Data Extraction Methods in Medium and Large Databases

Anthony M. Dymond, Dymond and Associates, LLC, Concord, CA

ABSTRACT

Data warehouse projects involve populating databases with unique subsets of records. These record subsets are frequently extracted from much larger sets that can sometimes contain many millions of records. This paper focuses on data extractions based on lookup tables, which can be implemented using several different techniques. These techniques are reviewed, and their performance differences are examined using several test data sets and flat files. The results of this study can help guide the warehouse designer's decisions regarding sorting, indexing, and choice of an extraction method.

INTRODUCTION

It is characteristic of data warehouses to scan larger source files and databases and to extract subsets of records that are then moved into the data warehouse. Data extraction can be done by performing logical operations on one or more variables in the records, or by comparing variables to values contained in a lookup table. This study focuses on using a lookup table to extract records.

Merges, SQL joins, formats, indexes, arrays, and macros all offer mechanisms that can be used to implement table lookups (Rafiee, 1997). These techniques, in particular merge and SQL joins, have been reviewed and compared (Bachler, 1996; Haase, 1996; Foley, 1997). Speed differences and the effects of indexing the source database have been identified. The use of formats to replace merge procedures and fuzzy SQL joins has been described (Levine, 1996; Zdeb, 1996).

All these approaches involve trade-offs. Merge and SQL will only work if the source data is in a data set, and are not applicable when the large data source is a flat file. Merge will not work unless both the source data set and the lookup table have been sorted or indexed, which can pose a problem when one of the data sets becomes excessively large. The various table lookup techniques also have performance differences that become very important in applications when the source database exceeds 10-100 million records, such as is encountered in telecommunications data warehouses.

The present study compares four of these table lookup techniques working against a range of source databases. The results should help warehouse designers in selecting and optimizing extraction methods.

METHODS

In this study, a SAS® data set with 10,000 records is used as the lookup table. Merge, SQL, key, and format methods are used to extract records from a data source that is either another data set or a corresponding flat file. The data sets can be unsorted, sorted, or indexed. The following notation will be used to describe these various combinations:

T = lookup table (data set)
B = big source data set
F = big source file

S = sorted
N = indexed

For example, SB would indicate the sorted big data set, and NT would indicate the indexed lookup table.

The MERGE operator with the following code can be used to extract records from the sorted big data set SB that have values for the variable VAR matching values in the sorted lookup table ST.

```
data temp;
  merge SB(in=inbig)
        ST(in=intab);
  by VAR;
  if inbig and intab;
run;
```

A similar result can be obtained by using the SQL procedure with the following code. In this case, neither the big data set B nor the lookup table T are sorted.

```
proc sql;
create table temp as
select *
from B,
     T
where B.VAR=T.VAR
;
quit; run;
```

The KEY option, as shown in the following simplified code, can also be used to extract records from a large data set. The code reads a record containing the variable VAR from big data set B, then searches in the lookup data set T that has been indexed on VAR. If a match is found, the system variable _IORC_ is set to zero and the code will continue processing the record from data set B. Since the big data set B will be read sequentially, it does not have to be either sorted or indexed. The lookup table T must be indexed on the key variable.

```
data temp;
  set B;
  set NT key=VAR;
  if _iorc_=0;
run;
```

With the following modification, this code can also be used to search a big flat file F. The code reads the variable VAR, looks for a matching value in the lookup table T indexed on VAR, and continues if the value is found.

```
data temp;
  infile F;
  input @1  VAR   varinfmt.  @;
  set NT key=VAR;
  if _iorc_=0 then do;
    input < more code >
          < more code >;
    end;
run;
```

The SAS FORMAT procedure implements an efficient memory-based lookup table that can be used for record extraction. Its use requires some preparation, as shown in the following code. The lookup table T is used to create a new data set TOCNTLIN containing the special variables START, FMTNAME, and LABEL. This data set is then used with the CNTLIN option in PROC FORMAT. Applying the FORMAT will then return the result "Y" whenever a variable is found that matches a value of VAR from the original lookup table T.

```
data tocntlin;
  set T(keep=VAR
         rename=(VAR=start)
         );
  retain fmtname 'fmtfmt'
          label 'Y';
run;
proc format cntlin=tocntlin; run;
```

The format is used to extract records, as shown in the following code.

```
data temp(drop=test);
  set B;
  test=put(VAR,fmtfmt.);
  if test='Y';
run;
```

The FORMAT method can also be used to extract records from a flat file.

```
data temp(drop=test);
  infile F;
  input @1  VAR   varinfmt.  @;
  test=put(VAR,fmtfmt.);
  if test='Y' then do;
    input < more code >
          < more code >;
    end;
run;
```

These four methods were applied to combinations of test data. The lookup table was a SAS data set containing 10,000 unique records, where each record had one variable holding a fully populated 8 digit number. The big data set or flat file contained from 10,000 to 10,000,000 records. Each record had three numeric and three character variables in the data set and equivalent characters in the flat file. One of the variables was a fully populated 8 digit number that had been randomly sampled to build the lookup table.

