Indexing SAS® Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling

Transcript

Indexing SAS® Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling Transcript was developed by Mark Stranieri. Additional contributions were made by David Ghan, Cynthia Johnson, Mark Jordan, and Warren Repole. Editing and production support was provided by the Curriculum Development and Support Department.

Indexing SAS® Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling Transcript

Book code E1447, course code RLSPID1, prepared date 13Apr2009. RLSPID1_001

ISBN 987-1-60764-122-3

Table of Contents

Lecture Description

This SAS e-Lecture shows how to use indexes to improve the performance of processing SQL WHERE clauses.

To learn more…

For information on other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the Web at support.sas.com/training/ as well as in the Training Course Catalog.

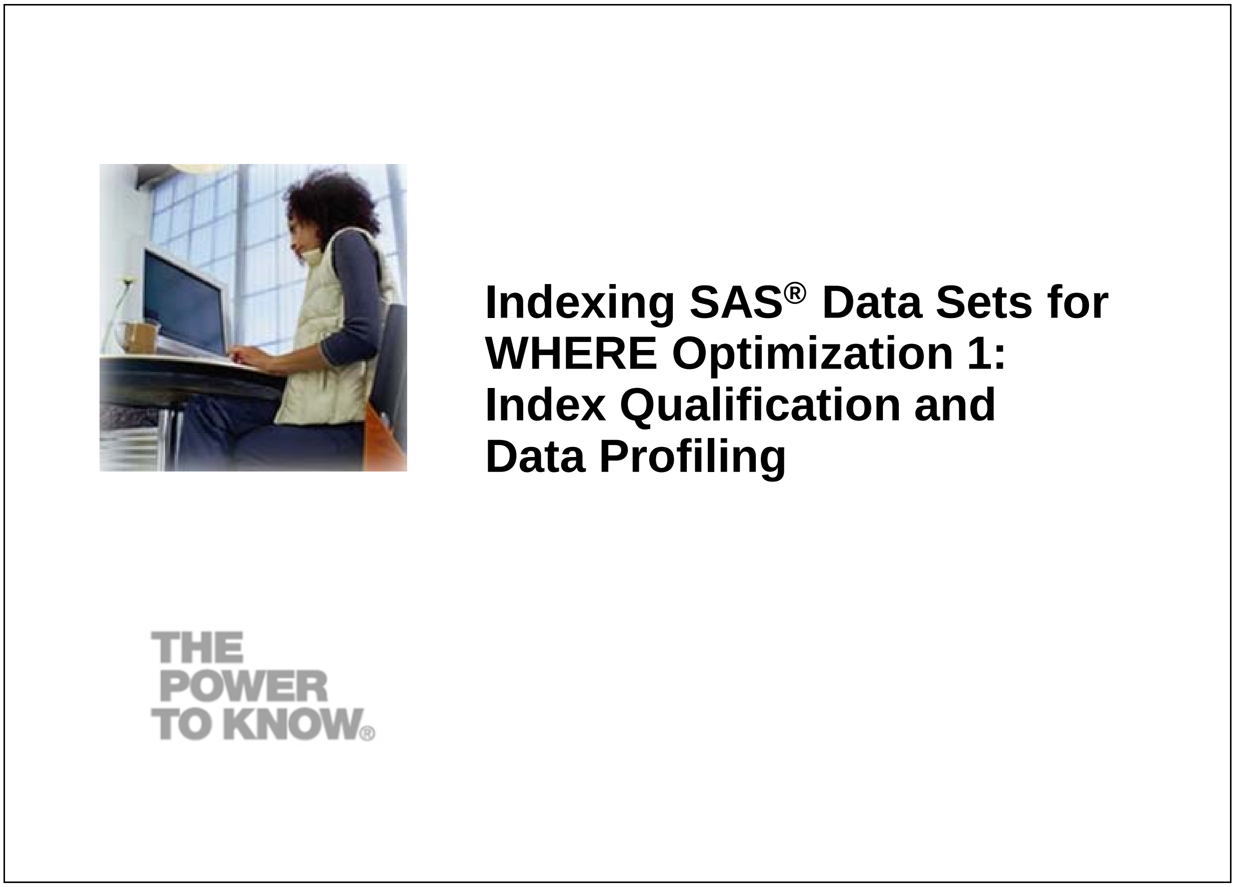
For a list of other SAS books that relate to the topics covered in this Course Notes, USA customers can contact our SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the Publications Catalog on the Web at support.sas.com/pubs for a complete list of books and a convenient order form.

Prerequisites

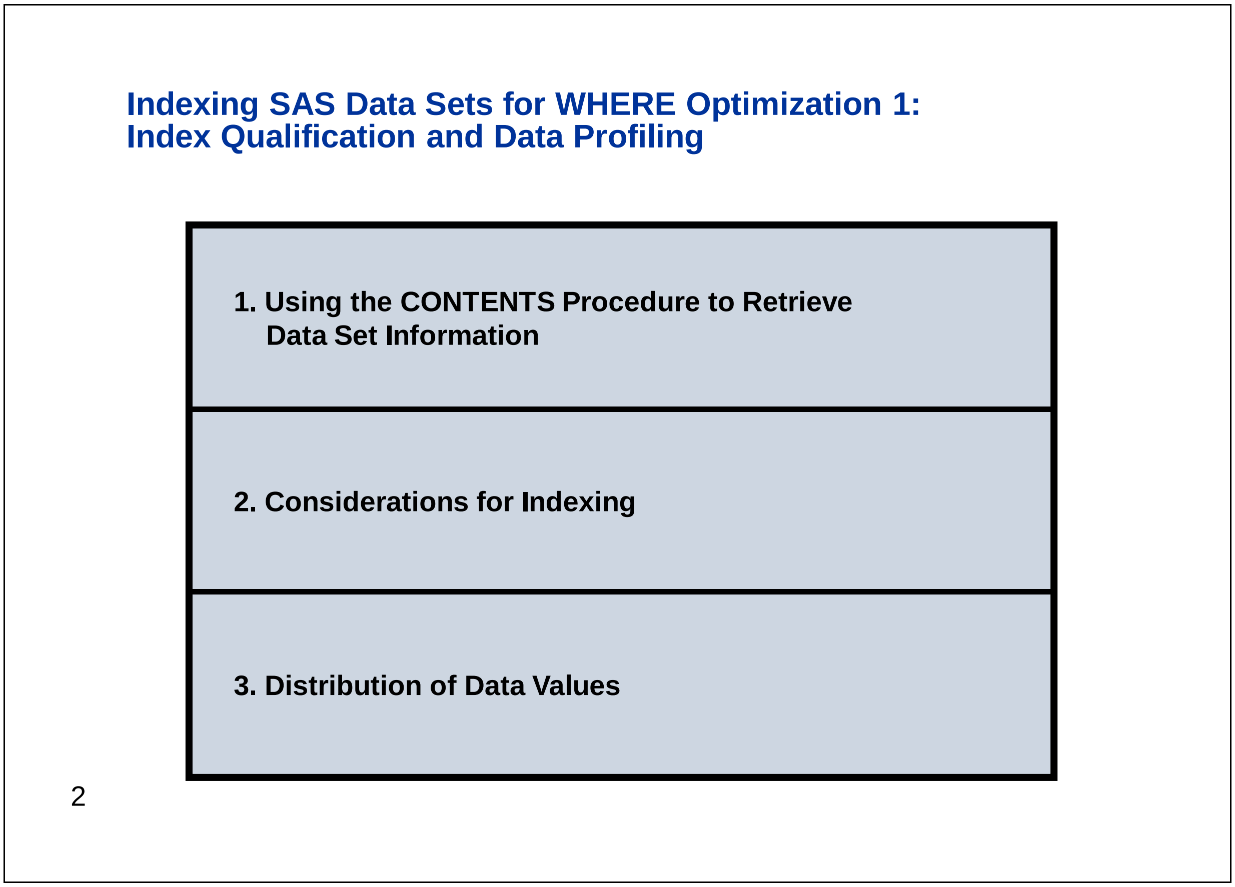
Before listening to this lecture, you should have completed the SAS® Programming 1: Essentials course or have equivalent knowledge. Completion of the SAS® Programming 2: Manipulating Data with the DATA Step course would be helpful.

Indexing SAS® Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling



Indexing SAS® Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling

Welcome to this SAS e-Lecture, *Indexing SAS® Data Sets for WHERE Optimization*. My name is Mark, and I am an instructor for SAS. Today we will be using indexes to improve the performance of processing SQL WHERE clauses.



**Indexing SAS Data Sets for WHERE Optimization 1:
Index Qualification and Data Profiling**

1. Using the CONTENTS Procedure to Retrieve Data Set Information
2. Considerations for Indexing
3. Distribution of Data Values

2

This e-lecture will cover three topics concerning indexing.

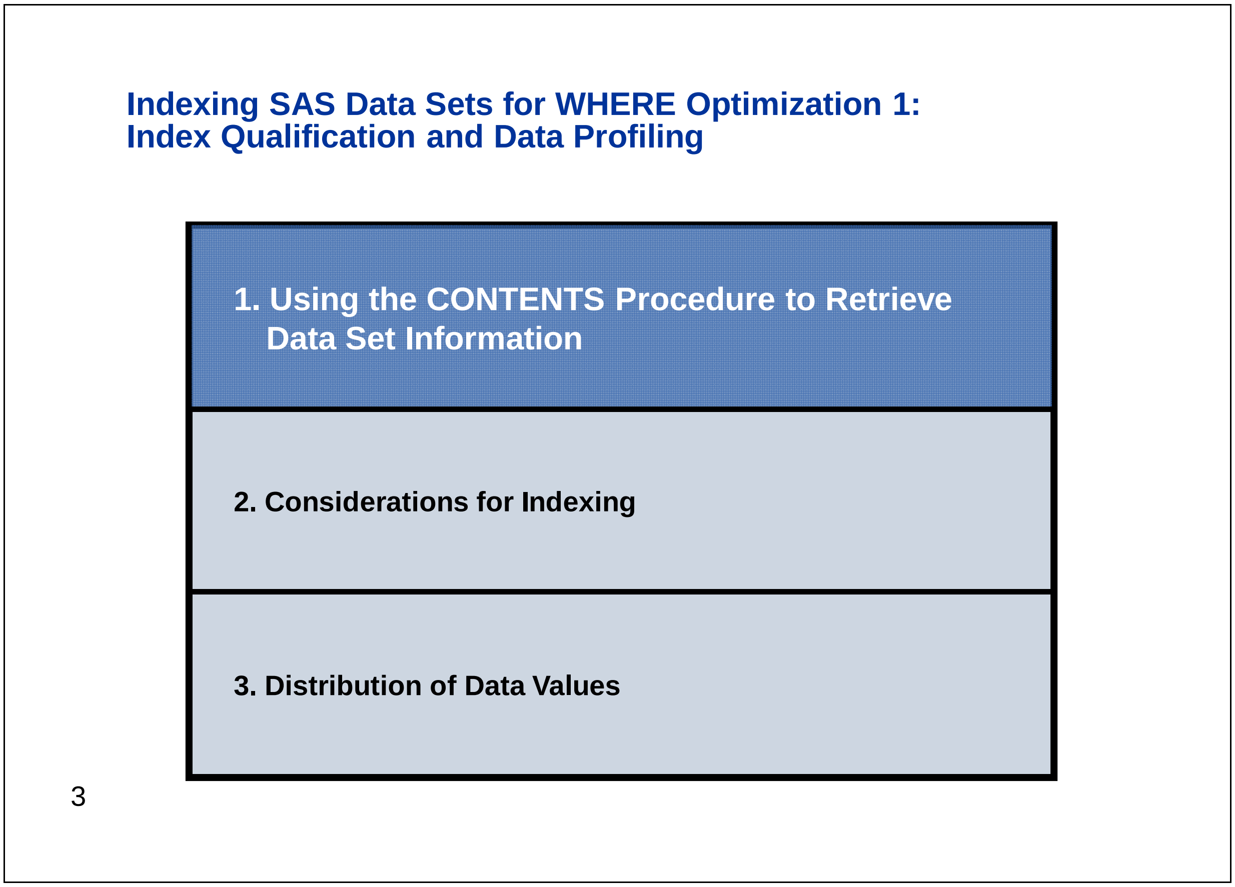
First, we will use the CONTENTS procedure, or PROC CONTENTS, to retrieve information about data set size and indexes available.

Next, we’ll review factors to consider when creating an index.

And third, we’ll analyze data distributions and determine which ones work well with indexing.

Please note that the traditional SAS terms of data set, observation, and variable can be used interchangeably with the SQL terms of table, row, and column, respectively.

1. Using the CONTENTS Procedure to Retrieve Data Set Information



Indexing SAS Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling

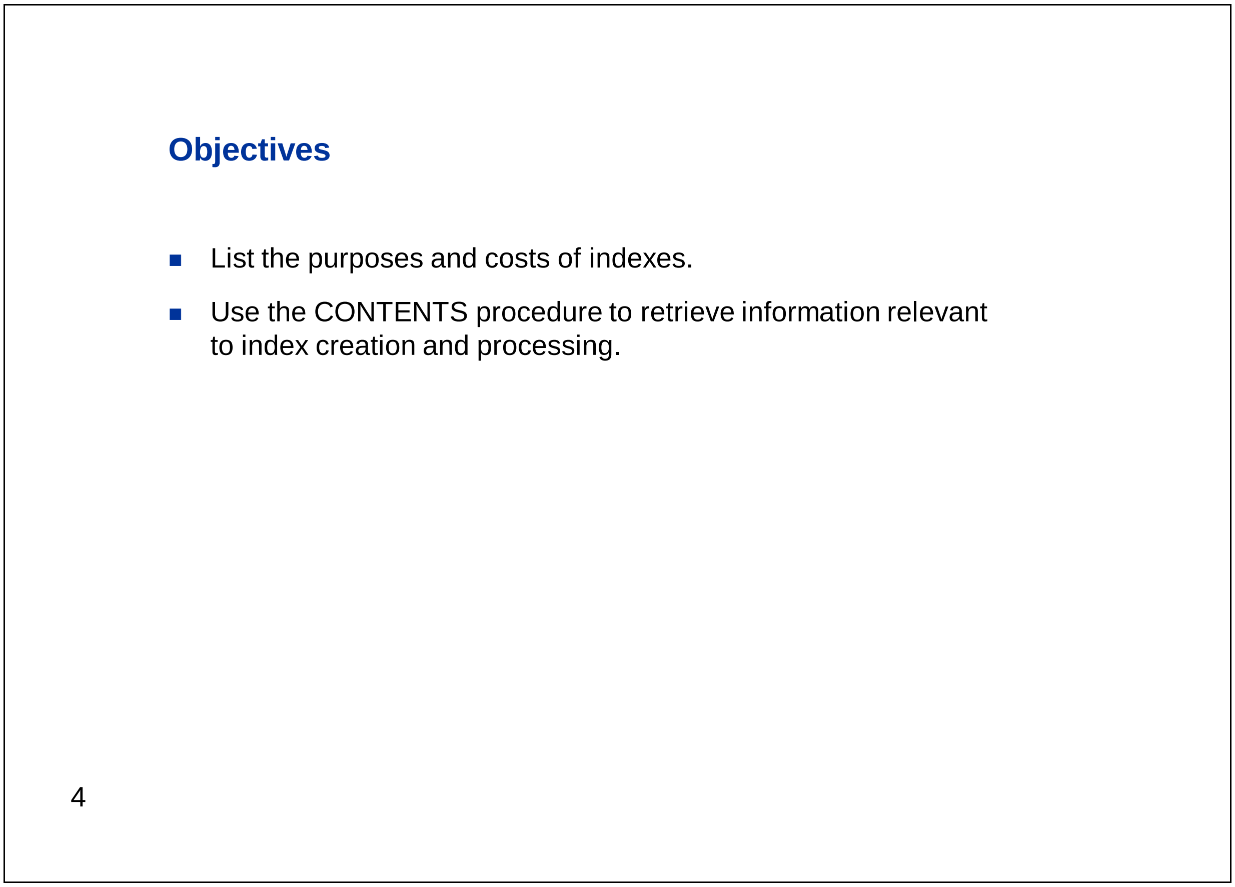
1. Using the CONTENTS Procedure to Retrieve Data Set Information

2. Considerations for Indexing

3. Distribution of Data Values

3

We will use PROC CONTENTS to retrieve data set information regarding space used by data sets and indexes.



