Performance Aspects of Data Archiving

Factors for optimal results in archiving projects

Version 1.0
November 2005
Copyright

© Copyright 2005 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.

IBM®, DB2®, OS/2®, DB2/6000®, Parallel Sysplex®, MVS/ESA®, RS/6000®, AIX®, S/390®, AS/400®, OS/390®, and OS/400® are registered trademarks of IBM Corporation.

ORACLE® is a registered trademark of ORACLE Corporation.

INFORMIX®-OnLine for SAP and INFORMIX® Dynamic Server™ are registered trademarks of Informix Software Incorporated.

UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.

Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.

HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA® is a registered trademark of Sun Microsystems, Inc.

JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, SAP Logo, R/2, RIVA, R/3, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP Business Suite Logo and mySAP Business Suite are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

Symbols used

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>⚠️</td>
<td>Caution</td>
</tr>
<tr>
<td>🔄</td>
<td>Example</td>
</tr>
<tr>
<td>➡️</td>
<td>Note</td>
</tr>
<tr>
<td>📖</td>
<td>Background information</td>
</tr>
</tbody>
</table>
# Table of Contents

1 INTRODUCTION ............................................................................................................................... 4  
1.1 THE BENEFITS OF DATA ARCHIVING ................................................................. 4  
1.2 DATA ARCHIVING AND PERFORMANCE ......................................................... 4  
1.3 SCOPE AND METHODOLOGY ............................................................................. 6  
1.4 TARGET GROUP ................................................................................................. 6  

2 FROM PERFORMANCE TO DATA ARCHIVING ................................................................. 7  
2.1 PERFORMANCE MONITORING IN SAP ECC ................................................... 7  
2.2 MONITORING DATABASE PERFORMANCE ...................................................... 8  
2.3 PERFORMANCE AND THE ROLE OF DATA AGE .......................................... 11  
2.4 FROM THE DELETION OF OLD DATA TO THE ARCHIVING OF BUSINESS-COMPLETE DATA ..................................................... 12  
2.5 I/O RATE – THE PERFORMANCE KEY FIGURE MOST AFFECTED BY ARCHIVING ............................................................................... 12  

3 TAKING A CLOSER LOOK AT THE DATABASE ................................................................. 13  
3.1 PERFORMANCE IMPACT OF ARCHIVABLE DATA ON THE DATABASE .......... 13  
3.2 USAGE OF DATA BLOCK IN INDEXES ................................................................. 14  
3.2.1 Chronologically Sorted Indexes .................................................................. 15  
3.2.2 Non-Chronologically Sorted Indexes ......................................................... 15  
3.2.3 A Mixture of Chronologically and Non-Chronologically Sorted Indexes ......................................................................................... 16  
3.3 USAGE OF DATA BLOCKS IN TABLES ............................................................... 17  
3.4 SPECIAL CONSIDERATIONS FOR MASTER DATA ........................................ 18  
3.5 ARCHIVING VS. INDEX OR PROGRAM OPTIMIZATION .................................... 19  
3.6 OPTIMIZER STATISTICS AND DATABASE PARAMETERS ............................. 21  
3.7 INDEX FRAGMENTATION AND REORGANIZATION ...................................... 23  
3.8 TABLE FRAGMENTATION AND REORGANIZATION ..................................... 24  
3.8.1 Online reorganization ................................................................................. 24  

4 SETTING PRIORITIES FOR ARCHIVING PROJECTS ............................................... 24  
4.1 PERFORMANCE VS. DATABASE SPACE ............................................................. 24  
4.2 STRATEGIES TO MINIMIZE FRAGMENTATION ............................................. 25  

5 PERFORMANCE AND ACCESSES TO ARCHIVED DATA ............................................ 27  
5.1 A FEW WORDS ON DATA ARCHIVING .............................................................. 27  
5.1.1 Technology and Main Functions ................................................................. 28  
5.1.2 Processes in Data Archiving ...................................................................... 31  
5.2 RELATIONAL VS. OBJECT-ORIENTED STORAGE ......................................... 33  
5.3 ARCHIVE INFORMATION SYSTEM ................................................................. 34  

6 SUMMARY AND CONCLUSION .................................................................................. 35  

© SAP AG 2005
1 Introduction

1.1 The Benefits of Data Archiving

Data archiving is becoming increasingly more important as CIOs and IT managers are trying to optimize their budgets in today’s tense economic environment. Although IT budgets have been frozen or even reduced, data volumes continue to grow at enormous rates. As databases are being loaded down with more and more data, IT managers are faced with a difficult dilemma: how can they speed up processes, help increase productivity and run operations more efficiently without spending more money?

The following graph shows the project database growth of an actual SAP customer who began data archiving. With a 400 GB system in January 2002 and a projected database growth of 25 GB per month, this company’s database would have reached its maximum capacity in less than a year. There are only a few options to combat the reality of database growth. A company can either expand its system or it can get rid of old data through deletion or through archiving.

![Projected Database Growth Graph](image)

Figure 1: Projected Database Growth

Many companies have realized that data archiving can be an important element of their IT strategy and a better long-term solution than expanding their system. Data archiving helps keep databases efficient by taking data that is no longer needed in everyday business processes, removing it from the database and writing this data to archive files; later it can be moved to less expensive media, if required (see “The Basic Data Archiving Flow”). If implemented soon after a system goes live, and if performed regularly, data archiving can have dramatic effects on the growth of a company’s database, which in turn can lead to noticeable improvements in the cost of running the company’s systems. Apart from this financial view there is another important aspect that is often overlooked: data archiving can improve a system’s performance.

1.2 Data Archiving and Performance

For years performance itself has been one of the most important issues in the IT world, especially for large companies. As a result, enterprises have been working towards fine-tuning their systems and perfecting methods, such as indexing, and have been largely successful in this endeavor. However, after all optimizations are in place, it is quite common to continue to see a small performance degradation over time. This can be seen in the following graph:
Although the performance degradation in terms of response time is not necessarily critical, the graph does clearly reflect that the slowdown is directly related to the database. There seems to be no strong relationship to the system load, which correlates with the number of dialog steps. Therefore, one can clearly assume that the increase in response time is caused by the growth of the data volumes in the database. If calculated in terms of person-hours, this performance degradation results in 19.5 hours lost per day. More importantly, such an increase in response time will affect the acceptance of the system by the users. When database growth is responsible for a performance degradation, then archiving should be the way to stop this trend.

Improving system performance has been used as a valid argument by companies for justifying data archiving projects. However, the priorities in such projects were always set considering mainly the effects on disk space. This was done under the assumption that the relationship between saved disk space and performance is more or less the same for all archiving objects. To shed some light on this relationship we carried out an evaluation to clarify how data archiving influences the performance of a database, and also give advice on how to set the appropriate priorities in a data archiving project and correctly estimate the results of data archiving.

In an initial analysis based on data in a customer system we monitored the growth of the table MSEG, which holds the line items of material documents, and also measured how long it takes to display a material document in transaction MB51.
The result of our analysis, shown in the above diagram, indicated a clear relationship between an increase in table size and a slowdown in response times. However, this example cannot claim to be generally valid since here the relationship between response time and table size is much stronger than in a usual scenario.

In this paper we will try to shed more light on the question of performance and how data archiving can improve it. To better understand the issues involved, we must first take a look at some fundamental aspects of data archiving and performance, and examine the factors that slow down a system. These factors need to be understood in order to recognize how strong the relationship between performance and table size is in different scenarios. Whenever a strong relationship is recognizable that cannot be overcome by other methods, such as a different indexing method, then data archiving is the right method for improving performance.

Apart from questions related to the performance improvement during system operation, we cover additional aspects, such as the performance of the archiving process itself and the performance of archive accesses. To ensure an excellent system performance through an optimal use of data archiving, all aspects mentioned here must be taken into account. Not only from a pragmatic point of view, but also from the perspective of performance it is unwise to entirely relocate the data from the database of the production system to the archive as soon as this becomes possible. This strategy would have some negative consequences: The more archive accesses are required, the more the archive system becomes the performance bottleneck of the entire system.

1.3 Scope and Methodology

To effectively tackle these issues, one must have a clear understanding of the nature of the accesses performed for the individual types of data. Also, one must know that the access behavior and the distribution of old and new data in the database tables and indexes changes considerably as the data “matures”. Both criteria are equally important for correctly assessing the performance aspects of a system, and are therefore also discussed in this paper. Depending on the type of data accessed and the access methods, data archiving can affect performance in various ways with respect to the following: data that is still in the database (online data), archived data, the archiving process.

Our fundamental discussion on the performance aspects of data archiving is not confined to SAP systems. In this paper concrete examples from SAP systems are only used to back up the assumptions made. Our discussion proves that the approaches described can actually be set into practice. It also brings benefits to archiving projects for SAP systems, because it helps companies use data archiving more efficiently in order to achieve the predetermined performance KPIs. Furthermore, our discussion provides a good opportunity to compare SAP’s data archiving solution to the concepts available for other systems or the concepts of other vendors.

1.4 Target Group

This paper is primarily aimed at system and database administrators as well as technology consultants who do not merely want a cookbook describing the tasks necessary in an archiving project, but who want to have a better understanding of how to best use available resources to optimally run a system. Therefore, the aspects discussed should also be interesting for anyone utilizing or implementing data archiving, such as solution consultants or members of archiving projects.

Readers can benefit most from this paper if they have at least a basic knowledge of data archiving and system tuning (system performance). These topics are dealt with in a comprehensive way in the books “Archiving your SAP Data” (Helmut Stefani, ISBN 1-59229-008-6) and “SAP Performance Optimization Guide” (Thomas Schneider, ISBN 1-59229-022-1), which we recommend as additional reading.