Studies were carried out on a 75 MHz Pentium® PC with 24M RAM, running the Windows® 95 operating system and SAS 6.12, and on an IBM® Model 9021 mainframe running the MVS operating system and SAS 6.09E. Studies were repeated in randomized order to minimize effects such as buffer preloading. The results are shown in figures 1 through 3.

RESULTS

Figure 1 shows the times required to extract 10,000 records from data sets and files containing 100,000 records on the PC. The various combinations of the merge, SQL, key, and format methods applied to the datasets and files are shown across the bottom of Figure 1. The total times shown are the sum of the actual extraction time plus any other preparatory operations required before the extraction could be performed. For example, the total time required for merging the sorted lookup table and the sorted big data set, MERGE(ST,SB), is the sum of the times for sorting both datasets and then using MERGE to do the record extraction.

The longest total times, as well as the longest extraction times, belonged to merges where the big data set was indexed, and to key extractions for

both the big data set and the big file. The shortest total times belonged to three of the SQL operations. The shortest extraction by itself belonged to the merge where the lookup table and the big data set were sorted, although the total time for this merge must include the time required to sort the data sets.

For the four merge combinations, indexing the large data set or the lookup table increased the extraction time and the total time. The effect was substantial when the large data set was indexed.

For the nine SQL combinations, sorting the lookup table or the large database increased the total time but had little effect on the extraction time. However, indexing the large data set not only added overhead time for the indexing but also significantly increased the extraction time.

In both the key and format extractions, the time to extract from the flat file F was slightly greater than from the unsorted data set B. The format technique was significantly faster than the key technique.

Figure 2 shows total times to extract 10,000 records from source data sets and files of 10,000, 100,000, and 1,000,000 records on the PC.

Data for the largest merge extraction is not shown because the computer used in this study was unable to sort or index the source data set with one million records. This does make the point that these operations can be very resource intensive and will become impractical at some point on any computer.

Extractions with an indexed key take the greatest amount of time. The difference is substantial, with key times of 20 minutes to extract 1,000,000 records, versus one or two minutes for the other techniques. The SQL extraction is the fastest technique, in part because it avoids sorting or indexing overhead.

Figure 3 shows these studies repeated on the mainframe and using up to 10,000,000 records. The results are very similar to the PC, with extractions taking about an order of magnitude less time than on the PC.

DISCUSSION

It is necessary to consider both overhead and actual extraction times when comparing table lookup methods. Figure 1 shows that the merge procedure operating on sorted data sets has the fastest extraction time. But when the sorting times are added, the best overall time belongs to SQL operating on unsorted data sets. In this situation, merge would be selected only if the data had to be sorted for some other reason.

Although SQL does well in these comparisons, it should be noted that the SQL code is only being asked to do lookups against a single unique variable in the lookup table. Experience has shown that SQL performance can deteriorate substantially when confronting more complex problems that require development of the full Cartesian product or that do not allow activation of internal SQL code optimizers. For these reasons, SQL is probably the technique of choice only when the problem is a simple table lookup such as used here.

Previous studies (Bachler, 1996) have noticed the adverse effect of indexing on both SQL and merge extractions. The present study confirms that the time for the extraction component for either merge or SQL increases when indexing is used. If the data set has to be indexed, then it might be preferable to index it as a separate step after major extractions are completed.

Only the key and format techniques can be applied to both data sets and flat files. Since key takes an order of magnitude longer than the other techniques, it does not appear to be a method of choice for data extraction.

Format extraction has times comparable to merge and SQL, and can operate on both data sets and flat files. Format does not require the source data set to be sorted or indexed, but it is necessary to have the format reside in memory. This will impose an upper limit on the lookup table size for any computer, although a table of 10,000 records was used with a format technique on the PC with no difficulty, and format with tables of over 100,000 records is used routinely on the mainframe. More sophisticated format techniques (Levine, 1996; Zdeb, 1996) can duplicate the results of complex merges or SQL fuzzy joins and should always be considered as alternatives to merge and SQL.