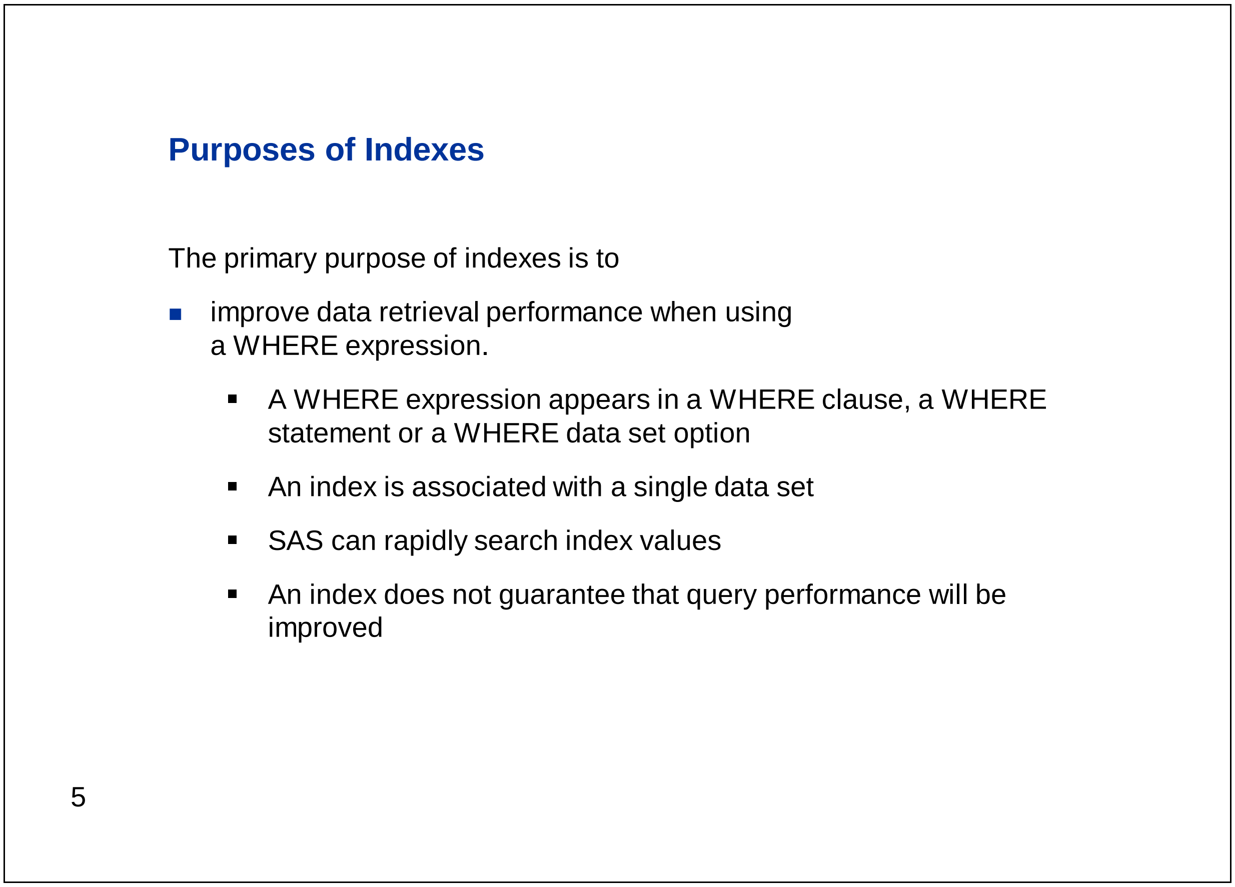
Objectives

- List the purposes and costs of indexes.
- Use the CONTENTS procedure to retrieve information relevant to index creation and processing.

4

In this section we’ll review the purposes and costs of indexes.

Our second objective is to use PROC CONTENTS to retrieve important information related to index creation, use, and justification.



Purposes of Indexes

The primary purpose of indexes is to

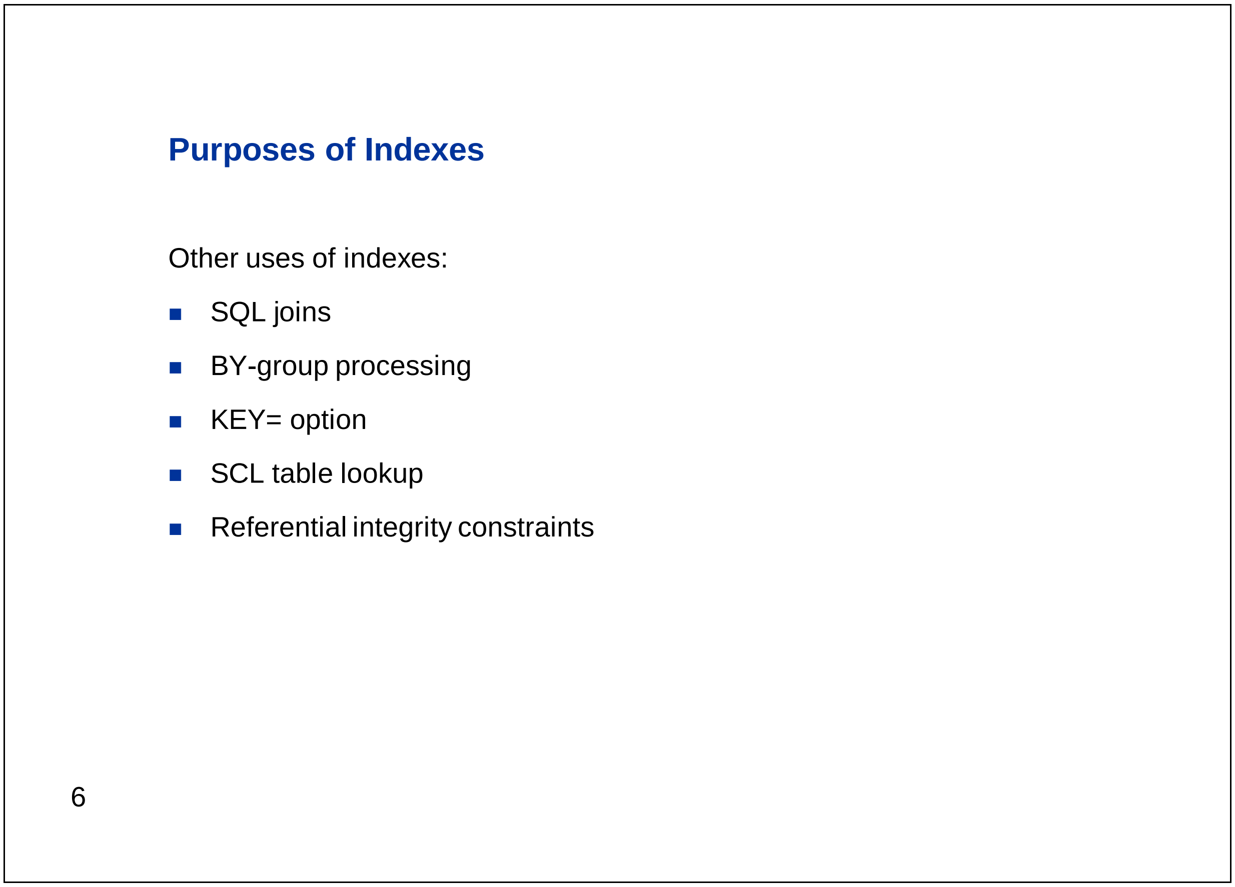
- improve data retrieval performance when using a WHERE expression.
  - A WHERE expression appears in a WHERE clause, a WHERE statement or a WHERE data set option
  - An index is associated with a single data set
  - SAS can rapidly search index values
  - An index does not guarantee that query performance will be improved

5

The primary purpose of an index is to improve the performance of retrieving subsets of data when using a WHERE expression within a WHERE clause, a WHERE statement, or a WHERE data set option.

An index is associated with a single data set, and is built using the sorted values from one or more variables of that data set. SAS can rapidly search index values specified in a WHERE expression and then directly access observations that contain those values in the data set. Without an index, SAS must search sequentially through the observations in the data set.

However, simply creating an index does not guarantee that query performance will be improved. There are times when it’s faster for SAS to retrieve data without using the index, in which case SAS will sequentially access the observations in the data set, effectively bypassing the index.



Purposes of Indexes

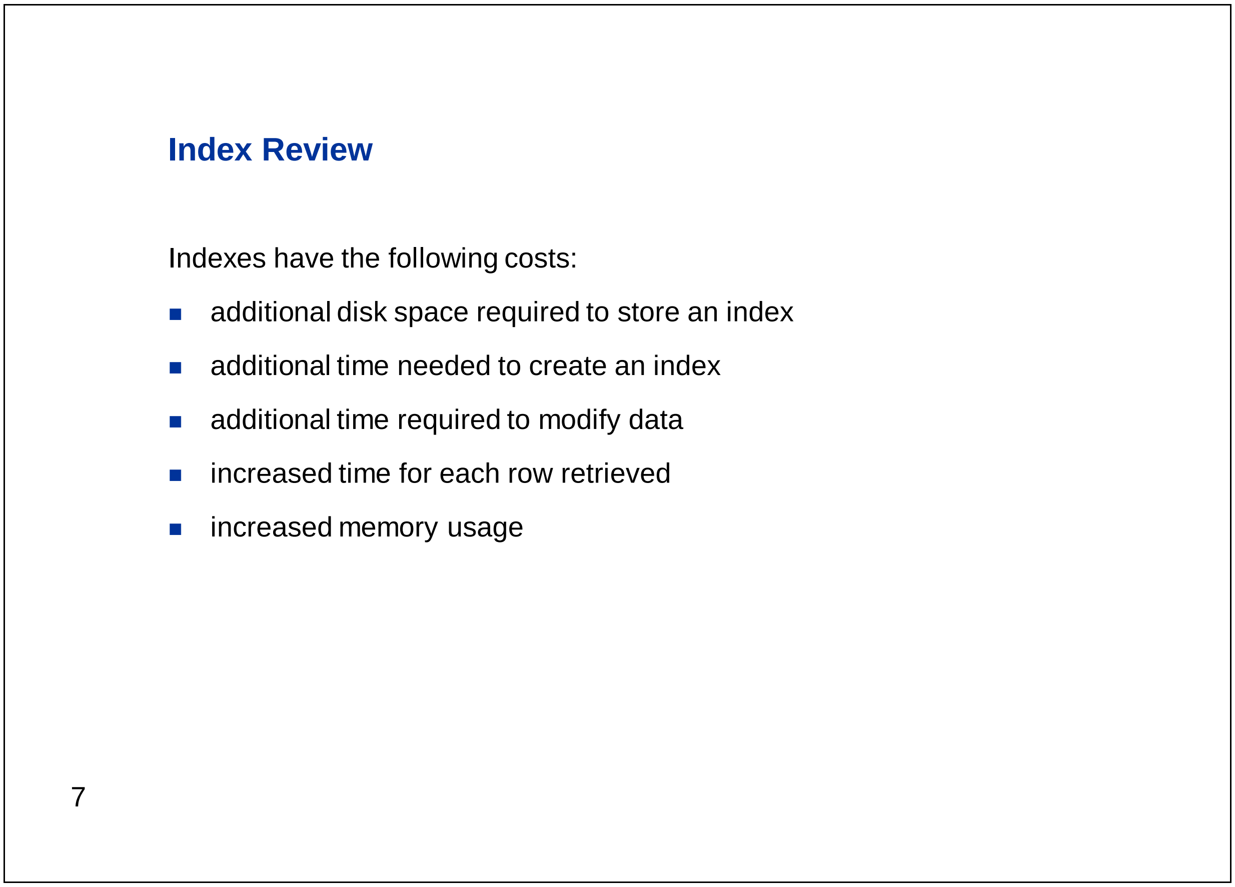
Other uses of indexes:

- SQL joins
- BY-group processing
- KEY= option
- SCL table lookup
- Referential integrity constraints

6

This lecture addresses using indexes for WHERE optimization. However, it’s important to note that there are many other uses for indexes that will not be covered here. Some of these uses include

- the SQL join optimization
- BY-group processing in the DATA step and procedures that require data to be pre-sorted
- the KEY= option in the SET and MODIFY statements
- SAS Component Language (SCL) table lookup
- referential integrity constraints.



Index Review

Indexes have the following costs:

- additional disk space required to store an index
- additional time needed to create an index
- additional time required to modify data
- increased time for each row retrieved
- increased memory usage

7

Even when indexes do improve the performance of data retrieval, it’s important to be aware of the possible costs of indexes.

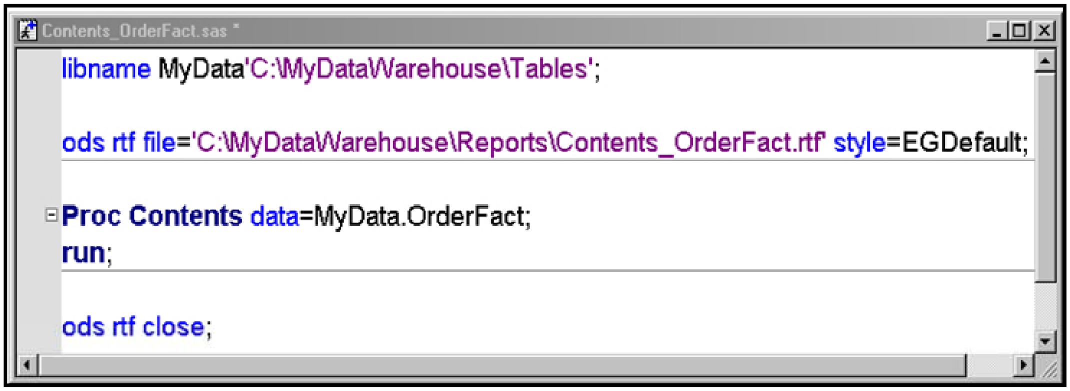
For example, indexes take additional time to create, and they use additional disk space. So if a large data set is being re-created periodically, an associated index will also need to be re-created and will use these additional resources.

Further, whenever a data set is modified, any associated indexes must be maintained by SAS. So if an observation is added, updated, or deleted from a data set that has an index, SAS must also add, update, or delete data from the index.

There is also some overhead associated with using an index to retrieve data. This overhead makes accessing an individual record slightly slower, but this is typically offset by the fact that only required observations are being accessed. But this is something to consider when retrieving large subsets of data.

Because of the costs associated with indexes, it’s important to investigate your data to be sure that a data set will show an increase in query performance if an index is created on that data set.

Displaying Data Set Attributes



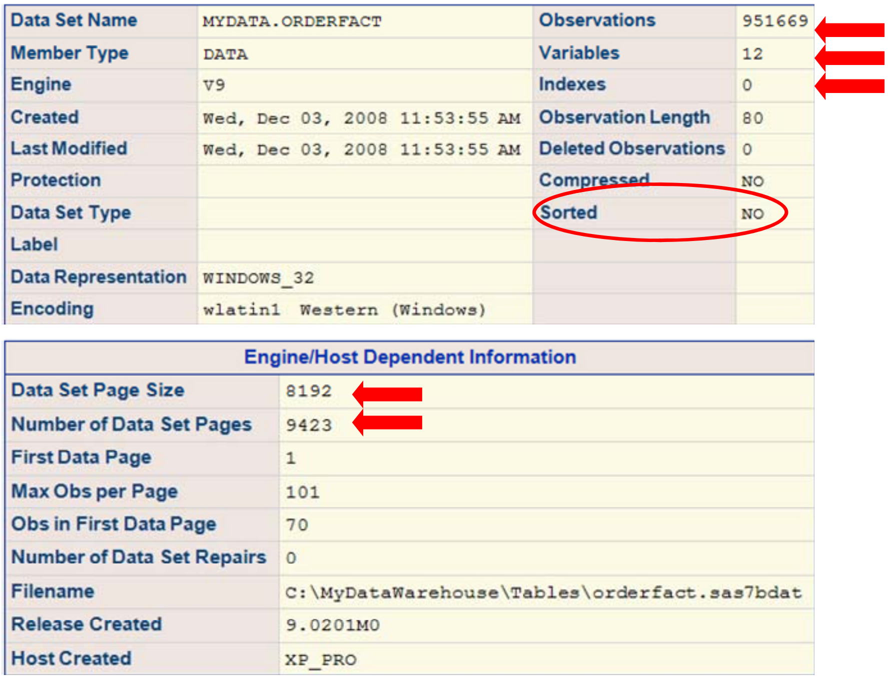
```
libname MyData'C:\MyDataWarehouse\Tables';

ods rtf file='C:\MyDataWarehouse\Reports\Contents_OrderFact.rtf' style=EGDefault;

Proc Contents data=MyData.OrderFact;
run;

ods rtf close;
```

8

We’ll begin investigating our data sets by using PROC CONTENTS. The data set we’re interested in is called **`OrderFact`** and is located in the Windows folder C:\MyDataWarehouse\Tables.

This SAS program establishes a SAS libref to the Windows folder and uses PROC CONTENTS to display the descriptor portion of the SAS data set. The two ODS statements above and below PROC CONTENTS are used to reformat the output into Rich Text Format (or RTF).

