Paper 162-30

Benchmarking Character String Replacement

Bruce Gilsen, Federal Reserve Board

INTRODUCTION

A user wanted to change all occurrences of 5 characters (", ', <, >, &) in a character variable to a space. To help determine which of the many methods provided by the SAS ® system to recommend, benchmark tests were conducted for various coding methods and data scenarios. This paper shows the results of these tests, and makes some recommendations based on the data being processed.

CODING METHODS AND DATA SCENARIOS

Four coding methods were considered.

1. TRANWRD function.
2. TRANSLATE function.
3. INDEXC and SUBSTR functions.
4. Perl regular expressions, available starting in Version 9.

The number of observations, the length of the character string, and the frequency of changes could affect the results, so data sets containing 1,000, 100,000, 1,000,000, and 5,000,000 observations were tested using the following data scenarios.

Data scenario	Character variable length	% of observations with changes
1	30	100%
2	30	0.4%
3	125	100%
4	125	0.4%

The benchmark tests were conducted with SAS for UNIX (Solaris) Version 9.1.

DATA USED IN THE BENCHMARK TESTS

The four DATA steps used to generate the input data are as follows. In each case, the number of iterations in the DO I = 1 TO loop was modified to generate the appropriate number of observations.

Data scenario 1. Character variable with length 30, changes in all observations.

```
data testdata;
  drop i;
  length xx $30;
  do i = 1 to 1000000; /* upper loop bound one of: 200, 20000, 200000, 1000000 */
    /* record with none of characters */
  xx='321cba32&1c”ba3>721 <cba';
  output;
    /* record with some of chars 1x */
  xx='abc< 127>3ab"c1&23abc123';
  output;
    /* record with some of chars more than 1x */
  xx='abc<<<12>3a<b"c1&23ab"c123';
  output;
    /* record with some of chars more than 1x */
  xx="abc<<<12>3a<b'c1&23ab'c123";
  output;
    /* record with some of chars more than 1x */
    /* including the last character */
  xx="abc<<<12>3a<b'c1&23ab'c123dds'";
```

```
    output;
    end;
  run;
```

Data scenario 2. Character variable with length 30, changes in 0.4% of observations.

```
  data testdata;
    drop i j;
    length xx $30;
    do i = 1 to 5000; /* upper loop bound one of: 1, 100, 1000, 5000 */
        /* record with none of characters */
      xx='abc123abc123abc123';
      output;
        /* record with some of chars 1x */
      xx='abc< 127>3ab"c1&23abc123';
      output;
        /* record with some of chars more than 1x */
      xx='abc<<<12>3a<b"c1&23ab"c123';
      output;
        /* record with some of chars more than 1x */
      xx="abc<<<12>3a<b'c1&23ab'c123";
      output;
        /* record with some of chars more than 1x */
        /* including the last character */
      xx="abc<<<12>3a<b'c1&23ab'c123dds'";
      output;
        do j = 1 to 995;
            /* record with none of characters */
          xx='abc123abc123abc123';
          output;
        end;
    end;
  run;
```

Data scenario 3. Character variable with length 125, changes in all observations.

```
  data testdata;
    drop i;
    length xx $125;
    do i = 1 to 5000000; /* upper loop bound one of: 1000, 100000, 1000000, 5000000 */
        /* record with some of characters */
xx='abc123abc123abc123abc<127>3ab"c1&23abc123abc<<<12>3a<b"c1&23ab"c123"abc<<<12>3a<b"c1&
23ab"c123abc<<<12>3a<b"c1&23ab"c123dds""';
      output;
    end;
  run;
```

Data scenario 4. Character variable with length 125, changes in 0.4% of observations.

```
  data testdata;
    drop i j;
    length xx $125;
    do i = 1 to 5000; /* upper loop bound one of: 1, 100, 1000, 5000 */
      do j = 1 to 4;
          /* record with some of characters */
        xx='abc123abc123abc123abc<
127>3ab"c1&23abc123abc<<<12>3a<b"c1&23ab"c123"abc<<<12>3a<b"
c1&23ab"c123abc<<<12>3a<b"c1&23ab"c123dds""';
        output;
      end;
      do j = 1 to 996;
        /* record with none of characters */
xx='abc123abc123abc123abczz127z3abec1s23abc123abchhh12r3arbrc1r23abrc123qabcjui12s3a4bu
c1l23abzc123abc27612d3adbdd1x23ab4c123dds18';
        output;
      end;
    end;
  run;
```

CODE USED IN THE BENCHMARK TESTS

One DATA step was employed for each coding method.

