Information Warehouse Storage Management
Guidelines and Considerations

Document Number GG24-4336-00

April 1994

International Technical Support Organization
San Jose Center
Take Note!

Before using this information and the products it supports, be sure to read the general information under “Special Notices” on page xiii.
Abstract

This document describes some of the considerations pertaining to implementation of large Information Warehouse databases and decision support systems. It provides some guidance on tools, techniques, products, and offerings that are available to assist in the implementation of an effective Information Warehouse Architecture, particularly for large volumes of data.

This document is written for systems designers, application designers, and technical architects involved with the implementation or support of large informational databases. Some knowledge of IBM relational products such as DATABASE 2 and an understanding of the concepts and potential uses of informational databases are desirable.

DS ST (139 pages)
Contents

Abstract ................................................................. iii

Special Notices ........................................................... xiii

Preface ........................................................................ xv
How This Document Is Organized ........................................ xv
Related Publications ....................................................... xvi
International Technical Support Organization Publications ...... xvi
Acknowledgments .......................................................... xvi

Chapter 1. Information Warehouse Framework .............................. 1
1.1 The Problem ......................................................... 1
1.2 A Solution: The Information Warehouse Framework ............... 2
1.3 Information Warehouse Architecture ................................ 4
1.4 Information Warehouse Framework and Storage Management .... 5

Chapter 2. The Storage Hierarchy ............................................. 7
2.1 Storage Product Positioning .......................................... 7
2.2 3390 Model 9 and 3995 Model 153 .................................. 9
   2.2.1 3390 Model 9 ............................................. 9
   2.2.2 3995 Model 153 ........................................ 10
   2.2.3 Summary .............................................. 14

Chapter 3. Data Storage Selection Based on Access Requirements ...... 17
3.1 Identifying Classes of Data ........................................... 17
   3.1.1 Classes of DB2 Data .................................... 18
   3.1.2 Mapping Classes to Devices ............................ 20
3.2 Data Facility Storage Management Subsystem ......................... 22
   3.2.1 Introduction to System Managed Storage ................. 22
   3.2.2 Implementation of DFSMS and DB2 .................... 25
   3.2.3 Conclusions .......................................... 27

Chapter 4. DataPropagator Relational ......................................... 29
4.1 Product Function ...................................................... 29
4.2 Implementation ...................................................... 30
   4.2.1 Capture and Data Staging ................................ 31
   4.2.2 Applying the Changes ................................... 33
   4.2.3 Administration .......................................... 33
   4.2.4 Update and Refresh Considerations .................... 34
4.3 DataPropagator NonRelational ....................................... 35
4.4 Summary ........................................................... 35

Chapter 5. The Role of Data Compression .................................... 37
5.1 DB2 Data Compression .............................................. 37
   5.1.1 Dictionaries ............................................. 39
   5.1.2 Compression Statistics ................................... 39
5.2 Benefits and Considerations ......................................... 40

Chapter 6. Data Rolling or Wrapping ........................................ 43
6.1 The Problem ........................................................ 43
6.2 Partition Wrapping: A Possible Solution ............................. 45
| Chapter 6.2 | Implementation of Partition Wrapping | 45 |
| Chapter 6.2 | Example: Electronic Equipment Manufacturer | 47 |
| Chapter 6.2 | View Creation Considerations | 49 |
| Chapter 6.2 | Table Maintenance | 49 |
| Chapter 6.3 | Benefits of Partition Wrapping | 51 |
| Chapter 6.4 | Summary | 53 |

| Chapter 7.1 | Functions | 56 |
| Chapter 7.1 | Archive | 57 |
| Chapter 7.1 | Storage Management | 59 |
| Chapter 7.1 | Retrieve | 59 |
| Chapter 7.1 | Archive Maintenance Utility | 60 |
| Chapter 7.2 | Capacity and Performance | 60 |
| Chapter 7.3 | Recovery and Stability | 61 |
| Chapter 7.4 | An Implementation | 61 |
| Chapter 7.5 | Identifying the Need for Data Archive Retrieval Manager | 63 |