Displaying Data Set Attributes



Data Set Name	MYDATA.ORDERFACT	Observations	951669
Member Type	DATA	Variables	12
Engine	V9	Indexes	0
Created	Wed, Dec 03, 2008 11:53:55 AM	Observation Length	80
Last Modified	Wed, Dec 03, 2008 11:53:55 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	8192
Number of Data Set Pages	9423
First Data Page	1
Max Obs per Page	101
Obs in First Data Page	70
Number of Data Set Repairs	0
Filename	C:\MyDataWarehouse\Tables\orderfact.sas7bdat
Release Created	9.0201M0
Host Created	XP_PRO

9

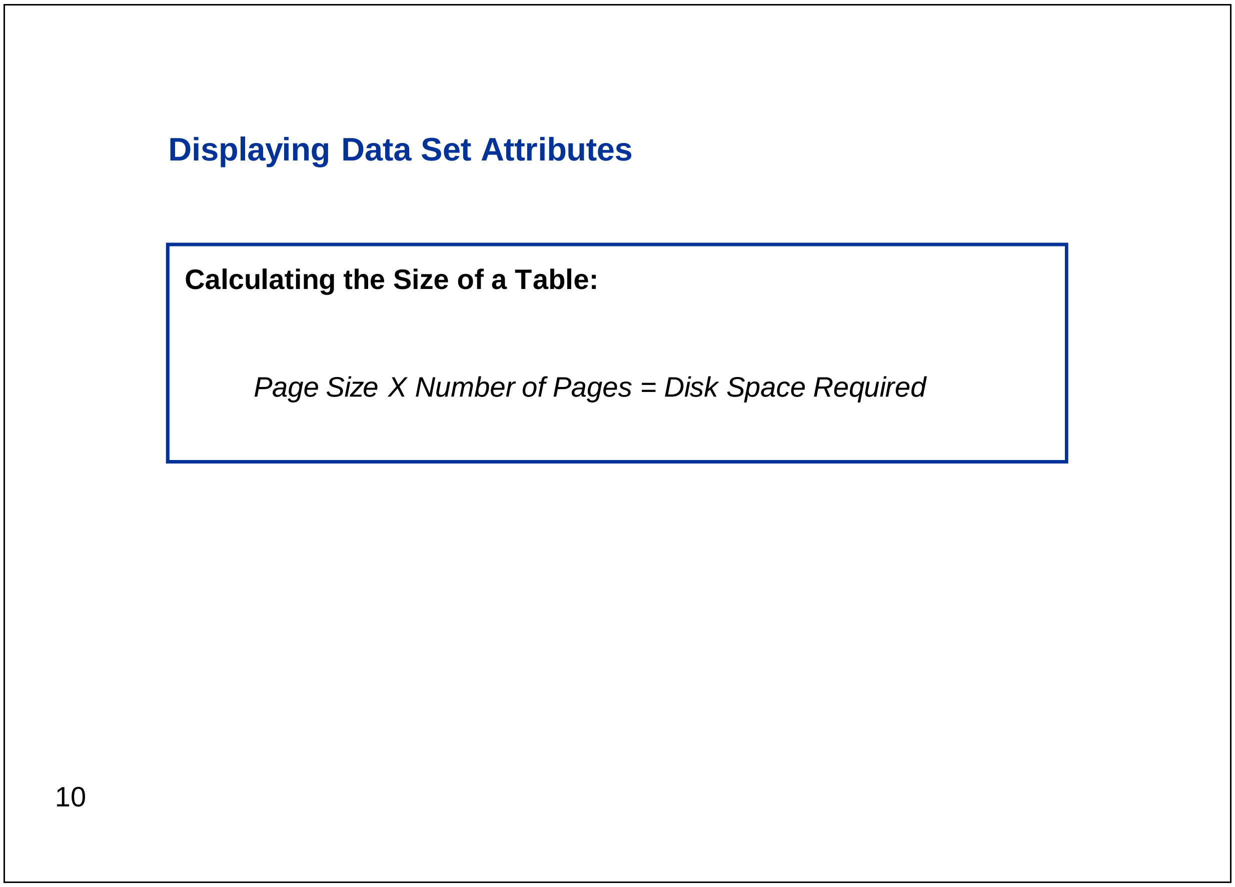
PROC CONTENTS returns information about the attributes of the data set. Among other things, we can see that this data set has 951,669 observations, 12 variables, and 0 indexes.

PROC CONTENTS also reports the size and number of pages used by the data set. A page is the amount of data (or the number of observations) that can be buffered and transferred between memory and disk space. (This is also known as an I/O request.)

For a small data set, retrieving rows sequentially may be faster than using an index. So a best practice is to avoid creating indexes on data sets that use fewer than three pages.

As a final note, notice that PROC CONTENTS tells us if a data set has been sorted. In this example the data is not sorted. If it had been, the procedure would display the names of the sort-by variables.

Index performance can be improved if the data is sorted on a variable prior to creating an index on that variable. We'll discuss this further in the next section.



Displaying Data Set Attributes

Calculating the Size of a Table:

Page Size X Number of Pages = Disk Space Required

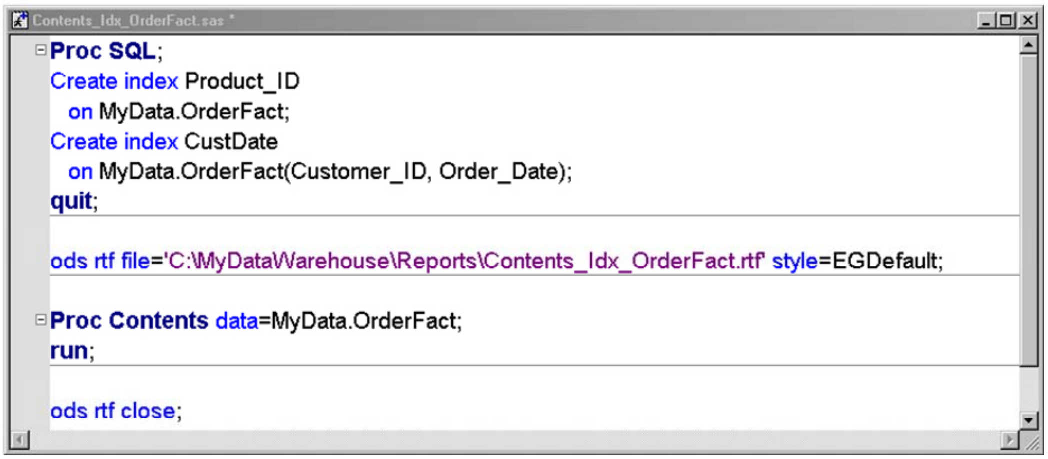
10

You can calculate the size of a table by multiplying the page size by the number of pages. In this case,

8192 X 9423 = 77,193,216 bytes, or about 74 mb

In order to demonstrate that extra space is needed for indexes, let’s create a couple of indexes on this table.

Displaying Data Set Attributes



```
Proc SQL;
Create index Product_ID
  on MyData.OrderFact;
Create index CustDate
  on MyData.OrderFact(Customer_ID, Order_Date);
quit;

ods rtf file='C:\MyDataWarehouse\Reports\Contents_Idx_OrderFact.rtf' style=EGDefault;

Proc Contents data=MyData.OrderFact;
run;

ods rtf close;
```

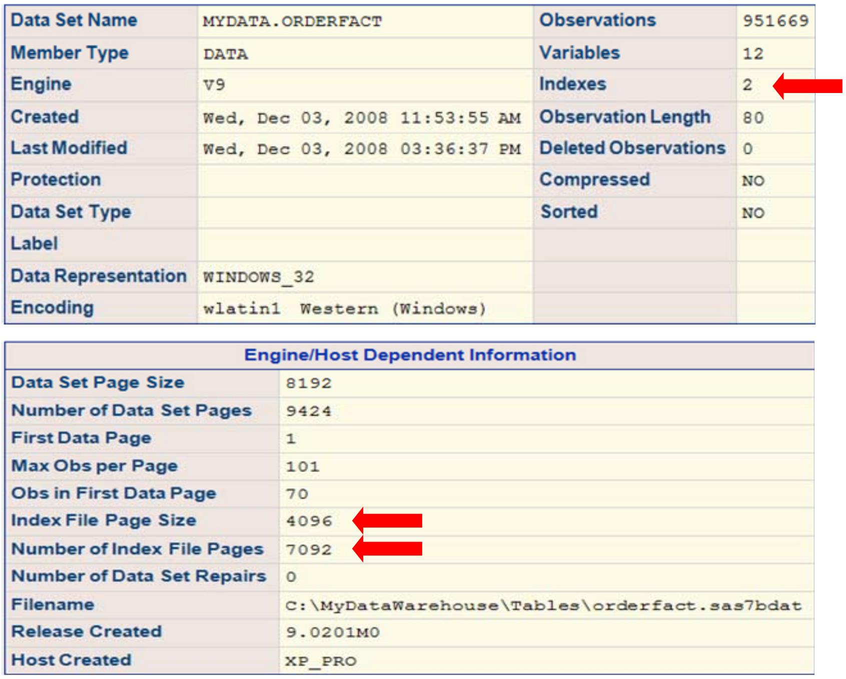
11

Here we use the SQL procedure to create two indexes on the **OrderFact** table. The first one is a simple index called Product_ID on the column named **Product_ID**. The second one is a composite index called CustDate on the columns **Customer_ID** and **Order_Date**.

Note that the column name does not have to be specified in parentheses after the table name for simple indexes.

After the indexes are created, we run the same PROC CONTENTS that was used in the previous example.

Displaying Data Set Attributes



Data Set Name	MYDATA.ORDERFACT	Observations	951669
Member Type	DATA	Variables	12
Engine	V9	Indexes	2
Created	Wed, Dec 03, 2008 11:53:55 AM	Observation Length	80
Last Modified	Wed, Dec 03, 2008 03:36:37 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	8192
Number of Data Set Pages	9424
First Data Page	1
Max Obs per Page	101
Obs in First Data Page	70
Index File Page Size	4096
Number of Index File Pages	7092
Number of Data Set Repairs	0
Filename	C:\MyDataWarehouse\Tables\orderfact.sas7bdat
Release Created	9.0201M0
Host Created	XP_PRO

12

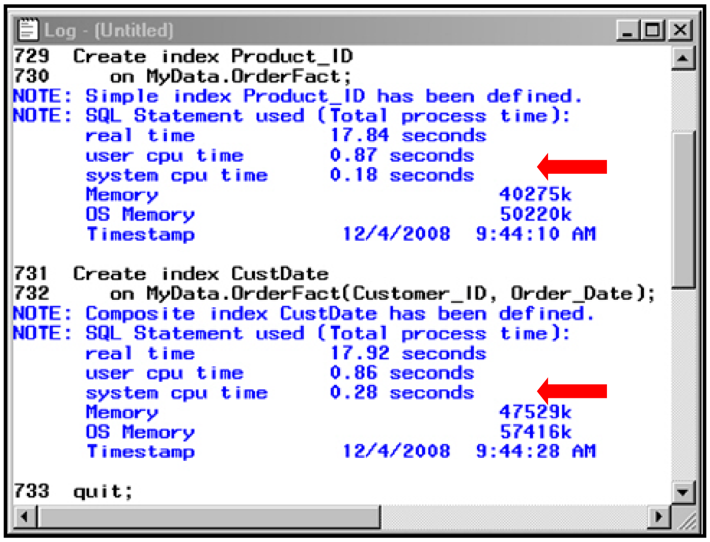
Notice that the number of rows and columns hasn’t changed and that the page size and number of pages for the data set are also unchanged. However, we can see that there are now two indexes, and they have a page size of 4096 and are using 7092 pages. We can calculate the size of the indexes by again multiplying page size by the number of pages:

4096 X 7092 = 29,048,832 bytes, or about 28 mb

So now, in addition to the 74 mb being used by the data table, we are using another 28 mb by adding two indexes.

If you find it necessary to estimate the size of an index before you create the index, SAS Technical Support can provide a program that estimates index size.

Displaying Data Set Attributes

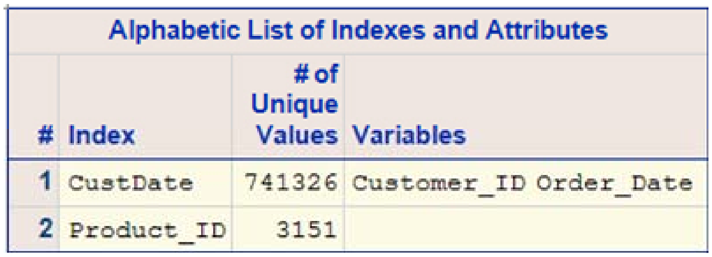


```
729  Create index Product_ID
730     on MyData.OrderFact;
NOTE: Simple index Product_ID has been defined.
NOTE: SQL Statement used (Total process time):
      real time           17.84 seconds
      user cpu time       0.87 seconds
      system cpu time     0.18 seconds
      Memory                            40275k
      OS Memory                         50220k
      Timestamp            12/4/2008  9:44:10 AM

731  Create index CustDate
732     on MyData.OrderFact(Customer_ID, Order_Date);
NOTE: Composite index CustDate has been defined.
NOTE: SQL Statement used (Total process time):
      real time           17.92 seconds
      user cpu time       0.86 seconds
      system cpu time     0.28 seconds
      Memory                            47529k
      OS Memory                         57416k
      Timestamp            12/4/2008  9:44:28 AM

733  quit;
```

13

This is a partial log of the PROC SQL statements that were submitted. The SQL STIMER option was turned on, which causes timing statistics to display for each SQL statement within the procedure. As you can see, creating indexes adds to the total time it takes to re-create a table. This additional time might be an important consideration for large tables that are refreshed periodically.

Displaying Data Set Attributes



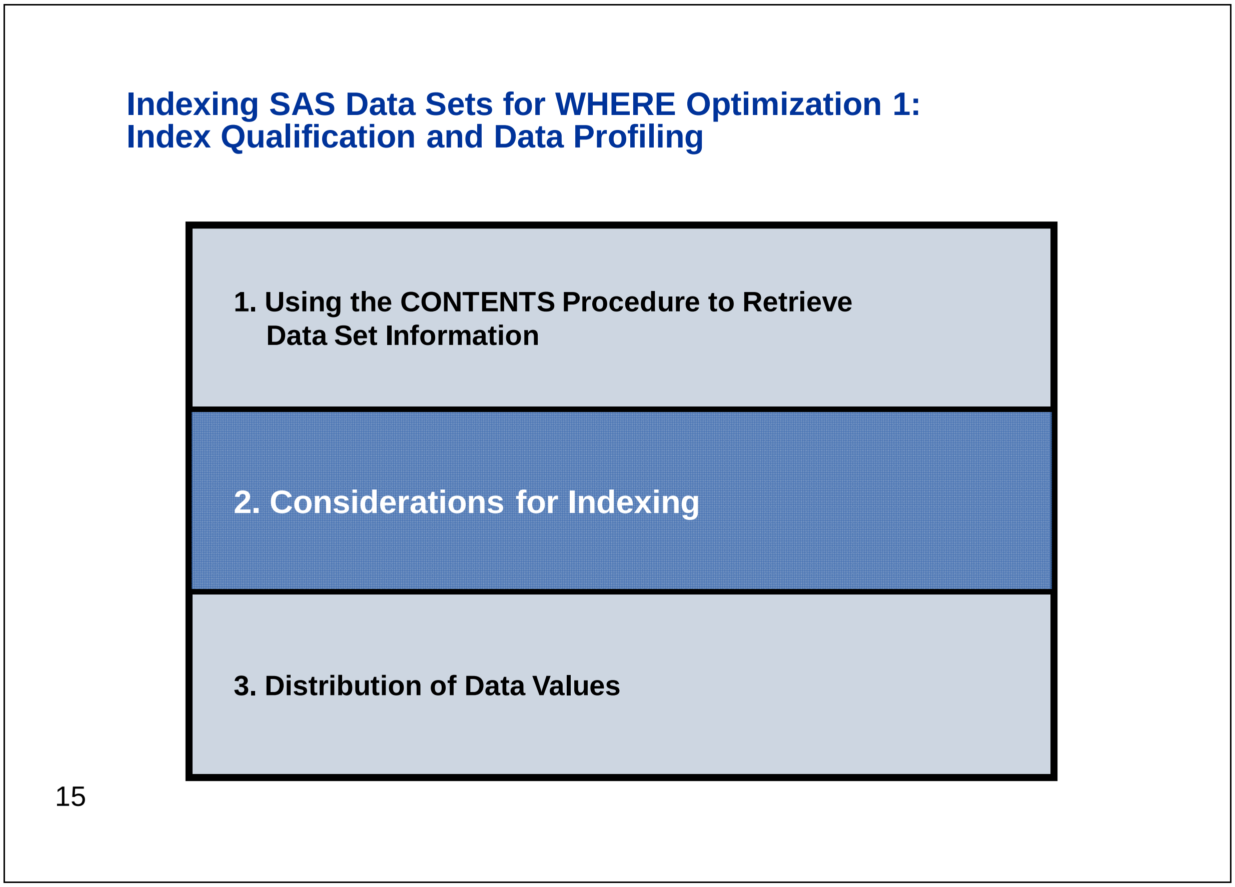
Alphabetic List of Indexes and Attributes			
#	Index	# of Unique Values	Variables
1	CustDate	741326	Customer_ID Order_Date
2	Product_ID	3151	

14

PROC CONTENTS also shows us index names, the number of unique values, and the columns contained in the index. We will use this information later to determine the size of the subset. For example, if we request a report for a single **`Product_ID`**, the average number of rows that will be returned is the number of unique values divided by the total number of rows, or 3151/951669, which is about 0.3% of the rows.

You can also use the CENTILES option in PROC CONTENTS to display information about the internal statistics maintained by indexes. This is discussed in a later section.