The goal of this paper is to enable readers to increase system performance by specifically leveraging the opportunities data archiving offers. This is particularly the case if the system has already been optimally tuned using conventional methods, and if there is no chance of further increasing system performance. Data archiving can help you achieve this goal, at least if certain prerequisites are fulfilled.
2 From Performance to Data Archiving

2.1 Performance Monitoring in mySAP ERP

The term “performance” denominates the capability of a data processing system of meeting predefined criteria with respect to the efficiency of the system. To be more precise, performance characterizes the response time behavior of the system during the (interactive) user dialog, background processing or other task types.

In an SAP system you can obtain a first overview of the various components that make up the response time by using transaction ST03N (see Figure 4:).

![Figure 4: Transaction ST03N](image)

Here, among other data the following performance KPIs are displayed, which are collected in the ABAP kernel during the runtime of the application programs:

- Average response time
- Average CPU time
- Average DB time
- Average DB procedure call time
- Average wait time
- Average roll in time
- Average roll wait time
- Average load and generation time
• Average lock time
• Average RFC interface time
• Average front-end network time
• Average GUI time

ST03N delivers a lot more details on the various performance aspects of an SAP system than we actually need for our discussion. The main role of the collected information is to help administrators set up the SAP system with respect to the number of work processes and the memory management for optimal use of the available hardware. Before using data archiving as a measure to increase performance, make sure these parameters are all set to reasonable values. You can find more information on these performance KPIs in the book “SAP Performance Optimization Guide” (see above).

However, even for well-tuned systems the collected information can help you locate performance bottlenecks in the following areas (the main indicators required are given in brackets):

• Front-end CPU / memory (average GUI time)
• Front-end communication (average front-end network time)
• Application CPU (average CPU time)
• DB CPU / I/O (average database time)

To be accurate, we must say that the figures given for the average CPU time and the average DB time do not match the figures that one would measure in a client/server environment with separate application and database servers using tools of the operating system. Apart from the CPU time that belongs directly to the application and which is listed in ST03N, the performance of the application server CPU is also affected by a part of the DB time. The reason for this is that, from an application point of view, the time for the database interfaces running on the application servers is already counted as DB time.

This database layer uses a sophisticated buffering concept that enable SAP applications to read a considerable amount of the data from buffers on the application, and not always straight from the database. These buffer accesses are usually more than ten times faster than accesses that are routed to the database. While this concept leads to a relatively large difference in the number of database accesses – a figure also displayed in ST03N – compared to the number of accesses measured by the database monitors, its effects on time are relatively small.

Apart from the figures discussed above, ST03N also lists the load and generation times as well as the roll-in time. Loading is the initial copying of user-specific data of a program from the program buffer into the user context. Generation time is the time needed to copy the program from the database to the program buffer of the application. The roll-in time specifies the time that is needed after a work process change, to copy the user context from the “roll area” back into the process-own memory of the work process. The corresponding “roll-out” time for copying the user context from the process-own memory into the roll area is not part of the user response time, and is therefore not listed in ST03N.

RFCs that an application may call play a special role when taking a closer look at the performance on the application and database side. As the target of these RFCs is very often another application server on the same system, additional CPU time will be used on the application server. Another point to consider is the database accesses that are possibly performed from within the coding processed during an RFC call. It is therefore not possible to allocate the time needed for these accesses to either the database or the application server.

2.2 Monitoring Database Performance

The database performance of a system is not only a decisive factor in the discussion about data archiving and performance improvements. It is true that in client/server environments the resources on the application and front-end side are infinitely scalable. It is also true that the performance of the database can continuously be adapted to the growing needs of a company through the addition of ever more powerful multiprocessors. However, there are several considerations that should not be neglected if you want to exploit the CPU potential of your servers to the fullest. For one, certain processes on the database must run in a serialized manner to ensure proper transaction processing. On the other hand, the I/O subsystem must also have sufficient potential to be able to provide the CPUs with data fast enough. Therefore, databases
must have a large data buffer in the memory. If the quality of this buffer is less than excellent, then the I/O subsystem is the decisive factor (size) for the performance and scalability of the entire system.

Let us clarify this concept by using an example: a select on a data record in a table. The WHERE condition is to be displayed entirely using an index which has a depth of four blocks. This means that for the entire process four index blocks and the data block of the table must be read. Experience has shown that this kind of a select can be executed ten times faster if all necessary blocks are already in the buffer, instead of having to be read from the database. If a total of 20 such selects have to be executed and of the 100 data blocks that are needed only two have to be read from the database, then in the total response time of the database the I/O part would already be as high as the part for the CPU. If the buffer quality of your system is less than 98%, it would be more beneficial to invest in the improvement of the I/O – that is, in the expansion of the database buffer – than in the performance increase of the CPU.

The factor 10 difference between the time needed for a database access without I/O and an access where a block is read, depends on the performance potential of both the used CPU and the I/O subsystem. However, in most systems this difference will not deviate too much from our assumption of factor 10. Even if the database accesses are not indexed single data record accesses, the conditions are similar. Only in the case of a full table scan are databases able to improve the I/O time per block through a multi block I/O, although only at the cost of a buffer quality of 0%.

The buffer quality and other indicators for the performance of the database can be monitored in the SAP system using transaction ST04. The transaction only displays indicators offered by the different databases and is therefore dependent on the database system being used. Figure 5 shows an example of an IBM DB6 database, Figure 6 of an Oracle database.

![Figure 5: ST04 in a DB6 Database System](image)
In both systems the buffer quality is roughly at a point which we have estimated as the break even point for the relationship between CPU and I/O, meaning that the system configurations are relatively balanced with respect to the available resources. Note that in the case of the Oracle database a buffer of 21 GB has been created in the memory. In the case of Oracle, the buffer quality for the index and data parts of the tables is shown together. In the case of DB6 the buffer data for the index and the data parts of the tables is displayed separately. As expected you can see in the latter case that the buffering for indexes is noticeably higher than the buffer quality of the data.

Another important indicator shown under Oracle is the value for buffer busy waits. Depending on the tasks a system must handle in parallel, the I/O waits in the system can increase considerably when the buffer quality drops. In Oracle databases the processes wait actively, meaning they use up CPU, if they are trying to read a data block that was already requested by another process. The number of buffer busy waits increases quadratically to the number of necessary physically read operations.

Although the reduction of the buffer quality is linear to the increase in database size, the effects on performance can be markedly non-linear (Figure 7). This is especially the case when the system is under heavy load. In order for your database operations to continue smoothly, it is necessary for you to react before these non-linear effects on performance become too severe. It therefore makes sense to track the history of the indicators we have mentioned and of the database size. This will allow you to recognize performance losses even if they are developing slowly, and help you take action before the state of your system becomes critical.
2.3 Performance and the Role of Data Age

To understand the role of archiving for performance, let us distinguish between “new” and “old” data, with old referring to data that is no longer needed for business purposes. In this paper, when talking about old and new data we always refer to transactional data, that is data pertaining to business objects, such as customer orders, billing documents, material movements, or financial documents. The ageing process of this type of data is more or less continuous. Ageing, however, is not only a matter of transactional data, it also affects master data, such as product or business partner data. Outdated master data records that are no longer used can also be referred to as old. However, in contrast to transactional data the ageing process of master data is not continuous. We will therefore discuss this type of data in a separate section.

Based on this definition we can assume, then, that all queries executed only access new data. This also holds true for reporting if the date after which the data is considered to be old is far enough in the past. If you want to call up a list of material documents for a specific material, it would not make sense, from a business perspective, to create a list of all the documents in your company if it was founded in 1870. This type of a query would clearly only be executed in rare cases and by users who are not familiar with how to restrict data searches. It is safe to say that any user who has had to wait for several minutes in front of his or her terminal to be presented with a list of documents from 1870 until now (assuming that all of the company's data has been transferred to a modern database system), will specify a time range the next time he or she starts a query. This does not only have to do with runtime, but also with the simple fact that the list contains mostly information that is not needed at all, which makes it difficult to find the useful information. On the other hand, it is realistic to say that not all users in a company know how to most optimally search for data by using time restrictions, and some will always execute inefficient queries. Even the very experienced users may accidentally repeat an error. In other words, although in theory it would make sense that all queries are made in the most efficient way possible, reality is a different matter.

With this in mind, we can say that in general the removal of data from the database from which the list is being created helps reduce the effect of inefficient queries, by enforcing a maximum range of data available for access during reporting. In this respect data archiving can help minimize the effects of user errors by purging all system resources, from the front-end to the database, of old data. This not only benefits the user who placed the inefficient query in the first place, but can also improve performance for other users who are concurrently working in the system.

Even if database queries for which neither an implicit nor explicit time limitation is used are considered to be errors, in practice they cannot be avoided completely. Assuming that the number of such inefficient queries
and the data volume increase almost proportionally to the number of users, then the share of these queries of the entire resource consumption increases about quadratically – at any rate the increase is disproportional to the system size. This clearly shows that the deletion of old data is imperative for maintaining system operation at a reasonable level. We also see, however, that in the case of sound queries the deletion of obsolete data can at best only ease the pressure on the database, but it has no influence on the resource consumption at the application, front-end communication, and front-end level. Chapter 3 will therefore examine more thoroughly in which cases old data that is no longer used in queries affects performance.