Method 1. The TRANWRD function.

```
data one;
  set testdata;
  xx = tranwrd(xx,'&',' ');
  xx = tranwrd(xx,'<',' ');
  xx = tranwrd(xx,'>',' ');
  xx = tranwrd(xx,'"',' ');
  xx = tranwrd(xx,"'"," ");
run;
```

Method 2. The TRANSLATE function.

```
data two ;
  set testdata;
  xx = translate(xx,' ','&<>"''');
run;
```

Method 3. The INDEXC and SUBSTR function.

```
data three;
  set testdata;
  drop i colcount;
  i = indexc(xx,"&<>""'");     /* returns 0 or location of a match */
  colcount = 0;                /* keep track of where we are in xx */
  do while (i ne 0);
    substr(xx,i+colcount,1) = " ";     /* change bad char to space */
      /* note: we want to set colcount = colcount+i, but that causes
         an error in the indexc(substr()) function if the last character
         of xx is a match, since colcount+1 would be larger than the
         variable length */
    colcount = colcount + i-1;
      /* look for match starting where the last
         match was found, NOT the start of xx */
    i = indexc(substr(xx,colcount+1),"&<>""'");
  end;
run;
```

Method 4. Perl regular expressions.

```
data four ;
  set testdata;
  retain regexp;
  drop regexp;
    /* Change &<>"' to space, represent " as "". */
  if _n_ = 1 then regexp = prxparse("s/[\&<>\""\']/ /");
  call prxchange(regexp,-1,xx);
run;
```

BENCHMARK TESTS: RESULTS

The following tables show the CPU time in seconds for each test. The same results are shown graphically in the appendix.

1. Character variable with length 30, changes in all observations.

	OBSERVATIONS			
	1,000	100,000	1,000,000	5,000,000
TRANWRD	0.07	1.84	18.32	92.94
TRANSLATE	0.02	0.66	6.57	34.10
INDEXC/SUBSTR	0.03	1.60	16.04	81.04
PERL REGEXP	0.14	4.13	41.01	206.78

2. Character variable with length 30, changes in 0.4% of observations.

	OBSERVATIONS			
	1,000	100,000	1,000,000	5,000,000
TRANWRD	0.04	1.58	15.27	77.97
TRANSLATE	0.02	0.67	6.52	33.66
INDEXC/SUBSTR	0.01	0.48	4.59	24.76
PERL REGEXP	0.10	1.11	10.60	56.77

3. Character variable with length 125, changes in all observations.

	OBSERVATIONS			
	1,000	100,000	1,000,000	5,000,000
TRANWRD	0.10	5.23	53.59	268.42
TRANSLATE	0.02	1.18	12.21	63.08
INDEXC/SUBSTR	0.08	6.24	63.39	318.04
PERL REGEXP	0.26	19.00	191.61	959.47

4. Character variable with length 125, changes in 0.4% of observations.

	OBSERVATIONS			
	1,000	100,000	1,000,000	5,000,000
TRANWRD	0.07	3.74	38.95	194.48
TRANSLATE	0.03	1.16	12.54	63.13
INDEXC/SUBSTR	0.03	1.08	11.55	56.91
PERL REGEXP	0.08	2.49	22.40	113.73

BENCHMARK TESTS: SUMMARY AND RECOMMENDATIONS

1. When the variable is modified in most or all records (data scenarios 1 and 3), TRANSLATE was by far the best method. Perl regular expressions were by far the worst method.

2. When the variable is modified infrequently (data scenarios 2 and 4), INDEXC/SUBSTR was the best method. It was about 10-30 percent faster than TRANSLATE and 2-4 times faster than the other methods.

3. The same tests were conducted with SAS for Linux Version 9.1.2 and SAS for Windows Version 9.1. The results were

similar, except for the relative performance of INDEXC/SUBSTR and TRANSLATE when the variable is modified infrequently (data scenarios 2 and 4).

For Linux, INDEXC/SUBSTR performed only 5-10 percent better than TRANSLATE.

For Windows, INDEXC/SUBSTR performed between 10 percent better and 10 percent worse than TRANSLATE.

4. Unless you are certain that the variable will be modified very infrequently on UNIX or Linux, TRANSLATE is the best method.

BENCHMARK TESTS: NOTES

1. I suspected that Perl regular expressions would run slowly for a small number of observations because of fixed startup costs, but faster for a large number of observations. In fact, there was no per-observation speedup as the number of observations increased. According to a SAS Institute employee, Perl regular expressions are sufficiently complex and powerful that using them for simple constructs is "overkill", regardless of the number of observations and frequency of changes.

2. Unlike the other three methods, the frequency of changes does not affect the performance of TRANSLATE. According to a SAS Institute employee, this is because when the statement xx = translate(xx,' ','<>"") is processed, XX must be copied to XX even if there are no changes.

3. Perl regular expression functions and call routines, introduced in Version 9, begin with PRX. SAS regular expression functions and call routines, introduced in Version 8, begin with RX. SAS regular expressions were not considered as a fifth test method because I suspect that they will not be widely used now that Perl regular expressions are available.