2. Considerations for Indexing



Indexing SAS Data Sets for WHERE Optimization 1:
Index Qualification and Data Profiling

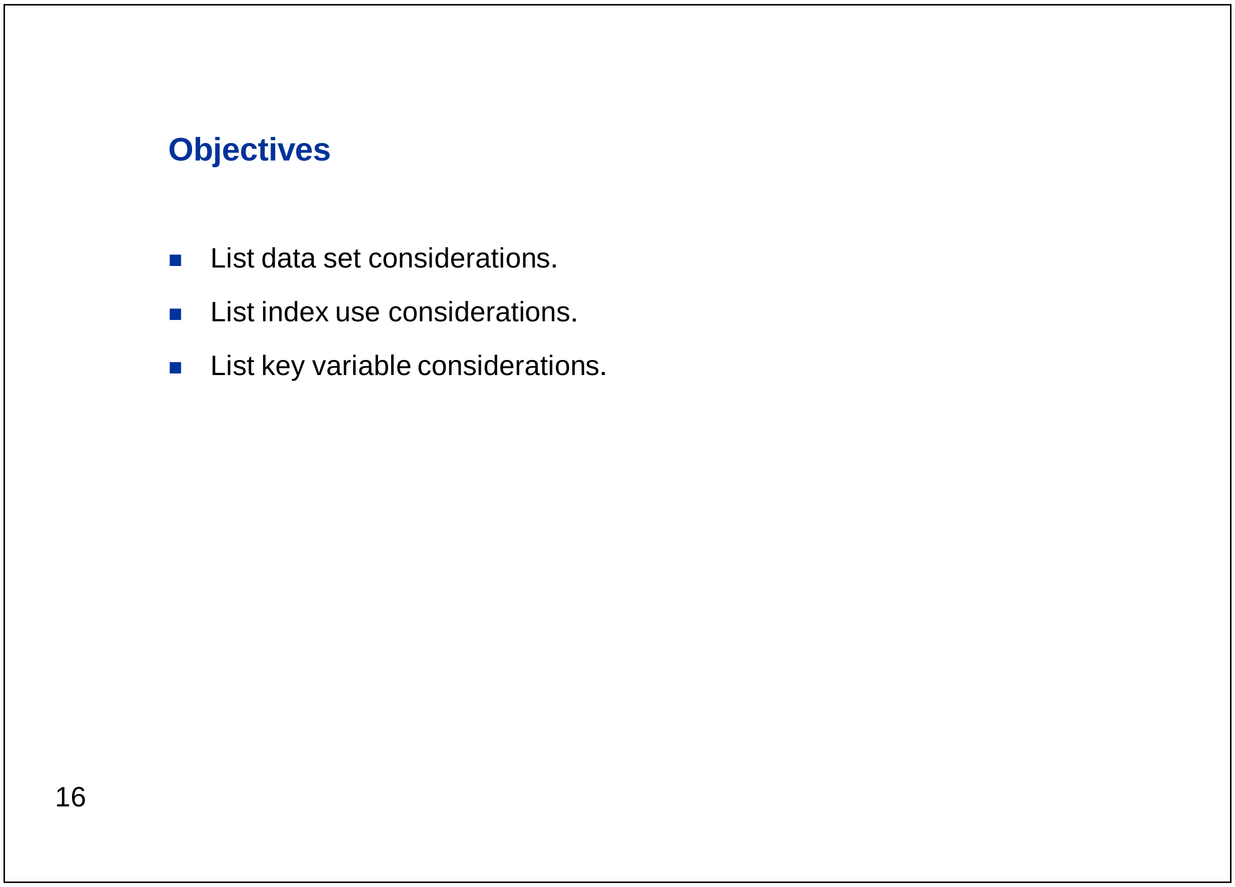
1. Using the CONTENTS Procedure to Retrieve Data Set Information

2. Considerations for Indexing

3. Distribution of Data Values

15

In this section we'll look at methods that can be used to justify adding an index to a data table.

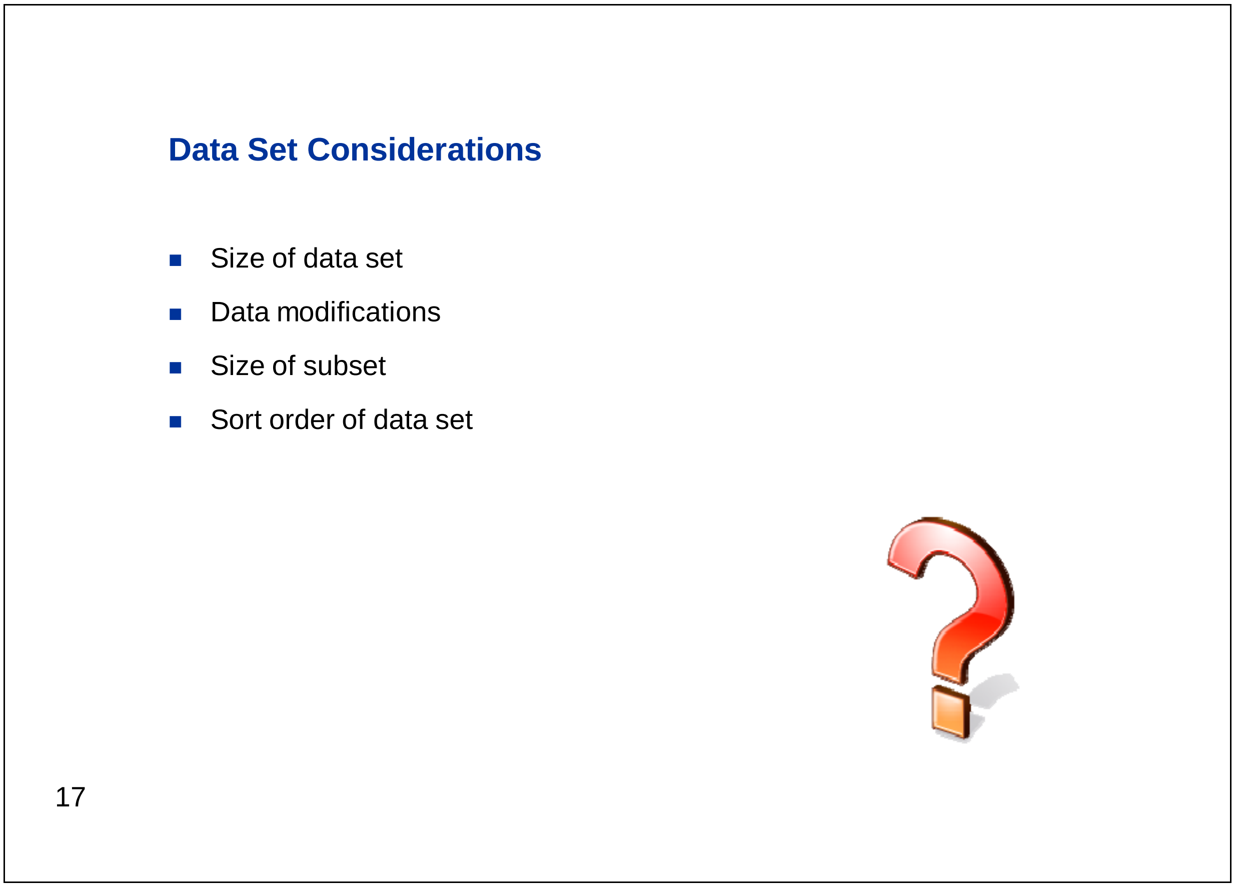


Objectives

- List data set considerations.
- List index use considerations.
- List key variable considerations.

16

In this section you’ll learn about considerations relevant to data sets, index use, and key variables.



Data Set Considerations

- Size of data set
- Data modifications
- Size of subset
- Sort order of data set

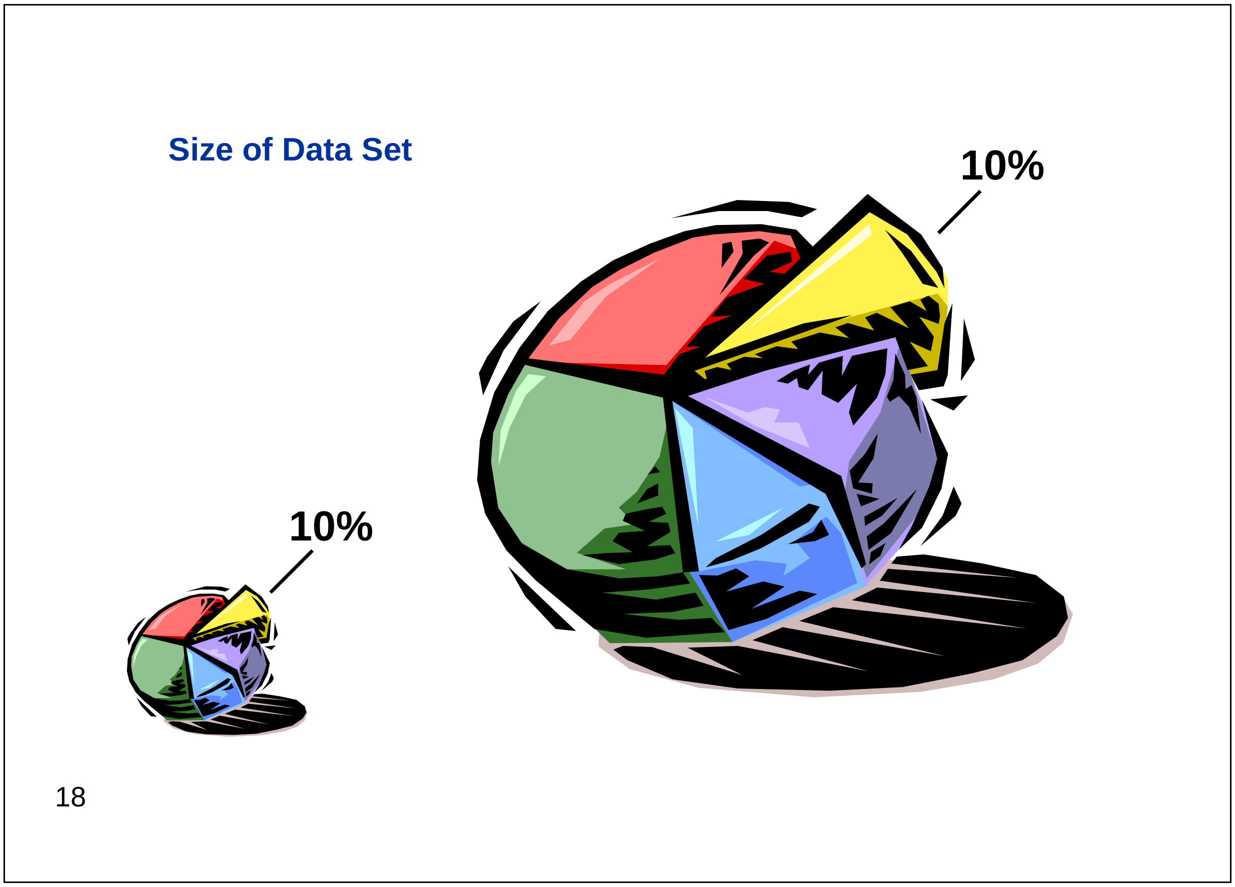
17

The size of the data set is important because performance gains using an index are directly proportional to the size of the data set.

Loss of performance during data update operations might negate performance gains during data retrieval.

Performance gains using indexes are inversely proportional to the size of the subset requested.

Index performance can be improved if the data is sorted on a variable prior to creating an index on that variable.



Size of Data Set

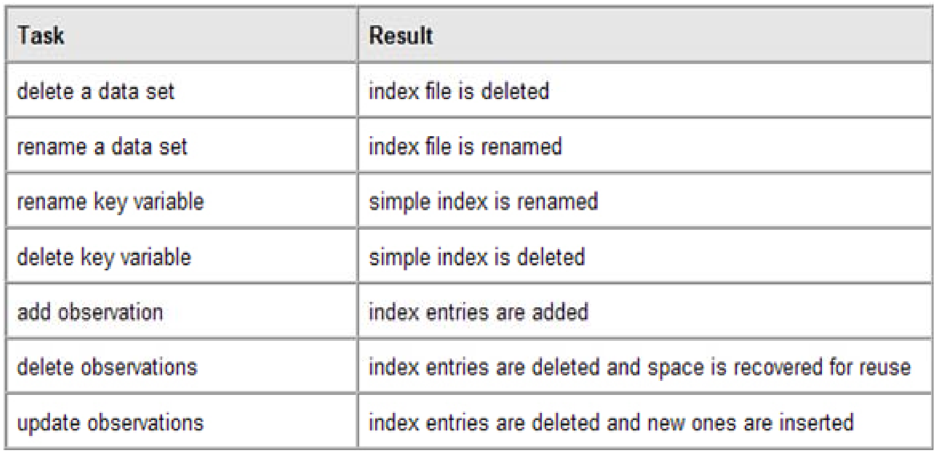
18

The size of the data set is important because performance gains using an index are directly proportional to the size of the data set.

The bigger the data set, the bigger the potential gains. For example, using an index to retrieve a 10% subset of a data set with 100,000 observations means I could avoid reading as many as 90,000 rows. If the data set contains 10,000,000 observations, I can avoid reading up to 9,000,000 rows!

Data Modifications

Maintenance tasks and index results



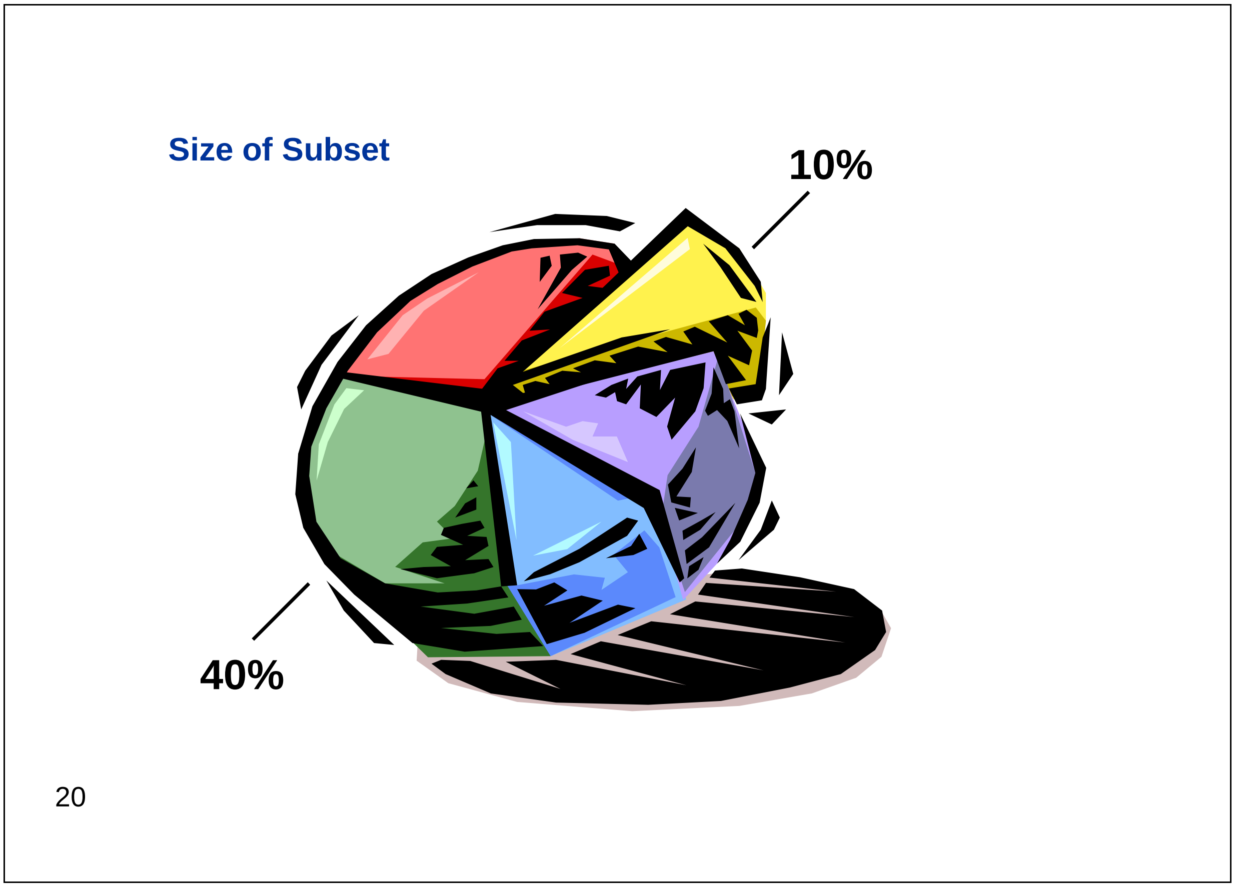
Task	Result
delete a data set	index file is deleted
rename a data set	index file is renamed
rename key variable	simple index is renamed
delete key variable	simple index is deleted
add observation	index entries are added
delete observations	index entries are deleted and space is recovered for reuse
update observations	index entries are deleted and new ones are inserted

19

Loss of performance during data update operations might negate performance gains during data retrieval.

So if a data set is used for transaction processing and is being updated several times every hour (or every minute), an index would slow down every update and should not be considered. However, if you have a data warehouse table that is refreshed only once a day (or once a week), then an index should be considered.

