Paper 101-30

The SQL Optimizer Project: _Method and _Tree in SAS®9.1

Russ Lavery
(Thanks to Paul Sherman)

ABSTRACT

This paper discusses two little known options of SAS® Proc SQL: _method and _tree. The main body of information, and the major opportunity to learn about the topics, exists in the heavily annotated appendix. Efficient use of this material might involve reading, and printing, this paper then working through the appendix. This paper will also attempt to describe some of the logic that the Optimizer employs.

Proc SQL has a powerful subroutine, the SQL Optimizer, that examines submitted SQL code and the state of the system (file size, index presence, buffersize, sort order etc). The SQL Optimizer creates a "run plan" for optimally running the query. Run plans describe executable programs that the Optimizer will create to produce the desired output. These executable programs can be quite complicated and often involve the creating, sorting and merging of many temporary files. Consistent with the Optimizer's goal of minimizing run times, the executable programs will trim variables and observations from the input file(s)/working file(s) as soon as they can be removed.

Many details of the run plan can be determined by using two Proc SQL options (_Method and _Tree) and this paper explains output from these two options. _Method and _Tree produce different output that present different aspects of the run plan. Learning to interpret _Method and _Tree can help programmers explain why small variations in code, or system conditions, can cause substantial variations in run time.

INTRODUCTION

This paper introduces two little known options of SAS Proc SQL: _Method and _Tree. The majority of the information, and the opportunity to learn about the topics, exists in the heavily annotated appendix. Reading the appendix is strongly recommended.

In SQL, you tell SQL what you want for results, not how you want the results produced. SAS SQL has a powerful subroutine, the SQL Optimizer, that decides how the SQL query should be executed in order to minimize run time. The Optimizer examines submitted SQL code and characteristics of the SAS system and then creates efficient executable statements for the submitted query. The created code can be quite complicated and often involves the creating, sorting and merging of many temporary files as well as the trimming of variables and observation at times that will minimize run time.

All versions of SQL have Optimizers and, while they perform very well, performance can sometimes be improved by manual intervention (AKA different coding of the query). Tuning SQL, or coding for optimum SQL efficiency, requires that the programmer understanding the logic of the Optimizer's choices and to change her/his submitted code in a way that allows the Optimizer to make better choices. Basic to the tuning process is the understanding of what the Optimizer did when it ran a "too slow" SQL Query. _Method and _Tree show much of this information.

Unfortunately _Method and _Tree options produce large amounts of output. This paper will present an overview of the subject and the reader is encouraged to work thorough the annotated logs in the appendix of this paper (included on the on the CD). The annotated log files, in the appendix, is one of the larger, more detailed, collections of annotated _Method and _Tree output.

This is part of a planned series of papers on the SAS SQL Optimizer. The papers will build on each other and, hopefully, create a coherent body of knowledge on the Optimizer.

THE LIST OF KNOWN MESSAGES AND BASIC SQL PROCESSES

_Method and _Tree will show much about how a query executed, but use many abbreviations to indicate SAS SQL processes. In order to understand the cost penalties/implications of a particular execution plan, we must understand the abbreviations and some of the details of the processes used by SAS SQL.

A comment on, and apology for, the incomplete nature of this paper is in order. It is only too likely that the Optimizer currently has more subtlety than has been uncovered by the author and presented here. Additionally, SAS Inc. is constantly improving its products. As time goes by, the Optimizer will only become more powerful and more subtle and this paper will only become more incomplete.

TOP-TO-BOTTOM READ:

In certain situations SQL will perform a top to bottom read of the data. It will often do so in a query that does not have a where clause, or has a where clause without a usable index. The query might not have a usable index because 1) there is no index on the variable in the where clause, or 2) the code/syntax in the where clause might have prevented the Optimizer from using the index. SAS has put lots of time into making a top-to-bottom read fast and has been successful. Each read of a single observation is fast, but when millions of observations must be read, the total time (time=Number of Obs. * seconds per observation read) can still be unacceptably long.

Here are two examples of queries that will be executed in a top to bottom read.

```
Proc sql;
Select * from dsn;
```

```
Proc sql;/*no index on sub*/
Select * from dsn
WHERE SUB=”001”;
```

EQUIJOIN

An equijoin is the name for a join that has an equality in the where clause. SAS has not implemented the code that an academic might consider a “true equijoin” however it has some fast techniques that the Optimizer can use on equality relationships. As a result, the SQL Optimizer is often able to process equijoins quickly. Some SUGI/NESUG articles have shown techniques to speed up queries by converting them to equijoins.

```
Proc sql;/*Equijoin*/
Select * from dsn
WHERE SUB=”001”;
```

```
Proc sql;/*not an Equijoin*/
Select * from dsn
WHERE SUB LE ”001”;
```

```
Proc SQL; /*Equijioin*/
SELECT L.SUBJID, L.NAME, R.AGE
FROM LeftT as L, Right as R
Where L.subjid=r.subjid;
```

```
Proc SQL; /*not Equijoin*/
SELECT L.NAME, L.Age, R.Name, R.Age
FROM Left as L, RightT as R
Where L.Age GE R.Age;
```

SQL, whenever it thinks it can save time, pushes operations down to lower SAS processes. In simple cases of an equijoin, like where Age=5, the where clause can be pushed down to the data engine. The data engine will perform this equijoin (think of it as passing only obs where Age=5) and pass the result to SQL. If the where clause contains code like where age*12=60, the observation would be brought into SQL The multiplication and filter happen in SQL. The data engine is usually not smart enough to perform multiplication/division and similar operations.

CARTESIAN PRODUCT OR STEP-LOOP JOIN

Consider merging two tables (LeftT and RightT) in a SQL from clause, one table on the left (LeftT) of the comma and one on the right (RightT). LeftT stands for the table on the left of the comma and RightT stands for the table on the right of the comma. Cartesian products and step loops are related merging processes and the SQL Optimizer employs them as a last resort. They are slow and very much to be avoided.

For these processes, a page of data is first read from the left table and then as much data as can fit in memory is read from the right table. All merges are made (between any observation in the page from LeftT and all observations in memory that came from RightT). The results are then output. Then the observations from RightT are flushed from memory and a new read of RightT pulls in as many observations as can fit in memory. All possible matches are made (between any observation in the page from LeftT and all observations in memory that came from RightT) and results are output. This process continues until SQL has looped through all of the table RightT.

Then SQL takes a step in the left table and reads a new page of data from LeftT into memory. The process of looping through right table is repeated for the second page from the left table. Then a third page is read from the left table and the looping through the right hand table is performed again. The process continues until all the observations in the left table have been read and matched against every observation in the right hand table.

If there is a where clause in the query, it will be applied before the observations are output. SQL joins that are not based on an equality are candidates for the step loop process processes and are therefore to be avoided.

```
Proc sql;
Select * from LeftT,RightT;
```

```
Proc sql;/*no index on sub*/
Select * from LeftT, RightT
Where LeftT.sub < RightT.sub;
```

Both examples above are likely to be executed using a step-loop join (depending on system conditions and the decision of the SQL Optimizer).

INDEX JOIN

Even if an index exists on a variable in the where clause, the where clause (the code used for the query logic) can "prevent" the Optimizer from using the index. Additionally, even if the index exists and the code in the where clause does not disable the index, the Optimizer may decide not to use the index. The decision logic for the Optimizer is complex, but a firm index rule is: if index merge will return more than 15% of the indexed file to the result file, the index should not be used. In this case, there is a faster way to perform the query and the Optimzer will look for it. The Optimizer (or the data engine) can access metadata and will estimate output file sizes.

In the case of a one file select (see left box below) the Optimzer checks the percentage of observations that come from PA and their distribution in the file. If the Percentage is small, SQL reads the index file on the indexed variable "state". It locates the desired level(s) of "state" in the index and reads, from that index-observation, the hard drive page number(s) that contain observations where state="PA". In order, the page number(s) are retrieved from the index, page number(s) are passed to the disc controller and data is sent back to the CPU. As each page of observations is received by the CPU, it is parsed and observations with state="PA" are passed back to the working file for the query. SAS keeps track of the most recently read page, as a technique to minimize disk reads. If a page contains several observations that meet the where clause (e.g. state="PA"), a new page will not be read from the hard drive until all the observations in that page from PA have been sent to the query working file.

In an index merge of two files (see right box below), an index exists and the where clause code must be written in a way that allows the SQL Optimizer to use it. In the appendix, several different queries are used to test how indexes are used by the SQL Optimizer. The basic process for index join is that one file is read from top to bottom and the matching observations in are read from the other file via an index lookup.

```
Proc sql;/*index on state*/
Select * from LeftT
Where state="PA";
```

```
Proc sql;/*L_I_sub indexed in LeftT*/
Select * from LeftT  ,  RightT
Where Lft_tbl.L_I_sub = Rgt_tbl.subj
```

For more of an explanation of the process in the right box above; Assume two tables, LeftT and RightT, with an index on the variable L_I_sub in LeftT. The basic index merge process is that we are reading the right file from top to bottom and using an index to find observation(s) with matching subject Ids from left.

1) Select the next observation from RightT (at start, this is the first observation in the data set)
   -if end of file, stop
2) Pass subj from RightT to the SAS index subroutine
3) Search the index on the variable L_I_sub in the file LeftT for the value of subj
4) If found, go to disk and return the required fields for that observation from LeftT
   - output and go to 1)
5) If not found – go to 1)

If the information required by the query is in the index file itself the Optimizer will simply access the index, and not proceed to access the file associated with the index. This situation usually arises in queries when the where clause tests for existence of a match in another file (where a value in the variable subj in RightT is also in LeftT and Left_T has an index on subj). The fact that the Optimizer has automated this speed feature is just one indication of the level of detail that has gone into the designing and programming Optimizer, and of the difficulty of understanding it.

HASHING JOIN

Hashing can be a very fast technique and is automatically considered by the Optimizer. SQL hashing has been installed since V6.08 but since the Optimizer evaluates, and implements, the hashing technique without the programmer's intervention, its existence is not well known. General information on Hashing can be found in articles by Dr. Paul Dorfman in SUGI and NESUG online proceedings.

Hashing will not be considered as a join technique unless certain conditions are met. The SQL Optimizer accesses metadata on the file and "takes a good guess" at the size of the files it needs to join. After removing unneeded observations and variables, the Optimizer checks to see if 1% of the smaller of the two files being joined will fit into one memory buffer. If the smaller file appears to fit, the Optimizer will attempt a hash join. If the smaller file is too large, in relation to the buffer size, hashing will not be selected. A programmer can influence the Optimizer's choice of hashing as a merge technique by manually changing the buffer size with a SAS option.

SQL performs a hash join in the following way. The SQL Optimizer determines which of the two tables is smaller (after keeping only the appropriate variables from the select statement) and checks that smaller table size against the buffer. If the table meets the size criteria, it is loaded into a tree-like structure (a hash table) in memory. The structure of the hash object and the fact that is memory resident, allows for very fast searching. Then SQL processes the large file, from top to bottom, and for every observation that satisfies the where clause, it performs a HASH table lookup for the observation. Details of "data step" hashing are given in an article titled "An Annotated Guide: Resource use of common SAS Procedures" in the NESUG 2004 proceedings.

HASH & INDEX & WHERE USE

The Optimizer will dynamically adjust to new information. In certain situations it will switch from one join method to another-in the middle of execution - and create a hybrid join method. One such example is in the appendix (see examples 9A to 9D) and indicates that SQL simultaneously used a hash join and an index to produce the results of the query.

What happened is that, after tentatively trimming rows and columns from both files, the Optimizer estimated that 1%, of the smaller of the files being joined, would fit in a buffer. This is a strong hint/instruction for the Optimizer to use a hash join and so SQL loaded the smaller table into a hash table.

The Optimizer, as the hash table was being created, counted the number of unique key-variable values being loaded into the hash table. The number of unique values loaded into the hash table was found to be a small number (maybe below 1024) and the Optimizer dynamically changed the plan to take account of this information. In general, if there are fairly few unique values key in the hash table, the Optimizer will take the values from the hash table and use them to build an "in" phrase for a where clause (e.g. where state in("PA", "TX")).

SQL will then use the where clause to select observations but the Optimizer will again re-evaluate it's options in light of currently known information. The method selected for the join can be a top to bottom read or an indexed lookup. This adjustment of code to the details of a particular query is complex, dynamic and automatic. It can be seen in examples 9A to 9D.

SORT MERGE JOIN

This is similar, but not identical to, the data step merge. Under certain situations, the Optimizer determines that the fastest way to execute the query is to sort the tables and process both tables from top to bottom, using a merge that is similar (only similar) to that used by the DATA Step. This SQL merge will produce a Cartesian product, unlike the data step merge, and it does this by looping within the BY-group variables. SQL processes a page of data from the left table and loops through the appropriate by group right table.

GROUPING

Grouping, or aggregating observations, is a multi-step process and can take some time.

SELECT

Select statements specifies variables in the final data set. The optimizer, as part of it's run plan, creates "temporary working files" that are called result sets. Select logic is executed dynamically and as early as possible, keeping results sets small.

In the code below, only the variables name, age and sex are all brought into the original SQL query space (AKA result set). This initial removal of variables (height, weight) is handled by the data step engine and functions much like a keep option on a data set. Height and weight never become part of a SQL result set.

Observations with sex NE "M" are filtered out during the initial read of the data and that variable (sex) is eliminated from the query space, by the Optimizer, after completion of the read. After the completion of the first read, the result set contains name and age. The result set is then sorted (by calling Proc Sort) by age. After the data is sorted age is not required and the Optimizer eliminates that variable from the result set.

```
proc sql _method _tree;
create table lookat as select name from sashelp.class
where sex="M"   order by age;
```

As the above explanation details, the Optimizer has automated "Good Programming Practices" and eliminates both variables, and observations, as soon as it can.

HAVING

The having statement is very useful to SQL programmers but requires that SQL perform several steps. Please examine the code below.

```
proc sql _method _tree;
   title "this illustrates a having clause";
       select name, sex, age
           from sashelp.class
       group by sex
       having age=max(age);
   quit;
```

In the code above, SQL must process the entire table sashelp.class, reading in the only the three variables in the select clause. SQL stores the observations in a result set (a temporary table that the SQL Optimizer directs be created). Then SQL makes a pass through the temporary table to find the max age, within each group, and tries to store that information in a "pipe line". As a speed/storage technique, the Optimizer tries to avoid the creation of temporary files.

Sometimes applying a "having" requires additional passes through the working data set (AKA the result set) to check the having criteria against each observation. If there is no Note in the log mentioning re-merging, the Optimizer was able to produce the desired result in one pass using "pipe lines" to apply the having criteria. The query above produces the log below, where the word remerging indicates an additional pass was required to find the observations with the maximum age for each gender.

```
NOTE: The query requires remerging summary statistics back with the original data.
NOTE: SQL execution methods chosen are:
sqxslct
          sqxsumg
              sqxsort
                  sqxsrc( SASHELP.CLASS )
```

DISTINCT

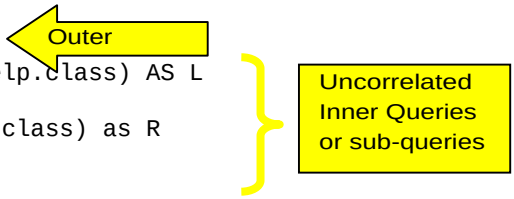
SQL usually implements distinct-ing by passing the result set to a Proc Sort with a nodup/nodupkey option. Distinct-ing is usually performed late in the query process as an additional pass through the data. This is a sorting and sorts are to be avoided because they take both time and space.

Under certain conditions (see examples 6A - 6D), the Optimizer can eliminate this last pass through if the query is distinct-ing a variable that has a unique index. The elimination of the distinct-ing saves time. There is an example of this, in compare-and-contrast format, in the appendix. It would be appropriate to use this as an example of how much effort has been put into making the Optimizer produce fast code. This situation does not happen often but the Optimizer has logic to help the programmer when it occurs.

UNCORRELATED SUB-QUERY

The code in an uncorrelated sub-query is processed just once by the SQL Optimzer. The results of this first evaluation are held in a result set until they are needed by the outer query. The code below is an example of an uncorrelated query. The result sets L and R are created once and accessed many times.

```
proc sql _method _tree;
 *title show inner join merge  using a comma;
 create table ex5 as
 select coalesce (l.name, r.name) as name
         FROM ( select Name, Height from sashelp.class) AS L
           ,
              ( select Name, sex from sashelp.class) as R
        where l.name  =r.name;
```



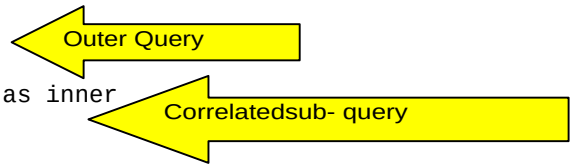
CORRELATED SUB-QUERY

A Correlated sub-query is processed differently from an un-correlated sub-query and again shows the power of the SQL Optimizer. A correlated sub-query uses information from each of the observations in the outer query to drive a "look up" process (a SQL query) against another table. In the worst case, SQL might have to execute the "look up process" for each row in the outer table. The look-up might have to be executed for every observation in the outer query but the Optimizer creates code that avoids that, whenever possible.

In the query below, as SQL processes each observation in the outer query, it seems to be passing the gender to the subquery and asking the subquery to find the maximum age for the current (in outer) value of gender. In fact, the Optimizer will process the sub-query for the first observation and store the results in a result set that is both temporary and indexed.

When additional observations from "outer" are processed the Optimizer first tries to find the needed information (in this case, has SQL calculated max age for that sex before) in the temporary, indexed result set. If it can find the required information in the temporary indexed result set, it takes information from the temporary indexed result set and does not execute the sub-query. If the information is not in the temporary indexed result set, SQL will run the sub-query, pass results to the outer query and then add the results of the query to the indexed result set. The temporary indexed result set grows in size as unique values of the equality variable are found in the outer file. When the query is done, the temporary indexed result set is deleted from the work library. The query below produces the _method output that follows it.

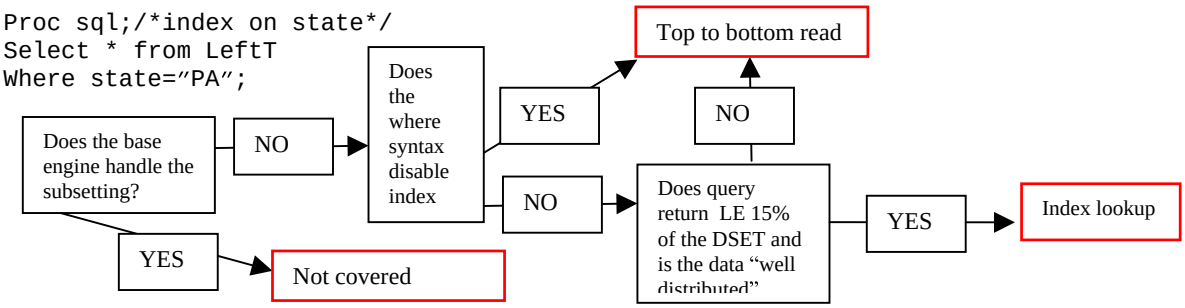
```
proc sql _METHOD _TREE;
TITLE "A SIMPLE CORRELATED QUERY";
select * from sashelp.class as Outer
 Where  Outer.AGE =
  (select Max(age) from sashelp.class as inner
        where outer.sex=inner.sex);
quit;
```




```
          NOTE: SQL execution methods chosen are:
            Sqxslct
                Sqxfil
                    sqxsrc( SASHELP.CLASS(alias = OUTER) )
          NOTE: SQL subquery execution methods chosen are:
            Sqxsubq
                Sqxsumn
                   sqxsrc( SASHELP.CLASS(alias = INNER)
```

SIMPLE SUBSETTING LOGIC

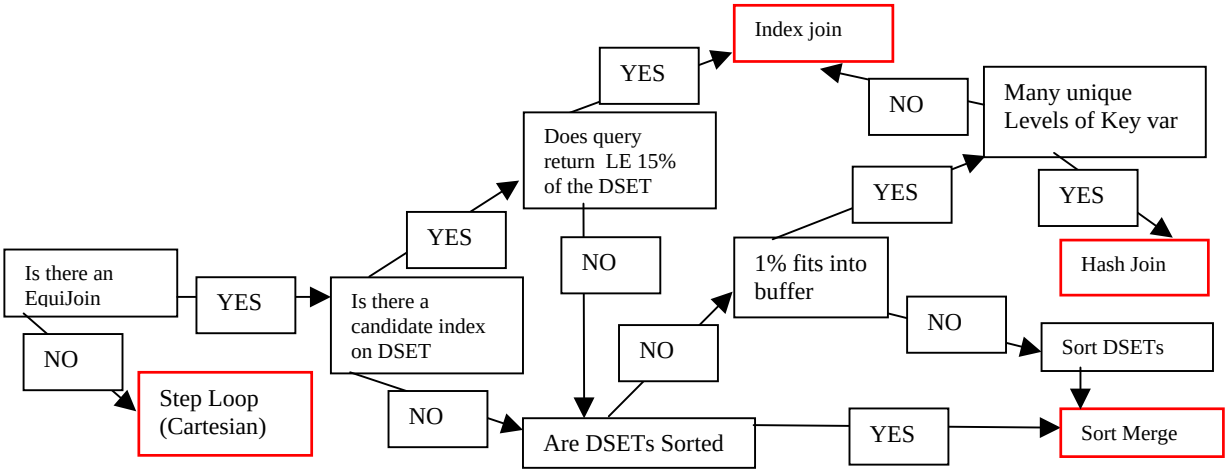
For a one file SQL query, as shown below, the Optimizer must first decide if it should push the where down to the data engine or handle it in SQL. The Optimizer's second decision is between using an index or a top-to-bottom read. The Optimizer has access to metadata on the file. The Optimizer considers both the percent of the file that will be returned and the distribution of the values in the file as it creates a plan to minimize run time.



The Optimizer has the ability to examine the metadata for the table and determine information useful for running the query. The metadata not only tells if there is an index on the variable in the where clause, but allows the Optimizser to determine both 1) what percent of the file will be returned by the where and 2) how the values are distributed through the file. This information lets the Optimizer make intelligent decisions on how to quickly access data.

MERGE LOGIC

The approximate logic for selecting a particular join is shown below (DSET means Data Set or table). Unfortunately this flowchart, like the one above, is a working model, rather than a definitive description of the Optimizer logic. The author's only consolation, is that whatever the current logic is, SAS Inc's commitment to improving it's product means that the current Optimizer will soon be replaced with one with more effective and subtle decision rules.



It is known that a where clause containing variables from two, or more, files can not be passed to the data engine.

OUTPUT FROM _METHOD

Method sends little output to the log. Below is typical output. It is important to note that an "indentation level" indicates the existence of a result set (working table, or temp file). Output in the appendix has been annotated. The query

```
proc sql _method;
  title Ex1 - show *  select;
  create table ex1 as  select *  from sashelp.class;
```

produces the following _Method output in the log.