CONCLUSION

This paper detailed benchmark tests for character string replacement. Four coding methods were considered: the TRANWRD function, the TRANSLATE function, the INDEXC/SUBSTR functions, and Perl regular expressions. Multiple data scenarios were considered, in which data varied by the number of observations and the frequency of replacement. In most cases, the recommended method is TRANSLATE function. If you are certain that the variable will be modified very infrequently on the UNIX or Linux platforms, the recommended method is the INDEXC and SUBSTR functions.

For more information, contact

Bruce Gilsen
Federal Reserve Board, Mail Stop 157
Washington, DC 20551
phone: 202-452-2494
e-mail: bruce.gilsen@frb.gov

REFERENCES

SAS Institute Inc (1999), "*SAS/GRAPH Software: Reference, Version 8*," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "*SAS Language Reference: Concepts*," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "*SAS Language Reference, Version 8*," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "*SAS Procedures Guide, Version 8*," Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

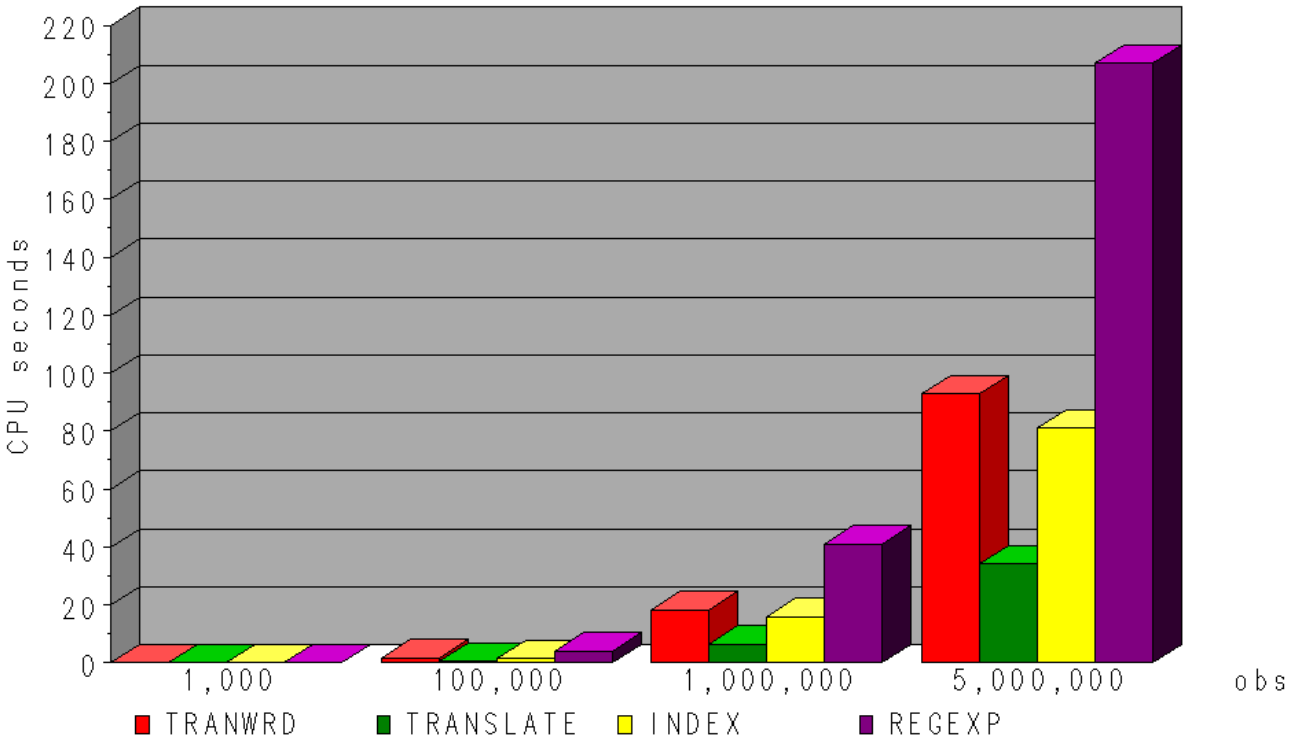
The following people contributed extensively to the development of this paper: Donna Hill, Scott Hoenig, and Steve Taubman at the Federal Reserve Board, Louise Hadden at Abt Associates, Inc., and Howard Schreier at the U.S. Department of Commerce. Their support is greatly appreciated.