| Chapter 8.1 | TEAMS Central Information System (CINFO) Description | 65 |
| Chapter 8.1 | Technical Architecture | 65 |
| Chapter 8.1 | Design and Implementation | 65 |
| Chapter 8.1 | Business Data Warehouse Layer | 66 |
| Chapter 8.1 | Process Data Warehouse | 68 |
| Chapter 8.1 | Implementation of User Tables | 72 |
| Chapter 8.2 | Conclusion | 74 |
| Chapter 8.3 | The Future | 75 |

| Chapter 9.1 | Summaries or Aggregates | 77 |
| Chapter 9.1 | Multiple Levels of Summary | 78 |
| Chapter 9.1 | Drill-Down Example | 79 |
| Chapter 9.2 | Summary Tables | 80 |
| Chapter 9.2 | Simple Cumulative Summary | 80 |
| Chapter 9.2 | Multilevel Cumulative Summary | 81 |
| Chapter 9.2 | Rolling Summary | 82 |
| Chapter 9.2 | Simple Direct Summary | 84 |
| Chapter 9.2 | Continuous Summary | 85 |
| Chapter 9.2 | Sample | 87 |
| Chapter 9.3 | Concluding Remarks | 88 |

| Chapter 10.1 | Design Steps for Informational Tables | 91 |
| Chapter 10.1 | Step 1: Remove Purely Operational Data | 91 |
| Chapter 10.1 | Step 2: Add One or More Elements of Time | 93 |
| Chapter 10.1 | Step 3: Add Derived Data | 94 |
| Chapter 10.1 | Step 4: Add Artifacts of Relationships | 96 |
| Chapter 10.2 | Summary Tables and Indexing | 99 |
| Chapter 10.2 | Summaries: Evaluating Alternatives | 99 |
| Chapter 10.2 | Indexes for Access | 104 |
| Chapter 10.2 | Other Considerations | 105 |
| Chapter 10.3 | Conclusion | 105 |

<p>| Appendix A. | An Example of Partition Wrapping | 107 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix B. Additional Coding Examples</td>
<td>123</td>
</tr>
<tr>
<td>B.1 Adjusting the Partition Number</td>
<td>124</td>
</tr>
<tr>
<td>B.2 REXX Example to HMIGRATE Partitions</td>
<td>130</td>
</tr>
<tr>
<td>B.3 Job to Move Tablespace</td>
<td>133</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>135</td>
</tr>
<tr>
<td>Index</td>
<td>137</td>
</tr>
</tbody>
</table>
Figures

1. Data Types within an Information System .................................. 3
2. The Information Warehouse Architecture I .................................. 5
3. Device Performance and Capacity Comparison ............................ 8
4. 3995 Models 153 and 113 ...................................................... 11
5. 3995 Model 153 Components .................................................. 12
6. Modeled Average Disk Mount Response Time .............................. 14
7. Data Set Response Time Range ............................................... 20
8. USING Block of CREATE TABLESPACE ................................... 23
9. Overview of DataPropagator Relational .................................... 31
10. DataPropagator Relational Implementation ............................... 32
11. Refresh and Update Considerations ........................................ 34
12. We Always Need to Store More Data Than We Have Space For ...... 37
13. The Compression and Decompression Life Cycle ....................... 38
14. Access Frequency over Time ................................................ 44
15. Possible Data and Index Implementations ................................ 45
16. Wrapping Lookup Table ...................................................... 47
17. Sample View Creation ........................................................ 48
18. Access Frequency and Partition IDs ....................................... 52
19. Data Access over Time ........................................................ 55
20. Data Growth ........................................................................ 56
21. Data Archive Retrieval Manager ............................................. 57
22. The Mean Data Volume Can Be Automatically Maintained .......... 58
23. CINFO BDW Component ....................................................... 66
24. CINFO PDW Component ....................................................... 68
25. Example of a DEM Configuration .......................................... 70
26. Extract and Load Process ...................................................... 72
27. Processing an Extract File ..................................................... 73
28. Example of Drill-Down Analysis ............................................. 79
29. Simple Cumulative Summary ................................................ 80
30. Multilevel Cumulative Summary ............................................ 81
31. Rolling Summary .................................................................. 83
32. Simple Direct Table ............................................................. 84
33. Continuous Summary Table Derived from Two Simple Direct Tables 85
34. Continuous Summary Table Derived from Previous Summary and Additional Simple Direct Data ........................................... 86
35. Sample of Unsummarized Transaction History Data .................. 87
36. Examples with Orders and Line Items ...................................... 92
37. Addition of Data Elements to Provide Time Basis ...................... 93
38. Addition of Derived Data to the Informational Table .................. 94
39. Addition of Artifacts of Relationships to Informational Table ...... 96
40. Alternative: Use Two Informational Tables instead of One .......... 97
41. Retail Credit Purchase History Informational Data: Unsummarized 100
42. Summary Table and Indexes ................................................... 101
43. Alternative Summary Table and Index ....................................... 102
44. Summary to Accommodate Different Types of Users (but with reduced detail) .......................................................... 103
45. Load Cards for LOAD Replacing a Partition .............................. 115
Tables