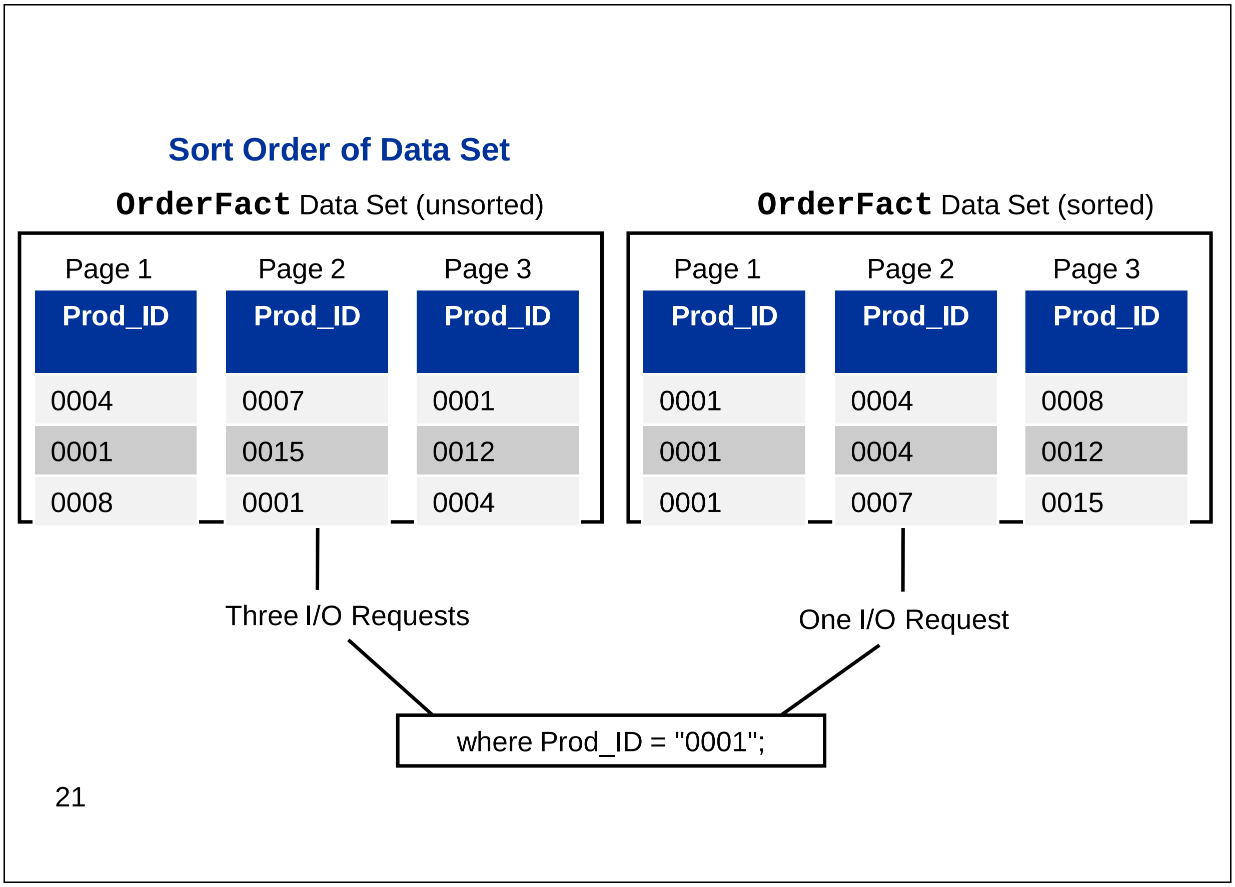
This table describes the overhead tasks that are performed by SAS when modifications are performed on a data set. Effectively, any time you perform a modification to a data set that has been indexed, SAS must perform a similar operation on all indexes associated with that data set.



Size of Subset

20

Performance gains using indexes is inversely proportional to the size of the subset requested. This is due to the added overhead necessary to retrieve each observation using an index. So if I request a report on every male employee, that might be close to half of the entire data set, which would make it faster for SAS to process sequentially without using the index.



Sort Order of Data Set

`OrderFact` Data Set (unsorted)

Page 1	Page 2	Page 3
Prod_ID	Prod_ID	Prod_ID
0004	0007	0001
0001	0015	0012
0008	0001	0004

`OrderFact` Data Set (sorted)

Page 1	Page 2	Page 3
Prod_ID	Prod_ID	Prod_ID
0001	0004	0008
0001	0004	0012
0001	0007	0015

Three I/O Requests

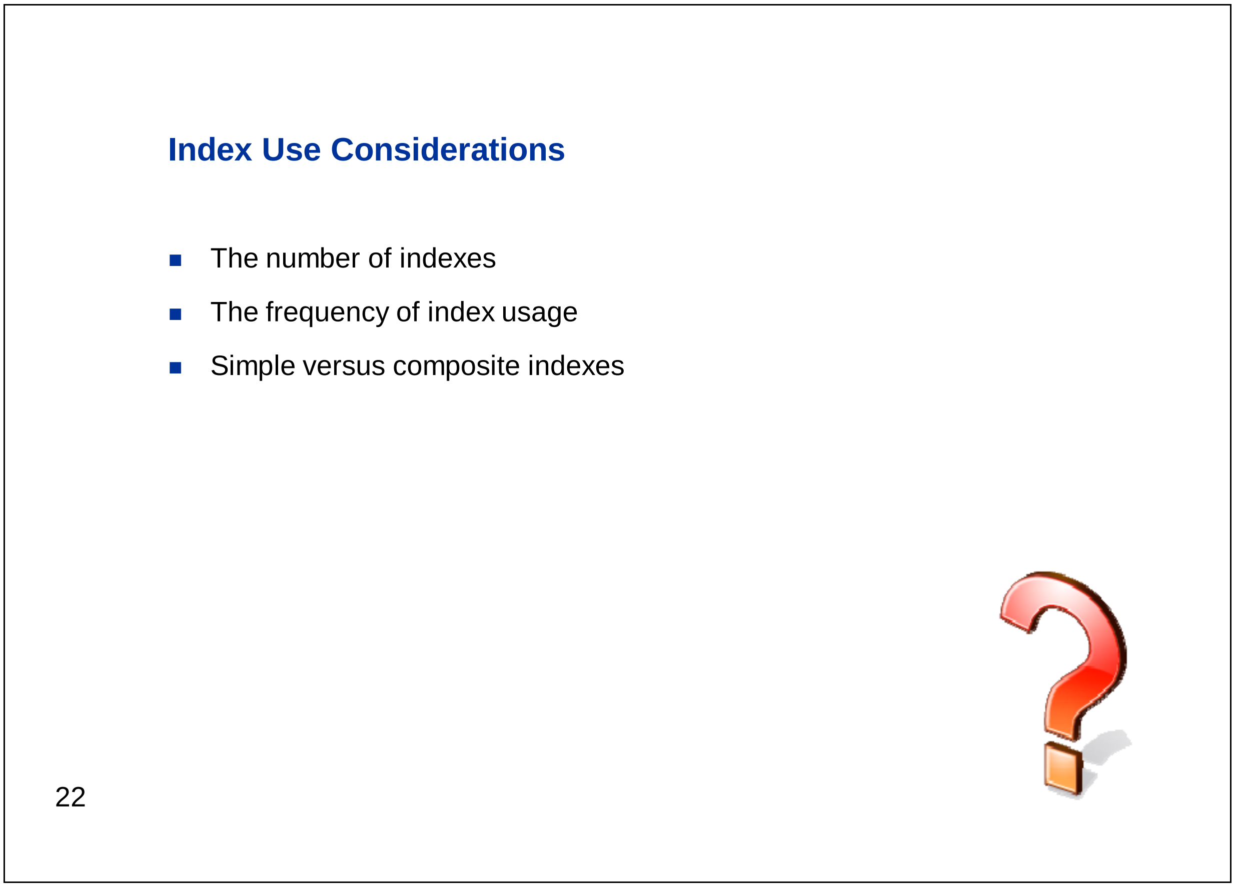
One I/O Request

where Prod_ID = "0001";

21

Index performance can be improved if the data is sorted on a variable prior to creating an index on that variable. Sorting the data can reduce I/O requests by physically moving rows together that would be requested together. If rows are located on a single page, SAS doesn’t have to make another I/O request, which speeds processing.

In this simplified illustration, the WHERE statement causes the index to make three I/O requests in order to retrieve the "0001" **`Prod_ID`** from the unsorted data set because the observations are located on three different pages. Only one I/O request is needed for the sorted data set because all the "0001" observations are together on a single page.



Index Use Considerations

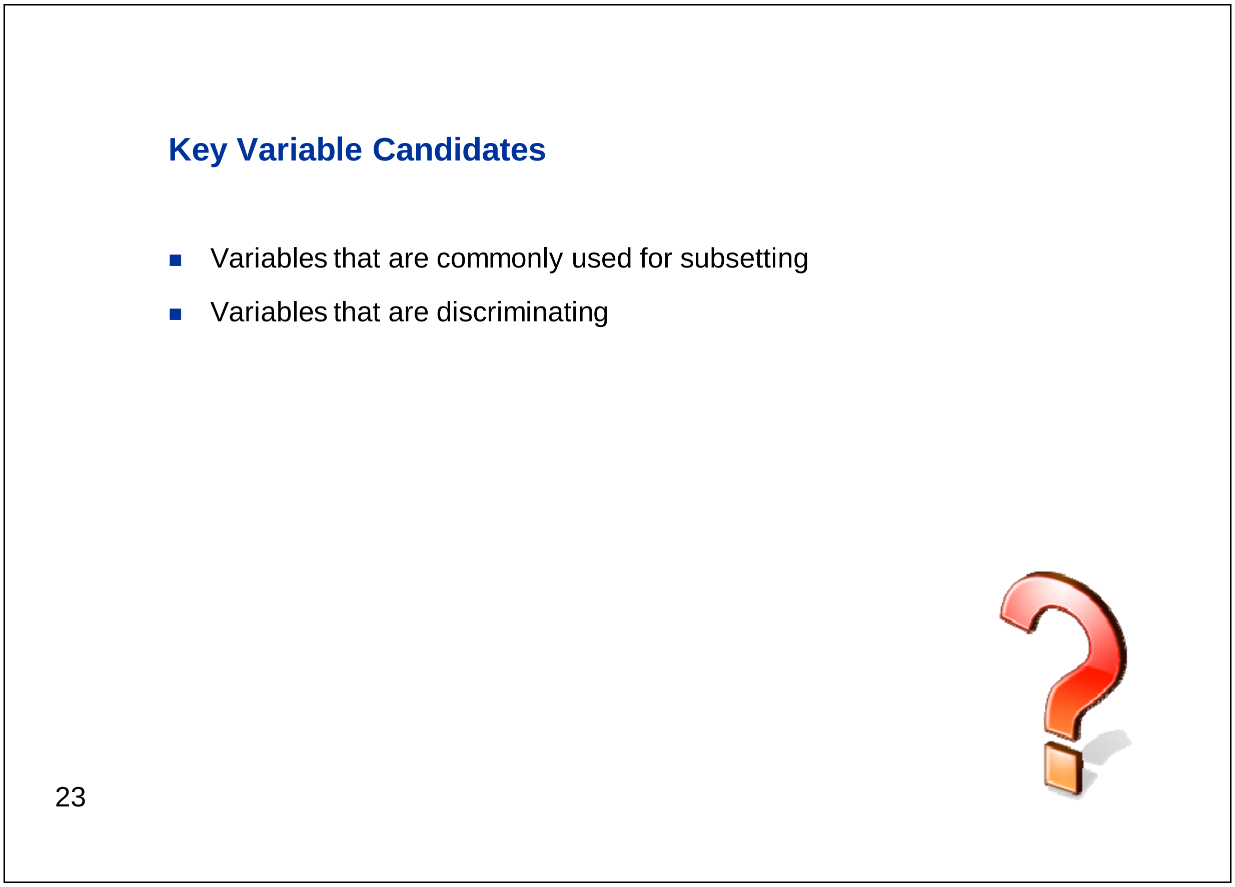
- The number of indexes
- The frequency of index usage
- Simple versus composite indexes

22

The number of indexes is an important consideration because each index that you create adds to the total costs. This is because each index takes time to create, takes up space, and must be maintained during data modifications. So the more indexes that you have on a single table, the more resources you will use to create and maintain the indexes.

Also, consider how often your applications will use an index. An index must be used often in order to make up for the resources that are used in creating and maintaining it. In other words, don't rely solely on resource savings from processing a WHERE expression. Take into consideration the resources it takes to create and maintain the index.

Now, based on what I just told you, it might be tempting to try to create one composite index that will satisfy all possible WHERE expressions. However, this will probably not work because a WHERE expression that references only one variable will only use an index if that variable is used in a simple index or is the first key variable in a composite index.



Key Variable Candidates

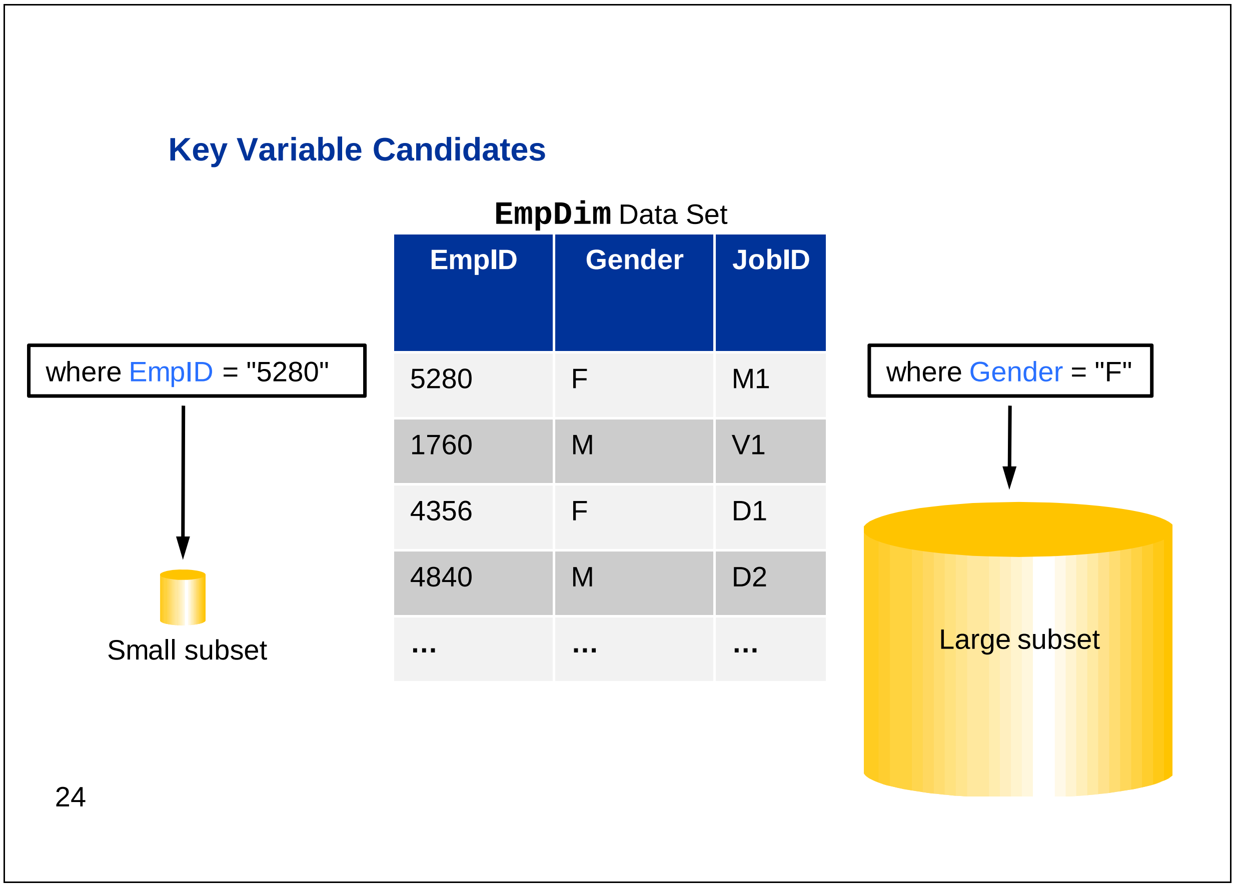
- Variables that are commonly used for subsetting
- Variables that are discriminating

23

In most cases, multiple variables are used to query a data file. However, it probably would be a mistake to index all variables in a data file, as certain variables are better candidates than others.

The variables to be indexed should be variables that are used in queries. That is, your application should require selecting small subsets from a large file, and the most common selection variables should be considered as candidate key variables.

A variable is a good candidate for indexing when the variable can be used to precisely identify the observations that satisfy a WHERE expression. That is, the variable should be discriminating, which means that the index should select the fewest possible observations. For example, the variable **`Gender`** is not discriminating because it's possible for a large number of observations to have the same value for **`Gender`**. However, a variable such as **`Employee_ID`** is discriminating because only a small number of observations will have the same **`Employee_ID`** value.



Key Variable Candidates

EmpDim Data Set

EmpID	Gender	JobID
5280	F	M1
1760	M	V1
4356	F	D1
4840	M	D2
...	...	...

where EmpID = "5280"

Small subset

where Gender = "F"

Large subset

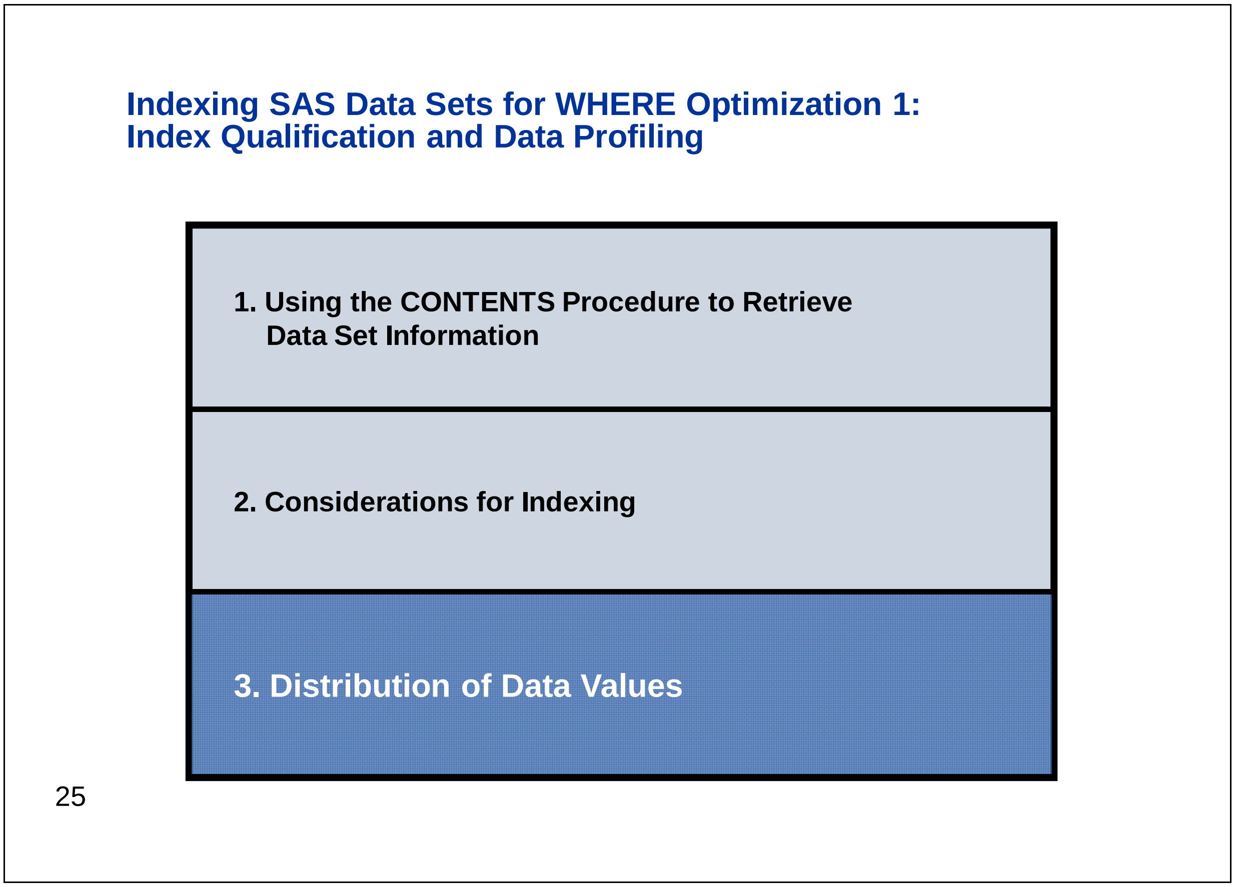
24

For example, consider a data file with variables **EmpID** and **Gender**.