TRADEMARK INFORMATION

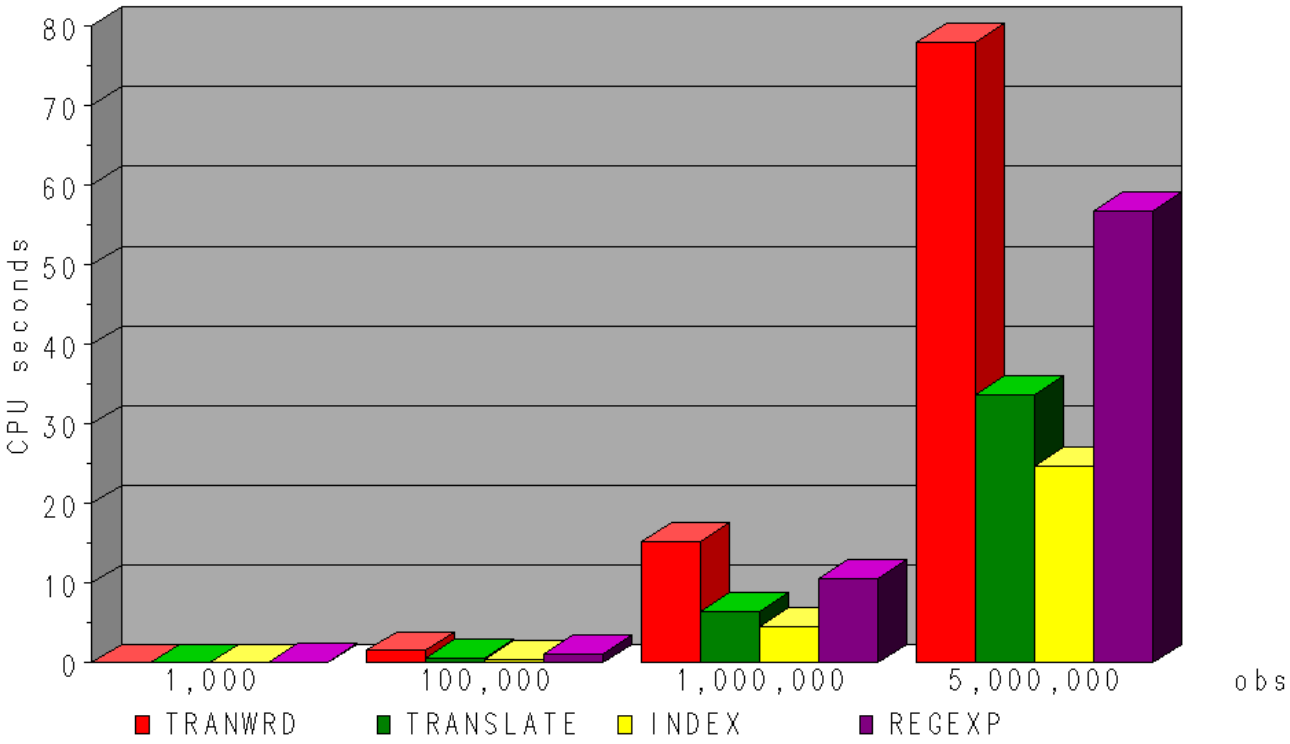
APPENDIX

The results of the benchmark tests are displayed graphically in the appendix using PROC GCHART, which is part of SAS/GRAPH ® software.

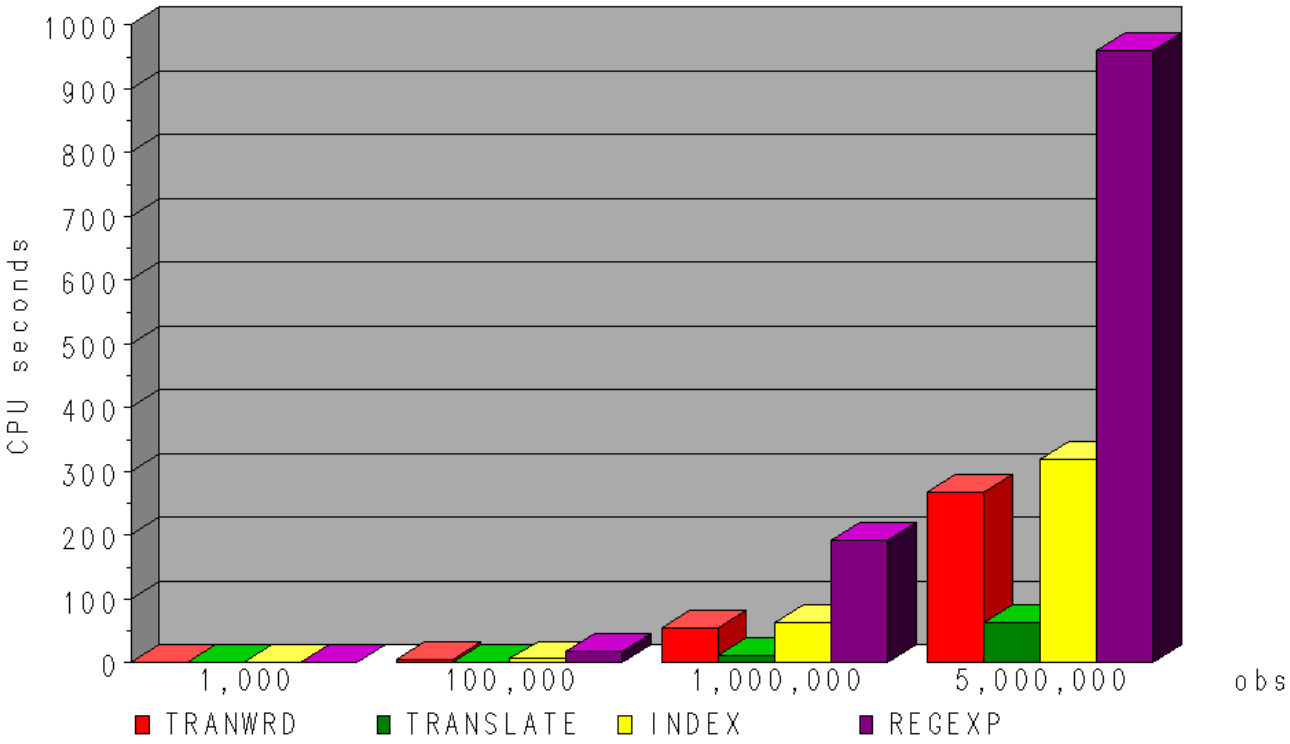
Scenario 1: Character length=30, changes in 100% of obs