2.4 From the Deletion of Old Data to the Archiving of Business-Complete Data

In reality it is most likely that your company would have deleted the material documents from 1870 long ago. If not, it would be a good idea to delete it now, without archiving this old data. It is possible to generate considerable performance gains by looking closely at the data in your system and moving the point in time or the date after which this data is considered to be old and deletable, closer to the present. There comes a point, of course, when this date cannot be moved up any more, because the possibility that the data may still be needed by different business applications increases. For a lot of data there also exist legal requirements that regulate the amount of time this data must be kept in the system. However, in order for you to generate the necessary performance gains you may need to remove more data from the database than allowed by such requirements. Because you cannot delete it, data archiving would be a useful alternative. During data archiving the data is copied to an archive before it is removed from the database. The archived data can then be used during reporting, in the few instances, that this data is still needed.

A point to consider when you decide to archive data is that, although it can still be accessed, archived data cannot be changed anymore. Thus, before the data is archived you need to be sure that it is “business complete”, meaning that it will no longer be needed or modified by the applications. How and when data is considered to be business complete or old depends on different factors, such as what type of data it is and in which application it was created. For example, the time when a business process is completed may vary widely from one sales document to another. In addition, when data can be considered old and how long it needs to remain in the system after it has become business complete largely depends on the sophistication of the tools used to access the archived data and how well this helps you meet your reporting requirements.

These considerations clearly show that it is important to closely analyze the nature of your data, so that you can determine which data can be removed from the database and whether this data still needs to be accessible or not after it has been removed. This will require a number of different actions and tools, such as the setting of residence times, possibly the choosing of appropriate selection criteria, and deciding what kind of tools you need for displaying the archived data.

2.5 I/O Rate – The Performance KPI Most Affected by Archiving

Figure 8 shows the situation in a system with an occupied disk space of about 2 TByte. Even with a size of 21 GB the database buffer only amounts to about 1% of the data volume. The high buffer quality of 98% can therefore only be obtained if there is a relatively high locality of the data during the queries.

As we have seen, the age of the data can affect how often it is accessed and (as to be demonstrated in Chapter 3.2) the different distribution of archivable data on the blocks of chronologically and non-chronologically sorted indexes, as well as in the data part of the tables. As a result, the contribution of the different accesses to the buffer quality can vary considerably. To better examine this let's take a look at the accesses that caused the largest number of (synchronous) I/O operations in our example system. This can be done by evaluating the statistics of the statements stored in the shared cursor cache in transaction ST04 (for Oracle database systems) using the option Detailed analysis menu → SQL request. The statements that are relevant for the examination of the I/O rate and the buffer quality are sorted by the total number of disk reads performed per statement (Figure 8).
3 Taking a Closer Look at the Database

3.1 Performance Impact of Archivable Data on the Database

Our argumentation that data archiving plays only a minor role in the resource consumption of the application programs was based on the assumption that archived data is usually not selected by the programs. But, under the condition that the select is optimally supported by an index, even the resource consumption of a select should always depend on the amount of returned data and not on the data volume stored on the database.

Whenever performance depends heavily on system size, often times poor indexing is the likely culprit. However, keep in mind that improving the indexing of a system is not a cure-all. The following aspects must be considered:

1. An optimal index must cover all accesses to a table and must also consider the database changes. Every index, for example, increases the time needed to perform an insert into the database. This means that an index setup that may seem optimal from a general point of view, may not be optimal for individual accesses. In certain cases it may therefore be more useful to perform less frequent accesses either via full table scans or via index range scans whose runtime is proportional to the total amount of data and can therefore be directly influenced by archiving. In such a case data archiving can be used to reduce the I/O load and the CPU consumption of the database.

2. Very often, when performing an access on several tables that are connected via a join, the selection fields are distributed over these tables. Creating a simple index on the individual tables would only help to achieve an optimal access path in rare cases. In this scenario, data archiving also has an effect on the I/O load and the CPU consumption of the database.

3. Even if the access is supported by an optimal index, more data is read into the memory than specified in the set of hits. The reason for this is that I/O operations are always performed blockwise and not by records. From the point of view of database performance it makes a great difference whether the data in a block is archivable or not. If it is, chances are high that it will not be required by other business processes. If it is new, it will most probably be needed by subsequent data accesses, which can then be

Figure 8: Statistics of Shared Cursor Cache (Oracle)
performed I/O free. In the first case data archiving can prevent the loading of unneeded data into the buffer. In the second case, however, data archiving would not affect the I/O load of the database.

To analyze the last point more thoroughly, let’s take a look at how data is stored in the blocks. A data block contains several rows of table entries, and an index block contains index information from a large number of table rows. In the case of the index block, when a new data row is inserted the insertion always takes place in those blocks of the affected indexes that are defined by the sorting order of the index. For the data block, data can be inserted in several different ways, depending on the database system used. In the next chapter we will take a closer look at the strategy for indexes and data blocks.

3.2 Usage of Data Block in Indexes

Indexes are sorted lists of some of the fields of a database table. They are used to speed up the search for data. A typical ratio for the amount of data stored in database tables and the data stored in indexes is 60 to 40 for SAP systems. Figure 9 shows a common example from a live SAP system. Under “Tables and Indexes” we can see the ratio of table data to index data. Based on the common 60/40 ratio, in this example, the amount of data in the indexes alone is so large that the necessary buffer quality cannot be achieved if we assume random access to all data in the indexes.

![Database Performance: Tables and Indexes](image)

**Figure 9: Tables and Indexes**

If we want to discuss the distribution of frequently accessed new data and archivable data for the indexes, we have to distinguish between chronologically sorted indexes and non-chronologically sorted indexes.

A **chronologically sorted index** is an index in which data is organized in reference to a time factor. This does not necessarily mean a specific date or time. Most often, chronologically sorted indexes organize documents according to when they were created. An index organized according to document number, for example, is a chronologically sorted index, in case the higher numbers are created at a later point in time. In chronologically sorted indexes old and new data does not reside in close proximity to each other. Therefore, it is unlikely that old and new data resides in the same data block.

A **non-chronologically sorted index** is organized in reference to a factor other than time, such as material number, etc. In such an index, data that is inserted into the table at a given time will as a rule be distributed over many data blocks spanning the whole width of the index.

It may well be that non-chronologically sorted indexes do contain some time-relevant information to be able to distinguish between old and new data. For example, the index could contain the booking date of a material
movement in addition to the material number. As long as the time information is not the first field in the index we can assume (without loss of generality) that all index blocks have to be accessed with the same frequency.

3.2.1 Chronologically Sorted Indexes

Assuming that only new (non-archivable) data is used in queries, we can see that only a small part of all index blocks, which contain the newest data, will ever be read into the memory of the database. All archivable data will reside in data blocks that are only rarely accessed. Archiving such data will free space on the disks of the database, but it will not have any influence on performance.

The following figure shows the data distribution on the index after some of the old data is archived.

![Figure 10: Data Distribution in Chronologically Sorted Indexes After Archiving](image)

The figure shows several data blocks of an index with three levels in the B-tree. The leaf blocks are located in the lower row. The index is chronologically sorted, which is indicated by the lower horizontal arrow depicting time. The newest data that was inserted is in the hatched green area on the right and is most often used in database queries. Before archiving, all leaf blocks were populated equally with data. The number of different data blocks frequently accessed in such an index is independent of the width of the index and thereby independent of the total amount of data it contains. During archiving there are some archivability checks for the data beside the criterion of age to make sure that the data is actually business complete. So, not all data of a given age becomes archivable at the same time. However, the percentage of archivable data increases with its age. The figure shows the distribution of free space after the archiving process.

In the case of chronologically sorted indexes the free space can only be used after the data blocks are completely empty, as only these blocks can be moved to other places in the index without disturbing the order of the data in the index. To be able to reuse the free space faster, the index needs to be reorganized after archiving. Like archiving the reorganization will have little effect on the performance of queries on this index.

3.2.2 Non-Chronologically Sorted Indexes

For our discussion of non-chronologically sorted indexes we also assume that all leaf blocks of an index are equally populated before archiving. The most recently inserted records that are needed for everyday business are distributed equally over the full width of the index. As the index width increases when more and more data is added to the system, the buffer quality for accesses to such an index goes down, eventually leading to a decrease in performance.
The chance to find an archivable record in a data block is independent from the position of the block in the index as well. After archiving, the index looks fragmented as shown in Figure 11.

![Figure 11: Data Distribution in Non-Chronologically Sorted Indexes After Archiving](image)

**Figure 11: Data Distribution in Non-Chronologically Sorted Indexes After Archiving**

In contrast to chronologically sorted indexes the free space in the index blocks can be used to insert new records into the index. As a result, the growth of the index is stopped. While this does not lead to a direct improvement of performance, the deterioration of the buffer quality and along with it the degradation of performance is stopped.

A reorganization of the index will reduce the size of the index, enhance the buffer quality, and thereby improve performance.

### 3.2.3 A Mixture of Chronologically and Non-Chronologically Sorted Indexes

The distinction between a chronologically sorted index and other types of indexes becomes somewhat more complex when we consider indexes in which the first field contains values that vary very little, and for the following fields the values are sorted chronologically. In such a case, only major parts of the index are chronologically sorted. However, for our discussion of the I/O behavior and the buffer quality the difference between indexes that are only partially and those that are completely sorted chronologically does not carry too much weight.

Also, the chronological order does not have to be very strict. Assume, for instance, that the index holds an integer or NUMC document number, and that a central number server is used. Number buffering in the application or just runtime effects with parallel use will lead to many instances where document numbers are not issued in a strictly chronological order. Also, if the index has only a rough chronological sorting the I/O shows the behavior we discussed for chronologically sorted indexes.

Figure 12 shows the relation of the age and the position of an index entry for the type of index described above. In general the clear distinction between chronologically sorted and non-chronologically sorted indexes may become difficult.