If many queries against the data file include **EmpID**, then indexing **EmpID** could be beneficial because the values are usually discriminating. That is, a single **EmpID** value would return few observations.

However, **Gender** is not a good candidate to be indexed, even though many queries will also include the **Gender** variable. This is because a WHERE expression using the **Gender** variable could result in a large subset of the observations to be returned.

3. Distribution of Data Values



Indexing SAS Data Sets for WHERE Optimization 1:
Index Qualification and Data Profiling

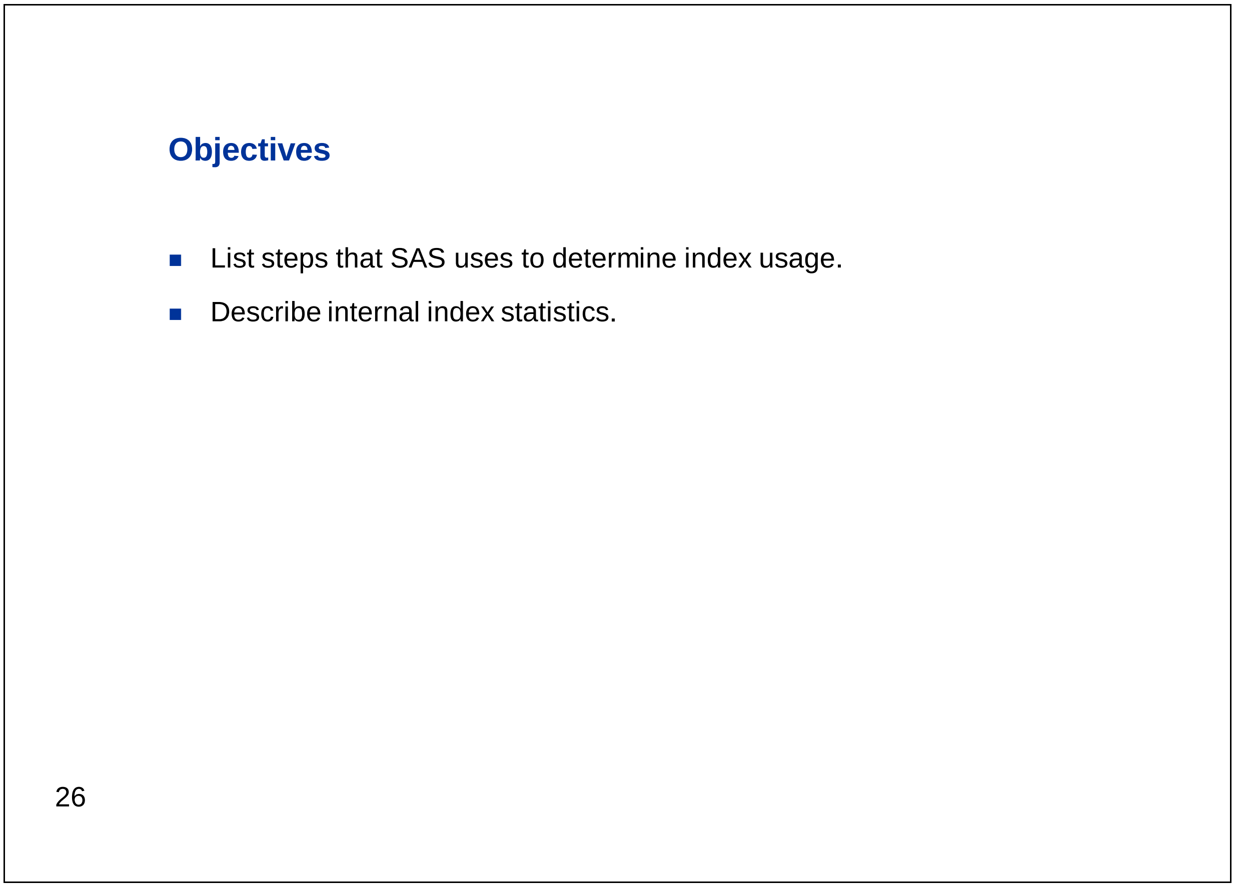
1. Using the CONTENTS Procedure to Retrieve Data Set Information

2. Considerations for Indexing

3. Distribution of Data Values

25

In this section we'll look at the factors that SAS uses in determining when to use an index and which index to use.



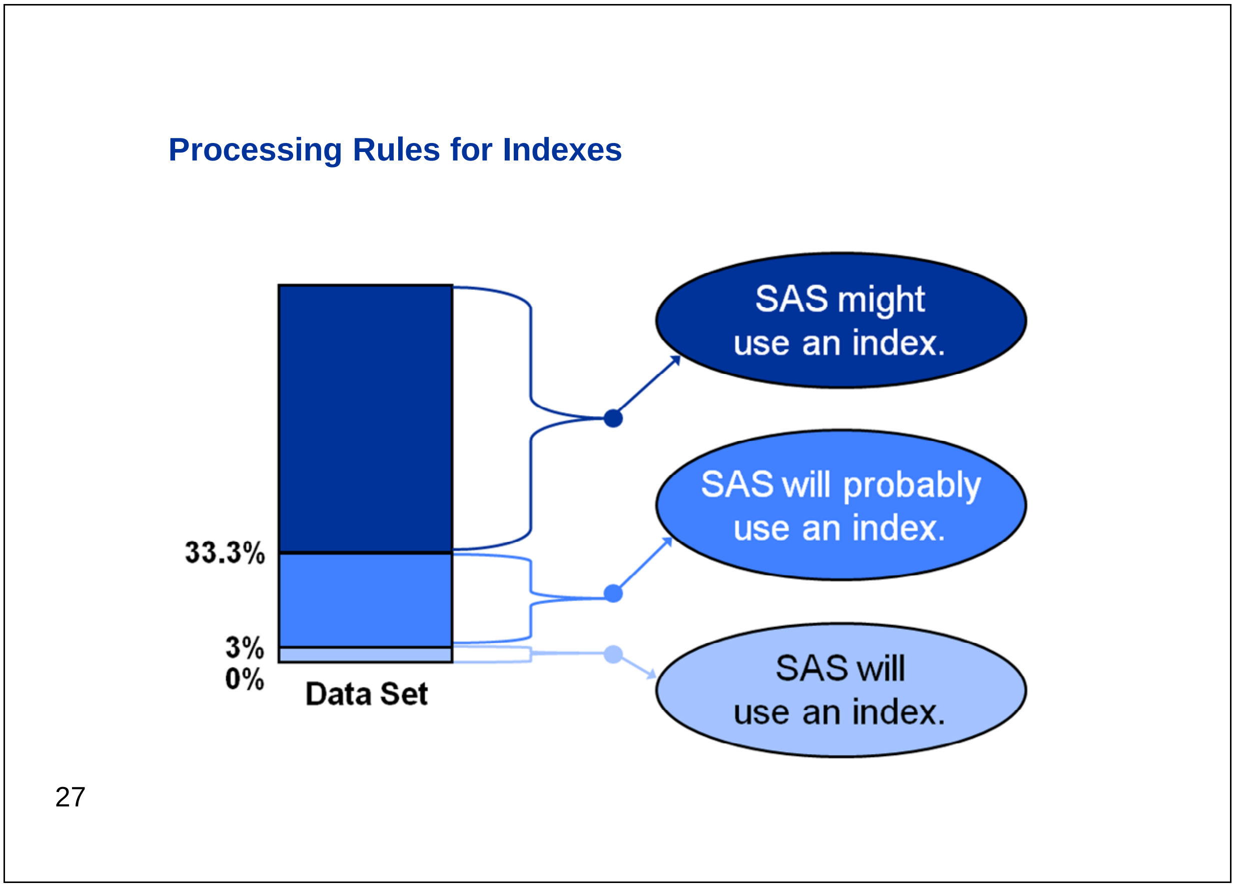
Objectives

- List steps that SAS uses to determine index usage.
- Describe internal index statistics.

26

In this section you’ll learn the steps that SAS goes through to determine whether or not to use an index.

You’ll then learn about the statistics that are automatically tracked within an index.



Processing Rules for Indexes

SAS might use an index.

SAS will probably use an index.

SAS will use an index.

27

WHERE processing conditionally selects observations when you issue a WHERE expression. Using an index to process a WHERE improves performance and is referred to as *optimizing* the WHERE expression.

When the WHERE is processed, SAS decides either to use an index or read the observations in the data file sequentially.

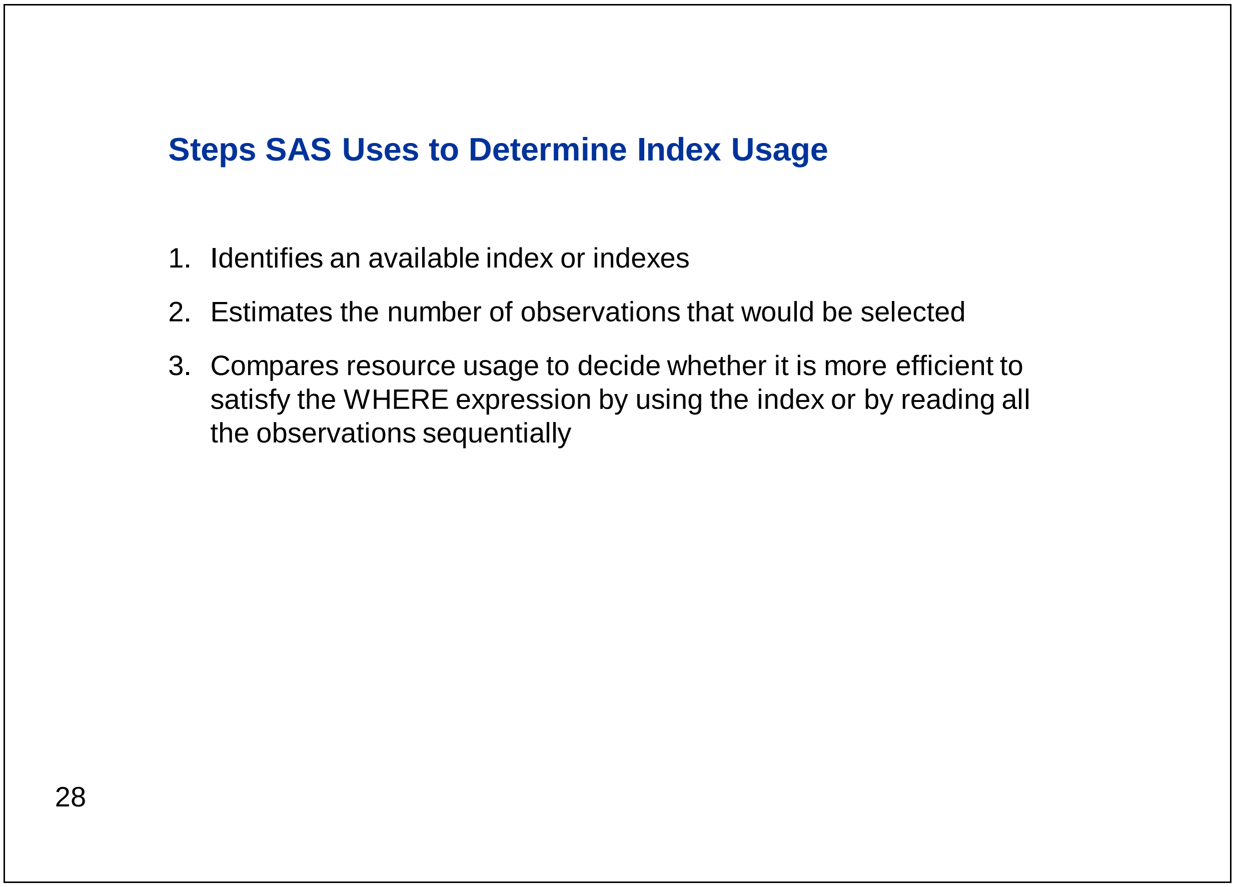
Processing rules for indexed variables dictate that when a WHERE expression returns less than 3 percent of the data, an index will always be used.

If a WHERE expression returns between 3 and 33 percent of the data, an index will probably be used.

And if a WHERE returns over 33 percent of the data, an index might be used depending on whether the data set is sorted by the indexed variable.

If a WHERE returns all the observations, SAS will not use an index.

So what are the steps SAS goes through to accurately determine the size of the subset and make this decision?



Steps SAS Uses to Determine Index Usage

1. Identifies an available index or indexes
2. Estimates the number of observations that would be selected
3. Compares resource usage to decide whether it is more efficient to satisfy the WHERE expression by using the index or by reading all the observations sequentially

28

To make this decision, SAS carries out three steps.

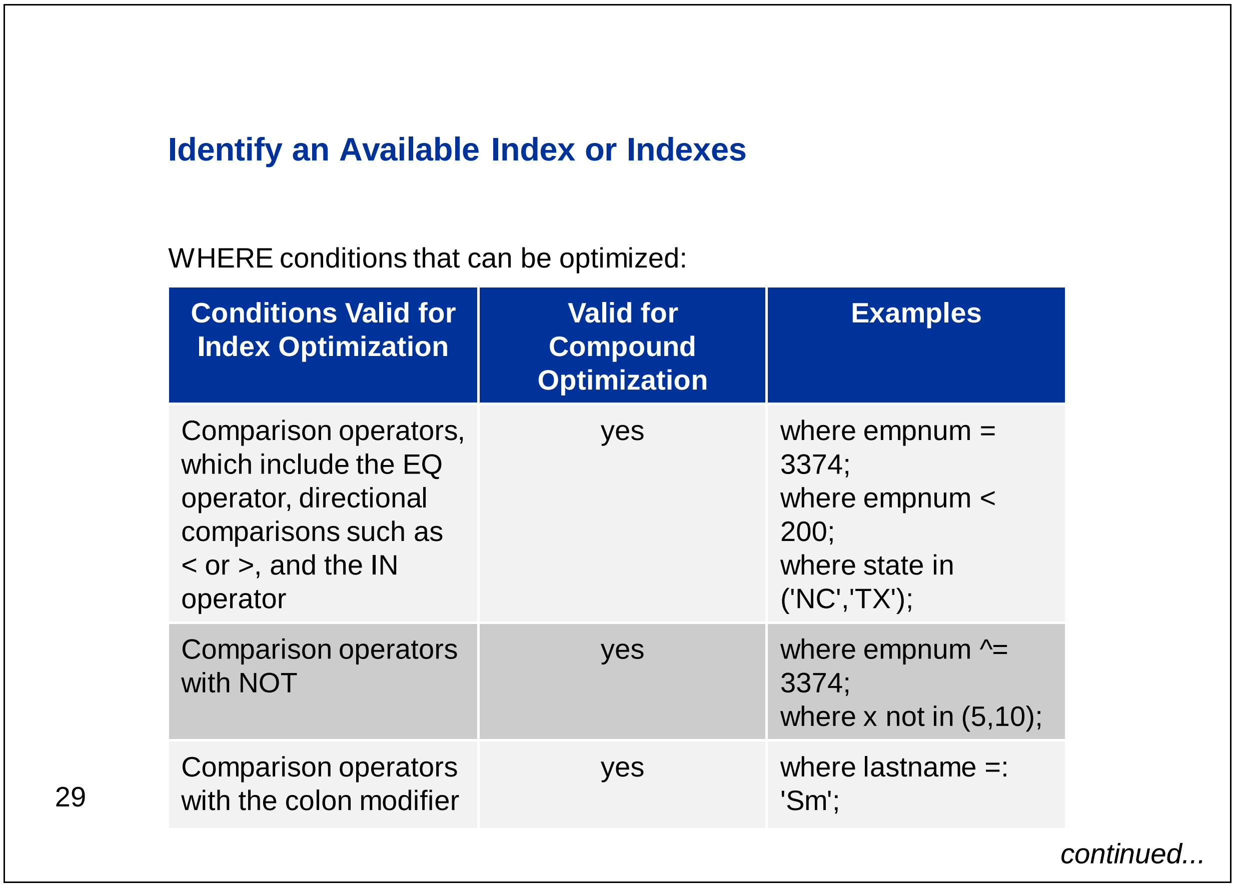
First, SAS identifies all the available indexes on the data set being queried.