Indexing datasets and using a key technique does not appear to be a good choice for any data extraction. For medium sized data sources, merge, SQL, and formats all appear to be acceptable. For simple extractions from data sets, SQL may often be the first choice. If the extraction becomes complex, then merge or format may be a better choice than SQL.

For large data sets, merge may not be an option because of the impracticality of sorting the source data set. In this situation, the choice will be between SQL and format. If the source is a large

flat file, as is often the case with warehouse feeds provided by a corporate IT service, then format may be the only practical choice. If format cannot be used because of a large lookup table, then compromises may be required such as reading the flat file into a data set followed by SQL extraction, or building several smaller formats and doing successive runs over the source data.

REFERENCES

Bachler,R., (1996), "PROC SQL as an Alternative to Multiple SORT/MERGE Statements," *Proceedings of the Fourth Annual Western Users of SAS Software Regional Users Group Conference*, San Francisco, California.

Haase, M., (1996), "Measuring Performance Between PROC SQL and the MERGE Approach," *Proceedings of the Fourth Annual Western Users of SAS Software Regional Users Group Conference*, San Francisco, California.

Levine, H., (1996), "Advanced Uses of SAS Formats", *Proceedings of the Twenty-First Annual SAS Users Group International Conference*, Chicago, Illinois.

Zdeb, M.,. (1996), "Rapid Record Matching via FORMATS and CNTLIN Datasets," ", *Proceedings of the Twenty-First Annual SAS Users Group International Conference*, Chicago, Illinois.

Foley, M.J., (1997) "Advanced MATCH-MERGING: Techniques, Tricks, and Traps," *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*, San Diego, California.

Rafiee, D., (1997) "No More MERGE -- Alternative Table Lookup Techniques," *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*, San Diego, California.

ACKNOWLEDGMENTS

CONTACT INFORMATION

Anthony M. Dymond, Ph.D.
Dymond and Associates, LLC
4417 Catalpa Ct.
Concord, CA 94521

(510) 798-0129
(510) 680-1312 (FAX)
dymond@ccnet.com

A SAS Institute Inc. Quality Partner

FIGURE 1

Time Components of Various Extraction Methods (PC/Windows 95)

(Extracting $1x10^4$ Records from a $1x10^5$ Record Source)