To be able to evaluate the behavior of a given index you will need to know the distribution of the different index types. This may seem more difficult than it actually is. In most cases having a rough knowledge of the data usage and a little bit of common sense, will be enough to carry out a sufficiently exact analysis.
Figure 12: Position in Index vs. Data Age of Data Records

Another aspect we would like to point out in this context is the use of GUIDs (Global Unique IDentifier) in indexes. Most GUIDs follow the RAW-16 format and are generated locally using the location (hardware ID) and time information. Standardized algorithms ensure that every GUID created is unique on a worldwide scale. However, there are slight variations of these algorithms that can lead to a different distribution of the location and time information in the RAW-16 field. Depending on the algorithm used, an index with GUIDs either has the properties of a chronologically sorted index, or it is completely unstructured in relation to time. This can lead to considerable differences regarding the performance and the effects of data archiving for such indexes.

3.3 Usage of Data Blocks in Tables

For the argumentation concerning database tables we have to distinguish between the following scenarios – depending on the database software used:

- **Scenario 1**: The data is stored in the same order as an index (usually the primary index).
- **Scenario 2**: The data is stored in data blocks that are determined by the use of freelists.

In **Scenario 1** we can say that the data belongs to the leaf block of the index. Therefore, our argumentation concerning the different behaviors of chronologically and non-chronologically sorted indexes can be applied directly in such cases. Access to tables that are clustered with respect to a chronologically sorted index does not show any performance degradation with the increase of the data volume in the table.

The primary key of transactional data very often contains a document number that increases (more or less) with time. Therefore, the performance of accesses to tables clustered with respect to such a key cannot be influenced by data archiving (with the exception of the effect of possible non-chronologically sorted secondary indexes in the data access.)

However, when the data is clustered with respect to a non-chronologically sorted index the performance degradation is much more pronounced. Here, the probability that a certain data block is needed in the database buffer is the same not only for all blocks that hold the index information, but for all blocks containing table data. This leads to a faster degradation of the buffer quality for such accesses.

Possible candidates for such tables are, for instance, tables clustered with respect to an index, which contains GUIDs generated at random (not in chronological order). The archiving of data from such tables will significantly reduce the degradation of performance with the increase of the data volume.
In Scenario 2, where the database uses freelists to decide where to insert the data, the argumentation is different. Starting from an empty table the data is put into the block at the beginning of the freelist until it is filled up to a predefined level. Then, the next block is filled. Typically, more than one freelist is used so that the data can be inserted into several blocks in parallel. However, even then the data in one block is created more or less at the same time. Thus, in this scenario we can say that the behavior during the table access can be compared to the behavior of accesses to a table clustered with respect to a chronologically sorted index.

However, especially when the blocks for the insertion of new data are maintained via a list of free blocks, there are several issues to consider about the reuse of the empty space created during data archiving. As with the data blocks of chronologically sorted indexes, all blocks that have been completely emptied can be reused without any restriction. You must remember, though, that half-empty data blocks are also put on the freelist and can therefore be used for the insertion of new data. This is an important factor to watch out for, if you are using other criteria in addition to the residence time to select the data to be archived. Depending on your selection criteria, your archiving session may cause fragmentation by leaving behind a lot of half-empty blocks.

![Figure 13: Data Block of a Table with Freelists](image)

The data in the hatched areas was inserted last filling the space that was previously emptied by archiving. Consequently, you will have blocks with both old and new data, which can lead to performance degradation during the insertion of new data, starting with an increased I/O load for the inserts themselves. This is because for storing a given amount of new records you need a greater number of half-empty data blocks than completely empty data blocks.

### 3.4 Special Considerations for Master Data

Until now our discussion of the distribution of the data was based on the assumption that the amount of data increases continuously with time. This applies to all tables in which data from business transactions, such as customer orders, deliveries, material documents, or documents from financial transactions, is stored. However, some tables are used for storing master data, such as data related to products, business partners, etc. The life cycle of this data is usually much longer than that of transactional data, and the frequency with which the data is accessed over a long period of time is very regular.

Making a clear distinction between master and transactional data is not easy and depends to a great extent on its definition in the individual business processes. The product information of a fashion clothes retailer, for example, may well have the access characteristics of short-lived transactional data. In contrast, the orders or contracts of an engineering company can reach a life cycle usually only attributed to master data.

But even master data is subject to changes that can cause the data to become obsolete, for example, if products are replaced from time to time by new, enhanced products. This does not only apply to fashion articles, but also to any other kind of products. As a result, the information on the old products is no longer or only rarely read.

It usually takes a long time until the growth of such master data reaches a magnitude large enough to merit data archiving in order to save disk space. However, in the context of our discussion about the chronological or non-chronological sorting order of data in indexes or table blocks, we must assume that as a rule master data is not chronologically sorted.
Even in the case of master data with a very low growth rate, we must expect that, due to the missing locality of the database accesses, the data accesses for all data blocks are much more homogeneous than for transaction data. Therefore, also the buffer quality for master data is worse than for transactional data.

### 3.5 Archiving vs. Index or Program Optimization

If examined individually, many of the expensive accesses can be sped up by applying relatively simple performance enhancement measures. This situation is clearly given for full table scans where in many cases the statement can be made considerably faster through the creation of an index to be used for the access. In theory, full table scans can be prevented by creating appropriate indexes for the queries that make sense for the business process, and by eliminating those queries that do not make sense. However, in practice it is difficult to completely eliminate all such statements.

More frequent than full table scans are searches where only a part of the WHERE clause can be evaluated using an index. Depending on the selectivity of the part of the WHERE clause that can only be evaluated after the table block has been read into memory, the performance of such statements can be increased substantially by adding fields to the index. If this would be done for all statements that can be found in a given system, one would end up with a rather complex index design.

However, creating an index to avoid expensive statements is not always the best solution if one considers the system performance as a whole. Creating indexes is also associated with costs: costs for disks to store the index information, and costs for maintaining the index in order to fight the performance degradation occurring during all inserts and during certain update operations. Moreover, practice has shown that if many indexes are created for a given database table the optimizer can become “confused” during a query. Thus, in reality an appropriate index design typically allows for some remaining full table scans – of course, none in the key business processes –, and several accesses that can only make limited use of an index.

#### Figure 14: Accesses to Table MSEG

The different factors that have to be considered when finding an optimal index design and the need to compromise is explained using the example of accesses to table MSEG.

Figure 14 shows an example of accesses to the table MSEG in a customer system. The different types of accesses are sorted by the overall number of disk reads that are caused by them. Clearly, per execution the most expensive statement is the view over MSEG and MKPF - the table containing header information of a
material document. The WHERE clause here only gives the material number MATNR and the client MANDT. This is an example of an incomplete WHERE clause, because there is no time range limitation given.

If a user reported any performance problems concerning accesses to table MSEG in this system, it was likely due to this statement. Whoever issued it had to wait a considerable amount of time. The first step to fixing this problem would be to either change the program or educate the user to specify a reasonable time range limitation. To obtain any useful results, it is necessary to enter at least the year, MJAHR, of the material document, or better still, the booking date. However, a look at the indexes on table MSEG (see Figure 15) shows that in this case, this would not have any influence on I/O at all: A select with MANDT, MATNR and MJAHR in the WHERE clause would use index MSEG=M. Because MJAHR is not included in the index, it has to be evaluated after the data is read.
A reduction of the I/O rate for this statement can only be obtained by creating another index. Such an index could be either MANDT, MATNR, MJAH or MANDT, MJAH, MATNR. The first alternative would have the advantage that the more selective field MATNR can be found in the index before the rather unselective field MJAH. This would be ideal for minimizing the effort to locate the desired data in the index. However, such an index would not be chronologically sorted. Thus, if the system has been up and running for a couple of years and the number of index blocks is growing, the buffer quality for such an index would go down. This would not be the case for the second alternative in which the year is in the index in front of the material number. Assuming that most requests are for material documents of the current year, the buffer quality would remain the same.

Before creating any of the indexes, however, we should remember that the overall influence on the system performance of the statement discussed so far is rather small, as it was only issued twice during the period monitored here. In this example the most expensive overall statement is the INSERT into MSEG. All indexes on the table have to be adjusted during an insert. Therefore, the number of data blocks that have to be touched is larger compared to a select that only needs to read one index (plus the table data). The I/O load caused by the inserts is therefore higher than that of the more frequent selects, which are overall the second most expensive statements. It will increase with every index added to the table. Even if the primary key is chronologically sorted (for example, material document number, which is an integer value increasing with time), the performance of the insert statement will degrade with the increasing table size, due to the presence of non-chronologically-sorted secondary indexes. As a result, the buffer quality decreases with the table size.

So, while the sorting order of the indexes is important here as well, more important is the overall number of indexes that has to be maintained. In this example it would be counterproductive for the system performance to try to speed up the long-running statements by changing the WHERE clause and adding an index.

This example shows that the effort to obtain the theoretically optimal index layout for a given system can be enormous and requires both regular monitoring and a lot of experience. An optimal index layout cannot be delivered by SAP or any other software vendor. It requires knowledge of the frequency of accesses for the different statements possible in the system. To be able to estimate the effect of the table growth on the buffer quality some basic understanding of the semantics of the fields, which are used in the WHERE clause, and the algorithms to determine their values is necessary.

Investing in a good index design can greatly reduce the influence of the system size on performance. However, even with an optimal layout performance will be somewhat affected. The decision of whether it is better to reduce database growth or aim for the most optimal index design to combat performance degradation depends largely on the size of the system and the available experience.