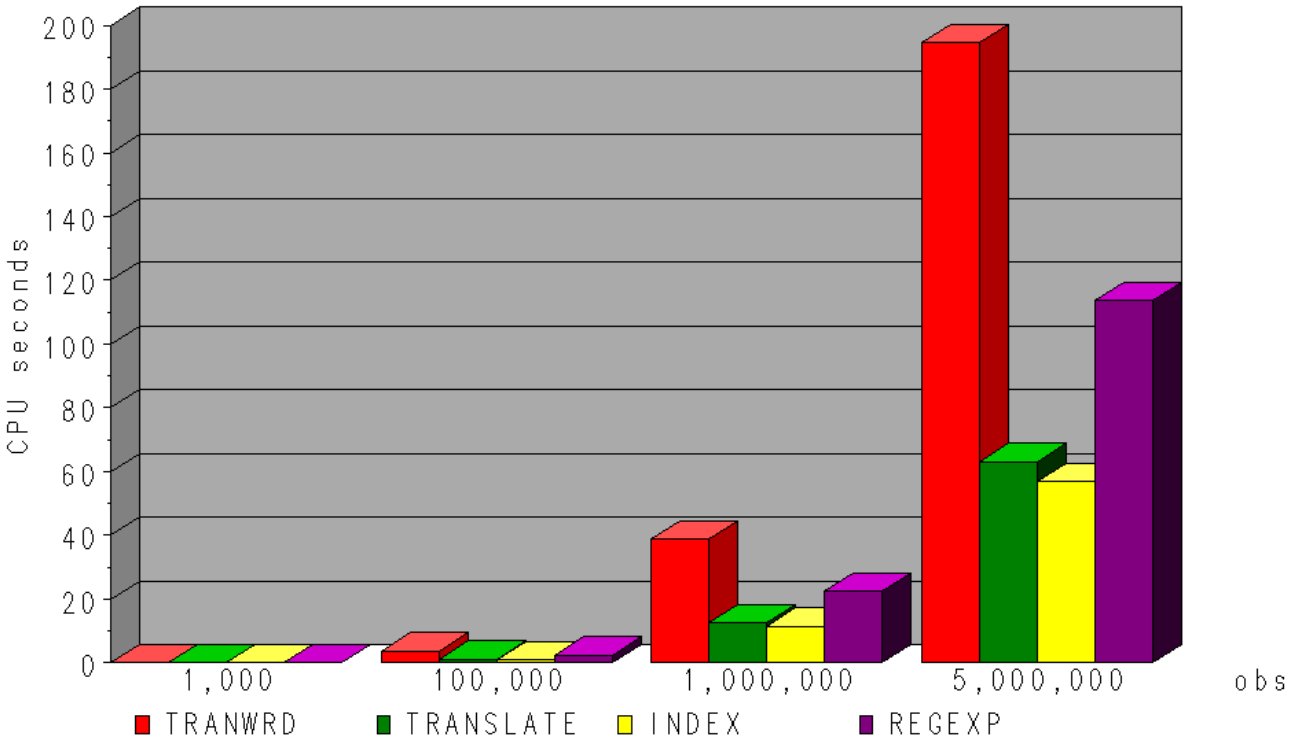
Scenario 2: Character length=30, changes in 0.4% of obs