Second, SAS estimates the number of observations that would be selected. If more than one index is available, SAS selects the index that returns the smallest subset of observations.

Third, SAS compares resource usage to decide whether it is more efficient to satisfy the WHERE expression by using the index or by reading all the observations sequentially.

Because SAS considers several factors when deciding whether to use an index, to get the best possible performance you should experiment using different WHERE expressions with and without an index and compare the results.

There are several options that allow you to control index usage, including the IDXWHERE= and IDXNAME= data set options.



Identify an Available Index or Indexes

WHERE conditions that can be optimized:

Conditions Valid for Index Optimization	Valid for Compound Optimization	Examples

Comparison operators, which include the EQ operator, directional comparisons such as < or >, and the IN operator	yes	where empnum = 3374;
		where empnum < 200;
		where state in ('NC','TX');

Comparison operators with NOT	yes	where empnum ^= 3374; where x not in (5,10);
Comparison operators with the colon modifier	yes	where lastname =: 'Sm';

29

continued...

The first step for SAS in deciding whether to use an index to process a WHERE expression is to determine if variables in the WHERE expression are indexed variables.

Even if a WHERE expression consists of multiple conditions with different variables, SAS will use only one index to process the WHERE expression. SAS will select the index that satisfies the most conditions and returns the fewest observations.

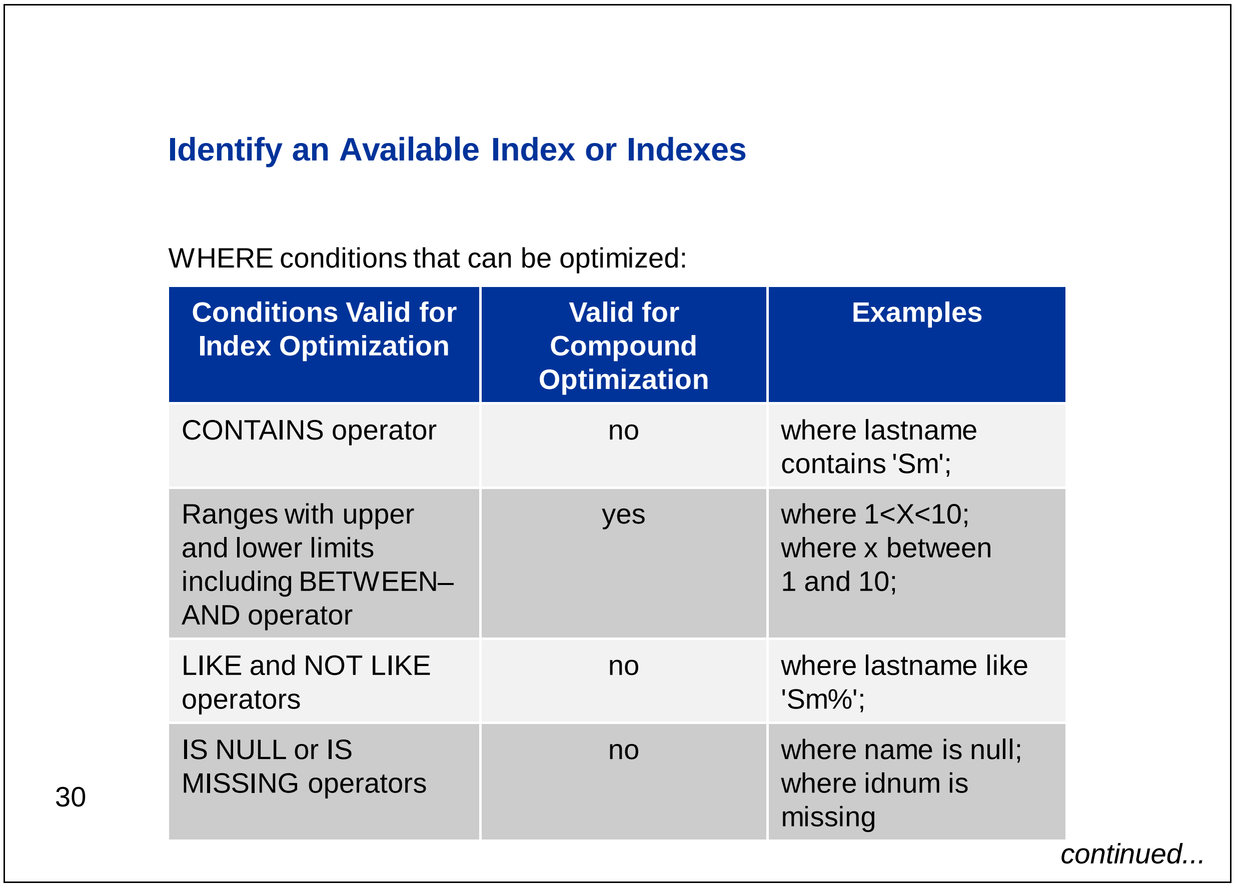
Note that the variable in the condition can either have a simple index or be the first variable in a composite index.

You can take advantage of multiple variables in a composite index by writing a WHERE expression that uses some or all of those variables. Note that in order to use all, or only some of the composite index variables, you're required to use them in order beginning with the first. For example, if a composite index called SCZ includes the variables **`State`**, **`City`**, and **`Zip`** (in that order), your WHERE expression can include all three variables, or can include just **`State`** or just **`State`** and **`City`**. But a WHERE expression that includes only **`City`** and **`Zip`** or only **`State`** and **`Zip`** could not use the SCZ composite index.

This table can help you write WHERE statements that can be optimized. The first column shows conditions that can be optimized using an index. The second column indicates if these conditions can also be used for compound optimization.

Compound optimization is the process of optimizing multiple WHERE conditions using a single composite index.

Note that you can replace the mnemonic operator with symbols, so for the equals operator you can use either EQ or the equals symbol (=).



Identify an Available Index or Indexes

WHERE conditions that can be optimized:

Conditions Valid for Index Optimization	Valid for Compound Optimization	Examples
CONTAINS operator	no	where lastname contains 'Sm';

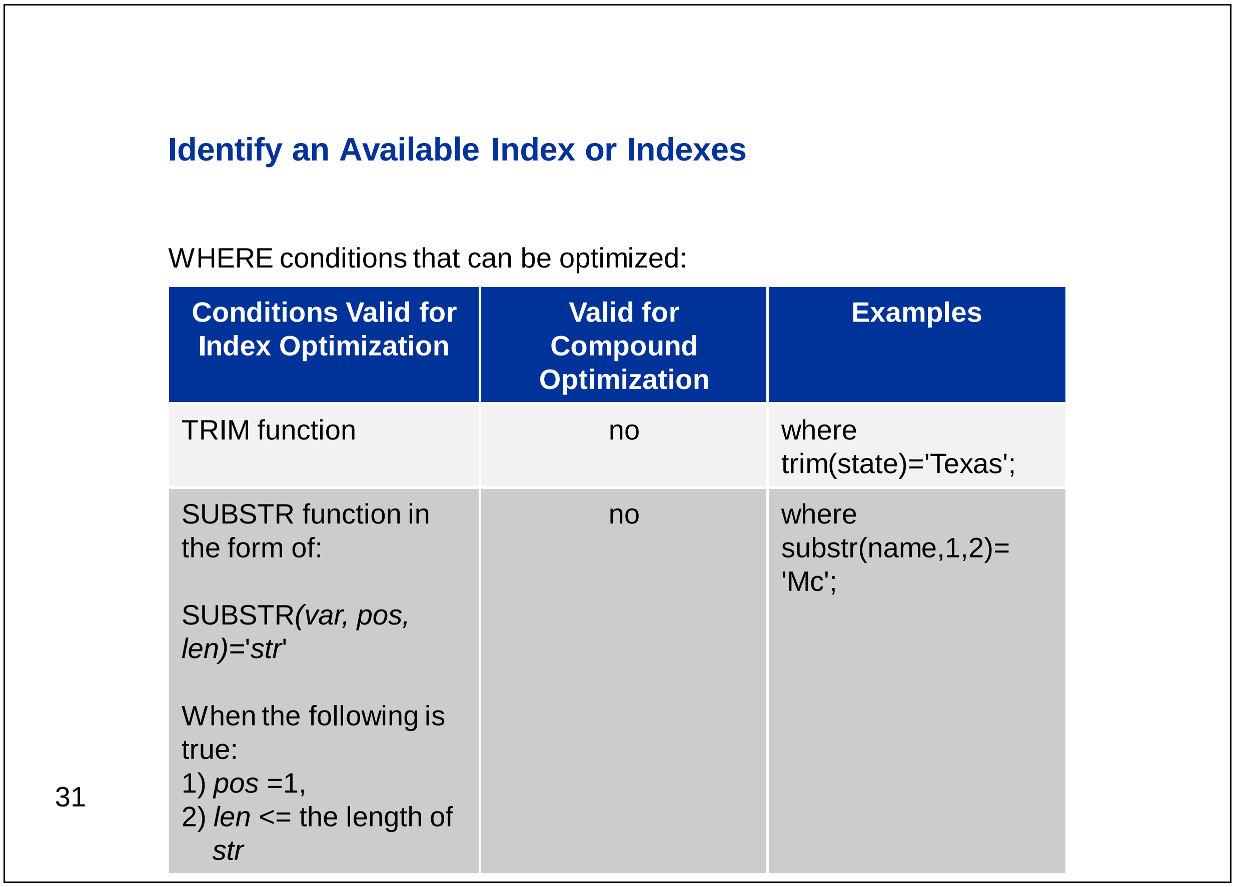
Ranges with upper and lower limits including BETWEEN–AND operator	yes	where 1<X<10;
		where x between 1 and 10;

LIKE and NOT LIKE operators	no	where lastname like 'Sm%';
IS NULL or IS MISSING operators	no	where name is null; where idnum is missing

30

continued...

This is a continuation of the previous table and shows operators that are unique to WHERE expressions.



Identify an Available Index or Indexes

WHERE conditions that can be optimized:

Conditions Valid for Index Optimization	Valid for Compound Optimization	Examples
TRIM function	no	where trim(state)='Texas';
SUBSTR function in the form of: SUBSTR(*var, pos, len)='str'* When the following is true: 1) *pos* =1, 2) *len* <= the length of *str*	no	where substr(name,1,2)= 'Mc';

31

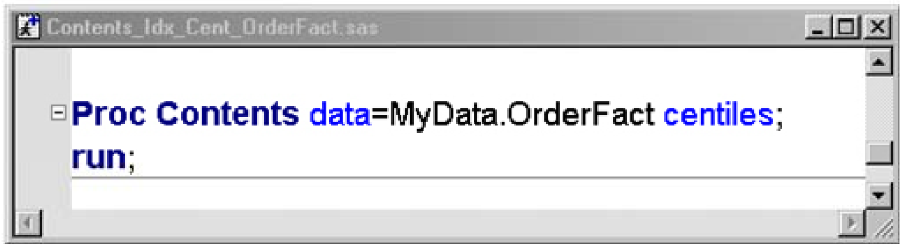
This is also a continuation of the previous tables and shows the SAS functions that are valid for index optimization.

Notice that only the SUBSTR and TRIM functions can be used in a WHERE expression that will be optimized with an index. All other functions will require SAS to process the data set sequentially.

In addition, conditions with the following will not be optimized with an index:

- arithmetic operators
- variable-to-variable comparisons
- the sounds-like operator
- conditions using the OR logical operator

Estimate the Number of Observations Selected



```
Proc Contents data=MyData.OrderFact centiles;
run;
```

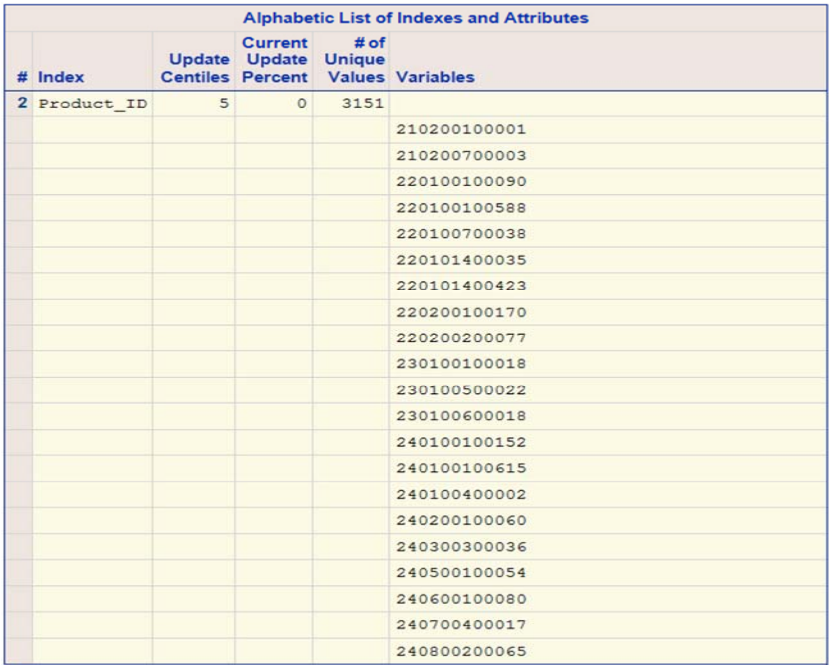
32

After SAS identifies the indexes that can satisfy the WHERE expression, the next step is to estimate the number of observations that will be returned by an available index. If multiple indexes exist, SAS will select the one that produces the fewest observations.

SAS estimates the number of observations that will be qualified by using stored statistics called *cumulative percentiles* (or *centiles* for short). Centiles information represents the distribution of values in an index so that SAS does not have to assume a uniform distribution.

To print centiles information for an indexed data set, we have included the CENTILES option in PROC CONTENTS.

Estimate the Number of Observations Selected



Alphabetic List of Indexes and Attributes

#	Index	Update Centiles	Current Update Percent	# of Unique Values	Variables
2	Product_ID	5	0	3151	
					210200100001
					210200700003
					220100100090
					220100100588
					220100700038
					220101400035
					220101400423
					220200100170
					220200200077
					230100100018
					230100500022
					230100600018
					240100100152
					240100100615
					240100400002
					240200100060
					240300300036
					240500100054
					240600100080
					240700400017
					240800200065

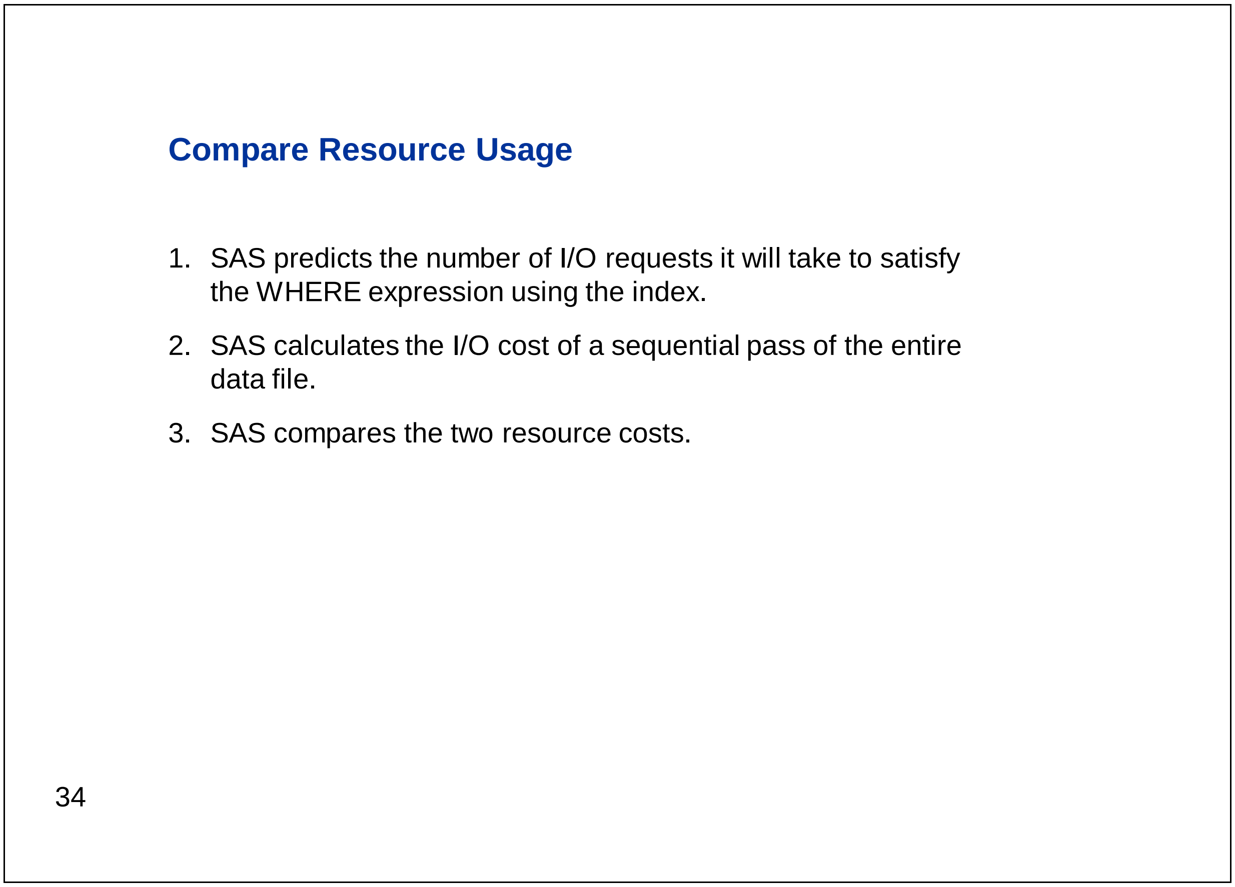
33

This slide displays a partial output of the report produced by the previous PROC CONTENTS.