1. Storage Device Performance and Capacities ........................................... 9
2. Media Suitability Matrix ................................................................. 15
3. Data Set and Device Suitability Matrix ........................................... 21
4. Telephone Data Summarization .......................................................... 77
5. Multiple Level Summary ................................................................. 78
This publication has been written to help systems designers, application designers, and technical architects design and implement Information Warehouse databases. The information in this publication is not intended as the specification of any programming interfaces that are provided by Database 2, Query Management Facility, Data Refresher, Data Propagator Relational, DFSMS/MVS, the IBM 3995 optical dataservers or the IBM 3390 storage devices. See the PUBLICATIONS section of the IBM Programming Announcement for these products for more information about which publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book is limited in application to those specific hardware and software products and levels discussed.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

DATABASE 2  
DB2  
DFSMS  
DFSMSSdss  
Distributed Relational Database Architecture  
DRDA  
IBM  
Information Warehouse  
QMF  

DataHub  
DB2/2  
DFSMSdfp  
DFSMShsm  
DProp  
DXT  
IMS/ESA  
OS/2
Preface

This document is intended to assist in the design and implementation of large informational databases as part of an Information Warehouse architecture. It contains information about various products, tools, techniques, and offerings available to assist in implementing systems based on IBM’s Information Warehouse framework.

This document is intended for systems designers, application designers, and technical architects involved in the implementation or support of large decision support systems and informational databases.

How This Document Is Organized

The document is organized as follows:

- Chapter 1, “Information Warehouse Framework”
  This chapter describes the role of the Information Warehouse framework by explaining some of the problems in obtaining information from the many data sources within organizations and how the framework attempts to solve them.

- Chapter 2, “The Storage Hierarchy”
  This chapter looks into some of the storage media that are currently available to installations for storing DB2 data. Emphasis is placed on higher volume, lower cost devices.

- Chapter 3, “Data Storage Selection Based on Access Requirements”
  This chapter considers the storage requirements of various components of an informational decision support system.

- Chapter 4, “DataPropagator Relational”
  This chapter describes the role of the DataPropagator Relational product and how it works.

- Chapter 5, “The Role of Data Compression”
  This chapter discusses the implementation of data compression made available with DB2 Version 3.

- Chapter 6, “Data Rolling or Wrapping”
  This chapter looks at a technique of handling large volumes of historical data efficiently.

- Chapter 7, “Data Archive Retrieval Manager”
  This chapter looks at a product offering that supports the archival and deletion of old data automatically.

- Chapter 8, “Decision Support System: Implementation Example”
  This chapter describes an example of an Information Warehouse implementation, including the components, architecture, and techniques used.
• Chapter 9, “Database Design Considerations”
This chapter looks at the issues related to the design and structuring of informational databases, emphasizing data summarization, aggregation, and sampling.