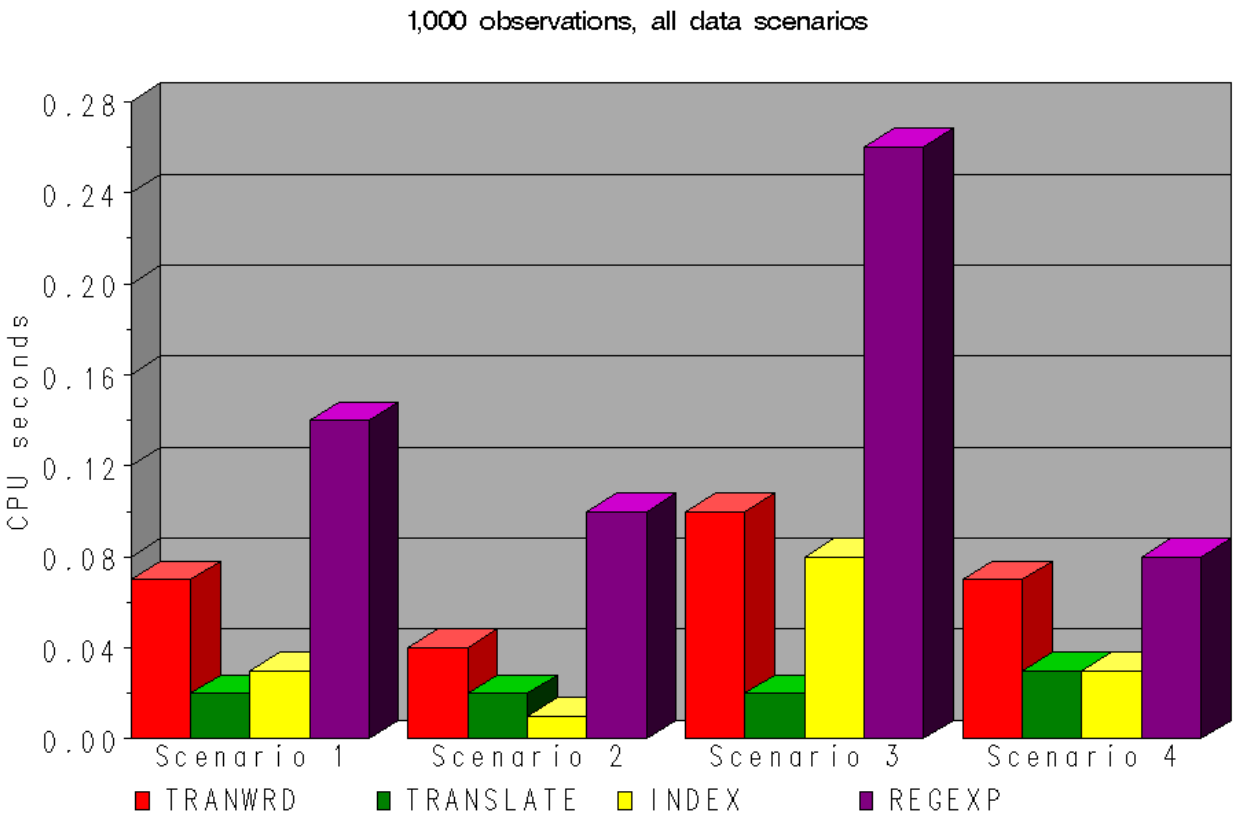


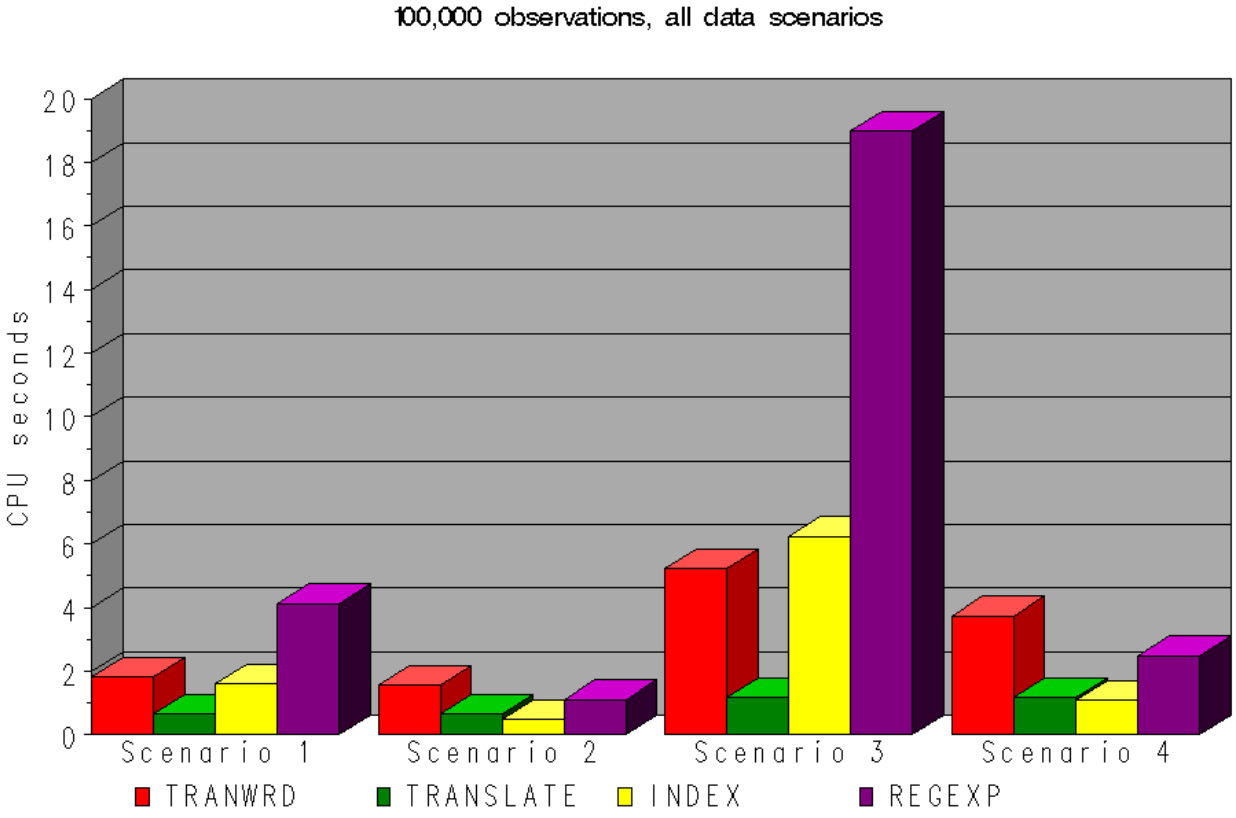
Scenario 3: Character length= 125, changes in 100% of obs