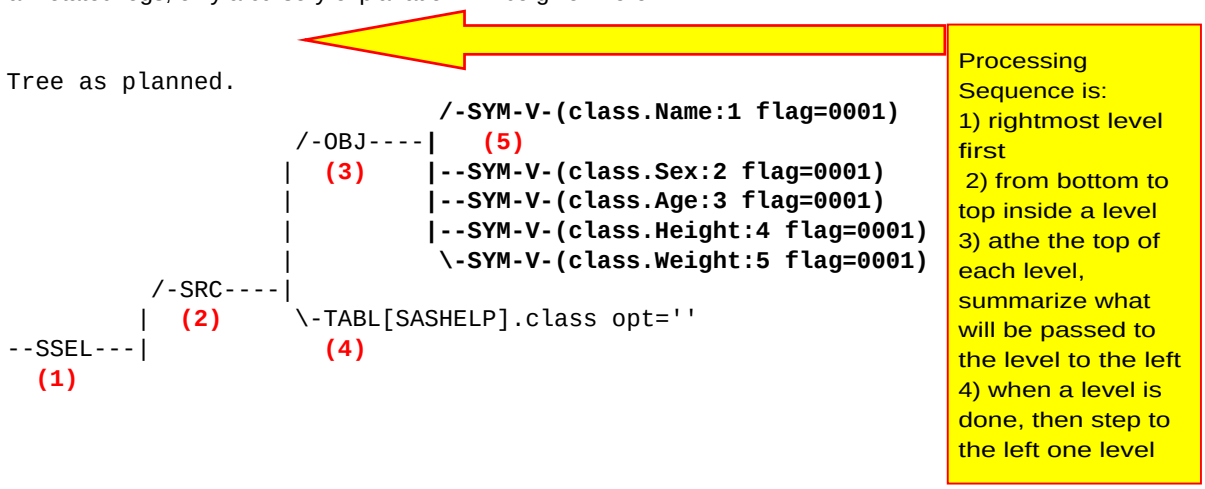
```
NOTE: SQL execution methods chosen are:
      Sqxcrta
          sqxsrc( SASHELP.CLASS)
```

A table of abbreviations is required to interpret the _Method output. Note that all abbreviations shown start with SQX. This prefix stands for "**SQ**L E**x**ecution" code. Below, please find the abbreviations I have been able to collect while investigating _Method and a short explanation of the abbreviations. It is likely that more exist.

Name code	Description
SqxCRTA	Create table as select
SqxSLCT	Select
SqxJSL	Step loop join (Cartesian)
SqxJM	Merge Join
SqxINDX	Index Join
SqxHASH	Hash Join
SqxSORT	Sort
SqxSRC	Source rows from table
SqxFIL	Filter rows
SqxSUMG	Summary stats with group by
SqxSUMM	Summary stats with NO group by

OUTPUT FROM _TREE

The Optimizer creates a program, a multi-step program, and the output from tree can go on for pages. Understanding the run plan requires information from both _Method and _Tree. Below is manually annotated (the red numbers in parenthesis) _Tree output from the query above. Since _Tree output is quite complex, and explained in the annotated logs, only a cursory explanation will be given here.



The query is processed, in levels, from right to left, and within "levels" from bottom to top. The idea of "passing data and information/instructions to the level to the left" is a useful mental concept. At the top of each level is a summary of the variables being passed to the level to the left. The result set being passed to the left is object (3) and it consists of variables (5).

In the output above,

(4) SQL reads SAShelp.Class with no options processes by SQL
(keep, drop etc are processed by the data set option processor).

(5) SQL reads variables from a SAS dataset (in sashelp) called class. The variables are:
Name: which is variable 1 in the data set class
Sex: which is variable 2 in the data set class
…..
Weight: which is variable 5 in the data set class

(3) is a summary of variables being passed to the level to the left.
(2) indicates that the branches to the right describe a data source
(1) indicates that this is a select type query (SAS developers can see other types)

Below, please find the abbreviations I have been able to collect while investigating _Tree and a short explanation of the abbreviations. Thanks to people at SAS for help with explanations.

	Abbreviation	This abbreviation can be found in Example Query Number found in the Appendix	Process
	ADIV	15	Divide
	AGGR	7, 8, 11	This indicates an aggregation, but SQL does several types of aggregation. This is associated with an aggregation operation like "select sex, sum(x) as totl group by sex", or "select name, Min(x) as smallest" Sometimes processing the AGGR requires a separate pass through the data set (look for re-merging note in the log as an indicator of a separate pass) and some times it does not.
	AMUL	14	Multiply **– A**rithmetic **Mul**tiplication
	ASC	3, 6A, 6C, 8, 13	Sort in **ASC**ending order

	ASGN	4, 5, 7	Assign. Create a new variable or **As**si**gn** a value to a new variable. If the SQL code is
			
			`"select sum(x) as totl"`
			
			X will be summed and the result **as**si**gn**ed to a variable named "totl". If the programmer names the new variable, the name will be used in output and the variable will be easy to identify/track through the output. If the programmer does not name the variable, it will be numbered and can be tracked via the number.
			
			It is suggested that created variables be named as shown below
			`Select coalesce(l.name, r.name) as Cname , r.age as Rgt_age`

	CEQ	4, 5, 8, 13	This indicates a logical instruction to be passed to the level to the left, where it is executed. CEQ means "check if these 2 leaves are equal" CEQ is the symbol for both numeric and character equality testing.
	DESC	13	The sort, on the level to the left, should be in descending order. This DESC code is "information passed to the left" on the output. The descending sort is performed (files are created and time is spent) in the sorting in the level to the left.
	Dlist		List of variables with distinct values that are participating in an aggregation/summation/grouping. See Slist and Tlist. Distincting gets rid of duplicates and Dlist reports on the distincting process. Variables on a dlist have to have duplicates removed before you can apply the aggregation.
	Empty	3, 4, 5, 6B, 6C, 7 , 8	This is a place holder in the output. The Optimizer has the capacity to do additional operations at this point-operations that were not performed.

	FCOA	4, 5,17,18,	This indicates a function, like a SAS function, of type coalesce.
	FIL	23	FILter is used to handle the situations that can not be handled by the data engine that is feeding into it. Fil indicates that SQL applied an additional predicate late in the processing. The clearest example of this is Ex 23. The where clause contains a variable that is not in the source data set (age_mo=age*12). It first must be created by SQL and then the filter predicate can be applied.
	Flags (`class.Sex:2 flag=0001`)		Flags are for developers and are also used internal to SQL. They are beyond the scope of this paper but, as one example, (001) means that the variable is used higher up in the SQL processing.
	JTAG	17, 17A, 19, 19A, 19B, 19C, 19D, 20A, 20B, 20c, 20D, 21A, 21B, 21C, 21D	This is a code that tells what type of join was applied. JDS=1 indicates a left join, JDS=2 Indicates a right join and JDS=3 indicates a full join
	FROM	4, 5	From indicates that data sources are being passed to higher (more leftward) process.
	GRP	8,	Group **–**is a multi-step process and can take some time to perform
	JOIN	4, 5	This indicates the SQL did a join, but not which type
	LAND	12	A **L**ogical **AND** should be performed. This is usually in a where or a select.
	LITC	Not shown	This indicates the use of a **Lit**eral **C**onstant (character string) as in: where state=”PA”
	LITN	13,14 ,	This indicates the use of a **Lit**eral **Number** (ie numeric constant) as in: Where age LT 12
	OBJ	1,2, 3, 4, 5, 6A, 6B, 6C, 7, 8, 13, 14,	This indicates the existence of an object, or result set. Obj indicates a description of columns in a result set that is passed leftward for more processing. Objects come from data sets or lower objects.
	OBJE	4, 5, 7,14, 15,	This indicates the existence of an evaluated object, the result of an assignment of a value to a variable. An OBJE is a variable that is typically added/merged to another object (result set) at the same level. When a programmer codes `Select sex, max(age) as Mage` The value of maximum age will be assigned to Mage and mage will be, for a brief moment before the merge into the result set for that level, an OBJE. See example 4.
	ORDR	3, 6A, 7, 8,	Order By, On the tree, contains ordering information that is passed to the sort to the left. Order is information and not an instruction. It is not, and does not create, a result set. The sort, to the left of the ORDR, creates the result set.
	OTRJ	17, 18, 19	This stands for any of the following joins: Left join, right join, full join. OTRJs are paired with JTAGs and the JTAG indicates the type of join.
	SLST	7, 8,15	Indicates a list of things that are involved in an aggregation/summarization. This is a “Not distinct” List of things that “participate” in the summarization **–** See Dlist and Tlist
	SORT	3, 7, 8, 13	Sort, at this level, in the order described to the right.

	SRC	1,2, 4, 5, 6A, 6B, , 6C,7 , 8,	Information to the right of this is a data source. Typically an object is being passed to the left.
	SSEL	1,2, 3, 4, 5, 6A, 6B, 6C,7 , 8	SSEL indicates that the query is a Select query. Not all queries are of type select. There are modify queries and drop queries and others.
	SUBP	25	This indicates an input to the subquery. It is a parameter passed to the subquery
	SYM-A	4, 5, 7, 14,	This identifies a variable as an assigned/created variable- one that did not get read from a data set. A SYM-A is the result of Assigning a value to a variable through a calculation or function as in: `Select name ,Age_yr=age*12 as YRS`

	SYM-G	7, 8	A SYM-G is the result of assigning a value to a variable through a **G**rouping calculation or function as in:
			`Select name ,sex, Max(age) as Mage`
			`Group by sex;`

	SYM-V	25	This identifies a variable as a having been read from a data set.
	Sym-v lib.name Flag=0001	1, 2, 3, 4, 5, 6A, 6B, 6C, 7, 8,14,	This shows a combination of abbreviations as they might appear in the log. A variable was read from a source table (SAS data set lib.name). See Flag above
	Table [lib].fname opt=" "	1, 2, 3,4 , 5 , 6A, 6B, 6C, 7, 8	This shows a combination of abbreviations as they might appear in the log. Tables are identified with a two part name. See opt=
	TLST	7, 8,15	Indicates a summarization. A temp List of things that "participate" in the summarization. See Dlist and Slist
	UNIQUE	6A, , 6C,	This is the message to the log when select distinct is coded. Creating unique values is usually implemented by passing the current result set to Proc Sort with a nodup/nodupkey
	opt=" "		SQL allows data set options inside the Query. An example might be `From class(keep=name age)` Some of these options are processed by Base SAS and some are processed by SQL. If an option is processed by SQL, it will show up in the opt=" " note.

CONCLUSION

The SQL Optimizer is a tremendous help to programmers, allowing them to write very efficient queries with absolutely no thought. The amount of work that the SQL Optimizer does, independently of programmer input and totally behind the scenes, is amazing.

In some cases the performance may be improved by re-coding the query and passing the Optimizer different instructions. These issues will be explored in future papers.

If SQL performance is causing problems, knowing what the Optimizer created for a plan of execution is essential if the programmer want to attempt to improve performance. _method and _tree allow the programmer to see how SQL executed the query and to see the effects of her/his programming changes.

REFERENCES

TS553–SQL Joins –the Long and the Short of it, by Paul Kent – available on the SAS web site
TS320-Inside PROC SQL's Query Optimizer, by Paul Kent – available on the SAS web site

Church (1999), Performance Enhancements to PROC SQL in Version 7 of the SAS® System
Performance Enhancements to PROC SQL in Version 7 of the SAS® System, Proceedings of the Twenty-fourth Annual SAS Users Group International Conference", 24 , paper 51

Kent, Paul (1995) "SQL Joins – The Long and The Short of IT Proceedings of the Twentieth Annual SAS Users Group
International Conference, Cary, NC: SAS Institute Inc., 1995, pp.206-215.

Kent, Paul (1996) "An SQL Tutorial – Some Random Tips",Proceedings of the Twenty-First Annual SAS Users Group International Conference pp. 237-241.

For non-SAS explanations of SQL execution, see the article by Dan Hotka at www.odtug.com

ACKNOWLEDGMENTS

The author wishes to thank the Paul Dorfman, Sigurd Hermansen, Kirk Lafler, and Paul Sherman for their contribution to the SAS community on SQL and for inspiring this paper. Thanks to Paul Sherman for his review and comments.

Thanks for the help from SAS institute, especially help from Paul Kent and Lewis Church.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Russell Lavery
9 Station Ave. Apt 1,
Ardmore, PA 19003,
610-645-0735 # 3
Email: russ.lavery@verizon.net

The Appendix Follows

The CONTENTS Procedure

Data Set Name	SASHELP.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
File Name	C:\Program Files\SAS\SAS 9.1\core\sashelp\class.sas7bdat		

A description of the data set

Variables in Creation Order

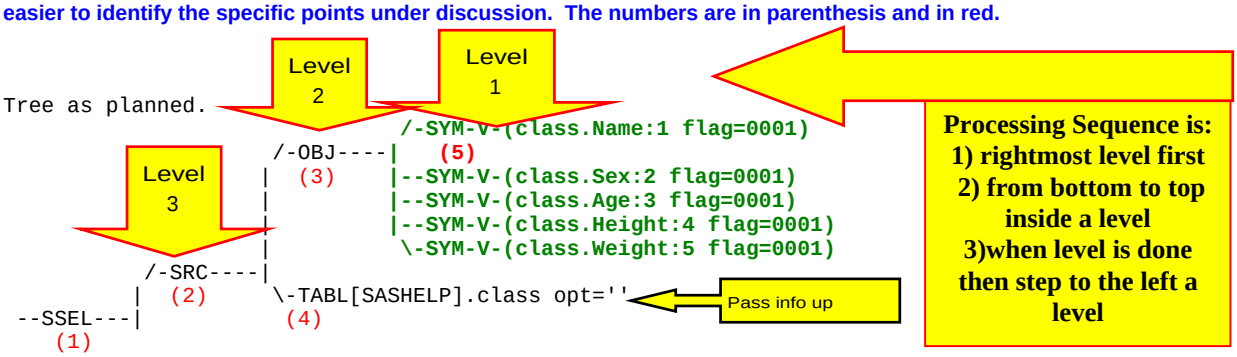
#	Variable	Type	Len
1	Name	Char	8
2	Sex	Char	1
3	Age	Num	8
4	Height	Num	8
5	Weight	Num	8

****Some Basic Terms and Background to method and tree;**
****Example 1;**

```
proc sql _method _tree;
title Ex1 - show *  select;
create table ex1 as  select *  from sashelp.class;
```

```
NOTE: SQL execution methods chosen are:
      Sqxcrta (1) apply the selection criteria
          sqxsrc( SASHELP.CLASS) (2) this indicates a source for the data
```

To make it easier to discuss details of the output, numbers in parenthesis were added manually to the log file to make it easier to identify the specific points under discussion. The numbers are in parenthesis and in red.



Red numbers have been added, manually, to make it easier to reference parts of the output.
Query processing proceeds from bottom to top inside a level and from right to left across levels. Not every level produces a temp file (result set). The first file actually produced above is the shown by the SRC (2). Items (3), (4) and (5) are information to be passed to SRC, which does the work. At the top of each level (3), the result set to be passed to the left is summarized in an obj.

Details about the above output follow:
(5) SYM-V indicates a variable from a SAS data set. Then we see the two part variable name. The :1 means that name is the first variable in the data set (see contents above). The flag is a complex notation that is used by developers and internal processing. Except to say that the 1 means that the variable is required in higher level processing, flag is beyond the scope of this paper

(4) This is an indication of the data source as SAShelp.class. You can put data set options(drop, keep, rename, etc.) in SQL. Some options are processed by base SAS and some by Proc SQL. If a dat step option were processed by Proc SQL, it would be mentioned in the opt=' '. This does not incicate a temporary table or processing.

(3)OBJ, at the top of a level, summarizes variables being passed on to higher processing.

(2) SRC indicates a source of data, a result set that is being passed on to higher processing, or display. SRC indicates a "result set".

(1) SSEL indicates that this is a select query, not that variable selection happens at this point. A drop query, or modify table query would have a different string at this point.

```
**Example 2;
proc sql _method _tree;
title Ex2 - show basic select;
create table ex2 as
        select Name, Height
        from sashelp.class;

NOTE: SQL execution methods chosen are:
      Sqxcrta (1) apply the selection criteria to observations???
          sqxsrc( SASHELP.CLASS ) (2) this indicates a source for the data

Tree as planned.
                                        /-SYM-V-(class.Name:1 flag=0001)
                        /-OBJ----|  (5)
                       |  (3)     \-SYM-V-(class.Height:4 flag=0001)
           /-SRC----|
          |  (2)     \-TABL[SASHELP].class opt=''
 --SSEL---|            (4)
    (1)
```

Instead of select *, specify the variables

The position number of the variable in the data set

Use this file (no options on the data set were processed by SQL

Reading the above output shows:
(5) SYM-V shows us selecting only two variables from a SAS data set. (Note that the Optimizer is following good programming practice and only using variables it needs.) Then we see the two part variable name. The :1 means that name is the first variable in the data set (see contents above). The flag is a complex notation that is used by developers and internal processing. Except to say that the 1 means that the variable is required in higher level processing, flag is beyond the scope of this paper.

(4) This is an indication of the data source as SAShelp.class. This shown information to be passed to the left, where work is done. SQL accepts data set options(drop, keep, rename, etc) and can show in the opt= section. Some options are processed by base SAS and some by Proc SQL. If a data step option were processed by Proc SQL, it would be mentioned in the opt=' '.

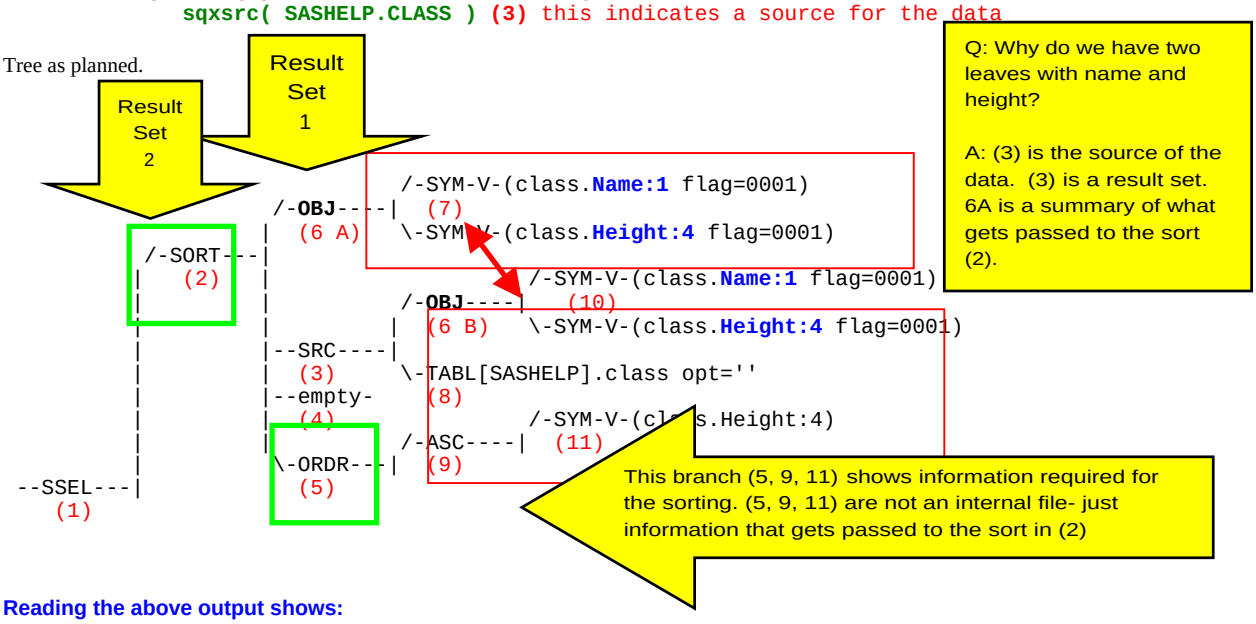
(3)OBJ, at the top of a level, summarizes variables being passed on to higher processing.

(2) SRC indicates a source of data, a result set that is being passed on to higher processing, or display. SRC indicates a "result set".

(1) SSEL indicates that this is a select query, not that variable selection happens at this point. A drop query, or modify table query would have a different string at this point.

```
**Example 3;
proc sql _method _tree;
*title Ex3 - show basic select;
create table ex3 as
        select Name, Height
        from sashelp.class
        order by height ascending;

NOTE: SQL execution methods chosen are:
      Sqxcrta (1) this indicates a selection of observations
          sqxsort (2) this indicates a sorting of the data
              sqxsrc( SASHELP.CLASS ) (3) this indicates a source for the data
```

Tree as planned.



Reading the above output shows:

Note that ORDER is not a command to execute a sort. It is information about how the sort should be processed and is passed to the sort (2).

Order (11, 9, 5) is not an instruction to put observations in order. Order is information to be passed up to higher SQL processing. No processing happens in (11), (9) or (5). This section is simply telling SQL that the data will be sorted by ascending height- at a later time. The sort happens at (2).

(4) is a place holder. The optimizer could do amazing things here, but has not been requested to do so. :-)

(3) is a source of data. What is in that source is specified to the right.

(6A) is a summarization of the data being passed to the next higher level of processing. At this level the information about sort order (5) becomes important.

(2) sorting is done by the SAS Proc Sort. The order information is passed to SAS sort from (5).

(1) SSEL indicates that this is a select query, not that variable selection happens at this point. A drop query, or modify table query would have a different string at this point.

```
**Example 4;
proc sql _method _tree;
*title Ex4 - show inner join merge;
create table ex4 as
select coalesce (l.name, r.name) as name
        FROM ( select Name, Height   from sashelp.class) AS L
                INNER JOIN
             (select Name, sex     from sashelp.class) as R
              on l.name  =r.name;
```

Note: the Optimizer knows it does not need Height or Sex to produce the requested results. It will not bring them into the "QUERY SPACE". See (12) and (13).

We specify Inner Join not using comma. For illustration, extra variables are included.

```
NOTE: SQL execution methods chosen are:
      Sqxcrta (1) this indicates a selection of observations
          Sqxjhsh (2) this indicates a hash join
              sqxsrc( SASHELP.CLASS ) (8) this indicates a source for the data
              sqxsrc( SASHELP.CLASS ) (9) this indicates a source for the data

Tree as planned.
                                    /-SYM-A-(name:1 flag=0031)
                        /-OBJ----|   (7)
              /-JOIN---|   (3)
             |   (2)    |                              /-SYM-V-(class.Name:1 flag=0001)
             |          |                 /-OBJ----|           (16)
             |          |          /-SRC----  (12)
             |          |         |  (8)    \-TABL[SASHELP].class opt=''
             |          |--FROM---|
             |          |  (4)    |                    /-SYM-V-(class.Name:1 flag=0001)
             |          |         |           /-OBJ----|           (17)
             |          |          \-SRC----|  (13)
             |          |            (9)    \-TABL[SASHELP].class opt=''
             |          |              y-
             |          |          /-SYM-V-(class.Name:1)
             |          |--EQ----|  (10)
             |          |  (5)    \-SYM-V-(class.Name:1)
             |          |--empty-
             |          |--empty-
             |          |                    /-SYM-A-(name:1 flag=0031)
             |          |          /-ASGN---|  (14)
             |          |         |  (11)   |          /-SYM-V-(class.Name:1)
             |          |         |          \-FCOA---|           (18)
             |          |         |            (15)   \-SYM-V-(class.Name:1)
             |           \-OBJE---|
 --SSEL---|              (6)
   (1)
```

Rule: Bottom SRC is the one that was put in the in Hash table

Do a hash join where names are equal.