### 3.6 Optimizer Statistics and Database Parameters

Data blocks that have space for the insertion of new data of a table are administered by Oracle in a freelist. The data blocks are not completely filled during the insert operations. The PCTFREE parameter determines how much freespase is to remain if a block is taken from the freelist. This space is then available for update operations and is used to avoid “chained rows”. The PCTUSED parameter defines the maximum level that a block must exceed due to the deletion of data before it is available again for insert operations. The default values delivered by SAP are 10% for PCTFREE and 40% for PCTUSED. The Oracle block size in SAP systems is 8,192 bytes. You can save approximately 7,300 bytes in an Oracle initial block (8,192 bytes – 150 bytes administration data - 10% PCTFREE). If, on the other hand, you use a block with a fill level of 40% (3,200 bytes), then only 7,300 bytes minus 3,200 bytes, that is, 4,100 bytes remain for the inclusion of new table entries. In the case of heavy fragmentation, this means that the number of blocks used to save new data may increase by up to 50%. If you cannot use suitable measures to avoid fragmentation, then the mechanism described here may lead to a deterioration in performance when you add new data. You can solve this problem using two strategies. Both strategies require a once-off reorganization of the affected tables.

#### 1. Reducing the PCTUSED Value of a Block

If you reduce the PCTUSED value from 40% to 20% or 10%, the blocks are subsequently used (under the problematic conditions mentioned above) for insert operations after the archiving runs because the blocks must be further emptied to a considerable extent. In this case, a PCTUSED value of 20% corresponds to approximately 1,600 bytes and a PCTUSED value of 10% to only 800 bytes. Since the changed PCTUSED value only affects newly allocated extents, it is essential that you reorganize the entire table. Note that for an archiving that is based exclusively on chronological characteristics, the selected PCTUSED value only has a
very limited effect because the blocks are completely emptied. The disadvantage of this solution is that, in certain circumstances, it takes considerably longer until blocks from the delete operations of the archiving are available again for insert operations. As a result, tables with a lower PCTUSED value will generally remain somewhat larger. The increase depends on when the filling level falls below the defined value. If PCTUSED is defined too low, these blocks may never be used again in certain circumstances. Therefore, we advise against using a PCTUSED value lower than 10%.

2. Using Partitioning

A more interesting alternative is to use Oracle partitioning. Partitioning is especially useful for tables that have a direct or indirect time key. VBRK, VBRP and VBFA are examples of these types of tables. The indirect time key of these tables consists of the VBELN field (or fields associated with this, for example VBELN_V, VBELN_N, and so on). Since the entries in these fields are sequentially incremented with a sequential time, you can use them as a range partitioning key. The idea here is to create a separate partition for a specific area of the VBELN numbers (for example, for a month). In addition, the unique index of the tables is defined as "local", that is, each table partition contains only one specific sub area of the data (ideally, a month). If you now carry out archiving, the data is deleted from these partitions only. No other partitions would be affected by the archiving runs. In this case, half-empty blocks do not represent a problem because, due to the partitioning key, no more data arrives in these blocks. Since partitions are dealt with as independent storage objects within Oracle, these partitions can be easily deleted or summarized after archiving. If, in addition, the partitions are created in separate table spaces, the memory space can be released directly at file system level. The disadvantage of this solution is that you constantly have to create new partitions because the partitioning key continually changes. However, this is not generally a problem because the number of partitions is manageable and you can create a large number of partitions in advance. First, a reorganization is also required here to transfer the normal table into a partitioned table. In this case, you should also create partitioned tables in separate table spaces and use locally managed table spaces.

The methods shown above cannot solve the problems that arise when initial archiving is started too late. In this case you will be deleting more data than will be used for subsequent inserts in the same table, assuming archiving is performed regularly from that point in time. To make this space available for other tables, a reorganization must be carried out.

Cost-Based Optimizer – Database Statistics

Databases use optimizers, which for a given access determine the most economical access path, based on the statistics about the distribution of data in the table and especially the indexes. The selectivity of an index is estimated using the number of different values of the fields contained in the index. A decisive value among the determined costs is the number of necessary I/O operations. Optimizers tend to underestimate the locality of the successive data accesses in an SAP application and herewith also underestimate the buffer quality. This is not very surprising: To be able to differentiate between chronologically and non-chronologically sorted indexes, as we have done in this white paper, you not only need to know how the data is distributed, but also roughly the semantics of the fields. The optimizers do not have any knowledge of the latter. The results suggest that the algorithms are based more on an even access probability across the entire index and therefore on a worse buffer quality than is actually the case in the most common chronologically sorted indexes. In the case of smaller tables or when the selectivity of an index is not very high, optimizers tend more towards a full table scan with the behavior optimized for multi-block I/Os, than towards reading many blocks into the memory individually.

In other words, optimizers tend towards full table scans more often than is good for optimal performance. With Oracle databases, for example, this is frequently balanced through the use of the OPTIMIZER_INDEX_COST_ADJ parameter, with which you can influence the costs determined for index accesses in the comparison of table scans. Unfortunately, this parameter has the disadvantage that even if it is used you cannot determine the different buffer qualities in the case of several accesses. As for an optimal index design, you would need knowledge about the access frequency and the data locality across several accesses. In SAP systems a value of between 10 and 20 would make sense (see SAP Note 483995).

Because the optimizer tends towards multi-block I/Os, the system may run index fast scans after the OPTIMIZER_INDEX_COST_ADJ parameter has been used, instead of using the B-Tree in as targeted a fashion as possible. This will not allow you to take advantage of the buffer quality, and the access is still not the most optimal, even though the correct index was used.

Also, the optimizer’s tendency of preferring a multi-block I/O increases as the size of the table on which it bases its calculations decreases. If you refresh the optimizer statistics after archiving, the probability that it
reaches an incorrect result is greater than if you would let it work with the old statistics, where it would base its calculations on a table that is bigger than the actual table (after archiving).

In sum, we can say then that it is not necessarily useful or even desirable to refresh the optimizer statistics after data archiving or after data has been deleted from a table.

### 3.7 Index Fragmentation and Reorganization

Index reorganization is the most important procedure for improving the performance of the database system. To achieve quick database access, you need a high database buffer hit rate for the data that is being read. The probability that the required index data is already in the database buffer is almost 100 percent, assuming that indexes require little working memory.

![Index B-tree Diagram](image)

**Figure 16: Index Fragmentation**

If data is archived or even just deleted, gaps appear in the indexes of the respective tables (see Figure 16). The database might not be able to fill these gaps, because the prescribed index entry sequence determines the data blocks that are to be used for new entries. If no more space is available in those blocks, new data blocks must be used. Despite data archiving, the space required for the indexes continues to increase. This lowers the buffer hit rate, which leads to a noticeable decrease in database system performance. You should be able to remedy this situation by reorganizing the index (ALTER INDEX ... REBUILD ...). It is considerably faster to reorganize an index than to delete it and recreate a completely new index.

As we have mentioned, during data archiving the old data is removed from a data block. However, this does not mean that performance will automatically improve. A performance increase will only occur when this free space is used again by new data. One way to be able to reuse the freed up space is through a reorganization of the index data, which can be performed online and is therefore not very resource intensive. We recommend that you run an index reorganization especially after you have archived a large amount of data in one archiving session. However, even if you do not run a reorganization of your indexes, data archiving will still have a positive influence on your system, by stopping the further degradation of your system’s performance.

In the case of non-chronologically sorted indexes, as is shown in the previous graph, the performance impact of data archiving can be noticeable, because the space freed up by data archiving can be reused for new entries in the table without the need to reorganize the index. This is because the data in the data blocks is not bound by any chronological order or restrictions. Moreover, if you carry out data archiving on a regular basis and only a small portion of the data is archived in each session, you do not necessarily need to run index reorganizations to stabilize the performance for these types of indexes.
In the case of chronologically sorted indexes the benefits of data archiving are more noticeable in terms of disk space saved than in terms of performance during data accesses. One would clearly see an effect if data archiving would reduce the index size so much that the depth of the B-tree can be reduced. However, in practice the amount of data that can be archived is seldom large enough to achieve this.

Nevertheless, if you have half emptied data blocks in a chronologically sorted index, you must run a reorganization of the index to be able to reuse the freed up space again. This is because the insertion of data in a half empty block that has not been reorganized would clearly violate the chronological order of the index. Half empty data blocks in chronologically sorted indexes are usually the result of using other selection criteria in addition to residence time. On the other hand, it is also true that with these types of indexes the chance that data blocks are completely freed of all data is rather high, especially if you only use the residence time to select the data to be archived. Blocks that are completely freed can be reused for the insertion of new data without any problem.

In sum, we can say that the impact of data archiving on performance can vary, depending on whether or not you reorganize your indexes. The impact of data archiving can also vary depending on whether the index is chronologically sorted or not. In the previous paragraphs we tried to touch upon the different scenarios that have to do with index reorganization and performance after data archiving. In the end, however, we recommend that you always reorganize your indexes. Doing so does not require any great amount of resources, can help improve your system performance, and may provide you with a more efficient use of your freed up space.

3.8 Table Fragmentation and Reorganization

In contrast to the gaps in indexes, badly utilized space in data blocks affects performance only slightly. The probability that the badly utilized data blocks need to be loaded to the buffer is much smaller than with badly utilized indexes. Since table reorganization costs considerably more than index reorganization, you should reorganize tables only if the cost is justifiable.