• Chapter 10, “Table Design Considerations”
This chapter describes additional design considerations for individual informational tables.

Related Publications
The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

• DB2 Version 2 Release 2 Design Guidelines for High Performance, GG24-3383
• DB2 Version 2 Release 3 Administration Guide, SC26-4888
• Information Warehouse: An Introduction, GC26-4876
• Information Warehouse Architecture I, SC26-3244
• IBM 3995 Optical Library Dataserver Family, G326-0118

International Technical Support Organization Publications
A complete list of International Technical Support Organization (ITSO) publications, with a brief description of each, may be found in:


Additional ITSO publications related to this document include:

   DB2 Usage in the DFSMS Environment - Part 1, GG24-3317
   DB2 Usage in the DFSMS Environment - Part 2, GG24-3371

Acknowledgments
The advisor for this project was:

Don Cameron
International Technical Support Organization, San Jose Center

The principal author of this document is:

Adrian Houselander
IBM United Kingdom

The material in Chapter 9, “Database Design Considerations” on page 77 and Chapter 10, “Table Design Considerations” on page 91 was adapted from an earlier monograph by:

Don Cameron
International Technical Support Organization, San Jose Center
This publication is the result of a residency conducted at the International Technical Support Organization, San Jose Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Mike Brook
IBM United Kingdom

Roy Cornford
IBM United Kingdom

Derek Ip
IBM Canada

Jose Manuel de la Lastra Haro
IBM Spain

Sally Montera
IBM ITSO, San Jose, California USA

Savur Rao
IBM, San Jose, California USA

Thanks also to Maggie Cutler of IBM ITSO, San Jose, for editing this book.
Chapter 1. Information Warehouse Framework

The IBM* Information Warehouse* framework is an approach to solving a problem that many organizations, both large and small, face today—access to data across the organization. This problem is typically caused by:

- The increased volumes of data being produced today
- The currency of data maintained
- The need for standard, easy to use access interfaces.

With computers firmly established in every department of every organization, the amount of information produced is staggering. The ability to make informed decisions often requires the decision maker to be conversant in many tools, applications, and environments. The situation is made worse by the locality of data:

- What data exists?
- Where is it?
- How do I access it?
- What format is it in?

The Information Warehouse framework can empower decision makers by giving them access to all of the data they need, providing them with methods of transforming the raw data into information.

This chapter gives a brief introduction to Information Warehouse concepts and some of the terminology associated with informational databases and decision support systems. The chapter is not intended to be a full, functional description of the Information Warehouse framework, which is discussed in more detail in other publications. It is intended to provide basic knowledge of Information Warehouse concepts as background for the subsequent chapters.

1.1 The Problem

Organizations perform a variety of data processing activities, and each activity has its own dedicated applications, data design, storage techniques, operating systems, and hardware. The data is said to be “tied” to applications.

Access to data is now commonly needed beyond the scope of the application maintaining it. For example, a head office may want to reduce costs by purchasing certain raw materials in bulk, but before making this decision it needs to analyze the use and costs of materials to determine the potential savings. The users of the data need to understand how to piece the data together and merge it with data from other areas.

Inconsistencies in the data from different parts of the organization can often cause the information worker to spend time determining which piece of information is correct. Different applications retrieve and maintain the data in different ways, causing problems in evaluation of results. One cause of such problems may be summarizations done for different departments in different ways.
Users need to know details about the data and the tools that are available to operate on the data and provide usable information, before the value of the data may be realized by the organization.

1.2 A Solution: The Information Warehouse Framework

The Information Warehouse framework offers a solution that addresses how information is presented to users and how that information is built and maintained behind the scenes.

The decision maker or knowledge worker wants easy access to all relevant data. Such access requires tools to help determine which data is available (such as an "information catalog") and a way of accessing the data. The method of accessing the data should be independent of the data location or data format, and access should be through a consistent user interface.