Sym-A (7) indicates a created variable. It is the result of Coalescing the variable name.

No Height or sex

No Height or sex

The equality test is not "done" here. (10 or 5). This info is passed "up" to the join (2)

Sym-A (14) indictes a created variable. It is the result of Coalescing the variable name as the come in from two sources. AssiGN the result to a new variable (of type SYM-A) called name.

(6) and to the right describe the coalescing process. FCOA (15) stands for Function. COAlesce

(6) OBJE "documents" OBJect Evaluation logic.
It tells SQL that how to calculate/evaluate the variable.
(11) Coalesce two variables, put the result in an Assigned variable (type=SYM-A).

(10 , 5) this is not an process, like getting data (9). It is used to pass data to the SQL. When the files are merged, the values of names from the data sets (variables of type SYM-V) must be equal.
(9, 8, 4) take data from the sources to the right.
(3) This summarizes the data to be passed to the right.

What type of join?
(2) says that a join takes place. (5) says it is an equality join on name. Sqxjhsh says that the join is a hash join.

Rule: The bottom file is the file that was loaded into the hash table.
NOTE that and OBJE, as well as an OBJ, can be passed up (7) started as (14). (14) was named, "name" (yeah, not too creative).

```
**Example 5;
 proc sql _method _tree;
 *title Ex5 - show inner join merge  using a comma;
 create table ex5 as
 select coalesce (l.name, r.name) as name
         FROM ( select Name, Height from sashelp.class) AS L
          ,
              ( select Name, sex from sashelp.class) as R
        where l.name  =r.name;

NOTE: SQL execution methods chosen are:
      Sqxcrta   (1) this indicates a selection of observations
          Sqxjhsh   (2) this indicates a hash join
              sqxsrc( SASHELP.CLASS )  (9) this indicates a source for the data
              sqxsrc( SASHELP.CLASS )  (10) this indicates a source for the data

Tree as planned.
                                  /-SYM-A-(name:1 flag=0031)
                        /-OBJ---|    (8)
           /-JOIN---|    (3)
          |    (2)  |                          /-SYM-V-(class.Name:1 flag=0001)
          |         |                 /-OBJ----|
          |         |          /-SRC---|    (13)
          |         |          |  (9)     \-TABL[SASHELP].class opt=''
          |         |          |
          |         |--FROM--  |
          |         |    (4)   |                          /-SYM-V-(class.Name:1 flag=0001)
          |         |          |                 /-OBJ----|
          |         |          \-SRC----|    (14)
          |         |            (10)    \-TABL[SASHELP].class opt=''
          |         |--empty-
          |         |    (5)   /-SYM-V-(class.Name:1)
          |         |--CEQ----|  (11)
          |         |    (6)   \-SYM-V-(class.Name:1)
          |         |--empty-
          |         |--empty-
          |         |                          /-SYM-A-(name:1 flag=0031)
          |         |           /-ASGN---|    (15)
          |         |          |  (12)   |          /-SYM-V-(class.Name:1)
          |         |          |         \-FCOA---|    (17)
          |         |          |                  (16)   \-SYM-V-(class.Name:1)
          |          \-OBJE---|
 --SSEL---|            (7)
   (1)
NOTE: Table WORK.EX5 created, with 19 rows and 1 columns.
```

Note: the Optimizer knows it does not need Height or Sex to process the query. See (13) and (14)

Specify Inner Join using comma. For illustration, specify unneeded variables

This output tells us the Optimizer used a hash join

Sym-A (3) indicates an Assigned/created variable. It is the result of Coalescing the variable name.

No Height or sex vars

No info here about type of join.

Bottom file (10) goes in Hash table

The equality test is not "done" here. (10). This passes information "up".

(7) and to the right describe the coalescing process. FCOA (16) stands for Function. COAlesce. Coalesce name from two files. Assign the result to a variable called name.

(7, 12, 15, 16, 17) OBJE "documents" object Evaluation logic. It tells SQL how to calculate/evaluate the variable.

(16, 15) Coalesce two variables (both called name), put result in an Assigned variable (type=SYM-A)called name.

(11 , 6) this is not an process, like getting data (9). It is used to pass data to SQL. When the files are merged in (2), the values of names from the data sets (variables of type SYM-V) must be equal.
(10, 9, 5) take data from the sources to the right.

(3) This summarizes the data to be passed to the right.

What type of join?
(2) says that a join takes place. (6) says it is an equality join on name. Sqxjhsh says that the join is a hash join. The bottom file is loaded into the hash table.

NOTE that and OBJE, as well as an OBJ, can be passed up (8) started as (15). (15) was named, "name" (yeah, not too creative).

```
**EXAMPLE SIX SERIES: COMPARE/CONTRAST AMONG EXAMPLES*;
** THIS IS AN EXAMPLE OF HOW MUCH THE OPTIMIZER DOES FOR US
Ask for distinct on an un-indexed variable, it sorts and selects in a final step -> 6A
If we have a unique index on that variable that can satisfy the select, it eliminates the sort -> 6B
**Example 6A ********************************;
 proc sql _method _tree;
 *title Ex6A- show distinct on unindexed variable;
  create table ex6A as  select distinct name, sex , age  FROM sashelp.class;

NOTE: SQL execution methods chosen are:
   Sqxcrta (1) this indicates a selection of observations
         Sqxuniq  (2)this indicates pass through the data, like a sort with nodup option
             sqxsrc( SASHELP.CLASS ) (4) this indicates a source for the data

Tree as planned.
                                  /-SYM-V-(class.Name:1 flag=0001)
                     /-OBJ----|   (7)
                    |  (3)    |--SYM-V-(class.Sex:2 flag=0001)
                    |          \-SYM-V-(class.Age:3 flag=0001)
          /-UNIQ---|
         |  (2)    |                    /-SYM-V-(class.Name:1 flag=0001)
         |         |          /-OBJ----|  (12)
         |         |         |   (8)   |--SYM-V-(class.Sex:2 flag=0001)
         |         |         |          \-SYM-V-(class.Age:3 flag=0001)
         |         |--SRC----|
         |         |  (4)     \-TABL[SASHELP].class opt=''
         |         |--empty-
         |         |  (5)               /-SYM-V-(class.Name:1 flag=0001)
         |         |          /-ASC----|  (13)
         |          \-ORDR---|   (9)
         |            (6)    |          /-SYM-V-(class.Sex:2 flag=0001)
         |                   |--ASC----|  (14)
         |                   |   (10)  /-SYM-V-(class.Age:3 flag=0001)
         |                    \-ASC----|  (15)
 --SSEL---|                      (11)
    (1)

*Example 6B*The Optimzer does not do the distinct-ing*;

 data class_W_indx(index=(name/ unique));
 set sashelp.class;  run;
 proc sql _method _tree;
     *title Ex6B - show distinct on unique indexed variable;
    create table ex6B as  select distinct name  FROM class_W_indx;
       quit;
NOTE: SQL execution methods chosen are:
      Sqxcrta  (1) this indicates a selection of observations
          sqxsrc( WORK.CLASS_W_INDX ) (2) this indicates a source for the data
Tree as planned.
                              /-SYM-V-(class_W_indx.Name:1 flag=0001)
                     /-OBJ----|   (5)
          /-SRC----|    (3)
         |   (2)    \-TABL[WORK].class_W_indx opt=''
 --SSEL---|              (4)
    (1)
```

Summary of what goes to higher level.

Uniq is done by proc sort with a nodup/ nodupkey option!

Select variables

Info passed up to higher levels – to the Proc Sort that does the "UNIQ-ing".

Create a unique index on Name. SQL knows the metadata on files.

SQL keeps track of different types of metadata

This Query did not even access the raw data. It was able to get all the information it needed from the index.

SQL reads the header information in the file and realizes, from the index information, that these obs. are all unique. It just prints the observations using the index as the data source. There is no mention of an index use in the log, in _Method or in _Tree, but the unique index was sensed by the optimizer and used to eliminate the "unique-ing process" in 6A. Note that there is no need for sorting/ordering in the query.

**Example 6C *1 variable index, distinct on 2 vars. **;

```
proc sql _method _tree;
*title Ex6C -distinct on unique indexed variable when selecting multiple variables;
create table ex6C as
             select distinct name ,sex
             FROM class_W_indx; quit;

NOTE: SQL execution methods chosen are:
      Sqxcrta (1) this indicates a selection of observations
          Sqxuniq (2) this indicates a source for the data
              sqxsrc( WORK.CLASS_W_INDX ) (4) this indicates a source for the data
Tree as planned.
                                    /-SYM-V-(class_W_indx.Name:1 flag=0001)
                     /-OBJ----|   (7)
                    |  (3)      \-SYM-V-(class_W_indx.Sex:2 flag=0001)
          /-UNIQ---|
         |   (2)   |                      /-SYM-V-(class_W_indx.Name:1 flag=0001)
         |         |           /-OBJ----|    (12)
         |         |          |  (8)     \-SYM-V-(class_W_indx.Sex:2 flag=0001)
         |         |--SRC----|
         |         |  (4)     \-TABL[WORK].class_W_indx opt=''
         |         |--empty-    (9)
         |         |  (5)                 /-SYM-V-(class_W_indx.Name:1 flag=0001)
         |         |           /-ASC----|    (13)
         |          \-ORDR---|  (10)
         |            (6)    |          /-SYM-V-(class_W_indx.Sex:2 flag=0001)
         |                   |          \-ASC----|    (14)
 --SSEL---|                    (11)
   (1)
```

We have a distinct index on ONE of the two variables in the select. The metadata can not be used to help us.

Summary of what goes to higher level.

Uniq is done by proc sort with a nodup/nodupkey option!

Info passed up to higher levels – to the Proc Sort that "does" the UNIQ.

The metadata information (index information) does not contain enough information to allow the Optimizer to help. Create a new dataset with a new index that is "on" both the variables and then use it in Proc SQL.

Example 6D ***Compound index and distinct ********;

```
data class_W_Cmpindx(index=(nm_sex=(name sex )/ unique));
set sashelp.class;
run;

proc sql _method _tree;
              *title Ex6D - show distinct on two vars with a unique compound index;
   create table ex6d as   select distinct name, sex    FROM class_W_Cmpindx;

NOTE: SQL execution methods chosen are:
                      Sqxcrta  (1) this indicates a selection of observations
          sqxsrc( WORK.CLASS_W_CMPINDX ) (2) this indicates a source for the data
                 Result
                 Set 1
Tree as planned.
                                    /-SYM-V-(class_W_Cmpindx.Name:1 flag=0001)
                     /-OBJ----|    (5)
                    |    (3)     \-SYM-V-(class_W_Cmpindx.Sex:2 flag=0001)
          /-SRC----|
          |  (2)    \-TABL[WORK].class_W_Cmpindx opt=''
 --SSEL---|            (4)
   (1)
```

Create a unique compound index on Name and sex – In real life we'd worry about names like Pat that can be male or female (Patrick and Patricia), but not in this small data set.

SQL reads the header information in the file and realizes that these obs. are all unique. It just prints the observations using the index itself as the source of the data. No mention of index use in the log, in _Method or in _Tree, but the unique index was sensed by the optimizer and used as the data source.

```
**Example 7 Grouping and calculations*********;
proc sql _method _tree;  *title Ex7 - Grouping and calculations;
  create table ex7 as   select sex , max(age) AS maxage, AVG(HEIGHT) AS AVG_HT
        FROM class_W_indx  GROUP BY SEX ;

NOTE: SQL execution methods chosen are:
      Sqxcrta  (1) this indicates a selection of observations
          Sqxsumg  (3) this indicates an aggregation – maps to AGGR in tree
              Sqxsort  (5) this indicates a sorting of observations
                  sqxsrc( WORK.CLASS_W_INDX ) (4) this indicates a source for the data

Tree as planned.
                                    /-SYM-V-(class_W_indx.Sex:2 flag=0001)
                          /-OBJ-----|  (13)
                         |  (4)     |-SYM-A-(maxage:1 flag=0039)
                         |           \-SYM-A-(AVG_HT:2 flag=0039)
                  /-AGGR-|
                 |  (3)  |                    /-SYM-V-(class_W_indx.Age:3 flag=0001)
                 |       |           /-OBJ----|    (21)
                 |       |          |  (14)   |--SYM-V-(class_W_indx.Height:4 flag=0001)
                 |       |          |         \-SYM-V-(class_W_indx.Sex:2 flag=0001)
                 |       |--SORT----|
                 |       |   (5)    |                    /-SYM-V-(class_W_indx.Age:3 flag=0001)
                 |       |          |           /-OBJ----|    (29)
                 |       |          |          |  (22)   |--SYM-V-(class_W_indx.Height:4 flag=0001)
                 |       |          |          |         |
                 |       |          |          |         \-SYM-V-(class_W_indx.Sex:2 flag=0001)
                 |       |          |--SRC-----|           (31)
                 |       |          |  (15)    \-TABL[WORK].class_W_indx opt=''
                 |       |          |--empty-     (23)
                 |       |          |                    /-SYM-V-(class_W_indx.Sex:2)
                 |       |          |           /-ASC----|
                 |       |           \-ORDR---|    (24)
                 |       |--empty-     (16)
                 |       |  (6)     /-SYM-V-(class_W_indx.Sex:2)
                 |       |--GRP-----|
                 |       |  (7)     |
                 |       |--empty-  |
                 |       |--empty-  |
                 |       |  (8)     |                 /-SYM-A-(maxage:1 flag=0039)
                 |       |          |          /-ASGN-|    (25)
                 |       |          |  (17)    |      \-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
                 |       |--OBJE----|          |           (26)
                 |       |   (9)    |          |      /-SYM-A-(AVG_HT:2 flag=0039)
                 |       |          |           \-ASGN-|    (27)
                 |       |          |            (18)  \-SYM-G-(#TEMG002:2 stat=3,0 from Height(0,0))
                 |       |          |--empty-             (28)
                 |       |           (10)   /-SYM-G-(#TEMG001:1 stat=5,0
                 |       |          LST---|  (19)
                 |       |          (11)    \-SYM-G-(#TEMG002:2 stat=3,0 from Height(1,0))
                 |       |                  /-SYM-S-(Age:3 ss=0008x)
                 |       |          SLST---|  (20)
                 |       |          (12)    \-SYM-S-(Height:4 ss=00E0x)
                 |       |--SSEL----|
                  (1)
```

Sex is just a Variable from a data set. We have Assigned values to Maxage and Avg_ht. through grouping.

Line Wrapping

(30)
(2)

AGGR = Aggregate

If there are no having clauses, and no re-merge, SQL can do this in one pass with a pipeline. Tlist and Slist contain info to be used by the AGGR

GRP (7) is Not a process! Passing info to higher level.

(0,0) The left 0 indicates position on on Slist the right 0 indicates non-unique.

stat function: Stat=5 -> max

Stat=3 -> mean function

Should be (1,0)

TList=TempLiST

Slist=Non-unique list

Slist Numbering starts at 0 height is position 1

Can also have Dlst= unique list

The Max(age) and Avg(height) values are not stored in temp files. They are stored in “Pipelines”. (25, 26, 27, 28)

In (26) SymG indicates that a grouping is involved. Stat=5 is a code indicating that an average is calculated for the groups. The left 0 in the (0,0) after age indicates the source of the data (position 0 in the slist). The right 0 indicates non-unique. There are three result sets here. (15) (5) and (3) are physical files.

```
*****Example 8 Having **************;
 proc sql _method _tree;
 *title Ex8 - Grouping and calculations;   /*create table ex7 as*/
       select NAME, sex , height, age  FROM class_W_indx
         GROUP BY SEX      HAVING AGE=MAX(AGE);
NOTE: The query requires remerging summary statistics back with the original data.

NOTE: SQL execution methods chosen are:
      Sqxslct  (1) this indicates a selection of observations
          Sqxsumg  (4) Summary Statistics With Grouping
              Sqxsort  (6) this indicates a sorting of observations
                  sqxsrc( WORK.CLASS_W_INDX ) (20)indicates a selection of observations

 Tree as planned                 /-SYM-V-(class_W_indx.Name:1 flag=0001)
                      /-OBJ----|
                     |  (5)    |--SYM-V-(class_W_indx.Sex:2 flag=0001)
                     |         |--SYM-V-(class_W_indx.Height:4 flag=0001)
                     |         \-SYM-V-(class_W_indx.Age:3 flag=0001)
              /-AGGR---|            (12)
             |   (4)   |                  /-SYM-V-(class_W_indx.Age:3 flag=0001)
             |         |          /-OBJ----|   (19)
             |         |         |   (13)  |--SYM-V-(class_W_indx.Sex:2 flag=0001)
             |         |         |         |--SYM-V-(class_W_indx.Name:1 flag=0001)
             |         |         |          \-SYM-V-(class_W_indx.Height:4 flag=0001)
             |         |--SORT---|
             |         |   (6)   |                   /-SYM-V-(class_W_indx.Age:3 flag=0001)
             |         |         |          /-OBJ----|
             |         |         |         |  (20)   |--SYM-V-(class_W_indx.Sex:2 flag=0001)
             |         |         |         |         |--SYM-V-(class_W_indx.Name:1 flag=0001)
   (3)       |         |         |         |          \-SYM-V-(class_W_indx.Height:4 flag=0001)
             |         |         |         |             (23)
   (2)       |         |         |--SRC----|
             |         |         |  (14)    \-TABL[WORK].class_W_indx opt=''
             |         |         |--empty-     (21)
             |         |         |                   /-SYM-V-(class_W_indx.Sex:2)
             |         |         |          /-ASC----|
             |         |          \-ORDR---|   (22)
             |         |            (15)
             |         |          /-SYM-V-(class_W_indx.Age:3)
             |         |--CEQ----|  (16)
             |         |  (7)     \-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
             |         |
             |         |          /-SYM-V-(class_W_indx.Sex:2)
             |         |--GRP----|
             |         |  (8)
             |         |--empty-
             |         |--empty-
             |         |--empty-
             |         |--empty-
             |         |  (9)     /-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
             |         |--TLST---|   (17)
             |         |  (10)    /-SYM-S-(Age:2 ss=0008x)
             |          \-SLST---|   (18)
 --SSEL---|              (11)
    (1)
```

Remerging is associated with the AGGR and takes another pass

This takes a pass through the data to apply the criteria

Line Wrapping

Passing info. to (6)

Stat=5 -> max

(0,0) The left 0 indicates position on on Slist the right 0 indicates non-unique.

Stat=5 -> max

Slist Numbering starts at 0 Age is position 0

We start with table (21) and create a result set (RS) (14). The contents of (14) are sorted (6) and then summarized in (20). The max age is stored in a pipeline named #TEMG001 (17). The CEQ (7) operator, and information to the right on its branch, is information passed to the higher level. Grp (8) is information passed to a higher level.

****Example 9 exploring index use in a join *********;**
Create data for join. Note that in the data set Left, we have an indexed variable called LindxName. Note that in the data set Right, we have an indexed variable called RindxName.

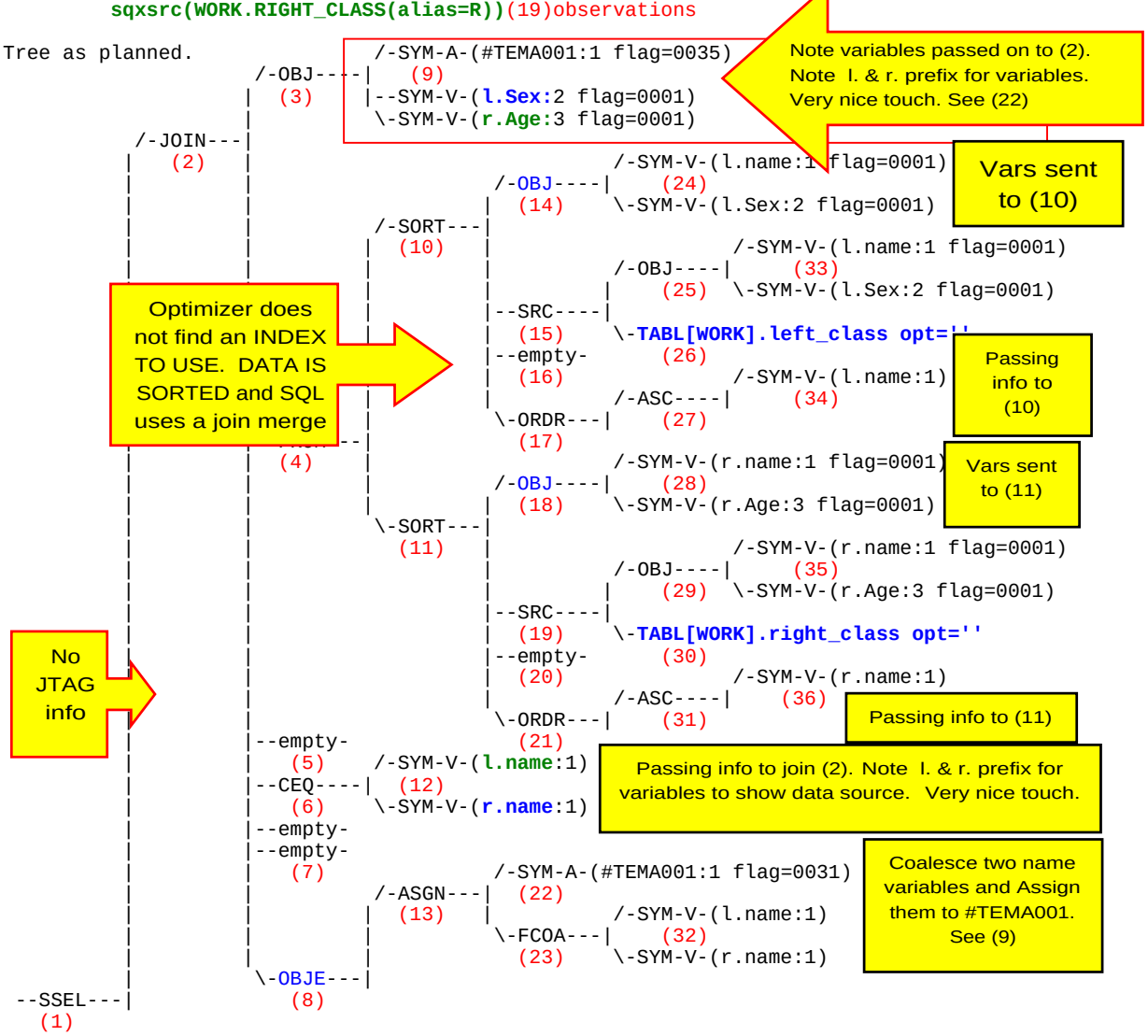
When we merge the two datasets, left will always be to the left of the join indicator and right will always be to the right.