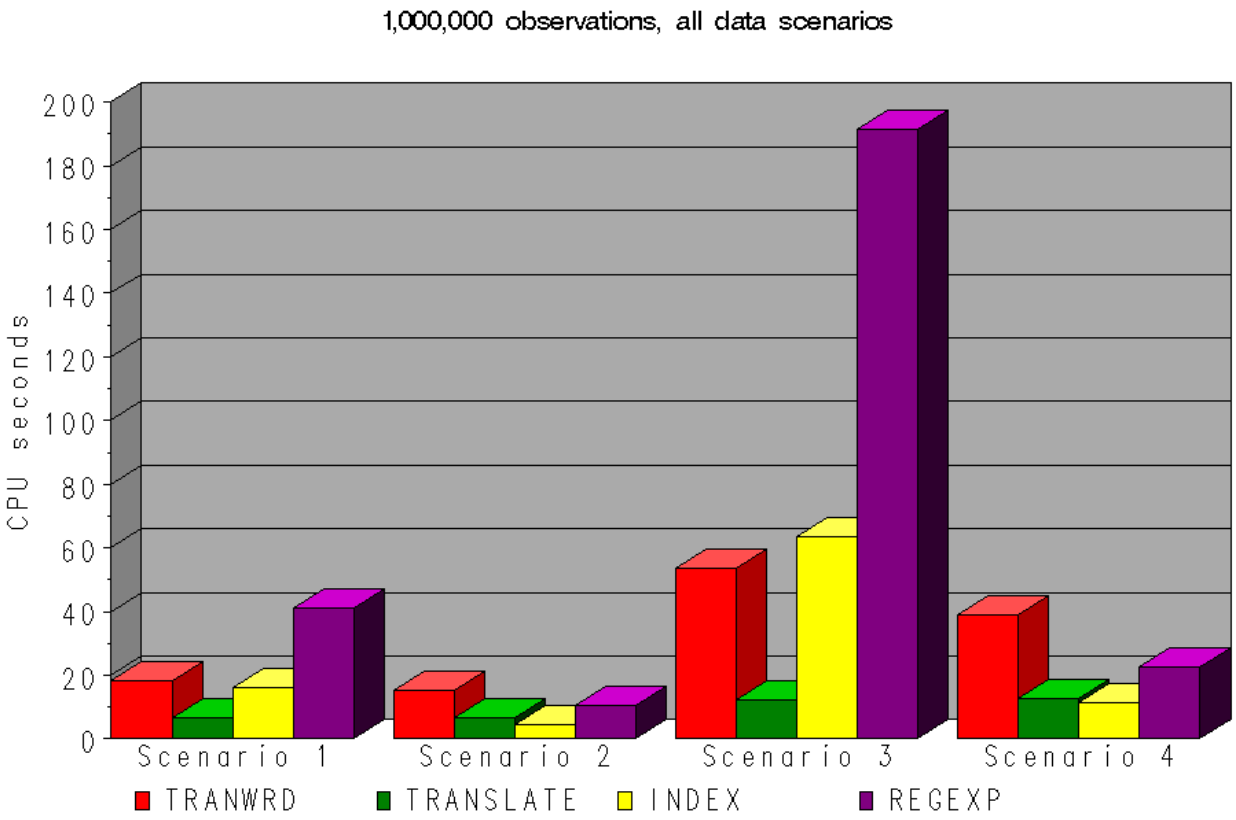


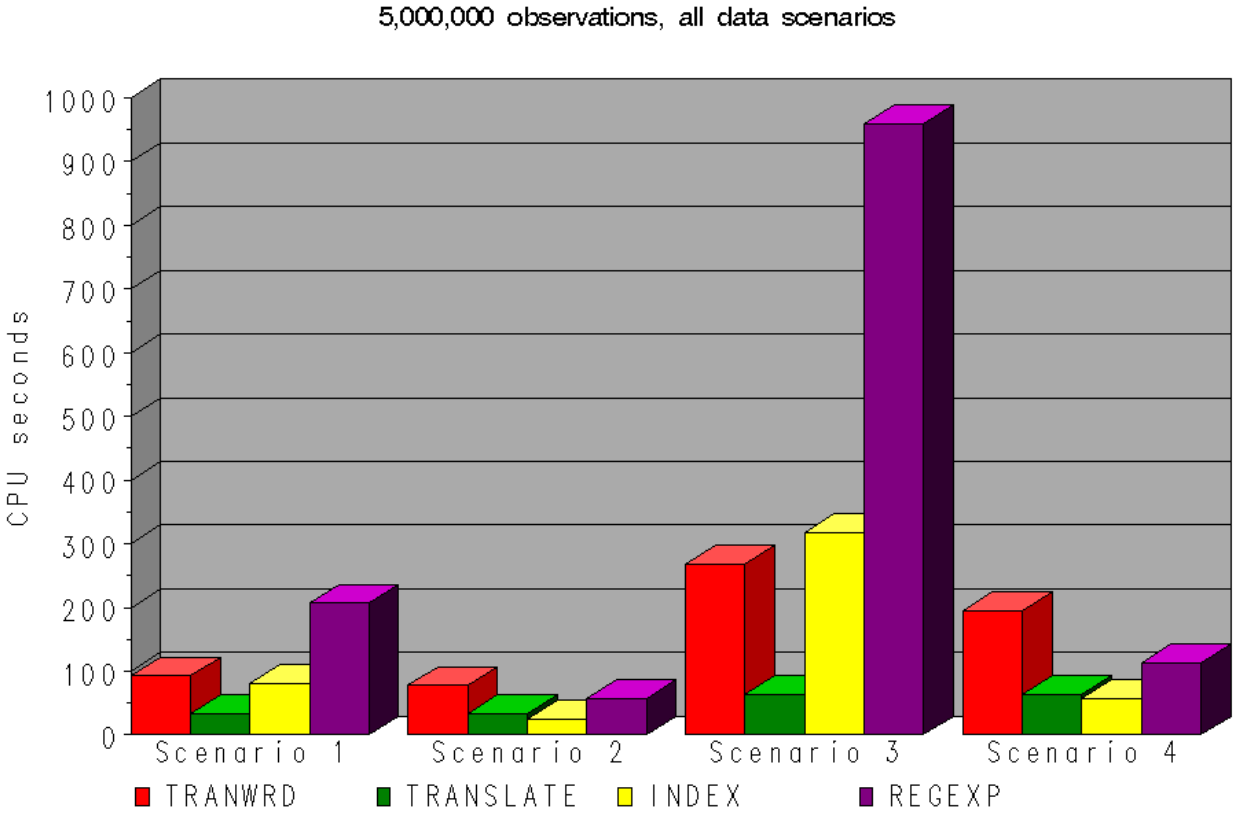
Scenario 4: Character length= 125, changes in 0.4% of obs




1,000 observations, all data scenarios
CPU seconds
0.28
0.24
0.20
0.16
0.12
0.08
0.04
0.00
Scenario 1
Scenario 2
Scenario 3
Scenario 4
TRANWRD
TRANSLATE
INDEX
REGEXP


100,000 observations, all data scenarios
CPU seconds
20
18
16
14
12
10
8
6
4
2
0
Scenario 1
Scenario 2
Scenario 3
Scenario 4
TRANWRD
TRANSLATE
INDEX
REGEXP


1,000,000 observations, all data scenarios
CPU seconds
200
180
160
140
120
100
80
60
40
20
0
Scenario 1
Scenario 2
Scenario 3
Scenario 4
TRANWRD
TRANSLATE
INDEX
REGEXP


5,000,000 observations, all data scenarios
CPU seconds
1000
900
800
700
600
500
400
300
200
100
0
Scenario 1
Scenario 2
Scenario 3
Scenario 4
TRANWRD
TRANSLATE
INDEX
REGEXP