An information catalog provides a description of the information available in terms familiar to the user. The description should include a general statement of the meaning and purpose of the information. It could also include:

- The current status of the information
- When it was created or last updated
- The frequency with which it will be changed
- The attributes of the data
- How the data is grouped
- Where the data comes from
- Data relationships
- How the data can be accessed (for example, systems and applications).

All information provided to the user in the information catalog should be accessible in a consistent way. If there are five different data sources, the method of access should be similar across the five systems. The access procedures should encompass not only the look and feel of the end-user interface but also the language used to access the data. The language should provide the user with an easy and flexible method of getting the data and presenting the requested information.

A user can be given access to the data needed in several ways. One approach is to let the user access the information at the information source. The user could sign on to the relevant applications to get the data, or user requests could be routed to other applications, with necessary language translations and reformatting done en route. For the end user, all relevant data appears to be on the common platform, although behind the scenes informational applications are transporting the data to and from the operational environment.

The needs of an operational application differ from an informational system required by the knowledge worker. The operational application’s data design would normally be geared to meeting the necessary transaction workload, which often means that only the latest state of the business is represented. There are many cases where the decision maker would require access to data covering not just the latest period, but going back many weeks, months, and possibly years. It would be out of the question for the operational applications to store this data.

The solution is to develop an information store to hold the data required by the user. This information store, or informational database, can be built from data that is extracted and derived from operational databases or files. The
informational data need not be just a copy of data taken from the operational applications. The data can be a selective subset, some form of aggregation, or in some way enhanced—for example, by obtaining the data from more than one source.

The data in the information store may be in any form that is most suitable for the users of the data. As described in *Information Warehouse Architecture I* (GC26-3244) and as shown in Figure 1, several types of data can be made available:

![Data Types within an Information System](image)

**Figure 1. Data Types within an Information System**

**Real-time data**

Real-time data is often referred to as operational or raw data. It is in the form needed by operational systems, containing fine detail about entities and events. It is not often of much use to the information worker in that form.

**Changed data**

Changed data contains a history of all changes that have been applied to the real-time data, possibly with sequencing information or time stamps. Depending on the activity against the operational data, the volume of changed data can be huge.

**Meta-data**

Meta-data provides descriptive information about entities and how they are represented, thus helping users access and understand the data.
Reconciled Data

Reconciled data is based on operational data but has been cleansed, enhanced, or adjusted to make it more consistent and suitable for use by informational applications. This type of data is vital where data coming from different sources is maintained in different ways, such as character representation of a number on one system and numeric representation on others.

Derived Data

Derived data provides users with some degree of summarization or enhancement of the raw data. Examples include aggregation of various reconciled or operational data records, or alternative views of data.

An information store may be implemented to provide one, several, or many types of data to the user. The objective of the information store is to provide users with what they need, in the way they would need to see it, and allow easy enhancement.

Getting the information into the data store in the form users need requires various tools to extract, filter, aggregate, and format the data. If the volume of data is high, the archival of data beyond certain dates or the compression of some of the data might be necessary.

These tasks should not be performed only when the user needs the data; many users may need the data at a moment’s notice. Some form of automation is desirable so that the users can be provided information when it changes, either immediately or at predefined intervals.

The Information Warehouse framework describes an approach to giving users easy access to data, almost regardless of where that information is. The framework consists of an architecture and a collection of products that support or use defined interfaces and protocols in the architecture.

1.3 Information Warehouse Architecture

The Information Warehouse Architecture defines a structure for building customized implementations that gives users easy access to data. As shown in Figure 2 on page 5, it has five components:

- **Applications** - to help decision makers analyze and present the information needed
Figure 2. The Information Warehouse Architecture

- **Tools** - to assist users in accessing information and to provide data replication and copy management capabilities (getting information from operational systems to the information store and managing data in the information store).

- **Access enablers** - such as the Structured Query Language (SQL) to provide users with an efficient way to get data.