E.G. (From left as L, right as R)

```
*left and right both have an un-idexed variable called name and a variable with a two part name
"L" or "R" and the suffix "IndName";

This naming convention will make it easier to understand if a variable has an index or not.

data left_class(  drop= RIndxName index=(LIndxName))
     Right_class( drop= LIndxName index=(RIndxName));

        length name $ 13;
        set sashelp.class;

     do i=1 to 1200; /*expand file so we do not hash*/
        name=name||put(i,5.0);
        RIndxName=name;
        LIndxName=name;
        Output;
     end;

run;
```

The data step above just expands the class data set (it uses a loop to increase the file size in an effort to have files so large that 1% of the file is larger than the buffersize) to reduce the chance of using a hash in the merging. The concatenation is done so that there are unique values of name so that we can make a unique index.

```
Proc SQL _method _tree;
title "EX9A inner join without an index on the variable";
create table hope as
select coalesce(l.name, r.name), l.sex, r.age
       From left_class as l inner join right_class as r
       on l.name=r.name; /*these are not indexed*/
NOTE: SQL execution methods chosen are:
Sqxcrta  (1) this indicates a selection of observations
 Sqxjm  (2) this indicates a Merge join
     Sqxsort (10) SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L))(14)observations
     sqxsort (11) SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R))(19)observations

Tree as planned.
```



NOTE that and OBJE, as well as an OBJ, can be passed to left.

```
Proc SQL _method _tree;
title "EX9B inner join with an index on the variable from LEFT table";
create table hope as
        select coalesce(l.name, r.name), l.sex, r.age
        From left_class as l inner join right_class as r
        on l.LIndxName=r.name; /* LIndxName  IS indexed*/


NOTE: SQL execution methods chosen are:
 Sqxcrta (1) this indicates a selection of observations
     Sqxjhsh (2) this indicates a HASH join
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (9)indicates a selection of observations
         sqxsrc(WORK.RIGHT_CLASS(alias=R))(10)indicates a selection of observations

Tree as planned.
                                    /-SYM-A-(#TEMA001:1 flag=0035)
                     /-OBJ----|   (8)
                    |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                    |          \-SYM-V-(r.Age:3 flag=0001)
           /-JOIN---|
          |  (2)    |                                  /-SYM-V-(l.LIndxName:7 flag=0001)
          |         |                       /-OBJ----|
          |         |                      |  (13)   |--SYM-V-(l.name:1 flag=0001)
          |         |                      |          \-SYM-V-(l.Sex:2 flag=0001)
          |         |             /-SRC----|
          |         |            |   (9)    \-TABL[WORK].left_class opt=''
          |         |--FROM---|             (14)
          |         |  (4)    |                          /-SYM-V-(r.name:1 flag=0001)
          |         |         |           /-OBJ----|
          |         |         |          |  (15)    \-SYM-V-(r.Age:3 flag=0001)
          |         |          \-SRC----|
          |         |            (10)    \-TABL[WORK].right_class opt=''
          |         |--empty-              (16)
          |         |  (5)     /-SYM-V-(l.LIndxName:7)
          |         |--CEQ----|  (11)
          |         |  (6)     \-SYM-V-(r.name:1)
          |         |--empty-
          |         |--empty-
          |         |  (7)                /-SYM-A-(#TEMA001:1 flag=0031)
          |         |           /-ASGN---|  (17)
          |         |          |  (12)   |          /-SYM-V-(l.name:1)
          |         |          |          \-FCOA---|   (19)
          |         |          |                    (18)   \-SYM-V-(r.name:1)
          |          \-OBJE---|
 --SSEL---|            (8)
   (1)
```

HASH JOIN indicated "incorrectly"

Note variables passed on (L. & R. prefixes). Note #TEMA See (17)

HASH JOIN above and index below

Passing info to (2). Note L. & R. prefix for variables to show data source. Nice touch.

Coalesce two name variables and Assign them to #TEMA001. See the OBJE at (8)

(8) is an object created by an evaluation process

INFO: Index LIndxName selected for WHERE clause optimization.

What happened is that, after tentatively trimming rows and columns from both files, the Optimizer estimated that 1%, of the smaller of the files being joined, would fit in a buffer.

This is a strong hint/instruction for the Optimizer to use a hash join and so SQL loaded the smaller table into a hash table. The Optimizer, as the hash table was being created, counted the number of unique key-variable values being loaded into the hash table. If the number of unique values loaded into the hash table is small (maybe below 1024), the Optimizer will dynamically change the plan to take account of this information.

If there fairly few unique values key in the hash table, the Optimizer will take the values from the hash table and use them to build an "in" phrase for a where clause (e.g. where state in("PA", "TX")). The Optimizer dynamically adjusted the plan to use an index lookup to effect the merge.

```
Proc SQL _method _tree;
 title "EX9C inner join with an index on the variable from RIGHT table";
 create table hope as
 select coalesce(l.name, r.name), l.sex, r.age
        From left_class as L inner join right_class as r
        on L.name=r.RIndxName; /* RIndxName  IS indexed*/
NOTE: SQL execution methods chosen are:
 Sqxcrta (1) this indicates a selection of observations
     Sqxjm (2) this indicates a MERGE join
       Sqxsort (11) SORT
           sqxsrc(WORK.LEFT_CLASS(alias=L))))(15)indicates a selection of observations
       sqxsort (12) SORT
           sqxsrc(WORK.RIGHT_CLASS(alias=R))))(19)indicates a selection of observations
Tree as planned.                  /-SYM-A-(#TEMA001:1 flag=0035)
                      /-OBJ----|  (10)
                     |  (4)     |--SYM-V-(l.Sex:2 flag=0001)
                     |          \-SYM-V-(r.Age:3 flag=0001)
              /-JOIN---|
             |  (3)    |                          /-SYM-V-(l.name:1 flag=0001)
             |         |                /-OBJ----|  (25)
             |         |               |  (15)    \-SYM-V-(l.Sex:2 flag=0001)
             |         |      /-SORT---|
             |         |     |  (11)   |                    /-SYM-V-(l.name:1 flag=0001)
             |         |     |         |          /-OBJ----|   (34)
             |         |     |         |         |  (26)    \-SYM-V-(l.Sex:2 flag=0001)
             |         |     |         |--SRC----|
             |         |     |         |  (16)    \-TABL[WORK].left_class opt=''
             |         |     |         |--empty-    (27)
             |         |     |         |  (17)                /-SYM-V-(l.name:1)
             |         |     |         |          /-ASC----|   (35)
             |         |     |          \-ORDR---|  (28)
             |         |--FROM---|                  (18)
             |         |  (5)    |                  /-SYM-V-(r.RIndxName:7 flag=0001)
             |         |         |          /-OBJ----|  (29)
             |         |         |         |  (19)   |--SYM-V-(r.name:1 flag=0001)
             |         |         |         |          \-SYM-V-(r.Age:3 flag=0001)
             |         |         |\-SORT---|
             |         |         |  (12)   |                    /-SYM-V-(r.RIndxName:7 flag=0001)
             |         |         |                      |  (36)
             |         |         |          /-OBJ----|
             |         |         |         |  (30)   |--SYM-V-(r.name:1 flag=0001)
             |         |         |         |          \-SYM-V-(r.Age:3 flag=0001)
             |         |         |--SRC----|
             |         |         |  (20)    \-TABL[WORK].right_class opt=''
             |         |         |--empty-    (31)
             |         |         |  (21)                /-SYM-V-(r.RIndxName:7)
             |         |         |          /-ASC----|     (37)
             |         |          \-ORDR---|  (32)
             |         |--empty-             (22)
             |         |  (5)     /-SYM-V-(l.name:1)
             |         |--CEQ----|   (13)
             |         |  (7)     \-SYM-V-(r.RIndxName:7)
             |         |--empty-
             |         |--empty-
             |         |   (8)                /-SYM-A-(#TEMA001:1 flag=0031)
             |         |          /-ASGN---|  (23)
             |         |         |  (14)   |          /-SYM-V-(l.name:1)
             |         |         |          \-FCOA---|  (33)
             |         |         |                    (24)    \-SYM-V-(r.name:1)
             |         |          \-OBJE---|
             |          \-SSEL---|    (9)
              \-SSEL---|  (1)
                (1)
```

Inner Join.
Left as L right as R

Note variables passed on to (4).
Note l. & r. prefix for variables.
Very nice touch. See (22)

No RindxName in (4)

Vars sent to (11)

Even though an index existed, it was not used here. Method shows a Merge join.

(11) sorts by name

Vars sent to (12)

No RindxName in (4)

(12) sorts by RIndxName

Passing info to (3)

Names are passed up to the coalesce and dropped ASAP. We Coalesce the names and store in SYM-A-(#TEMA001

(9) is an object created by an evaluation

Please note that the optimizer gets rid of RindxName as soon as it can. At (4) there is no need for RindxName to be passed "up".

```
Proc SQL _method _tree;
title "EX9D inner join with an index on the variables from BOTH tables";
create table hope as
select coalesce(l.name, r.name), l.sex, r.age
        From left_class as l inner join right_class as r
        on l.LIndxName=r.RIndxName; /* Both variables are indexed */

NOTE: SQL execution methods chosen are:
  Sqxcrta  (1) this indicates a selection of observations
    Sqxjhsh  (2) this indicates a HASH join
        sqxsrc(WORK.RIGHT_CLASS(alias = R)) (10)indicates a selection of observations
        sqxsrc( WORK.LEFT_CLASS(alias = L) )(11)indicates a selection of observations
```

HASH JOIN indicated "incorrectly"

```
Tree as planned.                 /-SYM-A-(#TEMA001:1 flag=0035)
                      /-OBJ----|  (9)
                     |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                     |          \-SYM-V-(r.Age:3 flag=0001)
            /-JOIN---|
           |  (2)    |                              /-SYM-V-(r.RIndxName:7 flag=0001
           |         |                    /-OBJ----|   (20)
           |         |                   |  (14)   |--SYM-V-(r.name:1 flag=0001)
           |         |                   |          \-SYM-V-(r.Age:3 flag=0001)
           |         |          /-SRC----|
           |         |         |  (10)    \-TABL[WORK].right_class opt=''
           |         |--FROM---|            (15)
           |         |  (4)    |                              /-SYM-V-(l.LIndxName:7 flag=000
           |         |                    /-OBJ----|  (21)
           |         |                   |  (16)   |--SYM-V-(l.name:1 flag=0001)
           |         |                   |          \-SYM-V-(l.Sex:2 flag=0001)
           |         |          \-SRC----|
           |         |            (11)    \-TABL[WORK].left_class opt=''
           |       --empty-               (17)
           |         (5)      /-SYM-V-(r.RIndxName:7)
           |       --CEQ----|  (12)
           |         (6)      \-SYM-V-(l.LIndxName:7)
           |       --empty-
           |       --empty-
           |         (7)                  /-SYM-A-(#TEMA001:1 flag=0031)
           |                   /-ASGN---|  (18)
           |         |        |   (13)  |          /-SYM-V-(l.name:1)
           |         |        |          \-FCOA---|  (22)
           |         |        |            (19)    \-SYM-V-(r.name:1)
           |          \-OBJE---|
 --SSEL---|            (8)
    (1)
```

Name is Passed up, & dropped after coalesce

Name is Passed up, & dropped after coalesce

HASH JOIN above and index join below

The indexed variables are passed up to the CEQ and dropped ASAP

INFO: Index RIndxName selected for WHERE clause optimization.

What happened is that, after tentatively trimming rows and columns from both files, the Optimizer estimated that 1%, of the smaller of the files being joined, would fit in a buffer.

This is a strong hint/instruction for the Optimizer to use a hash join and so SQL loaded the smaller table into a hash table. The Optimizer, as the hash table was being created, counted the number of unique key-variable values being loaded into the hash table. If the number of unique values loaded into the hash table is small (maybe below 1024), the Optimizer will dynamically change the plan to take account of this information.

If there fairly few unique values key in the hash table, the Optimizer will take the values from the hash table and use them to build an "in" phrase for a where clause (e.g. where state in("PA", "TX")). The Optimizer dynamically adjusted the plan to use an index lookup to effect the merge.

```
**example 10 **when does select happen???***********;
 proc SQL _method _tree;  /*EX10A the timing of the selects: variables is early */
 select name , sex, age
        from sashelp.class
        order by sex, age;
NOTE: SQL execution methods chosen are:
  Sqxslct (1) this indicates a selection of observations
      Sqxsort  (2) this indicates a Sort
          sqxsrc( SASHELP.CLASS ) )  (3) indicates a selection of observations
Tree as planned.                  /-SYM-V-(class.Name:1 flag=0001)
                     /-OBJ----|   (7)
                    |  (3)    |--SYM-V-(class.Sex:2 flag=0001)
                    |          \-SYM-V-(class.Age:3 flag=0001)
           /-SORT---|
          |   (2)   |                  /-SYM-V-(class.Name:1 flag=0001)
          |         |          /-OBJ----|  (12)
          |         |          |  (8)    |--SYM-V-(class.Sex:2 flag=0001)
          |         |          |          \-SYM-V-(class.Age:3 flag=0001)
          |         |--SRC----|
          |         |  (4)     \-TABL[SASHELP].class opt=''
          |         |--empty-    (9)
          |         |  (5)                /-SYM-V-(class.Sex:2)
          |         |          /-ASC----|  (13)
          |         \-ORDR---|  (10)
          |           (6)    |          /-SYM-V-(class.Age:3)
          |                  \-ASC----|  (14)
 --SSEL---|                     (11)
   (1)
NOTE: PROCEDURE SQL used (Total process time):  real time  1.66 seconds    cpu time  0.45 seconds
```

The Optimizer implements good programming practices. Variables are Selected Early. Un-needed variables (height, weight) are not brought into the SQL space.

(6, 10,11,13 &14) show information that is passed onto the sort (2). Sorting is done by Proc Sort.

Note how the Optimzer only brings in variables it needs for the query.

*DYNAMICALLY Trimming Extra (not needed) variables As they become redundant ;

```
proc sql _method _tree;title "EX10B timing of selects:of observations";
select name  from sashelp.class
       where sex="M"     order by age;
NOTE: The query as specified involves ordering by an item that doesn't appear in its SELECT clause.
NOTE: SQL execution methods chosen are:
 sqxslct
     sqxsort
         sqxsrc( SASHELP.CLASS )

Tree as planned.                  /-SYM-V-(class.Name:1 flag=0001)
                     /-OBJ----|   (7)
           /-SORT---|   (3)    |
           |  (2)   |          |                  /-SYM-V-(class.Name:1 flag=0001)
           |        |          |          /-OBJ----|  (12)
           |        |          |  (6)     \-SYM-V-(class.Age:3 flag=0001)
           |        |--SRC----|
           |        |  (4)    |--TABL[SASHELP].class opt=''
           |        |         |  (9)     /--(Sex:2)
           |        |         |  \-CEQ---|  (13)
           |        |         |    (10)  \-LITC('M')
           |        |--empty-
           |        |  (5)                /-SYM-V-(class.Age:3)
           |        |          /-ASC----|  (14)
           |        \-ORDR---|  (11)
 --SSEL---|          (6)
   (1)
NOTE: PROCEDURE SQL used (Total process time): real time 1.43 sec. cpu time  0.01 sec.
```

Note is from SQL

Age Not Needed in output and is not stored here

Age Needed for ordering

OBS. & Vars. are Selected Early. Base SAS Compares the value in sex to the literal character value "M"

Age Not Needed after ordering

Note how the optimizer trims variables that it no longer needs. It needs to select for sex="M", and then drops sex. It orders by age, and then drops it as well.

```
*****example 11***The Having Clause ********************************;
 proc sql _Method  _tree;
 title "EX11 this illustrates a having clause";
     select name, sex, age  from sashelp.class
     group by sex    having age=max(age);

NOTE: The query requires remerging summary statistics back with the original data.

NOTE: SQL execution methods chosen are:
 Sqxslct  (1) this indicates a selection of observations
    Sqxsumg (2) Aggreagate is associated with a having- requires a pass through the data
        sqxsort  (4) this indicates a Sort
            sqxsrc( SASHELP.CLASS ) (12) this indicates a selection of observations

Tree as planned.                  /-SYM-V-(class.Name:1 flag=0001)
                        /-OBJ----|   (10)
                       |  (3)    |--SYM-V-(class.Sex:2 flag=0001)
                       |         \-SYM-V-(class.Age:3 flag=0001)
              /-AGGR---|
             |   (2)   |                    /-SYM-V-(class.Age:3 flag=0001)
             |         |          /-OBJ----|  (19)
             |         |         |   (11)  |--SYM-V-(class.Sex:2 flag=0001)
             |         |         |         \-SYM-V-(class.Name:1 flag=0001)
             |         |--SORT---|
             |         |   (4)   |                    /-SYM-V-(class.Age:3 flag=0001)
             |         |         |          /-OBJ----|   (23)
             |         |         |         |  (20)   |--SYM-V-(class.Sex:2 flag=0001)
             |         |         |         |         \-SYM-V-(class.Name:1 flag=0001)
             |         |         |--SRC----|
             |         |         |   (12)   \-TABL[SASHELP].class opt=''
             |         |         |--empty-     (21)
             |         |         |   (13)              /-SYM-V-(class.Sex:2)
             |         |         |          /-ASC----|
             |         |         \-ORDR---|  (22)
             |         |            (14)
             |         |          /-SYM-V-(class.Age:3)
             |         |--CEQ----|   (15)
             |         |  (5)    \-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
             |         |          /-SYM-V-(class.Sex:2)
             |         |--GRP----|   (16)
             |         |  (6)
             |         |--empty-
             |         |--empty-
             |         |--empty-
             |         |--empty-
             |         |  (7)    /-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
             |         |--TLST---|   (17)
             |         |  (8)    /-SYM-S-(Age:2 ss=0008x)
             |         \-SLST---|   (18)
 --SSEL---|            (9)
   (1)
```

Remerging is associated with the AGGR and takes another pass through the data

SYM-G-(#TEMG001:1 Max(Age) not in Obj passed to left.

Data is sorted and max ages by sex are in Pipeline. Take a pass to find the correct observations.

(14) is information for the sort (4)

The (5) equality comparison will be done between age in the table and the values of max age in the pipeline – by sex (6).

Age in to be compared (5) within Sex

SQL avoids temp files if it can.

The data was re-merged and a second pass was required to get the results.

```
*****example 12 **illustrates the and clause  ***************;
 proc sql _Method  _tree;
 title "EX12 this illustrates a AND clause";
     select name, sex, age  from sashelp.class
     group by sex
     having age=max(age) and sex="F";
NOTE: The query requires remerging summary statistics back with original data.

NOTE: SQL execution methods chosen are:
 Sqxslct (1) this indicates a selection of observations
   Sqxsumg (2) Aggreagate is associated with a having- requires a pass through the data
     Sqxsort  (4) this indicates a SORT
        sqxsrc( SASHELP.CLASS ) (12) this indicates a selection of observations

Tree as planned.                    /-SYM-V-(class.Name:1 flag=0001)
                        /-OBJ----|    (10)
                       |  (3)    |--SYM-V-(class.Sex:2 flag=0001)
                       |         \-SYM-V-(class.Age:3 flag=0001)
              /-AGGR---|
             |   (2)   |                      /-SYM-V-(class.Age:3 flag=0001)
             |         |           /-OBJ----|    (20)
             |         |          |   (11)  |--SYM-V-(class.Sex:2 flag=0001)
             |         |          |         \-SYM-V-(class.Name:1 flag=0001)
             |         |--SORT---|
             |         |  (4)    |                      /-SYM-V-(class.Age:3 flag=0001)
             |         |         |           /-OBJ----|    (26)
             |         |         |          |   (21)  |--SYM-V-(class.Sex:2 flag=0001)
             |         |         |          |         \-SYM-V-(class.Name:1 flag=0001)
             |         |         |--SRC----|
             |         |         |   (12)  \-TABL[SASHELP].class opt=''
             |         |         |--empty-     (22)
             |         |         |   (13)             /-SYM-V-(class.Sex:2)
             |         |         |           /-ASC----|   (27)
             |         |          \-ORDR---|   (23)
             |         |                    (14)  /-SYM-V-(class.Age:3)
             |         |                   /-CEQ----|   (24)
             |         |                  |   (15)  \-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
             |         |--LAND---|
             |         |   (5)   |          /-SYM-V-(class.Sex:2)
             |         |         |         \-CEQ----|   (25)
             |         |         |           (16)   \-LITC('F')
             |         |         |          /-SYM-V-(class.Sex:2)
             |         |--GRP----|  (17)
             |         |  (6)
             |         |--empty-
             |         |--empty-
             |         |--empty-
             |         |--empty-
             |         |   (7)    /-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
             |         |--TLST---|  (18)
             |         |   (8)    /-SYM-S-(Age:2 ss=0008x)
             |          \-SLST---|  (19)
 --SSEL---|                      (9)
    (1)
```

Remerging is associated with the AGGR and takes another pass through the data

Summarizing what gets passed up. No max(age)

Summarizing what gets sorted

The Data (12)

Result of query without the and sex='F' in the where clause

Name	Sex	Age
Mary	F	15
Janet	F	15
Philip	M	16

2 equalities w/ a Logical And (5)

Info to Aggr (2)-Group by sex (6)

EX12 this illustrates the AND clause

Name	Sex	Age
Mary	F	15
Janet	F	15

RESULT

Max(Age) is stored in a var called SYM-G-(#TEMG001:1

Having is applied in the AGGR (2) and requires a pass through the sorted data and **remerging**.
The key to the having is the LAND (logical And) is at (5). We do not de-dupe in this query. Everyone having an age= max(age) gets passed on.

```
**example 13 **illustrates variable=literal & sorting**;
 proc sql _Method  _tree;
 title "EX13 this illustrates a = literal and sorting";
     select name, sex, age  from sashelp.class
         where age = 12   order by  name desc ,   height asc ;
NOTE query as specified involves ordering by an item that doesn't appear in its SELECT
      clause.
NOTE: SQL execution methods chosen are:
 Sqxslct (1) this indicates a selection of observations
   Sqxsort  (2) this indicates a SORT
     sqxsrc( SASHELP.CLASS ) (4) this indicates a selection of observations

Tree as planned.                  /-SYM-V-(class.Name:1 flag=0001)
                      /-OBJ----|   (7)
                     |  (3)    |--SYM-V-(class.Sex:2 flag=0001)
                     |          \-SYM-V-(class.Age:3 flag=0001)
            /-SORT---|
           |  (2)    |                     /-SYM-V-(class.Name:1 flag=0001)
           |         |           /-OBJ----|   (13)
           |         |          |   (8)   |--SYM-V-(class.Sex:2 flag=0001)
           |         |          |         |--SYM-V-(class.Age:3 flag=0001)
           |         |          |          \-SYM-V-(class.Height:4 flag=0001)
           |         |--SRC----|
           |         |  (4)    |--TABL[SASHELP].class opt=''
           |         |         |   (9)    /-NAME--(Age:3)
           |         |          \-CEQ----|   (14)
           |         |             (10)   \-LITN(12)
           |         |--empty-             (15)
           |         |  (5)
           |         |
           |         |                     /-SYM-V-(class.Name:1)
           |         |           /-DESC---|   (16)
           |          \-ORDR---|   (11)
           |            (6)    |          /-SYM-V-(class.Height:4)
           |                    \-ASC----|   (17)
 --SSEL---|                       (12)
    (1)
```


The checking of single observations against the criteria age=12 is done early – by the data engine. The optimizer wants to make tables as small as possible and will filter out observations (and variables) as soon as possible. Since obs with age NE 12 have been removed, SRC (4) is a small data set.

