

Relational Processing of Tape Databases

Howard Levine, DynaMark - A Fair Isaac Company

Outline

This paper covers the following topics:

Explanation of Relational Processing

Simple Relational Processing

Why Use Tapes?

Setting Up the Files

Parallel Processing

General Joins with More than 2 Files

Limitations of Tapes

Conclusion

Explanation of Relational Processing

The essence of relational processing is to use more than one file to store your information in an efficient, easily maintained way. Figure 1 shows how a name file and Zip Code file can be related to show which city each person lives in. The name of the city is not on the file with the person's name. Instead, Zip Code is used to associate a name with a city. There are two advantages to this method: (1) the data can be stored in fewer bytes in most cases and (2) the files are easier to maintain. If the name of a city associated with a Zip Code changes, then only the entry on the Zip Code file will have to be changed. It will not be necessary to change a city field on every individual's record.

Desirable Features in RDBs

Normalized Files

Redundant data should be eliminated to the maximum extent possible consistent with processing efficiency. This reduced overall storage requirements and makes databases easier to maintain.

Keyed or Indexed Access to Files

In order to find records quickly and avoid unnecessary processing, files should be indexed or keyed. With tapes, there can be only one key. If possible, it should be a sensible field or fields that will provide a useful way of separating items in the file into groups. The files in the database will have to be sorted by the key field(s).

Referential Integrity

This is a set of rules that forces records to exist in one file if one or more records with the same key. For example, in a human resources data base, you may not want to allow any performance review records to exist unless there is an employee record that they can match to. Of course, it might still be possible to have an employee record with no performance records.

Types of Relationships

There are different kinds of relationships that have varying levels of complexity.

One to One

Files are split for convenience or because of Null Relationships. An example would be a file with many variables that are not often used. It would be reasonable to separate the file into two files: (1) frequently used variables and (2) infrequently used variables. This would reduce processing in most cases and still allow access to all variables.

Another example is when a certain group of variables have null (or missing) values for a significant portion of the records. Since it is not even necessary to store the null values, separating those variables into a separate file can reduce overall storage needs and processing time. The non-existence of a record will indicate that certain variables are null (missing) without wasting storage space.

One to Many

One record in a file can match to several in another file. An example would be one family record matching to several individual records and each

individual record matching to only one family record. This would show a nuclear family relationship.

This is typically a Hierarchical Relationship or a Look-up Table.

Many to Many

A record on one file matches to many records on the second file. A record that is matched on the second file may also match to other records on the first file.

Example: Using family and individual records as with the one-to-many relationship except that a person is allowed to belong to more than one family. This would represent an extended family relationship. For example, a person may share one family record with a spouse and children and a different family record with siblings and parents.

These relationships can sometimes be more easily expressed as multiple one-to-many relationships.

Null Relations

A record does not match to a record in another file. An example would be a family record with no matching individual records or an individual record with no matching family record. Sometimes, null relationships indicate a legitimate lack of data. In other cases, they indicate referential integrity problems.

Null relationships can make accessing more than two files at a time fairly tricky under some circumstances. This is particularly true when using SQL joins.

Simple Relational Processing

SAS has a number of nice tools for relational processing. They each accomplish their objectives in slightly different ways.

Merge Statement in Data Step

When accompanied by a BY statement, this is a powerful, yet simple, technique for relating files. It handles one-to-one relationships very well and can accommodate one one-to-many relationship. Many-to-many relationships are not handled well with this method. Null relationships are handled very easily.

SQL Joins

This technique is well suited to handling many-to-many relationships. Unfortunately, it is not well suited to handling null relationships as easily as the MERGE statement when more than two files are involved.

Set with Key= Option

This is a way of doing table look-ups. Table look-ups are one-to-many relationships. It allows data steps to conveniently handle more than one one-to-many relationship. The look-up table is a SAS data set with keyed access based on the value of a variable.

VSAM Files

This is another way of doing table look-ups. The look-up table is a VSAM file with keyed access based on the value of a variable.

SAS Formats

This is yet another way of doing table look-ups. The look-up table is a SAS format accessed with the PUT or INPUT functions. A characteristic of this technique is that the entire look-up table is stored in memory when a Data or Proc step is using it.

Why use Tapes?

Massive Amounts of Data

Huge volumes of data, such as the entire United States census, might not fit onto disk packs at many computer centers.

Large Amounts of Data Accessed Infrequently

Large files that could be stored on disk might not be accessed frequently enough to justify storage on disk. Although automatic restore capabilities are available, it may be more cost effective to process large files directly from Tape.