- **Organization asset data** - the operational and informational data within an organization, and the meta-data in the information catalog.

- **Infrastructure** - those components necessary to implement the system, such as the database manager (for example, DATABASE 2*), and a system to manage the flow of information.

The Information Warehouse framework is intended to provide access to data from IBM and non-IBM systems, to maximize the information available to a knowledge worker and to allow the correct decisions to be made.

### 1.4 Information Warehouse Framework and Storage Management

The concepts, techniques, products, and guidelines presented in this book are relevant to Information Warehouse planning and design for the following reasons:

- The amount of data needed to maintain an adequate historical base for ad hoc management decision making may be very large. Such data is included in the “Organization Asset Data” referred to in Information Warehouse Architecture I. The quantity of such informational data may be larger by an
order of magnitude than the operational databases used for transaction processing and reporting that supports the organization. Thus this book addresses storage hardware and media such as the 3995 Optical Library Dataserver, and storage management software such as Data Facility Storage Management Subsystem (DFSMS*) and Data Archive Retrieval Manager (DARM). These can help to store, control, and manage very large amounts of data and the costs involved. They also coexist with and support database management systems such as DB2*. Although these products are not architecturally defined as part of the Information Warehouse framework, they are important to consider as components of the solution to the problems that the framework is intended to address.

• Because Information Warehouse applications such as decision support systems (DSSs) and executive information systems (EIsSs) are highly suitable for a client/server approach, there is a greater need for data replication and copy management functions. Such functions would:
  − Move data from operational data sources to informational data stores
  − Transform the data to forms more suitable for DSS or EIS use, as or after they are moved
  − Distribute informational data, for example, to systems which may provide useful subsets that are easy to manage and access. Such subsets may be based on geography, organization entity (such as department), or workgroup. This data distribution could involve further derivations or transformations of the data.

Such products as DataRefresher and DataPropagator* Relational (DProp* Relational or DPropR) provide these functions, as described in Chapter 4, “DataPropagator Relational” on page 29, and are part of the “Tools” component of Information Warehouse Architecture I.

• In addition to the “staging” requirement just described, there is often a need for “aging” of informational data. Reference patterns for a particular set of data records may be highly skewed and may change significantly over time. Use of a hierarchy of storage technologies can balance access time requirements, storage capacity needs, and costs, as described in Chapter 3, “Data Storage Selection Based on Access Requirements” on page 17. Use of data rolling and wrapping techniques (as described in Chapter 6, “Data Rolling or Wrapping” on page 43) should be evaluated for managing “aging” data. The data replication products such as DPropR may also be relevant for maintenance of consistent subsets (say, the most recent) of historical data.

• Irrespective of physical characteristics and performance issues, informational data differs from operational data from a logical database design standpoint. Logical design characteristics specific to informational databases, and the relationship of different examples of informational data tables to the “Organization Asset Data” (real time, reconciled, derived, changed) as defined in Information Warehouse Architecture I are the subjects of Chapter 9, “Database Design Considerations” on page 77 and Chapter 10, “Table Design Considerations” on page 91.
Chapter 2. The Storage Hierarchy

This chapter describes some storage options available to mainframe computer installations. It does not specifically mention all IBM storage devices but concentrates on the more common 3390 DASD models and the 3995 Optical Library Dataserver and refers to the 3490 tape family. These devices are referenced because they offer characteristics that are required by or could be used in implementations of the Information Warehouse framework. They are also likely to be found in many other installations.

2.1 Storage Product Positioning

Traditionally many mainframe computer installations use two types of storage media: DASD and tape.

For online data, DASD offers good performance for the majority of data processing needs. In certain performance-critical situations, faster devices, some form of cache, or specific placement of data sets are used to improve performance to the level desired. Most application and system code, together with a portion of the data, is stored on DASD for speed of access.