In principle, tablespaces that are heavily fragmented after data archiving do not affect database system performance. Even though tablespace reorganization is therefore not necessary, it can still be used to

- Regain database storage space if data that is no longer required has been archived and the tables are subject to minimal change. However, if the archived tables are expected to grow at the same rate, then you should not reorganize the tablespace.
- Create storage space for other tables. Alternatively, you could assume that, after repeated archiving, the database system’s tools for managing free space will combine the gaps. Once these combined gaps are large enough, they can be reused for new data without tablespaces having to be reorganized.

3.8.1 Online reorganization

To guarantee high database system availability, even during reorganization, database manufacturers, third-party suppliers, and SAP offer programs for online reorganization, such as SAPDBA.

To summarize, you should optimize performance by always reorganizing the index after you have archived a large amount of data. You can use the average occupancy history of the database blocks to determine roughly when you should perform a table or tablespace reorganization.

Even without these additional reorganization measures, data archiving stabilizes the database size. Even if several archiving sessions are necessary, every data block should eventually be completely free of old data (and ready for new data). Unlike data archiving with subsequent reorganization, this eventual stabilization of the database size does not occur immediately, or at a higher level.

4 Setting Priorities for Archiving Projects

4.1 Performance vs. Database Space

The results of our evaluation (see chapter 1.2 ) can best be highlighted by comparing the archiving objects SD_VBRK and MM_EBAN. The main tables from which SD_VBRK archives data are VBRK (header table for
 invoices) and VBRP (item table for invoices). The main table for MM_EBAN is EBAN (item table for purchase requisitions).

Taking a closer look at the disk space occupied by these tables (transaction DB02) we found out that the tables VBRK and VBRP occupy a total of 12 GB of data, whereas EBAN occupies a mere 0.5 GB. Also taking into account the larger number of additional dependent tables for archiving object SD_VBRK, it was obvious that in terms of the database space used it was high time to start archiving invoices. But it did not seem to be necessary to worry about archiving purchase requisitions.

Using “SQL Cursor Cache” in transaction ST04 we determined the most expensive SQL statements for the main tables of the archiving objects using the total I/O rate as a sorting criterion. The I/O rate has the largest influence on the performance of a statement. Our findings were as follows:

**Tables VBRK and VBRP**

In the time range considered we found almost 18 million accesses to table VBRK that caused about 1.5 million I/O operations. In addition, there were about 12 million accesses to VBRP causing another 1.5 million I/O operations. All but one of these 30 million accesses that were found in the cursor cache used the primary key or any other chronologically sorted index of the corresponding table. The performance of this type of access cannot be influenced significantly by data archiving. One particular access that would benefit directly from data archiving caused 20,000 disk reads. In sum, one can say that the performance benefit gained by archiving invoices is limited to a small number of very expensive selects with only little influence on the overall system performance.

**Table EBAN**

In the same period of time there were only about 100,000 accesses to table EBAN. With 5 million I/O operations this caused a larger I/O load than the combined number of accesses to the invoice tables. Again we saw that most of the 100,000 accesses used chronologically sorted indexes and contributed only 20,000 disk reads, leaving about 1,000 queries to be responsible for the vast majority of the I/O load. The performance of these queries could be directly influenced by data archiving so that the archiving object MM_EBAN has not only a significant influence on those 1,000 queries but also on the overall I/O load of the system. To summarize, one can say that in terms of database space there was no reason for archiving purchase requisitions, whereas from a performance perspective this would have been a good idea.

In the case of chronologically sorted indexes the benefits of data archiving are more noticeable in terms of disk space saved than in terms of performance during data accesses. One would clearly see an effect if data archiving would reduce the index size so much that the depth of the B-tree can be reduced. However, in practice the amount of data that can be archived is seldom large enough to achieve this. The vast majority of queries use chronologically sorted indexes. In case of performance issues both index layout and the nature of the queries has to be examined.

To summarize, although fewer queries use non-chronologically sorted indexes, they can be extremely expensive. Data archiving can have an enormous impact on improving their performance. Through a noticeable reduction of the I/O rate, data archiving has an effect on overall system performance. In your archiving projects it may be necessary to choose different priorities and procedures, depending on the ultimate goal: improve system performance or gain database space.

### 4.2 Strategies to Minimize Fragmentation

To achieve the greatest performance improvements through data archiving, you must plan your data archiving strategy according to the data that exists in your system. SAP offers different tools to help you analyze your data.

**General Recommendations**

If you have never archived data from your system or have never deleted any large amounts of data, then we can safely say that fragmentation is not yet an issue. In this case you can achieve the biggest performance gains if you archive data mainly from tables where you have identified a high number of I/Os among accesses that take place using non-chronologically sorted indexes (or even full table scans), and see no possibility of reducing these I/Os by changing the index layout.

If you concentrate on tables that were identified as described, you can be assured that data archiving will help you keep one of the main performance bottlenecks in your system under control. However, in order to achieve the biggest gains, you should also analyze what selection criteria to use during archiving. In terms of
performance improvement for data accesses using non-chronologically sorted indexes, it is only important to remove as much data as possible. This stops the growth of the index without a reorganization, or reduces its size after an index reorganization is performed. To avoid fragmentation of the actual table, however, it is also of importance that all the data of a certain block is archived. Otherwise you will be left with partially filled data blocks after archiving, which can lead to a higher I/O during insert and subsequent read operations of new data.

Therefore, an archiving session is ideal if you select the data to be archived using only time-related criteria, and before archiving make sure that all data in the selected time frame is actually archivable. Unfortunately, this ideal situation is very difficult to achieve in a live system. It is often useful to archive at least part of the data in test mode, to see which archiving hurdles still exist for the data set you want to archive. If you then see that a large part of this old data is still not archivable, because, for example, the business processes were not closed properly, then it can pay off to correct these problems before running the archiving session.

Data Analyses with Transaction TAANA

Often times a company needs to archive data from different organizational units differently. In this case we recommend that you analyze the table contents to get an overview of how the data is distributed. This helps you determine the selection criteria to use for archiving and predict how much data can be archived using those selection criteria. Transaction TAANA can help you perform these types of table analyses. With this tool you can determine the distribution of data in a specific table using analysis variants. You can also save the results of the analysis for later use. The variants should be selected in such a way that they help you determine the data distribution across the most important organizational units and across time. However, the criteria should not return information that is too detailed, to avoid unnecessary data. For example, depending on the table, it may be sufficient to determine how many data records exist for a year or for different months (see Figure 17). Finding out how many records exist per day would be too detailed and not useful. You can configure the analysis variants to your liking, or you can use the standard analysis variants that are offered by SAP, which already contain the most important fields in the context of data archiving.

Figure 17 shows the result of an analysis carried out for table SWWWIHEAD (header table for work items) using the standard variant STANDARD. This variant contains the fields SMONTH and SYEAR and allows you to find out the distribution of data records per month. In the screenshot, the figures are sorted by the largest entries. You can see that the largest number of records, that is 1.098 from a total of 15.807 records stored in table SWWWIHEAD, is associated with May 2005.
If the data to be archived from a specific time period is distributed across organizational units that require separate archiving sessions, then it would make sense to try to execute the different archiving sessions in relatively short succession. This reduces the probability that the partially emptied data blocks will be used for the insertion of new data, and herewith helps avoid performance problems in the future (data insert and access operations on partially emptied data blocks are more performance intensive).

**Determining the Right Archiving Objects**

So far when talking about data archiving we have concentrated on individual tables that may affect performance due to a high I/O rate. However, data archiving does not take place on the level of individual tables, but rather based on objects. This means that during archiving, all entries that contain information belonging to one order, for example, are written together to the archive and then deleted, regardless of whether the entries reside in one or several different tables. For most tables it is unambiguous to which archiving object they belong. Order headers (table VBAK) and order items (table VBAP), for example, are archived using the archiving object for orders (SD_VBAK). You can use transaction **DB15** to find out these relationships, by determining the archiving objects for a specific table, or vice versa.

In some cases more than one archiving object is assigned to a table, as is the case for table VBFA. This table contains the predecessor/successor relationships between orders, deliveries and invoices, for example. The relationships are archived together with the predecessor, because the successor itself contains the information about its predecessors.

If you know the business process in question, it is not difficult to accurately estimate the contribution each of the involved objects makes to table VBFA. For example, in a simple document chain Order → Delivery → Invoice, 2/3 of the entries most probably form part of the order, because it has both delivery and invoice as successor documents. The remaining 1/3 is made up of deliveries, which only have invoices as successors. Because the invoice in this example does not have a successor, it does not have any entries in table VBFA. However, you can also use the table analysis function TAANA to determine more specifically, for example, how many entries exist for a specific time interval that can be attributed to the archiving objects involved. This is particularly important for determining how much of an influence those processes that are not exactly standard are having on performance.

Table VBFA is generally only accessed via the primary key, which is chronologically sorted via the predecessor document number. This means that VBFA is not one of those tables where you can expect great performance gains through archiving. However, through the archiving of every order, delivery or other possible documents in the document flow, data records are also removed from table VBFA. At the same time, it is clear that it is not possible to clear the table of large areas of data that falls into the same time interval, if you only use one object for archiving. To clear the table of data in this manner, you must archive orders and deliveries of a specific time period as successively as possible.

You may lose some of the performance gains you achieved through data archiving in other places in your database, if these tables are highly fragmented. However, you should only resort to technical countermeasures directly affecting the database if you are unable to avoid fragmentation through more optimal archiving and an adequate time-based grouping of archiving sessions. Nevertheless, this is not equally possible for all tables. Depending on the situation, it may be useful to adapt the storage parameters (such as PCTUSED), use partitioning, or run a reorganization, for those tables that are most heavily affected by fragmentation. If you need to reorganize your data, you should do so as quickly as possible after the deletion of a large set of data, to keep the number of simultaneously occurring insert operations in partially emptied data blocks as low as possible.