To increase speed, the Optimizer has eliminated both variables and observations.

```
*****example 14 ***This shows a calculation*********;
proc sql _Method  _tree;
title "EX14 this illustrates a = calculation";
    select name, sex, age*12 as agemo
        from sashelp.class ;

NOTE: SQL execution methods chosen are:
 Sqxslct (1) this indicates a selection of observations
   Sqxfil (2) this indicates the  application of a predicate "late" in the process
      sqxsrc( SASHELP.CLASS ) (12) this indicates a selection of observations

Tree as planned.
                      /-OBJ----|  (7)
                     |  (3)    |--SYM-V-(class.Sex:2 flag=0001)
                     |          \-SYM-A-(agemo:1 flag=0031)
            /-FIL----|
           |  (2)    |                    /-SYM-V-(class.Name:1 flag=0001)
           |         |          /-OBJ----|   (11)
           |         |         |  (8)    |--SYM-V-(class.Sex:2 flag=0001)
           |         |         |          \-SYM-V-(class.Age:3 flag=0001)
           |         |--SRC----|
           |         |  (4)     \-TABL[SASHELP].class opt=''
           |         |--empty-    (9)
           |         |--empty-
           |         |--empty-
           |         |--empty-
           |         |  (5)                /-SYM-A-(agemo:1 flag=0031)
           |         |          /-ASGN---|  (12)
           |         |         |  (10)   |          /-SYM-V-(class.Age:3)
           |         |         |          \-AMUL---|   (14)
           |         |         |            (13)    \-LITN(12)
           |          \-OBJE---|
 --SSEL---|            (6)
   (1)
```

A Calculation in SQL

The optimizer passed Agemo to (2) but not age

Fil is the late application of a predicate

Here we see the multiplication calculation and assignment (**AMUL**)in the SYM-A (10, 12, 13). The variable age is required in object (8) but is not passed through to object (3) Note that the variable, agemo, is passed through summary object (3).

Fil means that there is a filter that is applied here, that can not be applied earlier (to the right).

Since age is not in the output of the query, the data engine would 'like" to not bring age into the result set. However, the data engine can not do the multiplication required for agemo. The Optimizer directs that the data engine bring in age so that SQL can use it in the multiplication. SQL calculates agemo and passes up the result. At the next higher level, the filter (2) on the variable age can be applied to reduce the size of the data set.

NOTE that an OBJE, as well as an OBJ, can be passed to the left.

```
*****example 15 *******This illustrates a division************;
 proc sql _Method  _tree;
 title "EX15 this illustrates a = division";

     select name, sex, height/12 as Ht_feet
         from sashelp.class
         where age = 12
         order by  name desc ;

NOTE: SQL execution methods chosen are:
  Sqxslct (1) this indicates a selection of observations
      Sqxsort  (2) this indicates a SORT
          Sqxfil  (4) this indicates the application of a predicate late in the process
              sqxsrc( SASHELP.CLASS ) (9) this indicates a selection of observations

Tree as planned.                  /-SYM-V-(class.Name:1 flag=0001)
                      /-OBJ----|   (7)
                     |  (3)    |--SYM-V-(class.Sex:2 flag=0001)
                     |          \-SYM-A-(Ht_feet:1 flag=0031)
              /-SORT---|
             |  (2)    |                      /-SYM-V-(class.Name:1 flag=0001)
             |         |           /-OBJ----|  (13)
             |         |          |  (8)    |--SYM-V-(class.Sex:2 flag=0001)
             |         |          |          \-SYM-A-(Ht_feet:1 flag=0031)
             |         |--FIL----|
             |         |  (4)    |                 /-SYM-V-(class.Name:1 flag=0001)
             |         |         |          /-OBJ----|   (19)
             |         |         |         |  (14)   |--SYM-V-(class.Sex:2 flag=0001)
             |         |         |         |          \-SYM-V-(class.Height:4 flag=0001)
             |         |         |--SRC----|
             |         |         |  (9)    |--TABL[SASHELP].class opt=''
             |         |         |         |  (15)    /-NAME--(Age:3)
             |         |         |         |   \-CEQ----|   (20)
             |         |         |         |     (16)    \-LITN(12)
             |         |         |--empty-
             |         |         |--empty-
             |         |         |--empty-
             |         |         |--empty-
             |         |         |   (10)                /-SYM-A-(Ht_feet:1 flag=0031)
             |         |         |                /-ASGN---|   (21)
             |         |         |               |  (17)   |          /-SYM-V-(class.Height:4
             |         |         |               |         \-ADIV---|  (23)
             |         |         |               |            (22)   \-LITN(12)
             |         |          \-OBJE---|
             |         |--empty-     (11)
             |         |   (5)                 /-SYM-V-(class.Name:1)
             |         |          /-DESC---|  (18)
             |          \-ORDR---|   (12)
 --SSEL---|              (6)
    (1)
```

A Calculation in SQL

Assign to a variable called agemo the result of an Arithmetic Multiplication (AMUL) of age * 12 (a numeric literal).

The optimizer passed Ht_feet to (2) but not Height

Fil is the late application of a predicate

Note how early the Optimizer applies the where age=12 to make the file small.

Assign to a variable called Ht_feet the result of an Arithmetic Division (ADIV) of Height/ 12 (a numeric literal).

Note the ordering (12) and the division (**ADIV**).

Fil means that there is a filter that is applied here, that can not be applied earlier (to the right). The data engine handles the age=12 restriction.

Since height is not in the output of the query, the data engine would 'like" to not bring height into the result set. However, the data engine can not do division for Ht_feet. The Optimizer directs that the data engine bring in height so that SQL can use it in the division. SQL calculates Ht_feet and passes up the result. At the next higher level, the filter on the variable height, can be applied to reduce the size of the data set.

```
*****example 16 ***SUMMARY WITHOUT GROUPING***************;
 proc sql _Method  _tree;
 title "EX16 this illustrates a = summary without grouping";
     select name, sex, height, min(height) as shortest , max(height) as tallest
         from sashelp.class   order by name  ;
NOTE: The query requires remerging summary statistics back with original data.

NOTE: SQL execution methods chosen are:
 Sqxslct (1) this indicates a selection of observations
   Sqxsort  (4) this indicates a SORT
      Sqxsumn  (6) this indicates summation without grouping-a summary of the whole table
         sqxsrc( SASHELP.CLASS ) (11) this indicates a selection of observations
Tree as planned.                    /-SYM-V-(class.Name:1 flag=0001)
                        /-OBJ----|   (9)
                        |  (5)    |--SYM-V-(class.Sex:2 flag=0001)
                        |         |--SYM-V-(class.Height:4 flag=0001)
                        |         |--SYM-A-(shortest:1 flag=0039)
                        |         \-SYM-A-(tallest:2 flag=0039)
                /-SORT--|
                |  (4)  |                    /-SYM-V-(class.Name:1 flag=0001)
                |       |          /-OBJ----|  (18)
                |       |          |  (10)  |--SYM-V-(class.Sex:2 flag=0001)
                |       |          |        |--SYM-V-(class.Height:4 flag=0001)
                |       |          |        |--SYM-A-(shortest:1 flag=0039)
                |       |          |        \-SYM-A-(tallest:2 flag=0039)
                |       |--AGGR---|
                |       |  (6)    |                    /-SYM-V-(class.Height:4 flag=0001)
                |       |         |          /-OBJ----|  (26)
                |       |         |          |  (19)  |--SYM-V-(class.Name:1 flag=0001)
                |       |         |          |        \-SYM-V-(class.Sex:2 flag=0001)
                |       |         |--SRC----|
                |       |         |  (11)   \-TABL[SASHELP].class opt=''
                |       |         |--empty-    (20)
                |       |         |--empty-
                |       |         |--empty-
                |       |         |--empty-
                |       |         |  (12)              /-SYM-A-(shortest:1 flag=0039)
                |       |         |          /-ASGN---|  (27)
                |       |         |          |  (21)   \-SYM-G-(#TEMG001:1 stat=6,0 from
Height(0,0))    |       |         |--OBJE---|
   (3)          |       |         |  (13)   |          /-SYM-A-(tallest:2 flag=0039)
                |       |         |          \-ASGN---|  (28)
                |       |         |            (22)   \-SYM-G-(#TEMG002:2 stat=5,0 from
Height(0,0))    |       |         |--empty-
   (2)          |       |         |  (14)   /-SYM-G-(#TEMG001:1 stat=6,0 from Height(0,0))
                |       |         |--TLST---|  (23)
                |       |         |  (15)   \-SYM-G-(#TEMG002:2 stat=5,0 from Height(0,0))
                |       |         |                 /-SYM-S-(Height:3 ss=0018x)
                |       |          \-SLST---|  (24)
                |       |--empty-           (16)
                |       |  (7)                    /-SYM-V-(class.Name:1)
                |       |          /-ASC----|  (25)
                |       \-ORDR---|  (17)
 --SSEL---|             |  (8)
   (1)
```

Remerging is associated with the AGGR and takes another pass through the data

Newly created "assignment" variables continue to be passed leftward

New "assignment" vars. passed to (6)

This is a pass through the data to apply the criteria

Wrapping :-0

Wrapping :-0

Tlst (15) and Slst (16) help manage the criteria

Function 6 is "find minimum"

Create two variables via statistical functions

Function 5 is "find maximum"

TLST & SLST hold parameters for the aggr subroutine

Info. For sorting/ordering

Note the creation of the variables shortest and tallest in ASGNs (21) and (22)).)

```
*****************************************************;
*****************************************************;
*****************************************************;
************  JOINS *********************************;

 data left_class(  drop= RIndxName index=(LIndxName))
      Right_class( drop= LIndxName index=(RIndxName));
         length name $ 13;
         set sashelp.class;

      do i=1 to 1200; /*expand file so we do not hash*/
         name=name||put(i,5.0);
         RIndxName=name;
         LIndxName=name;
         Output;
      end;
 run;
```

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.LEFT_CLASS has 22800 observations and 7 variables
```

NOTE: Simple index LIndxName has been defined.

```
NOTE: The data set WORK.RIGHT_CLASS has 22800 observations and 7 variables.
```

NOTE: Simple index RIndxName has been defined.

```
NOTE: DATA statement used (Total process time):
      real time           4.05 seconds
      cpu time            0.27 seconds
```

Create two tables that allow us to examine joins done in several ways. We will examine joins with indexes in Left position, in right position, and in both positions.

```
*****example 17 ***** left join ****************;
 Proc SQL _method _tree;   title "EX17 Illustrating a left join" ;
 create table hope as   select coalesce(l.name, r.name), l.sex, r.age
   From left_class as l left join right_class as r  on l.name=r.name;
NOTE: SQL execution methods chosen are:
Sqxcrta  (1) this indicates a selection of observations
  Sqxjm  (2) this indicates a sort-merge type of join
    sqxsort  (11) this indicates a SORT
        sqxsrc(WORK.RIGHT_CLASS(alias=R))(16) this indicates a selection of observations
    sqxsort  (12) this indicates a SORT
        sqxsrc(WORK.LEFT_CLASS(alias=L)) (20) this indicates a selection of observations
Tree as planned.                    /-SYM-A-(#TEMA001:1 flag=0035)
                        /-OBJ----|  (10)
                       |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                       |          \-SYM-V-(r.Age:3 flag=0001)
              /-OTRJ---|
             |  (2)    |                                 /-SYM-V-(r.name:1 flag=0001)
             |         |                      /-OBJ----| (25)
             |         |                     |  (15)    \-SYM-V-(r.Age:3 flag=0001)
             |         |           /-SORT---|
             |         |          |  (11)    |                      /-SYM-V-(r.name:1 flag=0001)
             |         |          |          |           /-OBJ----|  (34)
             |         |          |          |          |  (26)    \-SYM-V-(r.Age:3 flag=0001)
             |         |          |          |--SRC----|
             |         |          |          |  (16)    \-TABL[WORK].right_class opt=''
             |         |          |          |--empty-    (27)
             |         |          |          |  (17)               /-SYM-V-(r.name:1)
             |         |          |          |           /-ASC----|  (35)
             |         |          |           \-ORDR---|  (28)
             |         |--FROM---|             (18)
             |         |  (4)    |                      /-SYM-V-(l.name:1 flag=0001)
             |         |          |           /-OBJ----|  (29)
             |         |          |          |  (19)    \-SYM-V-(l.Sex:2 flag=0001)
             |         |          |--SORT---|
             |         |            (12)    |                      /-SYM-V-(l.name:1 flag=0001)
             |         |                     |           /-OBJ----|  (36)
             |         |                     |          |  (30)    \-SYM-V-(l.Sex:2 flag=0001)
             |         |                     |--SRC----|
             |         |                     |  (20)    \-TABL[WORK].left_class opt=''
             |         |                     |--empty-    (31)
             |         |                     |  (21)               /-SYM-V-(l.name:1)
             |         |                     |           /-ASC----|  (37)
             |         |                      \-ORDR---|  (32)
             |         |--empty-                (22)
             |         |  (5)     /-SYM-V-(r.name:1)
             |         |--CEQ----|  (13)
             |         |  (6)     \-SYM-V-(l.name:1)
             |         |
             |         |--JTAG(jds=1, tagfrom=1, flags=0)
             |         |  (7)
             |         |--empty-
             |         |  (8)                /-SYM-A-(#TEMA001:1 flag=0031)
             |         |           /-ASGN---|  (23)
             |         |          |  (14)    |          /-SYM-V-(l.name:1)
             |         |          |          |-FCOA---|  (33)
             |         |          |          |  (24)    \-SYM-V-(r.name:1)
             |          \-OBJE---|
--SSEL---|              (9)
  (1)
```

No index. Use Sort merge join

Coalesced Var.

JTAG if ends in 1= left join, 2=right join 3=full join

and we have variations on above

JTAG if ends in 1= left join, 2=right join 3=full join

Coalesced Var. – note: the query did not specify a name for it

```
*****example 17A **** right join ***************;
 Proc SQL _method _tree;    title "EX17A Illustrating a right join" ;
 create table hope as    select coalesce( l.name, r.name), l.sex, r.age
  From left_class as l right join right_class as r on l.name=r.name;

NOTE: SQL execution methods chosen are:
 Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations

Tree as planned.                    /-SYM-A-(#TEMA001:1 flag=0035)
                     /-OBJ----|  (10)
                    |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                    |          \-SYM-V-(r.Age:3 flag=0001)
           /-OTRJ---|
          |  (2)    |                                 /-SYM-V-(l.name:1 flag=0001)
          |         |                      /-OBJ----|  (25)
          |         |                     |  (15)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |          /-SORT---|
          |         |         |  (11)   |                      /-SYM-V-(l.name:1 flag=0001)
          |         |         |         |           /-OBJ----|  (34)
          |         |         |         |          |  (26)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |         |         |--SRC----|
          |         |         |         |  (16)    \-TABL[WORK].left_class opt=''
          |         |         |         |--empty-    (27)
          |         |         |         |  (17)               /-SYM-V-(l.name:1)
          |         |         |         |          /-ASC----|  (35)
          |         |         |          \-ORDR---|  (28)
          |         |        |---|                  (18)
          |         |         |                    /-SYM-V-(r.name:1 flag=0001)
          |         |         |          /-OBJ----|  (29)
          |         |         |         |  (19)    \-SYM-V-(r.Age:3 flag=0001)
          |         |          \-SORT---|
          |         |            (12)   |                      /-SYM-V-(r.name:1 flag=0001)
          |         |                   |           /-OBJ----|  (36)
          |         |                   |          |  (30)    \-SYM-V-(r.Age:3 flag=0001)
          |         |                   |--SRC----|
          |         |                   |  (20)    \-TABL[WORK].right_class opt=''
          |         |                   |--empty-    (31)
          |         |                   |  (21)               /-SYM-V-(r.name:1)
          |         |                   |          /-ASC----|  (37)
          |         |                    \-ORDR---|  (32)
          |         |--empty-                      (22)
          |         |  (5)     /-SYM-V-(l.name:1)
          |         |--CEQ----|  (13)
          |         |  (6)     \-SYM-V-(r.name:1)
          |         |
          |         |--JTAG(jds=2, tagfrom=2, flags=0)
          |         |  (7)
          |         |--empty-
          |         |  (8)                  /-SYM-A-(#TEMA001:1 flag=0031)
          |         |                 /-ASGN---|  (23)
          |         |                |  (14)   |          /-SYM-V-(l.name:1)
          |         |                |         |  (33)
          |         |                |           (24)    \-SYM-V-(r.name:1)
          |          \-OBJE---|
 --SSEL---|                    (9)
   (1)
```

No index.
Use Sort merge join

Coalesced Var.

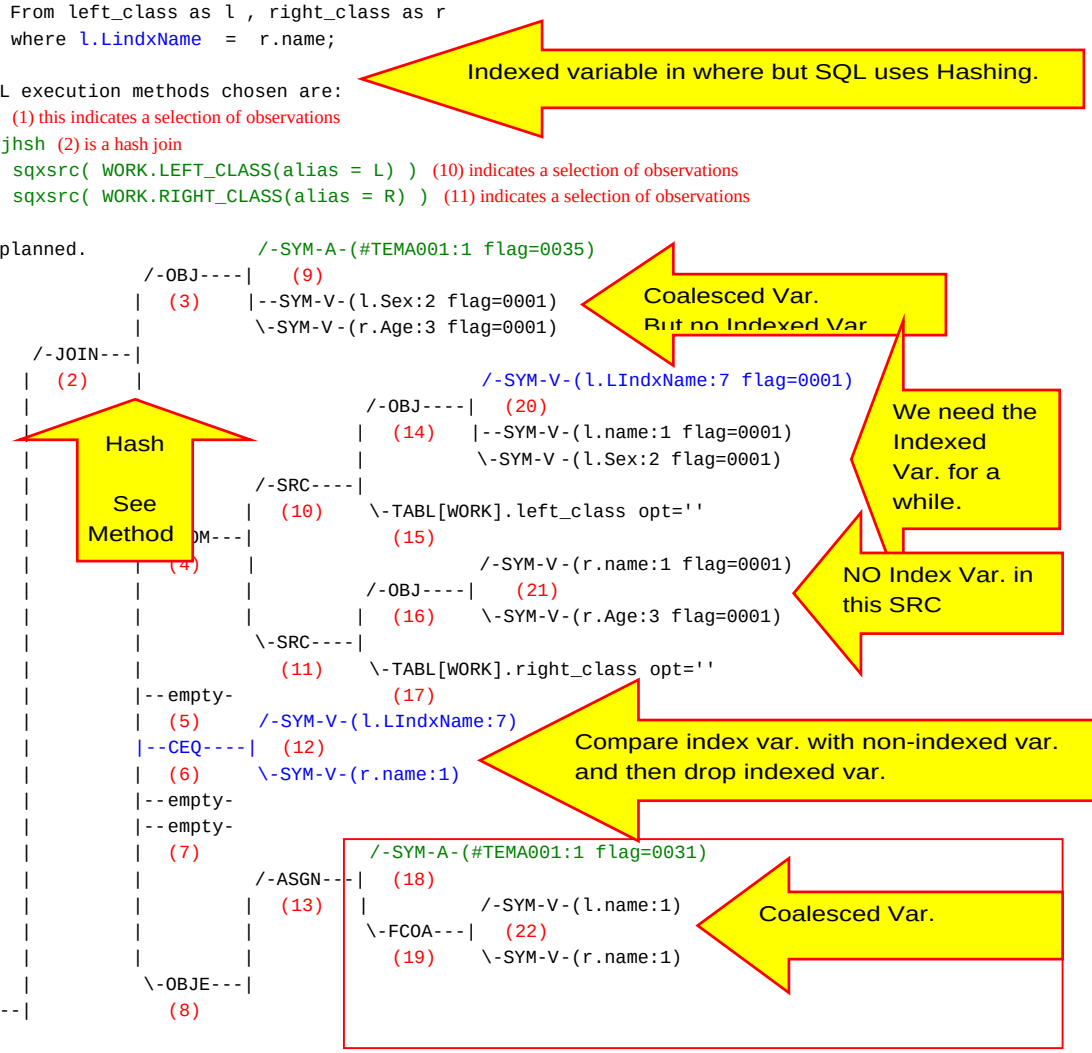
JTAG
if ends in
1= left join,
2=right join
3=full join

and we
have
variations
on above

JTAG if ends in
1= left join, 2=right join 3=full join

Coalesced Var. – note: the query did not specify a name for it

```
Proc SQL _method _tree;
 title "EX17B Illustrating an INNER JOIN WITH COMMA with index on left table" ;
 create table hope as
 select coalesce(l.name, r.name), l.sex, r.age
        From left_class as l , right_class as r
        where l.LindxName  =  r.name;

NOTE: SQL execution methods chosen are:
 Sqxcrta (1) this indicates a selection of observations
     Sqxjhsh (2) is a hash join
         sqxsrc( WORK.LEFT_CLASS(alias = L) ) (10) indicates a selection of observations
         sqxsrc( WORK.RIGHT_CLASS(alias = R) ) (11) indicates a selection of observations

Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|    (9)
                   |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                   |          \-SYM-V-(r.Age:3 flag=0001)
           /-JOIN---|
          |  (2)    |                              /-SYM-V-(l.LIndxName:7 flag=0001)
          |                                 /-OBJ----|   (20)
          |                                |  (14)   |--SYM-V-(l.name:1 flag=0001)
          |                                |          \-SYM-V-(l.Sex:2 flag=0001)
          |                      /-SRC----|
          |                     |  (10)    \-TABL[WORK].left_class opt=''
          |                     |            (15)
          |          |          |                    /-SYM-V-(r.name:1 flag=0001)
          |          |          |          /-OBJ----|    (21)
          |          |          |         |  (16)    \-SYM-V-(r.Age:3 flag=0001)
          |          |          \-SRC----|
          |          |            (11)    \-TABL[WORK].right_class opt=''
          |          |--empty-             (17)
          |          |  (5)     /-SYM-V-(l.LIndxName:7)
          |          |--CEQ----|  (12)
          |          |  (6)     \-SYM-V-(r.name:1)
          |          |--empty-
          |          |--empty-
          |          |  (7)                 /-SYM-A-(#TEMA001:1 flag=0031)
          |          |          /-ASGN---|   (18)
          |          |         |  (13)    |          /-SYM-V-(l.name:1)
          |          |         |           \-FCOA---|   (22)
          |          |         |                    \-SYM-V-(r.name:1)
          |           \-OBJE---|
 --SSEL---|             (8)
   (1)
```



L.name and R.name are coalesced (8, 13,18,19,22) and stored in a SYM-A variable (18) and kept as part of the output (9). While an index exists on the variable on the left side of the join, a hash join was selected by the optimizer.