For storage of this code and data, 3390 DASD and 3990 DASD controllers are commonly used. Models 3390-1 through 3390-3 provide excellent response times for online data, with different data retrieval speeds and capacities (see Table 1 on page 9). The caching features of the 3990 controllers allow certain operations to use a built-in memory cache, rather than accessing the disk for frequently referenced data. Use of data from the cache may result in significant reduction in access times (if properly tuned).

Because of the large volumes of data that an installation may need to store, and the costs of buying more DASD, data may be periodically off-loaded to tape.

Tape is suitable for data that does not need to be available at all times or can tolerate the delay of waiting for the tape to be loaded and its contents sequentially accessed and/or restored to DASD. In many installations, tape is largely an archive medium for backup and recovery purposes, offering a very low cost per gigabyte medium when stored offline. The time taken to retrieve a piece of data from tape varies greatly and depends on the number of drives available and the demand on the tape library.

Products such as DFSMShsm* can be used to simplify the automatic process of migrating infrequently used data to tape.

The introduction of large-capacity magnetic and optical DASD configurations has bridged the gap between high speed, high cost DASD and low speed, low cost tape drives.

The 3390 Model 9 DASD gives three times the capacity of the 3990 Model 3, albeit with a slower access time for direct access. The 3995 Model 153 Optical Library Dataserver gives an even larger amount of direct access storage, at a low cost, but with a slower data access time than the 3390 Model 3 and Model 9. See 2.2, “3390 Model 9 and 3995 Model 153” on page 9 for a functional description of the 3390-9 and 3395-153 devices.
The costs of storing large volumes of data online have made development of some business applications nonjustifiable. In some cases, tape could be used to store the volumes required. However, the sequential nature of tape technology and the variation in the tape access times make tape storage infeasible. Thus the 3995-153 and 3390-9 offer a storage medium for some previously nonjustifiable applications.

With the introduction of the 3990 Model 9 and the 3995 Models 153/113, there is now a complete range of storage products with a range of access times and capacities, as shown in Figure 3.

![Figure 3. Device Performance and Capacity Comparison](image)

With such a wide choice of storage technologies available, data may be stored on a medium that more closely matches the requirements and tradeoffs of cost per gigabyte, access time, and data transfer rate. Table 1 on page 9 compares the performance and capacity of the 3995-153 with the 3390 models.
Table 1. Storage Device Performance and Capacities

<table>
<thead>
<tr>
<th></th>
<th>3390-1</th>
<th>3390-2</th>
<th>3390-3</th>
<th>3390-9</th>
<th>3995-153</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes/track</td>
<td>56,664</td>
<td>56,664</td>
<td>56,664</td>
<td>56,664</td>
<td>56,664+</td>
</tr>
<tr>
<td>Tracks/cylinder</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15a</td>
</tr>
<tr>
<td>Cylinders/device</td>
<td>1,113</td>
<td>2,226</td>
<td>3,339</td>
<td>10,017</td>
<td>4,365a</td>
</tr>
<tr>
<td>Capacity/device (GB)</td>
<td>0.94</td>
<td>1.89</td>
<td>2.83</td>
<td>8.51</td>
<td>1.30b</td>
</tr>
<tr>
<td>Capacity/unit (GB)</td>
<td>11.35</td>
<td>22.70</td>
<td>34.05</td>
<td>102.17</td>
<td>178.00</td>
</tr>
<tr>
<td>Capacity/string (GB)</td>
<td>30.25</td>
<td>60.54</td>
<td>90.81</td>
<td>272.45</td>
<td>356.00c</td>
</tr>
<tr>
<td>Unit capacity/actuator (GB)d</td>
<td>0.94</td>
<td>1.89</td>
<td>2.83</td>
<td>8.51</td>
<td>44.5</td>
</tr>
<tr>
<td>Data transfer (MB/sec)</td>
<td>4.2</td>
<td>4.2</td>
<td>4.2</td>
<td>3.9</td>
<td>1.2 - 2.2</td>
</tr>
<tr>
<td>Latency (ms)</td>
<td>7.1</td>
<td>7.1</td>
<td>7.1</td>
<td>22.8</td>
<td>12.5</td>
</tr>
<tr>
<td>Min seek time (ms)</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>0/0/2.5e</td>
<td>0f</td>
</tr>
<tr>
<td>Ave seek time (ms)</td>
<td>9.5</td>
<td>12.5</td>
<td>15.0</td>
<td>22.5</td>
<td>60f</td>
</tr>
<tr>
<td>Max access time (ms)</td>
<td>18.0</td>
<td>23.0</td>
<td>33.0</td>
<td>38.0</td>
<td>10-20 secondsf</td>
</tr>
</tbody>
</table>