Data from Outside Sources on Frequent Basis

If you are getting data from outside sources and sending data outside your data center, then using tapes might be more convenient than disk.

Processing is Sequential rather than Direct Access

If all processing can be handled sequentially, it is more efficient than direct access. Data can be read much more efficiently.

Relational Processing Within BY Group

If all relationships are within a by-group, it is possible to have full relational processing in an efficient manner with tape data sets.

Assumptions About Data

Large Files Must be Sorted by a Common Key

A Typical Key is Region and Customer Number or Account Number

Typically, the most effective key for tape data sets is a variable that will group a large number of records together. Variables such as Region or State serve that purpose. That variable is combined with a variable such as customer number or account number that specifies a smaller group in order to form the complete database key.

Activity by One Customer does not Relate to Another

If this is not true, then direct access is required.

Comparison to Means or other Statistics is NOT possible (in one pass)

Since we cannot look at interactions between customers (or families or whatever), it is impossible to compare a record's values to any value based on a statistic based on other records. It is possible to calculate the mean and do a second pass. That is what disk based systems do anyway, but since there are no tapes to rewind, the complexity of doing that is hidden.

Setting Up the Files

Sort Files by Common Key

All of the files (except for small look-up tables) must be sorted by the same database key. This will allow matching within BY groups.

Store Files as SAS Data Sets

This allows SAS to perform BY group processing and eliminates the need to convert data into a SAS data set every time they are processed.

Consider Segmenting the Files based on the Key

This allows more direct access (as distinct from "direct access") to your tape data. If your data is segmented by state, you can access only the records for the state(s) needed. It is not necessary to waste processing time reading records that will not be used.

Index File on Disk if Data is Segmented

For segmented files, keep an index file on disk that shows which tape files have which records on them. For example, states 1,2 and 3 might be on tape 1. Tapes 2 and 3 might contain data for state 4. The directory would contain all of this information so your programs would know which tapes to read.

Look-up Files Should be on Disk

Any file used for table look-up s must be on a direct access device.

File Segmentation Techniques

Individually Segment every file of the database

This allows different files to remain physically separated. See Figure 2.

Segment Entire Database

This allows little mini-databases to be places on tape. See figure 3.

Look-up Files are not Segmented

These files will normally be on disk and will not normally be segmented.

Individually Segmented Files

Advantages

Allows only necessary records to be accessed

Enables faster processing since only needed records are accessed.

Disadvantages

File Maintenance is more difficult. The files must be segmented.

More Tape Drives might be needed. With several transaction file segments per customer file segment, the number of tape drives could increase because SAS must open all data sets at once.

Segmented Database

Advantages

Allows only necessary records to be accessed.

Enables faster processing because only necessary records are processed.

Allows for "true" direct access (Optical Drives). With DASD, each segment is truly a mini-database.

Fewer Tape Drives Necessary. Only one drive is needed. All data is copied from the tape to DASD for processing.

Disadvantages

File Maintenance is MUCH more difficult. Segmenting the files and updating SAS libraries on tape can be very difficult and incur substantial overhead.

Entire Volume MUST be copied to DASD for processing.

Parallel Processing

This technique allows a large database to be processed more quickly by having each of its segments processed simultaneously. As long as BY groups process independently, there is not problem with parallel processing.

Records or BY Groups processed Independently

Requires Segmenting Files

Each separate segment will be processed independently.

Requires Processing to Combine Results

Results from processing each segment must usually be combined to get a final result such as a SUM or COUNT.

Quicker Response

Since all segments can be run simultaneously (operating system willing), response time can be roughly the time to process one segment plus the time needed to combine the results.

Best with Multiple CPUs

If all parallel processes are run on the same CPU, then the full benefits of parallel processing will not be realized. If each segment must share its segment with another CPU, then it will not run as quickly as if it had its own CPU.

Lower Throughput

Because of extra overhead, throughput might go down.

Controlling Parallel Processing

Final Step Must Run After ALL Parallel Processes

Control Table

Process #	Done ?
1	Y
2	N
3	Y

When all processes are done, final step will begin.

Final Step Combines Results

Combine Summary Information

Combine Output Files

Produce Desired Reports

General Joins with More than 2 Files

This is a new, proprietary relational database accessing technique. It has advantages over the SQL2 standard for the following reasons:

Make Outer Joins as Easy as Inner Joins

SQL2 Supports Outer Joins Between Exactly 2 Tables

Some Databases do NOT have Referential Integrity

NULL Relationships Often Occur

Match Information "Best" Way Possible

The N Table join supports flexible outer joins involving more than two files. In situations with incomplete matches, it does the best job it can to match records. This is especially useful for marketing databases and other databases that might have poor data integrity.