**5 Performance and Accesses to Archived Data**

**5.1 A Few Words on Data Archiving**

In the preceding chapters we made the simple assumption that “old” data is not used anymore in everyday business and from this derived that data age affects the frequency of accesses to data blocks. Not used in everyday operation of course does not mean that the data is not needed from time to time. A discussion on performance would be incomplete if we did not extend it to include the performance of accesses to archived data. The general belief here is that accesses to archived data are always much slower than accesses to
online data. A natural concern might be, that as a result the negative performance impact of such accesses – seldom as they may be – offsets all the positive aspects achieved for online data through archiving.

However, although accesses to archived data have very different performance characteristics than do accesses to online data, the effects are not all negative and in many instances performance can even be better. Of course, this depends strongly on the technology used for data archiving. Therefore, while the information presented in the previous chapters can be applied to any archiving scheme on relational databases, the information in this chapter holds true for data archiving with SAP.

For a better understanding, we will provide a short introduction to the SAP data archiving technology. Those familiar with the general features of SAP’s archiving concept can move on to the following chapters covering the special performance aspects of object-oriented storage and the Archive Information System. If you want more in-depth information on SAP data archiving, refer to “Introduction to SAP Data Archiving”, a guide that can be found on the SAP Service Marketplace under [data-archiving → Media Library → Literature & Brochures]. Additionally, you are advised to consult the general archiving and archiving-object-specific documentation in the SAP Library or SAP Help Portal.

5.1.1 Technology and Main Functions

5.1.1.1 Archiving Object

The archiving object is a central element of data archiving. It defines the unit that can be archived and deleted in the database as a whole. An archiving object represents a business object that has been enhanced to include the following components required for data archiving:

- Data declaration part
  
  Used to describe all relevant data structures and database tables that characterize an application object.

- Archiving programs
  
  The archiving programs comprise
  - the write program, which writes the business object data sequentially to the archive files;
  - the delete program, which deletes the data that was previously copied successfully to the archive file from the database;
  - the preprocessing program (optional), which prepares the data for archiving. This could include marking the data that is to be archived (setting a deletion indicator);
  - the read program, with which the archived data can be displayed;
  - the postprocessing program (optional), which is used for postprocessing after archiving, for example, to update statistical data;

- Customizing settings
  
  The Customizing settings describe the archiving-object-specific parameters that can be set for an archiving session. These settings vary, because they are aligned to suit the individual archiving objects.

![Figure 18: Components of an Archiving Object](image-url)
5.1.1.2 Archive Development Kit

The Archive Development Kit (ADK), which is supplied with the SAP NetWeaver Application Server, is the technical foundation for SAP Data Archiving and provides the basic services for developing and running archiving programs. ADK contains functions for administering archiving sessions, it provides the application programs with program interfaces, and deals with handling the archive files. In other words, ADK creates, opens, writes, reads, and closes archive files.

In order to ensure the long-term readability of archived data, ADK writes metadata (for example, data type, codepage, number format) to the archive file. The archiving programs themselves then do not have to worry about this step. Long-term readability includes the legibility of the bit sequences and the conversion of the code page, in case there are any deviations. ADK also carries out automatic structure conversions, if any changes were made to the database objects. ADK centrally controls access to the archive files. To do this, ADK writes management data (for example, number and size of data objects, logical file path, physical path name) to the database. This information is then used later to read the archived files from the SAP system.

Broadly speaking, ADK consists of an administration environment (transactions SARA and DB15), the ADK runtime system, and an environment for developing archiving objects and archiving classes (transactions AOBJ or ACLA). ADK-specific tables (which make up the ADK repository) are used to administer the management data and metadata in the same SAP database.

Archive Administration

Archive Administration (transaction SARA) is the main function used for most user activities in data archiving, such as the scheduling of write and delete jobs, building and deleting the archive index, or storing and retrieving archive files. It is the central component used to control and monitor archiving sessions and it is a standard user interface used by all archiving objects in the SAP system. The interface will only vary in the amount of programs the corresponding archiving object offers. Archive Administration automatically receives all relevant data and information about the archiving processes from ADK.

In addition, Archive Administration offers the following functions to be used by the archiving administrator:

- Archiving Customizing
- Overview of the archiving jobs
- Assigning tables to archiving objects (transaction DB15)
- Data archiving statistics
- Archive Information System

Figure 19 shows the entry screen of Archive Administration, as it would look using the archiving object FI_DOCUMENT.

![Figure 19: Entry Screen of Archive Administration](image-url)
ADK as Runtime Environment

Figure 20 shows the interaction between the database, archiving programs, ADK components, and file storage as a function of the data and control flow during the write phase (for information on the processes in data archiving, see “The Basic Data Archiving Flow”).

![Figure 20: ADK as a Runtime Environment for Archiving Programs](image)

The write program can be scheduled as a job via Archive Administration. Within the write program, an ADK call generates a new archiving session that is entered in the ADK repository. The archiving-object-specific application data that is to be read and checked for archivability is transferred record-by-record to the ADK, which then uses functions to bundle the data to data objects.

Other ADK-internal data object services transform and compress a completed data object into a release- and platform-independent format. Depending on the data composition, compression rates of up to 1:10 can be achieved – even more, if there are large numbers of initial values. On the other hand, data stored in cluster tables cannot be compressed any further. On average, compression factors between 2 and 5 can often be reached.

Before writing the first data object to an archive file, ADK first transfers the metadata from the ADK repository and ABAP dictionary (required for technically interpreting the archive files). This metadata includes name tabs of all tables and structures belonging to the archiving object.

When accessing the archive, the ADK runtime system checks whether the system environment has changed since the archive was written. If necessary, ADK will perform a platform adjustment (changes in the code page or number format) or schema adjustment (changes to tables and structures) to make sure the data can be interpreted. These conversions occur only temporarily when the read, delete, or reload programs are run – the archive files remain unchanged. However, if the system environment has changed to an even greater extent than described above, you can use ADK to implement special conversion programs for permanently converting archives.

Using the other runtime system services shown in Figure 20, ADK relieves the archiving programs of further technical and application-independent tasks. The ADK file management function automatically creates and names new archive files during the write phase, as soon as the limits set in archiving-object-specific Customizing have been reached. ADK also selects the file access path that is valid for the respective syntax group.

ADK as a Development Environment

In addition to its function as a runtime environment for all archiving programs, ADK also acts as a development environment. In essence, this environment includes an application programming interface (API) consisting of several released function modules. ADK makes the API available to all archiving objects. The
ADK API helps programmers to develop new archiving objects and expand existing ones. Even the archiving solutions provided by SAP are based on the ADK API.

SAP develops archiving solutions for standard SAP business objects. If you want to archive data from your own user-defined tables, you can use ADK to develop an appropriate archiving solution.

You can find a detailed description of the functional scope of ADK in the SAP Library, and in the documentation of the function modules. The SAP BIT670 training course provides information and examples for developing your own archiving solutions.

5.1.2 Processes in Data Archiving

5.1.2.1 The Basic Data Archiving Flow

The flow of archiving data is divided into the following main phases:

1. Writing data from the database to the archive
   
   The write phase is the starting point of the archiving process. It starts with the execution of the write program, which you can schedule through Archive Administration. Enter the desired archiving object, select a variant or create a new one, then set the start date and the spool parameter. When entering a variant, make sure that it has not been used in another write session. Otherwise, you run the risk of archiving data multiple times, which can lead to corrupted results during the evaluation of the archived data volume. If you select a variant that has already been used, Archive Administration will inform you.
   
   Based on an archiving object, the data to be archived is read in the database and is sequentially written to newly created archive files.

2. Deleting data from the database
   
   After the data has been completely written to the archive during the write phase, it has to be deleted from the database during the delete phase. The deletion is performed by the ADK delete programs, which you can schedule through Archive Administration. In order to ensure that the data has been properly archived, the delete program first reads the archive files created during the write phase, and then deletes the corresponding data from the database.

3. Storing the archive files (optional)
   
   Archive files created during the write phase can be transferred to a storage location outside the file system. Technically speaking, this phase is not a part of the data archiving process itself. This process is finished, when the archive files are created in the file system, and the data is deleted from the database. Therefore, it is the data archiving user who decides what to do with the archive files that were created.
   
   Usually, it is not enough to replicate application data in the archive, and then to remove it from the database. The archive files themselves have to be made available for potential access to the data saved therein. Usually, this requires the use of a storage system and, if the archive files are stored manually, the introduction of an operating concept for their administration and safe storage. Archive files can be stored on a storage system, on an HSM (Hierarchical Storage Management) system, or manually using alternative media, such as tapes, CR-ROMs, optical disks etc.

In order to avoid data loss due to errors during archiving, a two-step procedure is used. During the first step, the data is written to archive files. Then the data is deleted from the database, but only after the archive file has been completely written and at least the file header has been read successfully. An advantage of this procedure is that it detects possible data transmission errors over the network from the database to the archive. If there is an error, a new archiving session can be started, because the data still exists in the database.

Data can be archived during live operation of the system, that is, the user can continue to work with the system during archiving. However, this can lead to performance bottlenecks, especially during large-scale archiving sessions, because tables are being accessed for normal operations, while records are being deleted in the parallel archiving session. For this reason, data archiving should be carried out during periods of low system load.

Given the huge amounts of data processed during data archiving in parallel with live operation, it is also important to consider system and program performance. There are several ways of controlling the write and delete phases to ensure that your system resources are used as efficiently as possible. One option would be to trigger the delete phase at different points during the archiving session:
• At the same time as the write phase
• After the write phase
• At a later time (triggered manually or by an event)

Nevertheless, even if you spread the load over the aforementioned times, you should always plan to archive data at times of low system load.