```
 Proc SQL _method _tree;
 title "EX17C Illustrating an INNER JOIN WITH COMMA with index on RIGHT table" ;
 create table hope as
 select coalesce(l.name, r.name), l.sex, r.age   From left_class as l , right_class as r
        where l.name = r.RIndxName;
NOTE: SQL execution methods chosen are:
 Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.              /-SYM-A-(#TEMA001:1 flag=0035)
                  /-OBJ----|  (9)
                 |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                 |          \-SYM-V-(r.Age:3 flag=0001)
          /-JOIN---|
         |  (2)    |                          /-SYM-V-(l.name:1 flag=0001)
         |         |                 /-OBJ----|  (24)
         |         |                |  (14)    \-SYM-V-(l.Sex:2 flag=0001)
         |         |          /-SORT---|
         |         |         |  (10)   |                  /-SYM-V-(l.name:1 flag=0001)
         |         |         |         |          /-OBJ----|   (34)
         |         |         |         |         |  (25)    \-SYM-V-(l.Sex:2 flag=0001)
         |         |         |         |--SRC----|
         |         |         |         |  (15)    \-TABL[WORK].left_class opt=''
         |         |         |         |-- empty-    (26)
         |         |         |         |  (16)              /-SYM-V-(l.name:1)
         |         |         |         |          /-ASC----|    (35)
         |         |         |          \-ORDR---|  (27)
         |         |--FROM---|            (17)
         |         |  (4)    |                   /-SYM-V-(r.RIndxName:7 flag=0001)
         |         |         |          /-OBJ----|  (28)
         |         |         |         |  (18)   |--SYM-V-(r.name:1 flag=0001)
         |         |         |         |          \-SYM-V-(r.Age:3 flag=0001)
         |         |          \-SORT---|            (29)
         |         |            (11)   |                   /-SYM-V-(r.RIndxName:7 flag=0001)
         |         |                   |          /-OBJ----|  (36)
         |         |                   |         |  (30)   |--SYM-V-(r.name:1 flag=0001)
         |         |                   |         |          \-SYM-V-(r.Age:3 flag=0001)
         |         |                   |--SRC----|
         |         |                   |  (19)    \-TABL[WORK].right_class opt=''
         |         |                   |-- empty-    (31)
         |         |                   |  (20)              /-SYM-V-(r.RIndxName:7)
         |         |                   |          /-ASC----|   (37)
         |         |                    \-ORDR---|  (32)
         |         |--empty-              (21)
         |         |  (5)     /-SYM-V-(l.name:1)
         |         |--CEQ----|  (12)
         |         |  (6)     \-SYM-V-(r.RIndxName:7)
         |         |--empty-
         |         |--empty-
         |         |  (7)                 /-SYM-A-(#TEMA001:1 flag=0031)
         |         |          /-ASGN---|  (22)
         |         |         |  (13)   |          /-SYM-V-(l.name:1)
         |         |         |          \-FCOA---| (33)
         |         |         |            (23)    \-SYM-V-(r.name:1)
         |          \-OBJE---|
 --SSEL---|            (8)
  (1)
```

Indexed variable in where but SQL uses merge join

Coalesced Var. But no Index Var.

NO Index Var. here

Index Var. passed up for equality check

Index Var. kept just for sort & equality check

Compare index var. with non-indexed var. and then drop indexed var.

Coalesced Var.

```
Proc SQL _method _tree;
 title "EX17D Illustrating an INNER JOIN WITH COMMA with index on BOTH tables" ;
 create table hope as
 select coalesce(l.name, r.name), l.sex, r.age   From left_class as l , right_class as r
        where l.LindxName  =  r.RIndxName;

NOTE: SQL execution methods chosen are:
 Sqxcrta (1) this indicates a selection of observations
     Sqxjhsh (2) is a hash join
         sqxsrc( WORK.LEFT_CLASS(alias = L) ) (10) indicates a selection of observations
         sqxsrc( WORK.RIGHT_CLASS(alias = R) ) (11) indicates a selection of observations

Tree as planned.
                                  /-SYM-A-(#TEMA001:1 flag=0035)
                      /-OBJ----|    (9)
                     |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                     |          \-SYM-V-(r.Age:3 flag=0001)
           /-JOIN---|
          |  (2)    |                                /-SYM-V-(r.RIndxName:7 flag=0001)
          |         |                     /-OBJ----|   (20)
          |         |                    |  (14)   |--SYM-V-(r.name:1 flag=0001)
          |         |                    |          \-SYM-V-(r.Age:3 flag=0001)
          |         |          /-SRC----|
          |         |         |  (10)    \-TABL[WORK].right_class opt=''
          |         |--FROM--                (15)
          |         |  (4)                            /-SYM-V-(l.LIndxName:7 flag=0001)
          |         |                      /-OBJ----|   (21)
          |         |                     |  (16)   |--SYM-V-(l.name:1 flag=0001)
          |         |                     |          \-SYM-V-(l.Sex:2 flag=0001)
          |         |          \-SRC----|
          |         |            (11)    \-TABL[WORK].left_class opt=''
          |         |--empty-               (17)
          |         |  (5)     /-SYM-V-(r.RIndxName:7)
          |         |--CEQ----|  (12)
          |         |  (6)     \-SYM-V-(l.LIndxName:7)
          |         |--empty-
          |         |--empty-
          |         |  (7)                  /-SYM-A-(#TEMA001:1 flag=0031)
          |         |          /-ASGN---|   (18)
          |         |         |  (13)   |          /-SYM-V-(l.name:1)
          |         |         |          \-FCOA---|   (22)
          |         |         |                    (19)    \-SYM-V-(r.name:1)
          |          \-OBJE---|
 --SSEL---|            (8)
   (1)
```

2 Indexed vars. in where, but SQL uses hash

SQL uses hash

Coalesced Var. But no Indexed Vars.

Indexed Var. needed for where CEQ

Hash Join- See Methods

Index Var. passed up for equality check

Indexed Var. needed for where CEQ

Index Vars. used in equality check

Coalesced Var.

```
*****example 18 ***INNER JOIN specifying INNER JOIN Phrase ***********;
 Proc SQL _method _tree; title "EX18A Showing INNER JOIN W/ INNER JOIN Phrase-no index";
 create table hope as    select coalesce( l.name, r.name), l.sex, r.age
From left_class as l inner join  right_class as r  on l.name  =  r.name;
NOTE: SQL execution methods chosen are:
 Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.                 /-SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|    (9)
                   |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                   |          \-SYM-V-(r.Age:3 flag=0001)
           /-JOIN---|
          |  (2)    |                              /-SYM-V-(l.name:1 flag=0001)
          |         |                    /-OBJ----| (24)
          |         |                   |  (14)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |          /-SORT---|
          |         |         |  (10)   |                    /-SYM-V-(l.name:1 flag=0001)
          |         |         |         |          /-OBJ----|    (32)
          |         |         |         |         |  (25)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |         |         |--SRC----|
          |         |         |         |  (15)    \-TABL[WORK].left_class opt=''
          |         |         |         |-- empty-
          |         |         |         |  (16)              /-SYM-V-(l.name:1)
          |         |         |         |          /-ASC----|    (33)
          |         |         |          \-ORDR---|  (26)
          |         |--FROM---|             (17)
          |         |  (4)    |                    /-SYM-V-(r.name:1 flag=0001)
          |         |         |          /-OBJ----|  (27)
          |         |         |         |  (18)    \-SYM-V-(r.Age:3 flag=0001)
          |         |          \-SORT---|
          |         |             (11)  |                    /-SYM-V-(r.name:1 flag=0001)
          |         |                   |          /-OBJ----|    (34)
          |         |                   |         |  (28)    \-SYM-V-(r.Age:3 flag=0001)
          |         |                   |--SRC----|
          |         |                   |  (19)    \-TABL[WORK].right_class opt=''
          |         |                   |-- empty-    (29)
          |         |                   |  (20)              /-SYM-V-(r.name:1)
          |         |                   |          /-ASC----|    (35)
          |         |                    \-ORDR---|  (30)
          |         |--empty-               (21)
          |         |  (5)     /-SYM-V-(l.name:1)
          |         |--CEQ----|  (12)
          |         |  (6)     \-SYM-V-(r.name:1)
          |         | -empty                   /-SYM-A-(#TEMA001:1 flag=0031)
          |         |  (7)     /-ASGN---|  (22)
          |         |         |  (13)    |          /-SYM-V-(l.name:1)
          |         |         |           \-FCOA---|  (31)
          |         |         |              (23)    \-SYM-V-(r.name:1)
  (1)     |          \-OBJE---|
 --SSEL---|             (8)
```

No indexed variables. in the inner join "where", SQL sorts

Coalesced Var.

Merge Join

See Methods

Sort Info passed up

Sort Info passed up

equality check info passed up

Coalesced Var. passed up

```
 Proc SQL _method _tree;
title "EX18B INNER JOIN W/ INNER JOIN Phrase w/ index on left table";
 create table hope as
 select coalesce(l.name, r.name), l.sex, r.age
        From left_class as l inner join right_class as r
        on l.LindxName  =  r.name;

NOTE: SQL execution methods chosen are:
 Sqxcrta (1) this indicates a selection of observations
     Sqxjhsh (2) is a hash join
         sqxsrc( WORK.LEFT_CLASS(alias = L) ) (10) indicates a selection of observations
         sqxsrc( WORK.RIGHT_CLASS(alias = R) ) (11) indicates a selection of observations
```

Use the inner join phrase
Left var has index

```
Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                     /-OBJ----|  (9)
                    |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                    |          \-SYM-V-(r.Age:3 flag=0001)
           /-JOIN---|
          |  (2)    |                              /-SYM-V-(l.LIndxName:7 flag=0001)
          |                               /-OBJ----|  (20)
          |                              |  (14)   |--SYM-V-(l.name:1 flag=0001)
          |                              |          \-SYM-V-(l.Sex:2 flag=0001)
          |                    /-SRC----|
          |                   |  (10)    \-TABL[WORK].left_class opt=''
          |                   M---|
          |                   |  (15)
          |                   |                    /-SYM-V-(r.name:1 flag=0001)
          |                   |          /-OBJ----|  (21)
          |                   |         |  (16)    \-SYM-V-(r.Age:3 flag=0001)
          |                   |          \-SRC----|
          |                   |            (11)    \-TABL[WORK].right_class opt=''
          |                   |--empty-             (17)
          |                   |  (5)     /-SYM-V-(l.LIndxName:7)
          |                   |--CEQ----|  (12)
          |                   |  (6)     \-SYM-V-(r.name:1)
          |                   |--empty-
          |                   |--empty-
          |                   |  (7)                 /-SYM-A-(#TEMA001:1 flag=0031)
          |                   |          /-ASGN---|  (18)
          |                   |         |  (13)   |          /-SYM-V-(l.name:1)
          |                   |         |          \-FCOA---|  (22)
          |                   |         |                    (19)    \-SYM-V-(r.name:1)
          |                    \-OBJE---|
 --SSEL---|                      (8)
   (1)
```

NO LindxName passed up

Used and discarded ASAP

Hash

See Methods

equality check info passed up

Put Into this var.

2 Coalesced Vars.

```
Proc SQL _method _tree; title "EX18C INNER JOIN W/INNER JOIN Phrase w/ index on RIGHT table " ;
create table hope as   select coalesce( l.name, r.name), l.sex, r.age
    From left_class as l inner join right_class as r    on l.name = r.RIndxName;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
```

No indexed variables. in the inner join "where", so SQL sorts

Coalesced Var.

NO RIndxName passed up

Merge Join

See Methods

Used and discarded ASAP

equality check info passed up

Coalesced Var.

```
Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                     /-OBJ----|  (9)
                    |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                    |          \-SYM-V-(r.Age:3 flag=0001)
           /-JOIN---|
          |  (2)    |                            /-SYM-V-(l.name:1 flag=0001)
          |         |                  /-OBJ----|  (24)
          |         |                 |   (14)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |          /-SORT---|
          |         |         |  (10)   |                    /-SYM-V-(l.name:1 flag=0001)
          |         |         |         |          /-OBJ----|  (33)
          |         |         |         |         |   (25)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |         |         |--SRC----|
          |         |         |         |   (15)    \-TABL[WORK].left_class opt=''
          |         |         |         |-- empty-    (26)
          |         |         |         |   (16)              /-SYM-V-(l.name:1)
          |         |         |         |          /-ASC----|   (34)
          |         |         |          \-ORDR---|   (27)
          |         |--FROM---|             (17)
          |         |  (4)    |                    /-SYM-V-(r.RIndxName:7 flag=0001)
          |         |         |          /-OBJ----|  (28)
          |         |         |         |   (18)   |--SYM-V-(r.name:1 flag=0001)
          |         |         |         |          \-SYM-V-(r.Age:3 flag=0001)
          |         |          \-SORT---|
          |         |            (11)   |                    /-SYM-V-(r.RIndxName:7 flag=0001)
          |         |                   |          /-OBJ----|  (35)
          |         |                   |         |   (29)   |--SYM-V-(r.name:1 flag=0001)
          |         |                   |         |          \-SYM-V-(r.Age:3 flag=0001)
          |         |                   |--SRC----|
          |         |                   |   (19)    \-TABL[WORK].right_class opt=''
          |         |                   |-- empty-    (30)
          |         |                   |   (20)              /-SYM-V-(r.RIndxName:7)
          |         |                   |          /-ASC----|   (36)
          |         |                    \-ORDR---|   (31)
          |         |--empty-               (21)
          |         |  (5)     /-SYM-V-(l.name:1)
          |         |--CEQ----|  (12)
          |         |  (6)     \-SYM-V-(r.RIndxName:7)
          |         |--empty-
          |         |--empty-     /-SYM-A-(#TEMA001:1 flag=0031)
          |         |  (7)     /-ASGN---|  (22)
          |         |         |  (13)    |          /-SYM-V-(l.name:1)
          |         |         |           \-FCOA---|  (32)
          |         |         |              (23)    \-SYM-V-(r.name:1)
   (1)    |          \-OBJE---|
 --SSEL---|             (8)
```

```
Proc SQL _method _tree; title "EX18D INNER JOIN W/INNER JOIN Phrase w/index on BOTH tables";
 create table hope as
 select coalesce(l.name, r.name), l.sex, r.age
        From left_class as l inner join right_class as r
        on l.LindxName  =  r.RIndxName;

NOTE: SQL execution methods chosen are:
 Sqxcrta (1) this indicates a selection of observations
     Sqxjhsh (2) is a hash join
         sqxsrc( WORK.LEFT_CLASS(alias = L) ) (10) indicates a selection of observations
         sqxsrc( WORK.RIGHT_CLASS(alias = R) ) (11) indicates a selection of observations

Tree as planned.
                                  /-SYM-A-(#TEMA001:1 flag=0035)
                        /-OBJ----|  (9)
                       |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                       |          \-SYM-V-(r.Age:3 flag=0001)
          /-JOIN---|
         |  (2)    |                                 /-SYM-V-(r.RIndxName:7 flag=0001)
         |         |                       /-OBJ----|  (21)
         |         |                      |  (15)   |--SYM-V-(r.name:1 flag=0001)
         |         |                      |          \-SYM-V-(r.Age:3 flag=0001)
         |         |           /-SRC----|
         |         |          |  (10)    \-TABL[WORK].right_class opt=''
         |         |--FROM---|            (16)
         |         |  (4)    |                       /-SYM-V-(l.LIndxName:7 flag=0001)
         |         |         |           /-OBJ----|  (22)
         |         |         |          |  (17)   |--SYM-V-(l.name:1 flag=0001)
         |         |         |          |          \-SYM-V-(l.Sex:2 flag=0001)
         |         |          \-SRC----|
         |         |            (11)    \-TABL[WORK].left_class opt=''
         |         |--empty-              (18)
         |         |  (5)     /-SYM-V-(r.RIndxName:7)
         |         |--CEQ----|  (12)
         |         |  (6)     \-SYM-V-(l.LIndxName:7)
         |         |--empty-    (13)
         |         |--empty-
         |         |  (7)                 /-SYM-A-(#TEMA001:1 flag=0031)
         |         |           /-ASGN---|  (19)
         |         |          |  (14)   |          /-SYM-V-(l.name:1)
         |         |          |          \-FCOA---|  (23)
         |         |          |            (20)    \-SYM-V-(r.name:1)
         |          \-OBJE---|
 --SSEL---|            (8)
   (1)
```

Use INNER JOIN Phrase in this example! Two vars have index

Coalesced Var.

Used and discarded ASAP

HASH Join

See Methods

Used and discarded ASAP

equality check info passed up

Coalesced Var.

```
*****example 19 ********LEFT JOINS ****************;
 Proc SQL _method _tree;  title "EX19A Illustrating a left join with no indexes" ;
 create table hope as   select coalesce( l.name, r.name), l.sex, r.age
    From left_class as l left join right_class as r  on l.name = r.name;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
```

Use LEFT JOIN Phrase!
No index to use

```
Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|  (10)
                   |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                   |          \-SYM-V-(r.Age:3 flag=0001)
           /-OTRJ---|
          |  (2)    |                              /-SYM-V-(r.name:1 flag=0001)
          |         |                    /-OBJ----|  (25)
          |         |                   |  (15)    \-SYM-V-(r.Age:3 flag=0001)
          |         |          /-SORT---|
          |         |         |  (11)   |                    /-SYM-V-(r.name:1 flag=00  )
          |         |         |         |          /-OBJ----|  (38)
          |         |         |         |         |  (26)    \-SYM-V-(r.Age:3 flag=0001)
          |         |         |         |--SRC----|
          |         |         |         |  (16)    \-TABL[WORK].right_class opt=''
          |         |         |         |--empty-    (27)
          |         |         |         |  (17)               /-SYM-V-(r.name:1)
          |         |         |         |          /-ASC----|  (39)
          |         |         |          \-ORDR---|  (28)
          |         |         |M---|        (18)
          |         |         |                    /-SYM-V-(l.name:1 flag=0001)
          |         |         |          /-OBJ----|  (29)
          |         |         |         |  (19)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |          \-SORT---|
          |         |            (12)   |                    /-SYM-V-(l.name:1 flag=00  )
          |         |                   |          /-OBJ----|  (40)
          |         |                   |         |  (30)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |                   |--SRC----|
          |         |                   |  (20)    \-TABL[WORK].left_class opt=''
          |         |                   |--empty-    (31)
          |         |                   |  (21)               /-SYM-V-(l.name:1)
          |         |                   |          /-ASC----|  (41)
          |         |                    \-ORDR---|  (32)
          |         |--empty-              (22)
          |         |  (5)     /-SYM-V-(r.name:1)
          |         |--CEQ----|  (13)
          |         |  (6)     \-SYM-V-(l.name:1)
          |         |--JTAG(jds=1, tagfrom=1, flags=0)
          |         |  (7)
          |         |--empty-              /-SYM-A-(#TEMA001:1 flag=0031)
          |         |  (8)     /-ASGN---|  (23)    /-SYM-V-(l.name:1)
          |         |         |  (14)    \-FCOA---| (33)
          |          \-OBJE--|             (24)    \-SYM-V-(r.name:1)
 --SSEL---|            (9)
  (1)
```

Coalesced Var.

R.Name is Passed up, used & discarded after coalesce.

JTAG if ends in 1= left join, 2=right join 3=full join

Merge join notation of JOIN Or OTRJ

Merg joins can be used for inner join or outer joins

L.Name is Passed up, used & discarded after coalesce

JTAG: if ends in 1= left join, 2=right join 3=full join

equality check info passed up the query

Coalesce Vars.

```
Proc SQL _method _tree;  title "EX19B Illustrating a right join with index on left table" ;
 create table hope as  select coalesce( l.name, r.name), l.sex, r.age
   From left_class as l left join right_class as r   on l.LindxName = r.name;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observation
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
```

Use LEFT JOIN Phrase! Left var has index