Example

Combine Account, Promotion, and Order Data for a Customer

See figure 4 for a diagram of a sample database. This shows records for one customer. In this database, all records are related within a customer only.

N Table Joining Options

Here is a proposed syntax for dealing with outer joins as simply as SQL deals with inner joins. A working prototype of this joining technique has already been developed.

Proposed Syntax

Options set for each Input Table

Set to Y for Yes or N for No

MUSTJOIN

This Input Table **MUST** be part of **EVERY** inner join when **MUSTJOIN=Y**. The joining process is a series of inner joins between all possible table combinations until all rows in all tables are used in at least one join. This is an oversimplification, but it conveys the general idea.

MUSTUSE

Every Row of this Table **MUST** be in at least one row of the Output Table when **MUSTUSE=Y**

Controls Outer Joining

Similar to **INNER**, **LEFT**, **RIGHT**, and **FULL** joins, but for N Tables instead of two.

Compare to SQL2 Outer Join

See Figure 5.

Notice that the **MUSTUSE** values are used to control whether the join is an **INNER**, **LEFT**, **RIGHT**, or **FULL** join. The **MUSTJOIN** values have no effect on a two table join. **MUSTJOIN** has meaning only when at least three tables are being joined.

Example with 3 Files

Figure 6 shows the results of doing the "fullest" join possible on the data depicted in Figure 4. The code for producing this is shown below.

Select *

From Account

(**MUSTJOIN=N,MUSTUSE=Y**) as A,

Promotion

(**MUSTJOIN=N,MUSTUSE=Y**) as P,

Order (**MUSTJOIN=N,MUSTUSE=Y**)

as O

where

(Account.Customer=Promotion.Customer) and

(Account.Customer=Order.Customer)

and

(Promotion.Customer=Order.Customer)

) and

(Account.Account=Promotion.Account)

and

(Account.Account=Order.Account) and

(Promotion.Promotion=Order.Promotion);

Example with 3 Files

Order Oriented View of Data

Get Orders and information
applying to them

Figure 7 shows a different view of the data than Figure 6. Notice that different items were joined based only on changing the **MUSTJOIN** and **MUSTUSE** values.

Select *

From Account (**MUSTJOIN=N,MUSTUSE=N**) as A,

Promotion (**MUSTJOIN=N,MUSTUSE=N**) as P,

Order (**MUSTJOIN=Y,MUSTUSE=Y**) as O

where

(Account.Customer=Promotion.Customer) and

(Account.Customer=Order.Customer) and

(Promotion.Customer=Order.Customer) and

(Account.Account=Promotion.Account) and

(Account.Account=Order.Account) and

(Promotion.Promotion=Order.Promotion);

Limitations of Tapes

Direct Access not allowed

SAS Libraries not as Flexible as on Disk

Reading and writing SAS Libraries on tape is more awkward and error prone than the same operations on disk.

Only One User can Access Data Simultaneously

It is possible for only one job to physically access the same tape. Segmented files can help to alleviate this problem.

Operator Intervention Required

Tape mounts must be performed unless automated equipment such as silo is used.

Relational Processing MUST be BY Group oriented

Because tape processing is sequential, all relational processing must occur within the BY group.

Summary

Explanation of Relational Processing

Simple Relational Processing

Why Use Tapes?

Setting Up the Files

Parallel Processing

General Joins with More than 2 Files

Limitations of Tapes

Conclusion

Relational Processing of Tapes is Possible

Relational processing and tapes are often thought to be mutually exclusive, but this is not true in many situations commonly encountered in data processing.

Non-Tape DASD Look-up Tables are Helpful

Disk look-up tables can help normalize a tape database and make file maintenance easier.

Much Relational Processing is BY Group Oriented

This is often true for disk based processing too. Often, little is lost by using tapes instead of disk.

Sequential Processing Simulating Relational Processing can be more Efficient for Large Files

Reading files more efficiently can be critical with very large files.

Relational Processing within BY Groups is the only way to Feasibly Process Large Files

Even with disk databases, relational processing outside of a BY group is likely to be very inefficient. This means that tape databases are often a good option.

For more information, feel free to contact the author

Howard Levine

DynaMark

4290 Fernwood Street

St. Paul, MN 55112-5730

612-486-1793

fax 612-481-8077

The author wishes to acknowledge the valuable assistance of David Sommer of Optimal Systems Inc. with clarifying the concepts of the N table join.