The archiving operation – or archiving session – is not complete until the write and delete phases have been completed without error. Accordingly, an incomplete archiving session is signaled by a yellow traffic light in the archive management status field. This status does not change until the delete phase is completed, at which point the color changes to green. This status must be monitored, because if the session is incomplete, the data selected for archiving resides in the archive file and in the database once the write phase is finished. If one were then to start an evaluation program that accesses data from both the archive files and the database, the results, such as totals, could be falsified. Thus, despite the above suggestion regarding later file deletion, the time between writing the archive file and deleting the data from the database should be kept as short as possible.

5.1.2.2 Retrieval of Archived Data

Data that is no longer used for day-to-day business (because it originates from previous fiscal years) is usually accessed far less frequently than data from the current fiscal year. Nevertheless, one must also be able to appropriately display and evaluate this “legacy data”. This is especially important if the system is to be accepted by end users who are accustomed to having complete access to all data records, regardless of their age.

Technically, data that has been relocated from the database as a result of data archiving can be accessed in several ways, of which the following are the most common:

• Access to individual business objects
  With this type of access – also known as single-document or direct access – the desired document is presented in a way that is more or less identical to the original transaction.

• Sequential access to multiple business objects
  The main focus of such an evaluation lies on totaling or comparative operations, or on creating print lists of the evaluated data.

There are also different ways of displaying archived data, of which the following are the most important:

• Displaying data in the (original) application transaction
  This type of display is already available for several archiving objects. The user accesses the archived data via the same method and transaction as that used when accessing data in the database. The user is made aware of the origin of the data, that is, a dialog box informs the user that the display program is accessing archived data.

• Displaying data using the Archive Information System
  The display (and previous access) of archived data in the Archive Information System (AS) is based on an index of archived data, referred to as archive information structure. Data can only be found in the archive and displayed if this kind of infrastructure exists, and is filled with elementary information and details concerning the exact position of the individual business objects in the archive file.

• Displaying data using the Document Relationship Browser
  With the Document Relationship Browser (DRB) you can display individual business objects that are linked to each other. Usually, these are documents that were created during a common business transaction or that are part of a common process. The DRB also displays the entire business process to which the business objects belong. This allows you to completely reconstruct a process chain that may have been broken by data archiving, regardless of whether the data resides in archive files or the database. DRB can be used within the Archive Information System, or it can be accessed by means of the user role SAP_DRB. The most frequently used archiving objects are connected to DRB.

Another type of access that is available for some archiving objects involves reloading archived data into the database. However, you should be aware that this function is inherently risky. These risks are linked to the fact that a company's structure, and therefore its organizational presentation in the SAP system, is subject to
change over time. The larger the time interval between data archiving and reloading, the greater the probability that the original system has changed.

These changes can be purely technical in nature, for example, a change in the table structure on which a business object is based. Or, they can be organizational, for example, the sale or acquisition of a company division, or the introduction of a new local currency. Although the concepts on which data archiving are based support changes in the system environment, the software cannot compensate for some of these changes. As a result, inconsistencies may arise when the data is reloaded into the database. Therefore, reloading should only be attempted if the wrong data was accidentally archived, and this operation needs to be reversed.

5.2 Relational vs. Object-Oriented Storage

In a relational database management system (RDBMS) data is stored in the form of related tables. For example, data pertaining to a specific business object, such as a sales order or a billing document, usually spreads across several database tables. The sales order, for example, includes nearly 150 tables. The knowledge about the related tables is stored within the definition of the business object and, from an archiving perspective, also in the archiving object definition. Whenever a document is being processed, for example, displayed or changed, the database needs to access all the associated tables in order to extract the entire set of data that make up the document. This may also include accessing data not directly pertaining to the document, such as master or organizational data. Depending on the complexity of the document, that is, the number of database tables that need to be accessed, this process can be quite time-consuming.

![Diagram: Relational vs. Object-Oriented Storage of Data](image)

**Figure 21: Relational vs. Object-Oriented Storage of Data**

Data archiving changes the way the data is stored from a relational to an object-oriented storage (see Figure 21). This means that all the data that belongs to a particular document is stored in a “flat”, object-like rather than in a relational form. This results in some differences when you access the data. For example, archived single documents can generally be displayed faster than documents from the database because the data is available in a single location and does not need to be collected from several tables. On the other hand, there are disadvantages when you evaluate (report) archived data. However, data archiving is not designed for this type of data access. A specially developed solution for the drilldown reporting of historical data is provided for this purpose with SAP Business Intelligence.

Data archiving is used to extract the data for business-complete processes from the database and write this data to the file system. By helping to restrict the data volume in a system to a reasonable amount, data archiving contributes greatly to reducing the TCO of a system. A possible loss of accessibility as a result of archiving is accepted, since business-complete processes are no longer the focus of daily work. If the costs involved in accessing and displaying archived data were the same as they are for current data on the
database, the cost and performance advantages that result from data archiving would be partially or even completely negated.

5.3 Archive Information System

Clearly, the fastest access to archived data occurs when you can directly access the desired object in the archive and do not need to scan the archive or parts of it to find the correct object. For this purpose, SAP provides the Archive Information System (AS), a function for indexing objects in the archive. AS allows you to create tables containing freely configurable attributes of an object and store these tables in the online system. These archive indexes, in the SAP environment better known as infostructures, contain the archive file name and the offset of the file via which you can access the object itself. Infostructures function similarly to database indexes, because they enable a direct access to the archive. However, compared to the indexed accesses in a relational database, there are two differences that occur with indexed archive accesses, which can be used to optimize the relationship between cost and performance:

- Because the infostructures are separated from the actual data in the system, it is not necessary to index all business objects of one type equally.
- The infostructures are object indexes. In them you can easily combine attributes that would be stored in different tables on the database.

The first point is important, because for every infostructure that you create for archived data, you reduce the space gained in the database. If we consider that in the database of a typical SAP system around 40% of the space is used for indexes, then it is clear that you would gain considerably less space through archiving, if you would create the same amount of indexes for archived data as you had in the live system.

However, with each index that you leave off, you also lose the option of a fast and direct archive access; in other words, reducing the index volume will always carry with it a reduction in the access functionality for archived data. In many cases this is not necessarily something negative, because as data gets older, the frequency of accesses does not go down in equal measure for all access types. Some accesses, such as the accessing of data to create a work list of open documents, are simply not needed anymore for archived data and do not need to be offered.

On the other hand, it is also important to determine how best to index your archived data to ensure the most balanced relationship between cost and functionality. One strategy may be to use the Archive Information System to stagger the efficiency of the indexes according to the age of the archiving sessions and in this way, also choose the access functionality according to the age of the data. You would still be able to subsequently build more exhaustive indexes for older data, if you expect, for example, that you will need to access archived data of a specific time period more often, because of an audit.

Although the use of infostructures may reduce some of the space gained through data archiving, they have practically no effect on the performance gains achieved. As we have seen, old data affects performance mainly if it resides in a data block which also contains new data that is accessed frequently. This can be avoided by moving the old data to infostructures, even if these remain in the database.

Storing the infostructures in a file system would therefore only make sense from a database space point of view. From a performance point of view, you would not notice much difference. However, you would face the disadvantage of not having any database support for the infostructure accesses, which could lead to problems.

Also important for performance is the fact that attributes originally kept in separate tables can be combined in a single infostructure. For access to archived material documents, for example, you can create an infostructure containing the field MATNR (material number) from MSEG and the field BUDAT (booking date) from MKPF. Because through this the infostructure already contains the time information necessary to limit the selection to a specific time period. An access to archived material documents from transaction MB51 is often faster than from the database, where MATNR and BUDAT cannot be combined in one index.
6 Summary and Conclusion

With the growth of a database, the performance of a system can go down in different manners. Archiving is a good way to stop performance degradation, because it curbs data growth. This is particularly true when other optimization measures in terms of index design and data access have been exhausted.

The influence of the database size actually affects performance for only a fraction of the accesses and tables. However, this can be significant enough to play an important role in the operation of your system. At the same time, data archiving can also negatively affect performance on other tables and accesses, because it may have left behind some fragmentation in the database. As a result, overall system performance could suffer.

In sum, you can only achieve an optimal performance effect if you take into account how the age of the data influences the probability that it will be accessed and the block-orientated storage of data in the database. This helps you understand the effect of the database size on the different accesses, and develop strategies to use data archiving specifically in those areas of the database, where performance is most affected. It will also help you avoid or reduce the negative influence of table fragmentation in a more targeted manner.

If the goal of your archiving project is mainly to improve system performance, you should use different analysis methods than if your goal would be to reduce used disc space. Moreover, the objective of your archiving project determines which archiving objects you should use first, which selection variants to opt for, when to archive and which follow-up tasks you need to carry out for the administration of the database. By leveraging the mechanisms described above, data archiving directly affects the performance of a database system. This effect can further be increased by performing a reorganization of the indexes and tables, but can also be noticed without these measures.

In summary, one can say that through data archiving, performance benefits can only be obtained if the following criteria are fulfilled:

- The database is optimally configured.
- Table accesses are performed using an optimal index.
- The database access time makes up the largest part of the response time.
- During database access the DBMS is mainly busy analyzing a large number of database blocks.

To check whether these criteria are met you need to analyze the SQL trace. Data archiving has the largest performance impact on a badly tuned system that contains many statements with incomplete WHERE clauses and missing indexes.

In your archiving projects it may be necessary to choose different priorities and procedures, depending on the ultimate goal: improve system performance or gain database space.