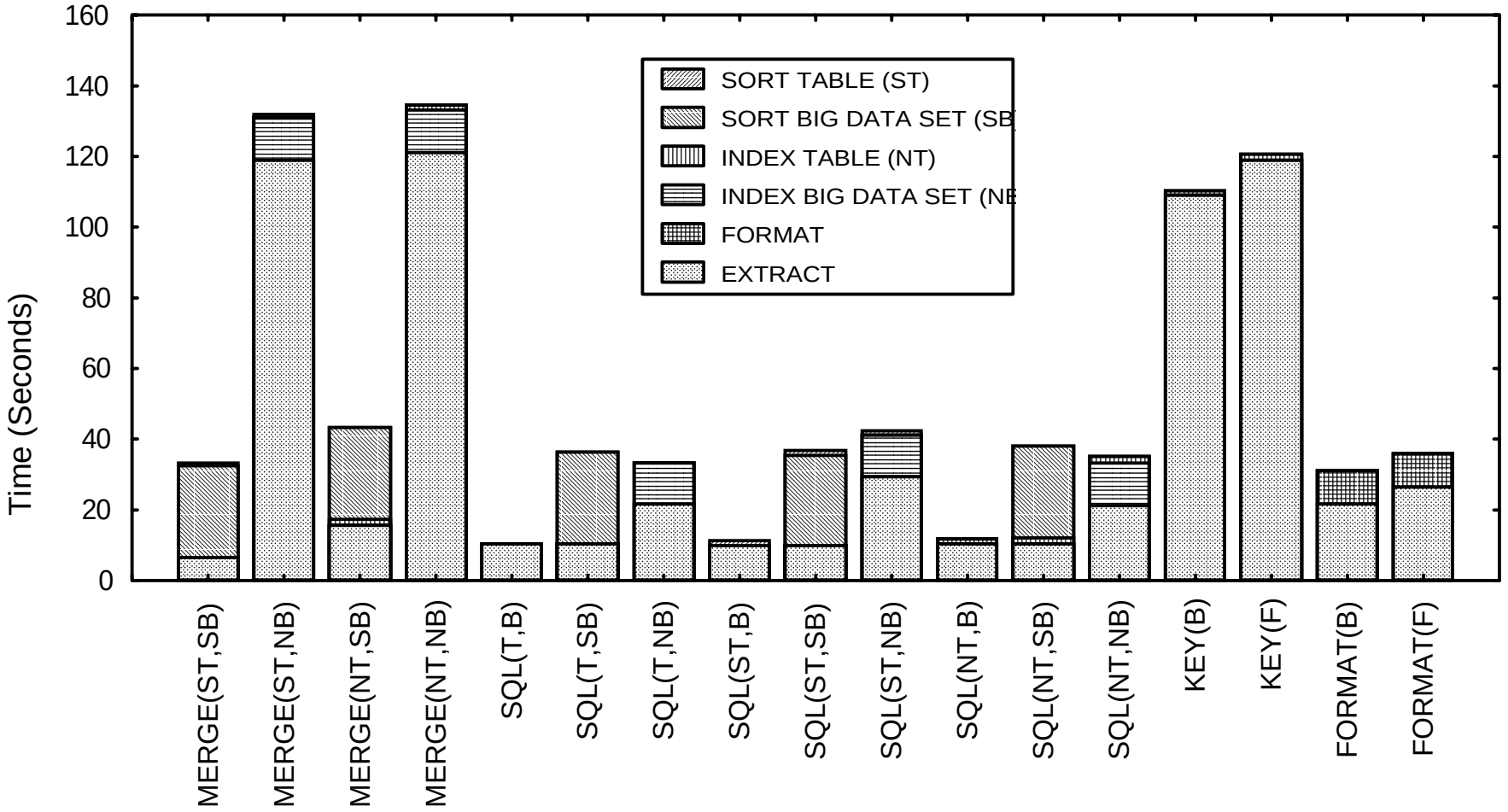

FIGURE 2

Time to Extract 1x10 ^{4} Records (PC/Windows 95)

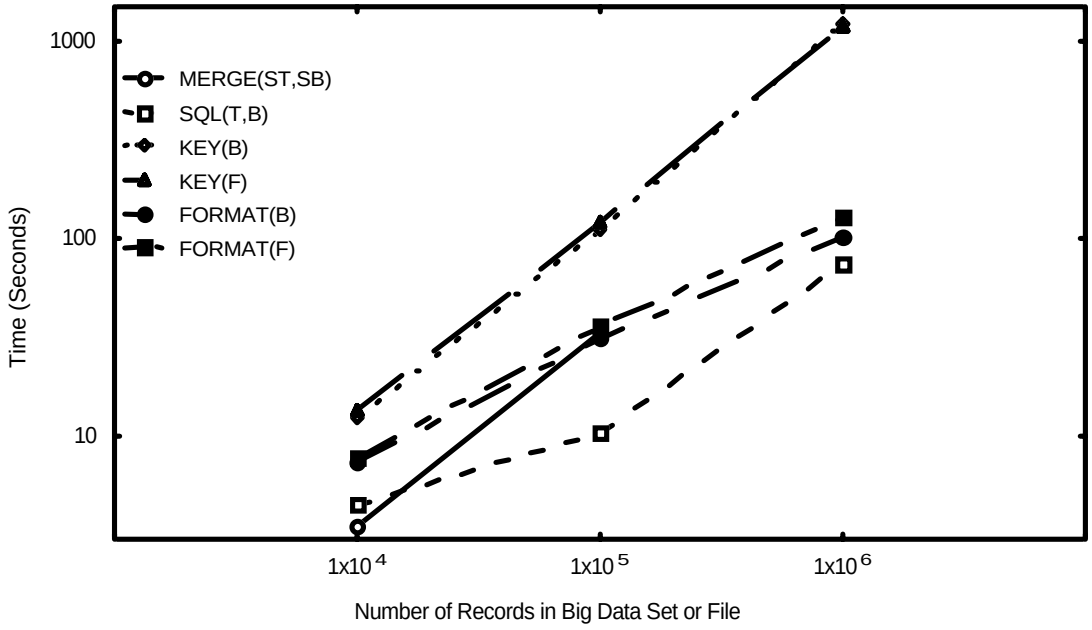

FIGURE 3

Time to Extract 1x10 ^{4} Records (mainframe/MVS)