SAS, SAS/AF, SAS/FSP, and SAS/STAT are registered trademarks of SAS Institute, Cary, NC

Figure 1

Name File

Name	ZipCode
Bill	01249
Glenn	19395
Harriet	39499
Harv	39282
Jane	01837
Mary	39499
Melissa	19395
Mike	39204
Steve	39204

ZipCode File

ZipCode	City	State
01249	New Hope	NH
01837	Little Hope	MA
19395	Friendly	PA
39204	Showme	MO
39282	Blue Grass	KY
39499	Coal Dust	WV
42822	Motown	MI

- Zip Code relates a name to a City and State

Figure 2

Customer File

File Name	State	Start Customer Number	End Customer Number
customer.grp001	MI	1	1000
customer.grp002	MN	1001	3000
customer.grp003	MN	3001	4500
customer.grp004	SD	4501	7000

Transaction File

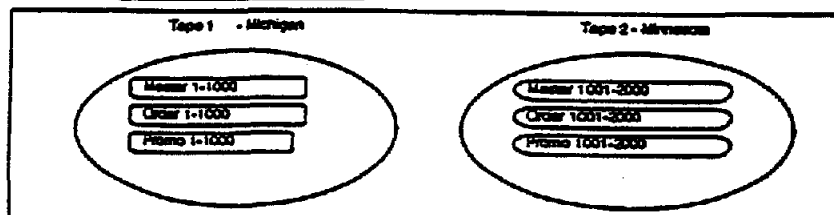
File Name	State	Start Customer Number	End Customer Number
trans.grp001	MI	1	500
trans.grp002	MI	501	750
trans.grp003	MI	751	1000
trans.grp004	MN	1001	2000
trans.grp005	MN	2001	3000
trans.grp006	MN	3001	4500
trans.grp007	SD	4501	5000
trans.grp008	SD	5001	7000

Look-up File
Keyed by ?

Figure 3

- Put segments from all files in EVERY volume

Keyed by Customer Number



Look-up Table Keyed by ?

Look-up Table

Figure 4

- Combine Account, Promotion, and Order Data for a Customer

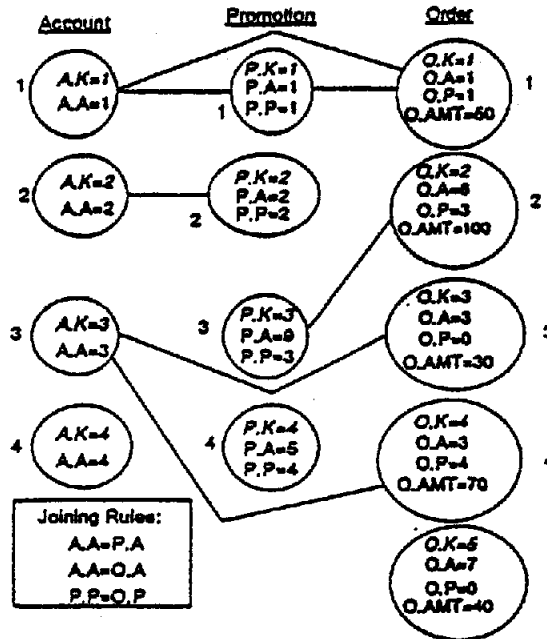


Figure 5-Compare to SQL2 Outer Join

- Simple Example

Names

Name	EmpNum
Bill	1
Bob	2
Babette	3

proc sql;

select *

from Names full join JobTitle

on Names.EmpNum =
JobTitle.EmpNum;

JobTitle

EmpNum	JobTitle
1	Manager
3	Applications Prog.
4	Systems Prog.

Select *

from

Names (MUSTJOIN=Y,MUSTUSE=Y),
JobTitle (MUSTJOIN=Y,MUSTUSE=Y)

where Names.EmpNum =
JobTitle.EmpNum;

Figure 6 - Result

Joining Step	Files	A.K Account Key	P.K Promotion Key	O.K Order Key
1	A, P, O	1	1	1
2	A, P	2	2	-
	A, O	3	-	3
		3	-	4
	P, O	-	3	2
3	A	4	-	-
	P	-	4	-
	O	-	-	5

**Figure 7
Result - Order
Oriented View of
Data**

Joining Step	Files	A.K Account Key	P.K Promotion Key	O.K Order Key
1	A, P, O	1	1	1
2	A, O	3	-	3
		3		4
	P, O	-	3	2
3	O	-	-	5