The information provided by centiles represents the distribution of values in an index. Twenty-one centiles are kept: 0, 5, 10,…, 95, 100 percentiles, where 0 percentile is the minimum value of the data, the 50th percentile is the median value, and the 100th percentile is the maximum value.

By default, centile information is updated after 5% of the data set has changed.

These internal statistics enable SAS to make an accurate estimate of how much data will be returned with a given WHERE condition, regardless of the distribution of the values. This means that you don’t need to think about the distribution of the data. So all variables can be considered for indexes whether they have values with a uniform, normal, or skewed distribution.



Compare Resource Usage

1. SAS predicts the number of I/O requests it will take to satisfy the WHERE expression using the index.
2. SAS calculates the I/O cost of a sequential pass of the entire data file.
3. SAS compares the two resource costs.

34

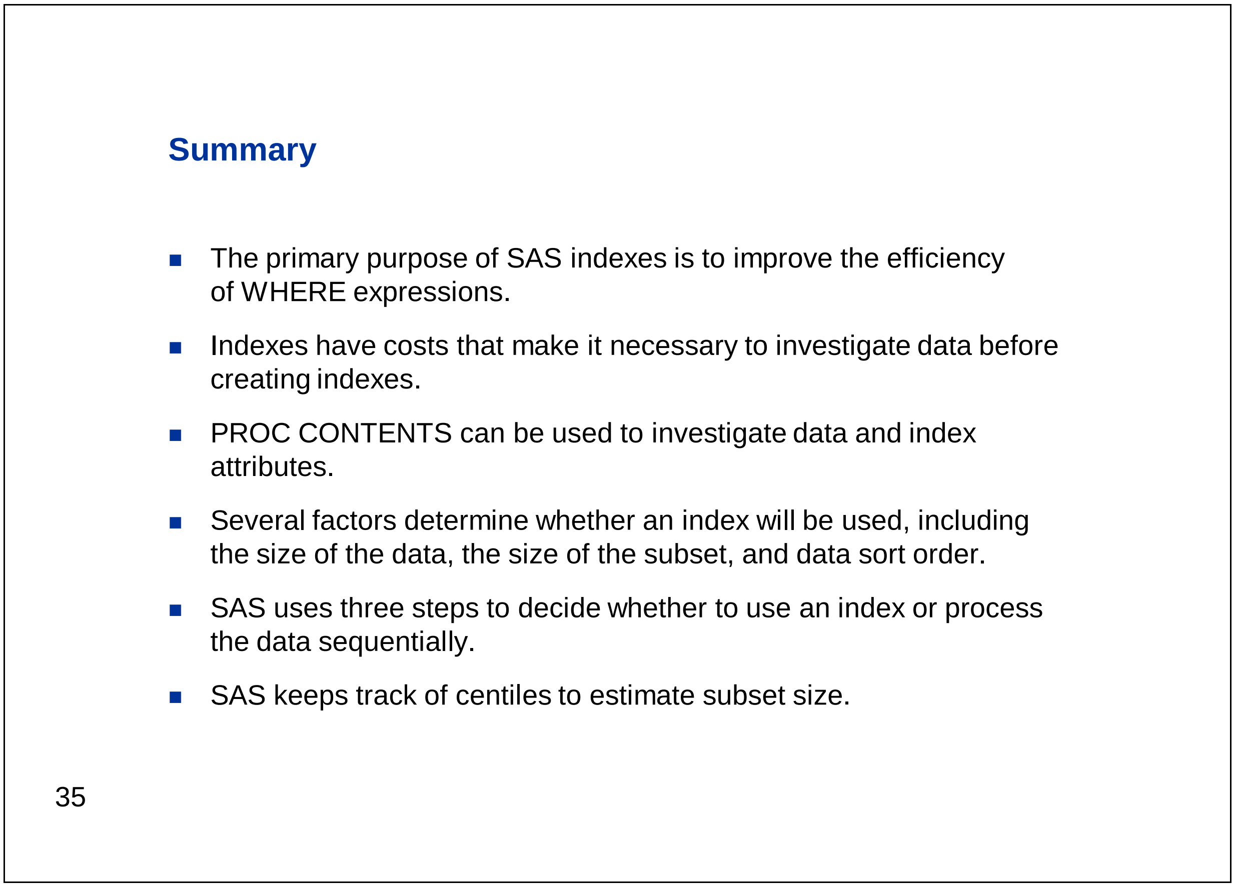
To compare resource usage, first, SAS predicts the number of I/O requests it will take to satisfy the WHERE expression using the index.

Next, SAS calculates the I/O cost of a sequential pass of the entire data file.

Last, SAS compares the two resource costs and chooses the one that is most efficient.

Factors that affect the comparison include the size of the subset, the sort order of the data set, the page size of the data set, the number of allocated buffers, and the cost to uncompress or compress a compressed data file.

Note that if comparing resource costs results in a tie, SAS will choose the index.



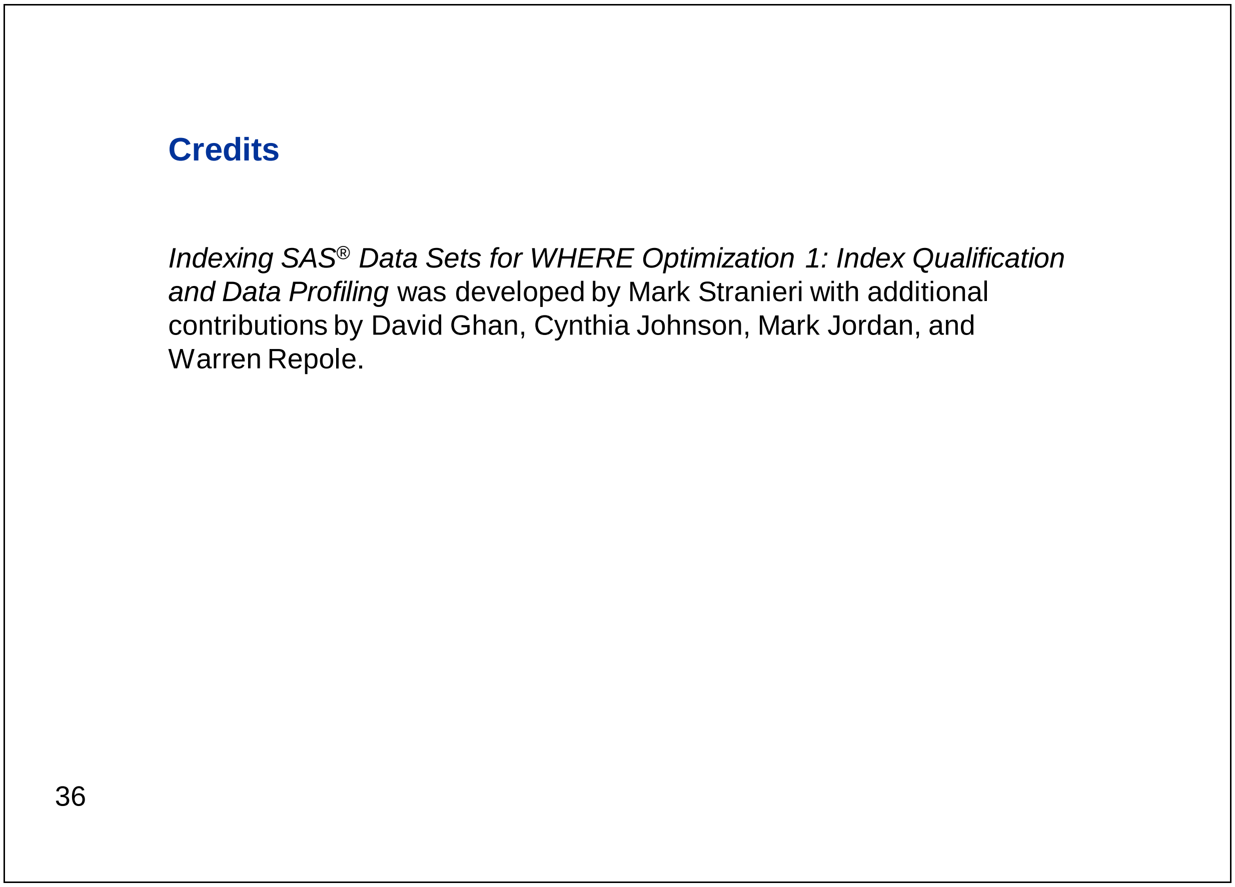
Summary

- The primary purpose of SAS indexes is to improve the efficiency of WHERE expressions.
- Indexes have costs that make it necessary to investigate data before creating indexes.
- PROC CONTENTS can be used to investigate data and index attributes.
- Several factors determine whether an index will be used, including the size of the data, the size of the subset, and data sort order.
- SAS uses three steps to decide whether to use an index or process the data sequentially.
- SAS keeps track of centiles to estimate subset size.

35

In summary:

- The primary purpose of SAS indexes is to improve the efficiency of WHERE expressions.
- Indexes have costs that make it necessary to improve the efficiency of WHERE expressions.
- To investigate data and index attributes, you can use the CONTENTS procedure.
- Several factors determine whether an index will be used, including the size of the data, the size of the subset, and data sort order.
- SAS uses three steps to determine whether to use an index or to process the data sequentially.
- SAS keeps track of centiles to estimate subset size.

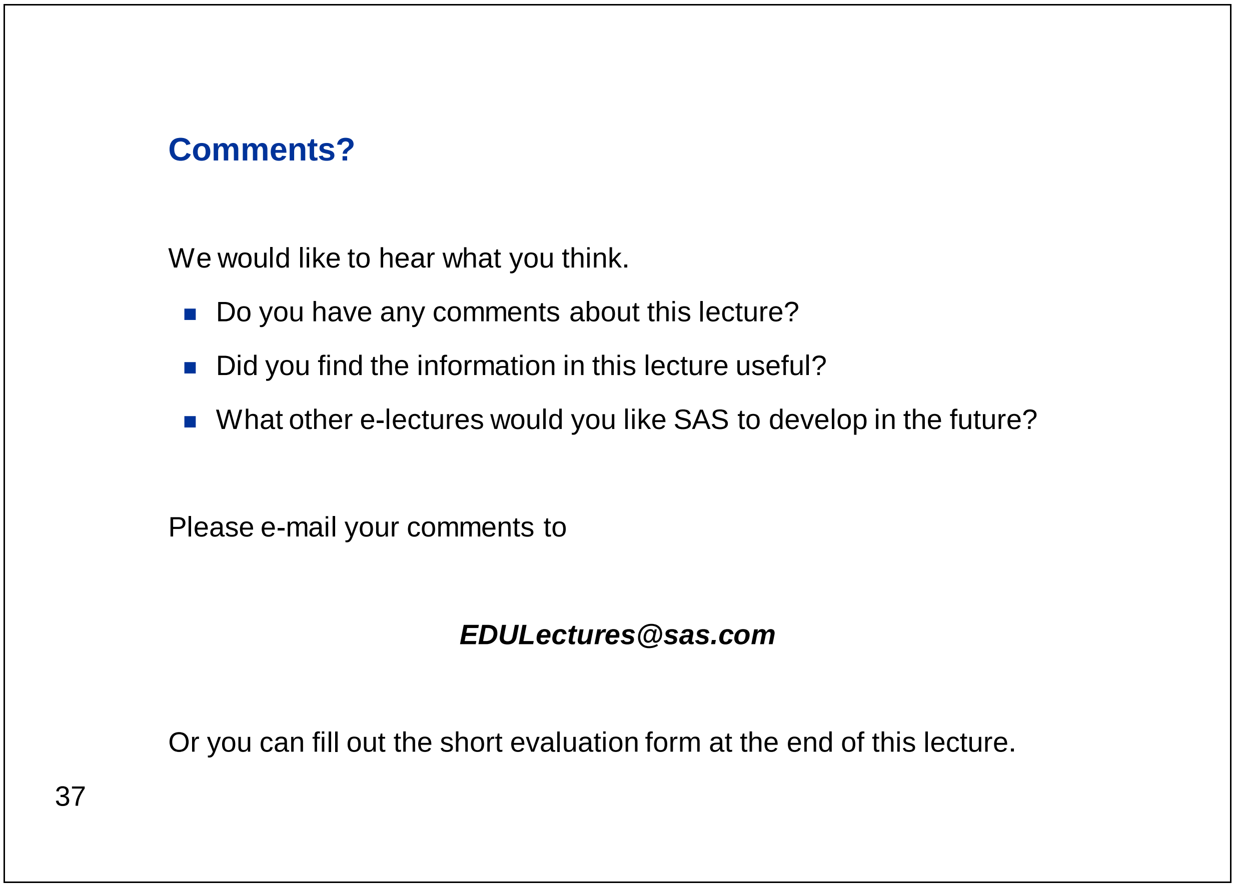


Credits

Indexing SAS® Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling was developed by Mark Stranieri with additional contributions by David Ghan, Cynthia Johnson, Mark Jordan, and Warren Repole.

36

This e-lecture was developed by Mark Stranieri with additional contributions from David Ghan, Cynthia Johnson, Mark Jordan, and Warren Repole. This concludes the first lecture on *Indexing SAS® Data Sets for WHERE Optimization 1: Index Qualification and Data Profiling*. I hope you find the material to be helpful for your work.



Comments?

We would like to hear what you think.

- Do you have any comments about this lecture?
- Did you find the information in this lecture useful?
- What other e-lectures would you like SAS to develop in the future?

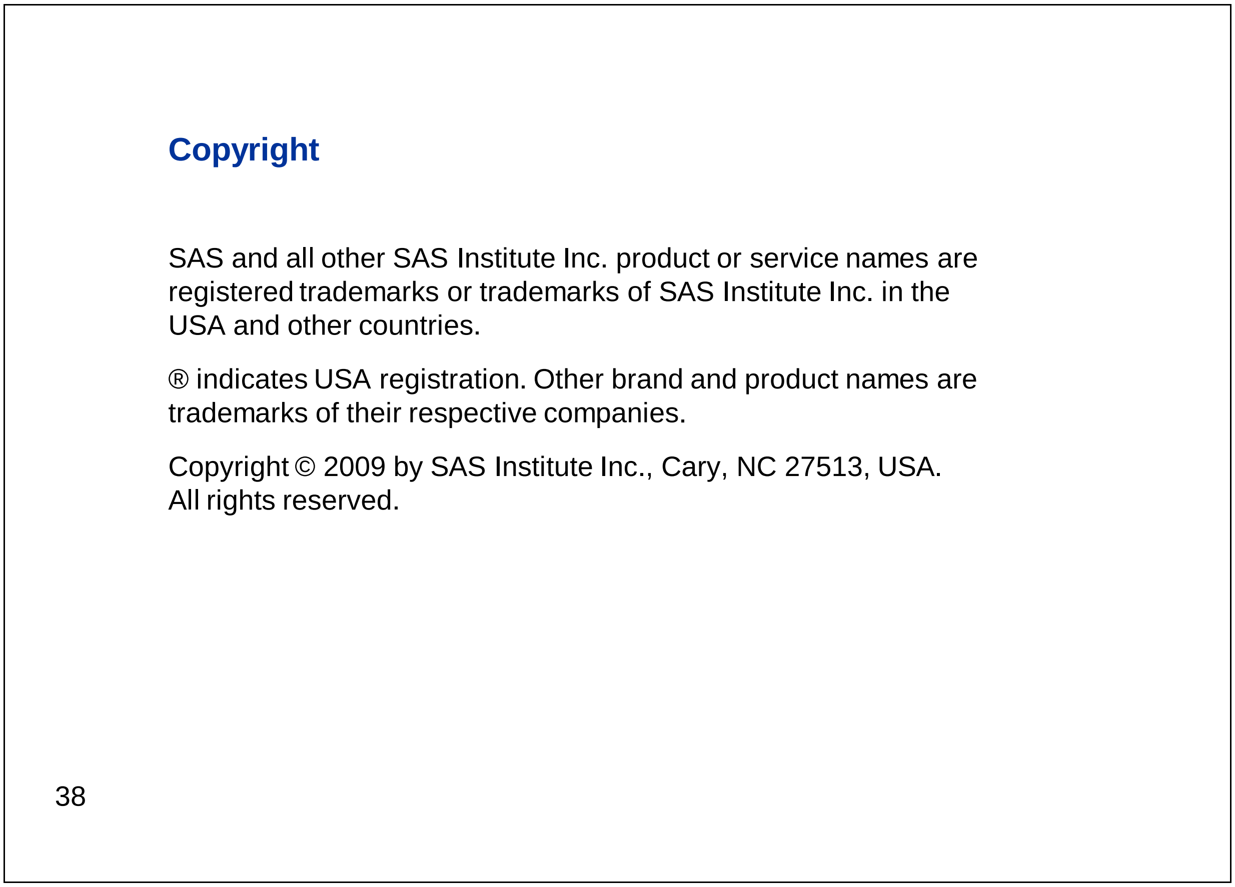
Please e-mail your comments to

EDULectures@sas.com

Or you can fill out the short evaluation form at the end of this lecture.

37

If you have any comments about this lecture or e-lectures in general, we would appreciate receiving your input. You can use the e-mail address listed here to provide that feedback, or you can complete the short evaluation form available at the end of this lecture.



Copyright

Thank you for your time.