```
Tree as planned.                  /-SYM-A-(#TEMA001:1 flag=0035)
                     /-OBJ----|  (10)
                    |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                    |          \-SYM-V-(r.Age:3 flag=0001)
           /-OTRJ---|
          |  (2)    |                              /-SYM-V-(r.name:1 flag=0001)
          |         |                    /-OBJ----|  (25)
          |         |                   |  (15)    \-SYM-V-(r.Age:3 flag=0001)
          |         |          /-SORT---|
          |         |         |  (11)   |                    /-SYM-V-(r.name:1 flag=0001)
          |         |         |         |          /-OBJ----|  (36)
          |         |         |         |         |  (26)    \-SYM-V-(r.Age:3 flag=0001)
          |         |         |         |--SRC----|
          |         |         |         |  (16)    \-TABL[WORK].right_class opt=''
          |         |         |         |-- empty-    (28)
          |         |         |         |  (17)              /-SYM-V-(r.name:1)
          |         |         |         |          /-ASC----|  (37)
          |         |         |         |           \-ORDR---|  (29)
          |         |--FROM---|         |              (18)
          |         |         |                    /-SYM-V-(l.LIndxName:7 flag=0001)
          |         |         |         |          /-OBJ----|  (30)
          |         |         |         |         |  (19)   |--SYM-V-(l.name:1 flag=0001)
          |         |         |         |         |         |  \-SYM-V-(l.Sex:2 flag=0001)
          |         |         |          \-SORT---|            (31)
          |         |         |             (12)  |                    /-SYM-V-(l.LIndxName:7 flag=000
          |         |         |                   |          /-OBJ----|  (38)
          |         |         |                   |         |  (32)   |--SYM-V-(l.name:1 flag=0001)
          |         |         |                   |         |         |  \-SYM-V-(l.Sex:2 flag=0001)
          |         |         |                   |--SRC----|
          |         |         |                   |  (20)    \-TABL[WORK].left_class opt=''
          |         |         |                   |-- empty-    (33)
          |         |         |                   |  (21)              /-SYM-V-(l.LIndxName:7)
          |         |         |                   |          /-ASC----|  (39)
          |         |         |                    \-ORDR---|  (34)
          |         |--empty-                           (22)
          |         |  (5)     /-SYM-V-(r.name:1)
          |         |--CEQ----|  (13)
          |         |  (6)     \-SYM-V-(l.LIndxName:7)
          |        --JTAG(jds=1, tagfrom=1, flags=0)
          |         |  (7)                /-SYM-A-(#TEMA001:1 flag=0031)
          |         |         /-ASGN---|  (23)
          |         |--empty- |  (14)   |          /-SYM-V-(l.name:1)
          |         |  (8)    |          \-FCOA---|  (35)
          |         |         |             (24)    \-SYM-V-(r.name:1)
     (1)  |          \-OBJE---|
 --SSEL---|             (9)
```

Coalesced Var.

R.Name is passed up, used & dropped after coalesce.

JTAG if ends in 1= left join, 2=right join 3=full join

L.Name and Lindxname are passed up, used & dropped coalesce

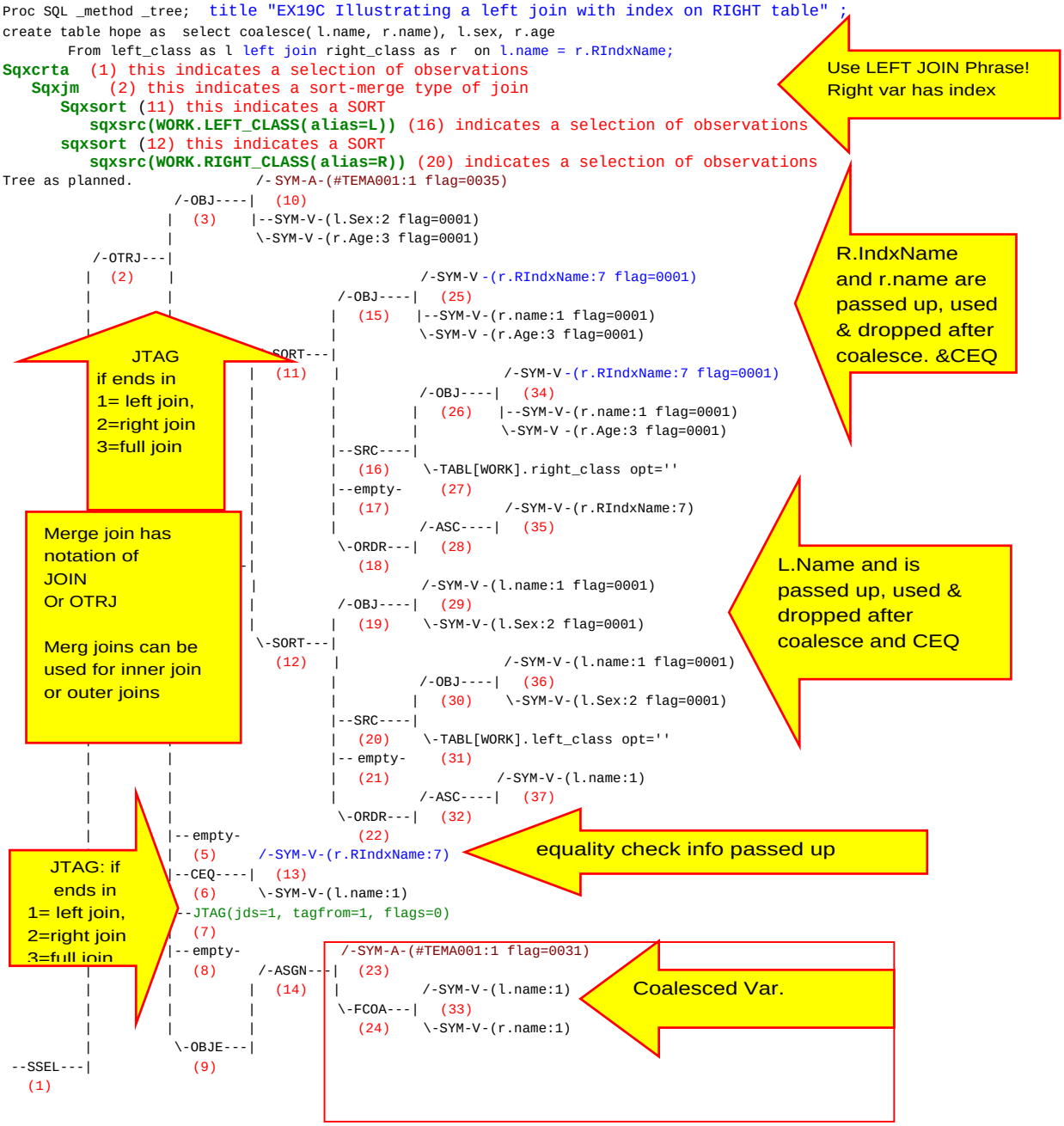
Merge join has notation of JOIN Or OTRJ

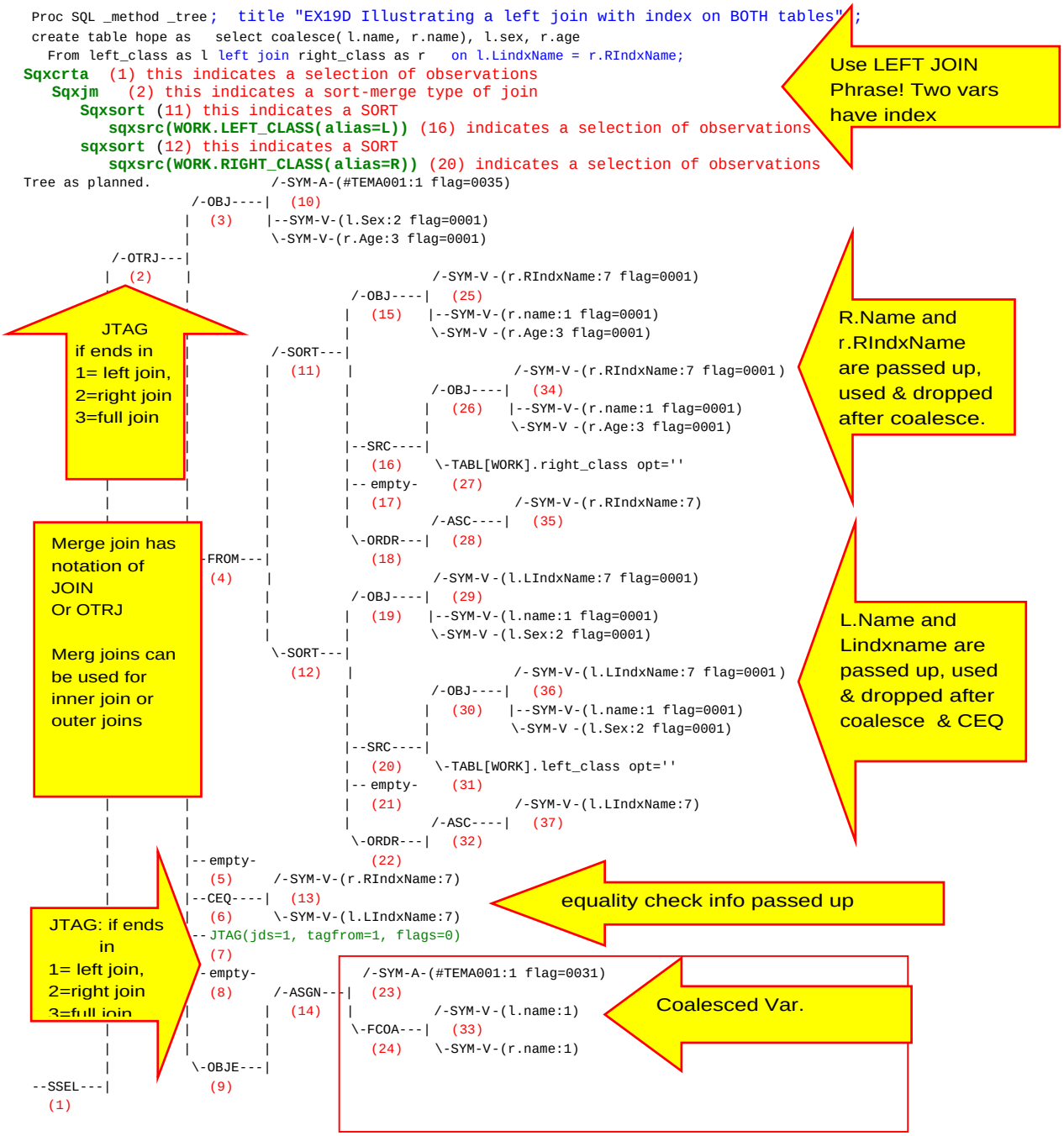
Merg joins can be used for inner join or outer joins

equality check info passed up. Note LIndxName

JTAG: if ends in 1= left join, 2=right join 3=full join

Coalesced Var. uses L.name and R.name but NOT LIndxName


Proc SQL _method _tree; title "EX19C Illustrating a left join with index on RIGHT table" ;
create table hope as select coalesce(l.name, r.name), l.sex, r.age
From left_class as l left join right_class as r on l.name = r.RIndxName;
Sqxcrta (1) this indicates a selection of observations
Sqxjm (2) this indicates a sort-merge type of join
Sqxsort (11) this indicates a SORT
sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
sqxsort (12) this indicates a SORT
sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.
Use LEFT JOIN Phrase!
Right var has index
R.IndxName and r.name are passed up, used & dropped after coalesce. &CEQ
JTAG if ends in 1= left join, 2=right join 3=full join
Merge join has notation of JOIN Or OTRJ
Merg joins can be used for inner join or outer joins
L.Name and is passed up, used & dropped after coalesce and CEQ
equality check info passed up
JTAG: if ends in 1= left join, 2=right join 3=full join
Coalesced Var.


Proc SQL _method _tree; title "EX19D Illustrating a left join with index on BOTH tables";
create table hope as select coalesce(l.name, r.name), l.sex, r.age
From left_class as l left join right_class as r on l.LindxName = r.RIndxName;
Sqxcrta (1) this indicates a selection of observations
Sqxjm (2) this indicates a sort-merge type of join
Sqxsort (11) this indicates a SORT
sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
sqxsort (12) this indicates a SORT
sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.
/-SYM-A-(#TEMA001:1 flag=0035)
/-OBJ----| (10)
|--SYM-V-(l.Sex:2 flag=0001)
\-SYM-V-(r.Age:3 flag=0001)
/-OTRJ---|
/-SYM-V -(r.RIndxName:7 flag=0001)
/-OBJ----| (25)
|--SYM-V-(r.name:1 flag=0001)
\-SYM-V -(r.Age:3 flag=0001)
/-SORT---|
/-SYM-V -(r.RIndxName:7 flag=0001)
/-OBJ----| (34)
|--SYM-V-(r.name:1 flag=0001)
\-SYM-V -(r.Age:3 flag=0001)
|--SRC----|
\-TABL[WORK].right_class opt=''
|-- empty- (27)
/-SYM-V-(r.RIndxName:7)
/-ASC----| (35)
\-ORDR---| (28)
-FROM---| (18)
/-SYM-V -(l.LIndxName:7 flag=0001)
/-OBJ----| (29)
|--SYM-V-(l.name:1 flag=0001)
\-SYM-V -(l.Sex:2 flag=0001)
\-SORT---|
/-SYM-V-(l.LIndxName:7 flag=0001)
/-OBJ----| (36)
|--SYM-V-(l.name:1 flag=0001)
\-SYM-V -(l.Sex:2 flag=0001)
|--SRC----|
\-TABL[WORK].left_class opt=''
|-- empty- (31)
/-SYM-V-(l.LIndxName:7)
/-ASC----| (37)
\-ORDR---| (32)
|-- empty- (22)
/-SYM-V-(r.RIndxName:7)
|--CEQ----| (13)
\-SYM-V-(l.LIndxName:7)
--JTAG(jds=1, tagfrom=1, flags=0)
(7)
-empty-
/-SYM-A-(#TEMA001:1 flag=0031)
/-ASGN---| (23)
/-SYM-V-(l.name:1)
\-FCOA---| (33)
\-SYM-V-(r.name:1)
\-OBJE---|
--SSEL---|
(1)
Use LEFT JOIN Phrase! Two vars have index
JTAG if ends in 1= left join, 2=right join 3=full join
R.Name and r.RIndxName are passed up, used & dropped after coalesce.
Merge join has notation of JOIN Or OTRJ
Merg joins can be used for inner join or outer joins
L.Name and Lindxname are passed up, used & dropped after coalesce & CEQ
equality check info passed up
JTAG: if ends in 1= left join, 2=right join 3=full join
Coalesced Var.

```
*****example 20 **** RIGHT JOIN ******************;
Proc SQL _method _tree;   title "EX20A Illustrating a right join no indexes" ;
create table hope as   select coalesce( l.name, r.name), l.sex, r.age
  From left_class as l right join right_class as r   on l.name = r.name;
Sqxcrta  (1) this indicates a selection of observations
  Sqxjm   (2) this indicates a sort-merge type of join
    Sqxsort (11) this indicates a SORT
      sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
    sqxsort (12) this indicates a SORT
      sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|  (10)
                   |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                   |          \-SYM-V-(r.Age:3 flag=0001)
          /-OTRJ---|
         |  (2)    |                            /-SYM-V-(l.name:1 flag=0001)
         |         |                  /-OBJ----|  (24)
         |         |                 |  (15)    \-SYM-V-(l.Sex:2 flag=0001)
         |         |          /-SORT---|
         |         |         |  (11)   |                  /-SYM-V-(l.name:1 flag=0001)
         |         |         |         |          /-OBJ----|  (34)
         |         |         |         |         |  (25)    \-SYM-V-(l.Sex:2 flag=0001)
         |         |         |         |--SRC----|
         |         |         |         |  (16)    \-TABL[WORK].left_class opt=''
         |         |         |         |-- empty-    (26)
         |         |         |         |  (17)               /-SYM-V-(l.name:1)
         |         |         |         |          /-ASC----|  (35)
         |         |         |          \-ORDR---|  (27)
         |         |--FROM---|             (18)
         |          (4)    |                            /-SYM-V-(r.name:1 flag=0001)
         |                 |          /-OBJ----|  (28)
         |                 |         |  (19)    \-SYM-V-(r.Age:3 flag=0001)
         |                  \-SORT---|             (29)
         |                    (12)   |                  /-SYM-V-(r.name:1 flag=0001)
         |                           |          /-OBJ----|  (36)
         |                           |         |  (30)    \-SYM-V-(r.Age:3 flag=0001)
         |                           |--SRC----|
         |                           |  (20)    \-TABL[WORK].right_class opt=''
         |                           |-- empty-    (31)
         |                           |  (21)               /-SYM-V-(r.name:1)
         |                           |          /-ASC----|  (37)
         |                            \-ORDR---|  (32)
         |          |--empty-
         |          |  (5)     /-SYM-V-(l.name:1)
         |          --CEQ----|  (13)
         |           (6)     \-SYM-V-(r.name:1)
         |          -JTAG(jds=2, tagfrom=2, flags=0)
         |           (7)
         |          --empty-                  /-SYM-A-(#TEMA001:1 flag=0031)
         |          |  (8)     /-ASGN---|  (22)
         |          |         |  (14)    |          /-SYM-V-(l.name:1)
         |          |         |          \-FCOA---|  (33)
         |          |         |                     (23)    \-SYM-V-(r.name:1)
   (1)   |           \-OBJE---|
 --SSEL---|             (9)
```

Use RIGHT JOIN Phrase!

L.Name is passed up, used & dropped after coalesce & CEQ

JTAG if ends in 1= left join, 2=right join 3=full join

r.name is passed up, used & dropped after coalesce. &CEQ

Merge join has notation of JOIN Or OTRJ

Merg joins can be used for inner join or outer joins

equality check info passed up

JTAG: if ends in 1= left join, 2=right join 3=full join

Coalesced Var.

```
Proc SQL _method _tree; title "EX20B Illustrating a right join witn index on left table" ;
 create table hope as   select coalesce( l.name, r.name), l.sex, r.age
  From left_class as l right join right_class as r   on l.LindxName = r.name;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.                /-(10)  SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|--SYM-V-(l.Sex:2 flag=0001)
                   |  (3)     \-SYM-V-(r.Age:3 flag=0001)
          /-OTRJ---|
         |  (2)    |                              /-SYM-V-(l.LIndxName:7 flag=0001)
         |         |                    /-OBJ----|  (25)
         |         |                   |  (15)   |--SYM-V-(l.name:1 flag=0001)
         |         |                   |         \-SYM-V-(l.Sex:2 flag=0001)
         |         |          /-SORT---|
         |         |         |  (11)   |                    /-SYM-V-(l.LIndxName:7 flag=0001)
         |         |         |         |          /-OBJ----|  (34)
         |         |         |         |         |  (26)   |--SYM-V-(l.name:1 flag=0001)
         |         |         |         |         |         \-SYM-V-(l.Sex:2 flag=0001)
         |         |         |         |--SRC----|
         |         |         |         |  (16)    \-TABL[WORK].left_class opt=''
         |         |         |         |-- empty-    (27)
         |         |         |         |  (17)              /-SYM-V-(l.LIndxName:7)
         |         |         |         |          /-ASC----|  (35)
         |         |         |          \-ORDR---|  (28)
         |         |-FROM---|                    (18)
         |          (4)    |                    /-SYM-V-(r.name:1 flag=0001)
         |                 |          /-OBJ----|  (29)
         |                 |         |  (19)    \-SYM-V-(r.Age:3 flag=0001)
         |                  \-SORT---|
         |                    (12)   |                    /-SYM-V-(r.name:1 flag=0001
         |                           |          /-OBJ----|  (36)
         |                           |         |  (30)    \-SYM-V-(r.Age:3 flag=0001)
         |                           |--SRC----|
         |                           |  (20)    \-TABL[WORK].right_class opt=''
         |         |                 |-- empty-    (31)
         |         |                 |  (21)              /-SYM-V-(r.name:1)
         |         |                 |          /-ASC----|  (37)
         |         |                  \-ORDR---|  (32)
         |         |--empty-                   (22)
         |         |  (5)     /-SYM-V-(l.LIndxName:7)
         |         |--CEQ----|  (13)
         |          (6)     \-SYM-V-(r.name:1)
         |         -(7)JTAG(jds=2, tagfrom=2, flags=0)
         |         --empty-                 /-SYM-A-(#TEMA001:1 flag=0031)
         |         |  (8)     /-ASGN---|  (23)
         |         |          |  (14)   |         /-SYM-V-(l.name:1)
         |         |          |          \-FCOA---|  (33)
 (1)     |         |          |                    (24)    \-SYM-V-(r.name:1)
--SSEL----|         \-(9)OBJE-|
```

Use RIGHT JOIN Phrase!

L.Name and L.LindxName are passed up, used & dropped after coalesce & CEQ

JTAG if ends in 1= left join, 2=right join 3=full join

r.name is passed up, used & dropped after coalesce & CEQ

Merge join has notation of JOIN Or OTRJ

Merg joins can be used for inner join or outer joins

equality check info passed up

JTAG: if ends in 1= left join, 2=right join 3=full join

Coalesced Var.

```
Proc SQL _method _tree; title "EX20C Illustrating a right join with index on RIGHT table " ;
 create table hope as   select coalesce( l.name, r.name), l.sex, r.age
    From left_class as l right join right_class as r  on l.name = r.RIndxName;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|  (10)
                   |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                   |         \-SYM-V-(r.Age:3 flag=0001)
          /-OTRJ---|
         |  (2)    |                                /-SYM-V-(l.name:1 flag=0001)
         |         |                     /-OBJ----|  (25)
         |         |                    |  (15)    \-SYM-V-(l.Sex:2 flag=0001)
         |         |          /-SORT---|
         |         |         |  (11)   |                     /-SYM-V-(l.name:1 flag=0001)
         |         |         |         |           /-OBJ----|  (34)
         |         |         |         |          |  (26)    \-SYM-V-(l.Sex:2 flag=0001)
         |         |         |         |--SRC----|
         |         |         |         |  (16)    \-TABL[WORK].left_class opt=''
         |         |         |         |-- empty-    (27)
         |         |         |         |  (17)               /-SYM-V-(l.name:1)
         |         |         |         |           /-ASC----|    (35)
         |         |         |         \-ORDR---|  (28)
         |         |--FROM---|                    (18)
         |         |  (4)    |                     /-SYM-V-(r.RIndxName:7 flag=0001)
         |         |         |           /-OBJ----|  (29)
         |         |         |          |  (19)   |--SYM-V-(r.name:1 flag=0001)
         |         |         |          |         \-SYM-V-(r.Age:3 flag=0001)
         |         |          \-SORT---|
         |         |            (12)   |                     /-SYM-V-(r.RIndxName:7 flag=0001)
         |         |                   |           /-OBJ----|    (36)
         |         |                   |          |  (30)   |--SYM-V-(r.name:1 flag=0001)
         |         |                   |          |         \-SYM-V-(r.Age:3 flag=0001)
         |         |                   |--SRC----|
         |         |                   |  (20)    \-TABL[WORK].right_class opt=''
         |         |                   |-- empty-    (31)
         |         |                   |  (21)               /-SYM-V-(r.RIndxName:7)
         |         |                   |           /-ASC----|    (37)
         |         |                    \-ORDR---|  (32)
         |         |--empty-                     (22)
         |         |  (5)     /-SYM-V-(l.name:1)
         |         |--CEQ----|  (13)
         |          (6)     \-SYM-V-(r.RIndxName:7)
         |         --(7)JTAG(jds=2, tagfrom=2, flags=0)
         |         --empty-             /-SYM-A-(#TEMA001:1 flag=0031)
         |         |  (8)     /-ASGN---|  (23)
         |         |         |  (14)    |          /-SYM-V-(l.name:1)
         |         |         |          \-FCOA---|  (33)
         |         |         |                     (24)    \-SYM-V-(r.name:1)
  (1)    |          \-OBJE---|
 --SSEL---|            (9)
```

Use RIGHT JOIN Phrase!

JTAG if ends in 1= left join, 2=right join 3=full join

L.Name is passed up, used & dropped after coalesce & CEQ

r.name and R.RindxNam e are passed up, used & dropped after coalesce. &CEQ

Merge join has notation of JOIN Or OTRJ

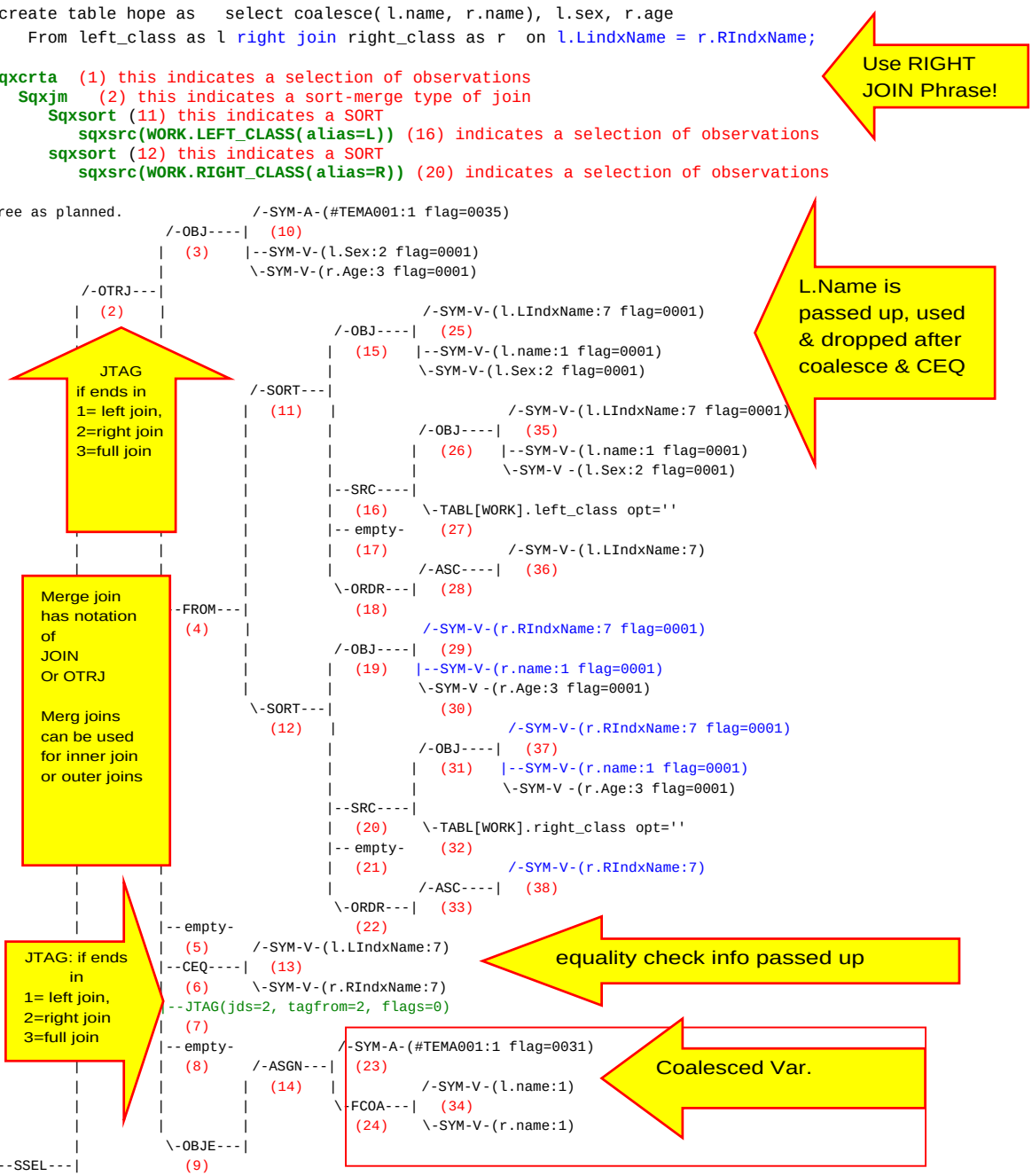
Merg joins can be used for inner join or outer joins

equality check info passed up

JTAG: if ends in 1= left join, 2=right join 3=full join

Coalesced Var.

