtable search.

These methods are useful when the key field values of a number of rows in an existing, large rdbtable is known in advance, a common situation.

One example of updating an rdbtable using these methods is as follows. First, 'nsq-search' is used to quickly obtain a new, small rdbtable consisting of the desired rows from the existing, large rdbtable. Then 'nsq-ed' is used to update the data in the new rdbtable, including generating new rows, changing some rows, and marking certain rows for deletion, if desired. The next stage would be to use 'nsq-merge' to combine the new and old rdbtables into a new, large rdbtable, which will still be in sort order (both the old rdbtable and the new, small one will still exist for backup and/or journaling purposes). Note that the last step could be done in the background.

Chapter 9

Limits, a few minor ones

The following limits apply.

There must not be any ASCII TAB characters in the data. This is the primary limit as the ASCII TAB character is the delimiter in rdbtables. There must not be any pipe characters '|' used or entered as data when using the operator 'nsq-ed'. All of the Compare operators and Logical constructors are reserved and should not be used as column names (they are all lower case and four characters or less). They are: 'gt', 'ge', 'lt', 'le', 'eq', 'ne', 'or', 'and', 'null', 'mat', and 'nmat'.

Chapter 10

References

1. "Unix Review" magazine, March, 1991, page 24, "A 4GL Language".
2. "UNIX Relational Database Management", R. Manis, E. Schaffer, R. Jorgensen, 1988, Prentice Hall.
3. "Programming PERL", L. Wall, R. Schwartz, 1991, O'Reilly & Associates.

Appendix A

Examples of perl expressions and statements

Following are some examples of PERL expressions and statements of the type that might be used with NoSQL operators, and their meaning. Note that the operator 'nsq-row' takes a PERL expression while the operator 'nsq-compute' takes a complete PERL statement.

Expressions:

```
COLA mat /XXX/

               -- column COLA contains the pattern 'XXX'.

COLA nmat /XXX/

               -- column COLA does NOT contain the pattern 'XXX'.

COLA mat /^XXX/

               -- column COLA starts with the pattern 'XXX'.

COLA mat /XXX$/

               -- column COLA ends with the pattern 'XXX'.

COLA ne null

               -- column COLA is not null (but it could contain blanks).

COLA mat /^\s*$/

               -- column COLA is null or contains only blank space.

COLA eq 'YYY'

               -- column COLA equals the literal 'YYY'.

COLA mat /X..Y/
```

```
                -- column COLA contains the pattern 'X..Y', which means
                   'X', followed by any two characters, then 'Y'.

COLA mat /X.*Y/

                -- column COLA contains the pattern 'X.*Y', which means
                   'X', followed by any number of (including zero)
                   characters, then 'Y'.

NUMC eq 12

                -- column NUMC equals 12.

COLA ne null && COLB ne null

                -- column COLA and column COLB are not null (empty).

COLA eq 'ABC' || COLA eq 'BCD'

                -- column COLA equals the literal 'ABC' or column COLB
                   equals the literal 'BCD'
```

Statements:

```
COLA = COLB ;

                -- set the value of column COLA to that of COLB.

NUMC = NUMC - 7 ;

                -- decrement the value of column NUMC by 7.

NUMC -= 7 ;

                -- (same as above).

NUMC = NUMC / 4 ;

                -- divide the value of column NUMC by 4.

NUMC *= 2.3 ;

                -- multiply the value of column NUMC by 2.3.

$abc++ ;

                -- increment the value of variable $abc by 1.

++$abc ;

                -- (same as above).
```

```
COLA = 'WORDS' ;

               -- set the value of column COLA to the literal 'WORDS'.

NUMC = 12 ;

               -- set the value of column NUMC to 12.

if( COLA mat /XXX/ ){ COLA .= 'YYY' ; }

               -- If column COLA contains the pattern 'XXX' then add the
                  literal 'YYY' to the end.

COLA .= 'YYY' if COLA =~ /XXX/ ;

               -- (same as above).

if( COLA eq 'ABC' || COLA eq 'BCD' ){ COLA = 'XXX' ; }

               -- If column COLA equals 'ABC' or 'BCD' set the value of
                  COLA to 'XXX'.
```

Appendix B

Copyright terms.

B.1 GNU General Public License

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                          675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
```

rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

```
  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
```

machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or

otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals

of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software

```
    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) 19yy name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```