```
Proc SQL _method _tree;  title "EX20D Illustrating a right join witn index on BOTH tables" ;
 create table hope as   select coalesce( l.name, r.name), l.sex, r.age
    From left_class as l right join right_class as r  on l.LindxName = r.RIndxName;

Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations

Tree as planned.                 /-SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|   (10)
                   |   (3)    |--SYM-V-(l.Sex:2 flag=0001)
                   |           \-SYM-V-(r.Age:3 flag=0001)
           /-OTRJ---|
          |  (2)    |                            /-SYM-V-(l.LIndxName:7 flag=0001)
          |         |                  /-OBJ----|   (25)
          |         |                 |   (15)   |--SYM-V-(l.name:1 flag=0001)
          |         |                 |           \-SYM-V-(l.Sex:2 flag=0001)
          |         |          /-SORT---|
          |         |         |   (11)   |                    /-SYM-V-(l.LIndxName:7 flag=0001)
          |         |         |          |           /-OBJ----|   (35)
          |         |         |          |          |   (26)   |--SYM-V-(l.name:1 flag=0001)
          |         |         |          |          |           \-SYM-V -(l.Sex:2 flag=0001)
          |         |         |          |--SRC----|
          |         |         |          |   (16)    \-TABL[WORK].left_class opt=''
          |         |         |          |-- empty-    (27)
          |         |         |          |   (17)               /-SYM-V-(l.LIndxName:7)
          |         |         |          |           /-ASC----|   (36)
          |         |         |           \-ORDR---|   (28)
          |         |         |                      (18)
          |         |-FROM---|                      /-SYM-V-(r.RIndxName:7 flag=0001)
          |         |  (4)    |           /-OBJ----|   (29)
          |         |         |          |   (19)   |--SYM-V-(r.name:1 flag=0001)
          |         |         |          |           \-SYM-V -(r.Age:3 flag=0001)
          |         |         |           \-SORT---|      (30)
          |         |         |              (12)   |                    /-SYM-V-(r.RIndxName:7 flag=0001)
          |         |         |                     |           /-OBJ----|   (37)
          |         |         |                     |          |   (31)   |--SYM-V-(r.name:1 flag=0001)
          |         |         |                     |          |           \-SYM-V -(r.Age:3 flag=0001)
          |         |         |                     |--SRC----|
          |         |         |                     |   (20)    \-TABL[WORK].right_class opt=''
          |         |         |                     |-- empty-    (32)
          |         |         |                     |   (21)               /-SYM-V-(r.RIndxName:7)
          |         |         |                     |           /-ASC----|   (38)
          |         |         |                      \-ORDR---|   (33)
          |         |         |                                 (22)
          |         |--empty-                    /-SYM-V-(l.LIndxName:7)
          |         |  (5)     /-SYM-V-(l.LIndxName:7)
          |         |--CEQ----|  (13)
          |         |  (6)     \-SYM-V-(r.RIndxName:7)
          |         |--JTAG(jds=2, tagfrom=2, flags=0)
          |         |  (7)
          |         |--empty-                  /-SYM-A-(#TEMA001:1 flag=0031)
          |         |  (8)     /-ASGN---|  (23)
          |         |         |  (14)    |           /-SYM-V-(l.name:1)
          |         |         |           \-FCOA---|   (34)
          |         |         |                     (24)    \-SYM-V-(r.name:1)
          |          \-OBJE---|
  --SSEL---|                   (9)
    (1)
```




```
NOTE: Table WORK.HOPE created, with 27360000 rows and 3 columns.
```

```
*****example 21 ***** FULL JOIN ******************;
 Proc SQL _method _tree;   title "EX21A Illustrating a full join with no indexes" ;
 create table hope as   select coalesce( l.name, r.name), l.sex, r.age
  From left_class as l full join right_class as r    on l.name = r.name;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                     /-OBJ----|  (10)
                    |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                    |          \-SYM-V-(r.Age:3 flag=0001)
           /-OTRJ---|
          |  (2)    |                              /-SYM-V-(l.name:1 flag=0001)
          |         |                   /-OBJ----|  (25)
          |         |                  |  (15)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |          /-SORT---|
          |         |         |  (11)   |                    /-SYM-V-(l.name:1 flag=0001)
          |         |         |         |          /-OBJ----|  (35)
          |         |         |         |         |  (26)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |         |         |--SRC----|
          |         |         |         |  (16)    \-TABL[WORK].left_class opt=''
          |         |         |         |--empty-    (27)
          |         |         |         |  (17)               /-SYM-V-(l.name:1)
          |         |         |         |          /-ASC----|  (36)
          |         |         |          \-ORDR---|  (28)
          |         |         |ROM---|              (18)
          |         |         |4)    |                    /-SYM-V-(r.name:1 flag=0001)
          |         |         |         |          /-OBJ----|  (29)
          |         |         |         |         |  (19)    \-SYM-V-(r.Age:3 flag=0001)
          |         |          \-SORT---|              (30)
          |         |            (12)   |                    /-SYM-V-(r.name:1 flag=0001)
          |         |                   |          /-OBJ----|  (37)
          |         |                   |         |  (31)    \-SYM-V-(r.Age:3 flag=0001)
          |         |                   |--SRC----|
          |         |                   |  (20)    \-TABL[WORK].right_class opt=''
          |         |                   |--empty-    (32)
          |         |                   |  (21)               /-SYM-V-(r.name:1)
          |         |                   |          /-ASC----|  (38)
          |         |                    \-ORDR---|  (33)
          |         |--empty-                     (22)
          |         |  (5)     /-SYM-V-(l.name:1)
          |         --CEQ----|  (13)
          |           (6)     \-SYM-V-(r.name:1)
          |         -JTAG(jds=3, tagfrom=3, flags=0)
          |           (7)
          |         --empty-                 /-SYM-A-(#TEMA001:1 flag=0031)
          |         |  (8)     /-ASGN---|  (23)
          |         |         |  (14)    |          /-SYM-V-(l.name:1)
          |         |         |          \-FCOA---|  (34)
          |         |         |            (24)    \-SYM-V-(r.name:1)
          |         |         |
 -(1)-SSEL|          \(9)OBJE-|
```

Use FULL JOIN Phrase!

JTAG if ends in 1= left join, 2=right join 3=full join

L.Name is passed up, used & dropped after coalesce & CEQ

Merge join has notation of JOIN Or OTRJ

Merg joins can be used for inner join or outer joins

JTAG: if ends in 1= left join, 2=right join 3=full join

equality check info passed up

Coalesced Var.

```
Proc SQL _method _tree;  title "EX21B Illustrating a full join witn index on left table" ;
 create table hope as   select coalesce( l.name, r.name), l.sex, r.age
     From left_class as l full join right_class as r  on l.LindxName = r.name;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
```

Use FULL JOIN Phrase!

```
Tree as planned.                  /-SYM-A-(#TEMA001:1 flag=0035)
                      /-OBJ----|  (10)
                     |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                     |          \-SYM-V-(r.Age:3 flag=0001)
            /-OTRJ---|
           |  (2)    |                              /-SYM-V-(l.LIndxName:7 flag=0001)
           |         |                    /-OBJ----|  (23)
           |         |                   |  (15)   |--SYM-V-(l.name:1 flag=0001)
           |         |                   |          \-SYM-V-(l.Sex:2 flag=0001)
           |         |          /-SORT---|
           |         |         |  (11)   |                    /-SYM-V-(l.LIndxName:7 flag=0001)
           |         |         |         |          /-OBJ----|  (33)
           |         |         |         |         |  (24)   |--SYM-V-(l.name:1 flag=0001)
           |         |         |         |         |          \-SYM-V-(l.Sex:2 flag=0001)
           |         |         |         |--SRC----|
           |         |         |         |  (16)    \-TABL[WORK].left_class opt=''
           |         |         |         |-- empty-    (25)
           |         |         |         |  (17)              /-SYM-V-(l.LIndxName:7)
           |         |         |         |          /-ASC----|  (34)
           |         |         |          \-ORDR---|  (26)
           |         |--FROM---|             (18)
           |         |  (4)    |                    /-SYM-V-(r.name:1 flag=0001)
           |         |         |          /-OBJ----|  (27)
           |         |         |         |  (19)    \-SYM-V-(r.Age:3 flag=0001)
           |         |          \-SORT---|
           |         |             (12)  |                    /-SYM-V-(r.name:1 flag=0001)
           |         |                   |          /-OBJ----|  (35)
           |         |                   |         |  (28)    \-SYM-V-(r.Age:3 flag=0001)
           |         |                   |--SRC----|
           |         |                   |  (20)    \-TABL[WORK].right_class opt=''
           |         |                   |-- empty-    (29)
           |         |                   |  (21)              /-SYM-V-(r.name:1)
           |         |--empty-           |          /-ASC----|  (36)
           |         |  (5)               \-ORDR---|  (30)
           |         |          /-SYM-V-(l.LIndxName:7)
           |         |--CEQ----|   (13)
           |         |  (6)     \-SYM-V-(r.name:1)
           |         --JTAG(jds=3, tagfrom=3, flags=0)
           |         |  (7)
           |         |--empty-                  /-SYM-A-(#TEMA001:1 flag=0031)
           |         |  (8)     /-ASGN---|  (21)
           |         |         |  (14)   |          /-SYM-V-(l.name:1)
           |         |         |          \-FCOA---|  (31)
  (1)      |         \-OBJE(9)-|              (22)    \-SYM-V-(r.name:1)
 --SSEL---|
```

JTAG if ends in 1= left join, 2=right join 3=full join

Merge join has notation of JOIN Or OTRJ

Merg joins can be used for inner join or outer joins

JTAG: if ends in 1= left join, 2=right join 3=full join

equality check info passed up

Coalesced Var.

```
Proc SQL _method _tree; Title "EX21C Illustrating a full join w/ index on RIGHT table" ;
 create table hope as  select coalesce( l.name, r.name), l.sex, r.age
   From left_class as l full join right_class as r    on l.name = r.RIndxName;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations

Tree as planned.                /-SYM-A-(#TEMA001:1 flag=0035)
                     /-OBJ----|  (10)
                    |  (3)    |--SYM-V-(l.Sex:2 flag=0001)
                    |          \-SYM-V-(r.Age:3 flag=0001)
           /-OTRJ---|
          |  (2)    |                              /-SYM-V-(l.name:1 flag=0001)
          |         |                     /-OBJ----|  (25)
          |         |                    |  (15)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |          /-SORT---|
          |         |         |  (11)   |                     /-SYM-V-(l.name:1 flag=0001)
          |         |         |         |                    /-OBJ----|  (34)
          |         |         |         |                   |  (26)    \-SYM-V-(l.Sex:2 flag=0001)
          |         |         |         |--SRC----|
          |         |         |         |  (16)    \-TABL[WORK].left_class opt=''
          |         |         |         |--empty-    (27)
          |         |         |         |  (17)               /-SYM-V -(l.name:1)
          |         |         |         |          /-ASC----|  (35)
          |         |         |          \-ORDR---|  (28)
          |         |--FROM---|            (18)
          |         |  (4)    |                     /-SYM-V-(r.RIndxName:7 flag=0001)
          |         |         |          /-OBJ----|  (29)
          |         |         |         |  (19)   |--SYM-V-(r.name:1 flag=0001)
          |         |         |         |          \-SYM-V-(r.Age:3 flag=0001)
          |         |          \-SORT---|
          |         |            (12)   |                     /-SYM-V-(r.RIndxName:7 flag=0001)
          |         |                   |          /-OBJ----|  (36)
          |         |                   |         |  (30)   |--SYM-V-(r.name:1 flag=0001)
          |         |                   |         |          \-SYM-V-(r.Age:3 flag=0001)
          |         |                   |--SRC----|
          |         |                   |  (20)    \-TABL[WORK].right_class opt=''
          |         |                   |--empty-    (31)
          |         |                   |  (21)               /-SYM-V-(r.RIndxName:7)
          |         |                   |          /-ASC----|  (37)
          |         |                    \-ORDR---|  (32)
          |         |--empty-              (22)
          |         |  (5)     /-SYM-V-(l.name:1)
          |          --CEQ----|  (13)
          |           (6)     \-SYM-V-(r.RIndxName:7)
          |          -JTAG(jds=3, tagfrom=3, flags=0)
          |           (7)
          |          --empty-
          |         |  (8)     /-ASGN---|  (23)
          |         |         |  (14)   |          /-SYM-V-(l.name:1)
          |         |         |          \-FCOA---|  (33)
   (1)    |          \-OBJE---|            (24)    \-SYM-V-(r.name:1)
   --SSEL-|            (9)
```

Note sorting

JTAG: if ends in 1= left join, 2=right join 3=full join

equality check info passed up

Coalesced Var.

```
Proc SQL _method _tree; title "EX21D Illustrating a full join w/ index on BOTH tables" ;
 create table hope as    select coalesce( l.name, r.name), l.sex, r.age
    From left_class as l full join right_class as r    on l.LindxName = r.RIndxName;
Sqxcrta  (1) this indicates a selection of observations
   Sqxjm   (2) this indicates a sort-merge type of join
      Sqxsort (11) this indicates a SORT
         sqxsrc(WORK.LEFT_CLASS(alias=L)) (16) indicates a selection of observations
      sqxsort (12) this indicates a SORT
         sqxsrc(WORK.RIGHT_CLASS(alias=R)) (20) indicates a selection of observations
```

Use FULL JOIN Phrase!

```
Tree as planned.                 /-SYM-A-(#TEMA001:1 flag=0035)
                    /-OBJ----|   (10)
                   |   (3)    |--SYM-V-(l.Sex:2 flag=0001)
                   |           \-SYM-V-(r.Age:3 flag=0001)
          /-OTRJ---|
         |   (2)    |                            /-SYM-V-(l.LIndxName:7 flag=0001)
         |          |                   /-OBJ----|   (24)
         |          |                  |   (15)   |--SYM-V-(l.name:1 flag=0001)
         |          |                  |           \-SYM-V-(l.Sex:2 flag=0001)
         |          |          /-SORT---|
         |          |         |   (11)   |                    /-SYM-V-(l.LIndxName:7 flag=0001)
         |          |         |          |          /-OBJ----|    (32)
         |          |         |          |         |   (25)   |--SYM-V-(l.name:1 flag=0001)
         |          |         |          |         |           \-SYM-V-(l.Sex:2 flag=0001)
         |          |         |          |--SRC----|
         |          |         |          |   (16)    \-TABL[WORK].left_class opt=''
         |          |         |          |-- empty-    (26)
         |          |         |          |   (17)              /-SYM-V-(l.LIndxName:7)
         |          |         |          |          /-ASC----|   (32)
         |          |         |           \-ORDR---|   (27)
         |          |--FROM---|             (18)
         |           (4)     |                    /-SYM-V-(r.RIndxName:7 flag=0001)
         |          |         |          /-OBJ----|   (28)
         |          |         |         |   (19)   |--SYM-V-(r.name:1 flag=0001)
         |          |         |         |           \-SYM-V-(r.Age:3 flag=0001)
         |          |          \-SORT---|
         |          |             (12)  |                    /-SYM-V-(r.RIndxName:7 flag=0001)
         |          |                   |          /-OBJ----|   (33)
         |          |                   |         |   (29)   |--SYM-V-(r.name:1 flag=0001)
         |          |                   |         |           \-SYM-V-(r.Age:3 flag=0001)
         |          |                   |--SRC----|
         |          |                   |   (20)    \-TABL[WORK].right_class opt=''
         |          |                   |-- empty-    (30)
         |          |                   |   (21)              /-SYM-V-(r.RIndxName:7)
         |          |                   |          /-ASC----|   (33)
         |          |                    \-ORDR---|
         |          |--empty-                     (22)
         |          |   (5)     /-SYM-V-(l.LIndxName:7)
         |          |--CEQ----|   (13)
         |           (6)     \-SYM-V-(r.RIndxName:7)
         |          -JTAG(jds=3, tagfrom=3, flags=0)
         |           (7)
         |          -empty-                 /-SYM-A-(#TEMA001:1 flag=0031)
         |          |   (8)     /-ASGN---|   (22)
         |          |          |   (14)   |          /-SYM-V-(l.name:1)
         |          |          |           \-FCOA---|   (31)
         |          |          |             (23)    \-SYM-V-(r.name:1)
         |           \-OBJE---|
 --SSEL---|            (9)
   (1)
```

JTAG if ends in 1= left join, 2=right join 3=full join

Merge join has notation of JOIN Or OTRJ

Merg joins can be used for inner join or outer joins

JTAG: if ends in 1= left join, 2=right join 3=full join

equality check info passed up

Coalesced Var.

```
*******Example 22  *** CORRELATED QUERY  ******;
* showing how  CORREALTED QUERY is processed ;
proc sql _METHOD _TREE;   TITLE "EX25 A SIMPLE CORRELATED QUERY";
select *  from sashelp.class as Outer
 Where  Outer.AGE = (select Max(age) from sashelp.class as inner  where outer.sex=inner.sex);

NOTE: SQL execution methods chosen are:
  Sqxslct  (1) this indicates a selection of observations
      Sqxfil (2) this indicates the application of a ?????
          sqxsrc( SASHELP.CLASS(alias = OUTER) ) (20) indicates a selection of observations

NOTE: SQL subquery execution methods chosen are:
  Sqxsubq (1) this indicates a selection of observations
      Sqxsumn (6) this indicates summation without grouping--a summary of the whole table
          sqxsrc( SASHELP.CLASS(alias = INNER) ) (20) indicates a selection of observations
```

Simple Correlated Query

Subquery

Select * from outer subject to the condition that outer.age = the max age for that sex.

Pass to FIL

Fil is the late application of a predicate

Get this from sashelp.class

Outer.age is part of outer query & used in the equality check

Subquery causes the creation of a temp, indexed table

SUBP: stands for subroutine parameter

Create a variable, using Grouping (SYM-G), containing the max age (stat function=5)

SUBC is short for SUBroutine Call. Printed out as a summary of the process

Pretend that what is inside the paren, in the query, is a function and we are passing , to the function, a paramater- we pass outer.sex from the outer query to the function.

```
Tree as planned.                 /-SYM-V-(Outer.Name:1 flag=0001)
                      /-OBJ----|   (6)
                     |   (3)    |--SYM-V-(Outer.Sex:2 flag=0001)
                     |          |--SYM-V-(Outer.Age:3 flag=0001)
                     |          |--SYM-V-(Outer.Height:4 flag=0001)
                     |          \-SYM-V-(Outer.Weight:5 flag=0001)
            /-FIL----|
           |   (2)    |                    /-SYM-V-(Outer.Age:3 flag=0001)
           |          |          /-OBJ----| (11)
           |          |         |   (7)    |--SYM-V-(Outer.Sex:2 flag=0001)
           |          |         |          |--SYM-V-(Outer.Name:1 flag=0001)
           |          |         |          |--SYM-V-(Outer.Height:4 flag=0001)
           |          |         |          \-SYM-V-(Outer.Weight:5 flag=0001)
           |          |-SRC----|
           |          |  (4)     \-TABL[SASHELP].class opt=''
           |          |            (8)
           |          |           /-SYM-V-(Outer.Age:3)
           |          |-CEQ----|   (9)
           |          |  (5)    |                    /-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0) flag=0001)
           |          |         |          /-OBJ----|   (19)
           |          |         |         /-AGGR---|   (14)
           |          |         |         |   (12)   |
           |          |         |         |          |                    /-SYM-V -(inner.Age:3 flag=0001)
           |          |         |         |          |          /-OBJ----|    (25)
           |          |         |         |          |--SRC----|   (20)
           |          |         |         |          |   (15)   |--TABL[SASHELP].class opt=''
           |          |         |         |          |         |-- empty-
           |          |         |         |          |         |   (21)    /-SUBP(1)
           |          |         |         |          |          \-CEQ----|   (26)
           |          |         |         |          |            (22)    \-NAME--(Sex:2)
           |          |         |         |          |-- empty-              (27)
           |          |         |         |          |-- empty-
           |          |         |         |          |-- empty-
           |          |         |         |          |-- empty-
           |          |         |         |          |-- empty-
           |          |         |         |          |-- empty-
           |          |         |         |          |   (16)    /-SYM-G-(#TEMG001:1 stat=5,0 from Age(0,0))
           |          |         |         |          |--TLST---|   (23)
           |          |         |         |          |   (17)    /-SYM-S-(Age:2 ss=0008x)
           |          |         |         |          |    \-SLST---|   (24)
           |          |         |          \-SUBC---|             (18)
           |          |         |            (10)    \-SYM-V-(Outer.Sex:2)
           |          |         |                      (13)
 --SSEL---|          |         |
    (1)
```

```
*******Example 23  *** calculated  ******;

15   proc sql _method _tree;
16   select name, age*12 as age_mo
17   from sashelp.class
18   where calculated age_mo LE 144;

NOTE: SQL execution methods chosen are:

      sqxslct
          sqxfil
              sqxsrc( SASHELP.CLASS )

Tree as planned.
                              /-SYM-V-(class.Name:1 flag=0001)
                     /-OBJ----|  (6)
                    |   (3)    \-SYM-A-(age_mo:1 flag=0031)
           /-FIL----|
          |  (2)    |                    /-SYM-V-(class.Age:3 flag=0001)
          |         |          /-OBJ----|    (11)
          |         |         |  (7)     \-SYM-V-(class.Name:1 flag=0001)
          |         |--SRC----|
          |         |   (4)    \-TABL[SASHELP].class opt=''
          |         |          /-SYM-A-(age_mo:1 flag=0070)
          |         |--CLE----|  (8)
          |         |   (5)    \-LITN(144)
          |         |--empty-    (9)
          |         |--empty-
          |         |                    /-SYM-A-(age_mo:1 flag=0070)
          |         |          /-ASGN---|   (12)
          |         |         |  (10)   |          /-SYM-V-(class.Age:3)
          |         |         |         \-AMUL---|   (14)
          |         |         |            (13)   \-LITN(12)
          |          \-ERLY---|
 --SSEL---|            (6)
    (1)
```

Call FIL subroutine in SQL (2) because the variable age_mo was not available to the data engine. It must be created by SQL and then logic can be applied.

"ERLY" is short for "early list."

It contains a list of expressions that must be evaluated before any other expressions on a step.

On that list the first one must be evaluated before the second one (if any), the second one must be evaluated before the third one (if any), etc.

For example, "age_mo" must be calculated (it's on the early list) BEFORE its value can be referenced in the "age_mo <= 144" expression.