Notes:

a These figures are for the emulation of a 3390 Model 2.
b Represents capacity of one optical cartridge (double density).
c Capacity of a Model 153 and Model 113 attached.
d This value indicates the demand on the optical drives. Each drive is therefore responsible for a very large amount of storage, resulting in limited parallel access capacity.
e Each cylinder of the Model 9 represents three cylinders of a Model 3. This means there is potentially a seek time of 0 in two out of three requests, compared to a Model 3.
f The Model 153 stores information in a spiral track format and uses the on-board controller to emulate the cylinder device geometry of the 3390 family. The max access time value includes seek time and time to change an optical cartridge, which could be necessary for an emulated seek. The min seek could be 0 if the record were in controller memory or following the record just accessed.

2.2 3390 Model 9 and 3995 Model 153

The 3390 Model 9 and 3995 Model 153 storage devices were announced in 2Q1993. They offer a computer installation direct access to large volumes of information at a low cost per gigabyte. The larger storage capacity must be evaluated against a longer data access time. The sections that follow discuss the 3390 Model 9 and 3995 Model 153 individually.

2.2.1 3390 Model 9

The 3390 Model 9 is architecturally similar to other models in the 3390 family, but with the following advantages:

- The cost per gigabyte is significantly lower than the other models, particularly when a fully configured 3390-9 is compared with the configuration of the equivalent capacity of other models.
- Improved reliability, availability, and serviceability have been achieved through use of new materials and as a result of slower disk rotation speed.
• The reduced power consumption and cooling requirements and increased
capacity have resulted in a lower cost per gigabyte of stored data.
• The physical dimensions are the same as the other 3390 models.

The 3390-9 units have a rotation speed of 1320 revolutions per minute compared
with 4260 on other models. One physical cylinder is used to store three logical
cylinders. This is possible through a new thin film deposition coating process
giving a smoother, higher quality disk surface that allows higher density
recording. The smoother surface allows the flying height of the heads to be
lowered, improving the recording precision.

The use of hardware-assisted data compression can further improve the cost per
gigabyte, giving the 3390-9 a capacity of up to 17GB per volume. The capacity of
a maximum configuration DASD subsystem (one 3990 with two fully configured
3390-9 strings attached) is 544GB, compared with 181GB for a Model 3
subsystem. One may also install a 3390-9 on the same string as a 3390-3,
providing a mix of access times and capacity on the same string.

The 3390-9 is not a replacement for the traditional general-purpose 3390-3; it is a
lower cost per gigabyte option suitable for applications where the slower access
time is acceptable. It is particularly suited to processing large sequential chunks
of data, as would be expected when running utility programs or performing
scans of large tables. The 3390-9 is not suitable in applications that require a
high degree of parallel access.

### 2.2.2 3995 Model 153

The 3995 Model 153 (see Figure 4 on page 11) is an improvement over the 3995
Model 151, having twice the storage capacity. It provides low-cost storage using
rewriteable optical storage technology. It is best suited to situations where the
application data is infrequently accessed, requires high storage capacity, and/or
requires a low cost of storage. It emulates a very large capacity 3390 Model 2
but is much slower in operation.
2.2.2.1 Configuration

The main components of the 3995 Model 153 are:

**Controller**

Receives extended count-key-data (ECKD) commands from the 3995's built-in channel adapter (see Figure 5 on page 12) and converts them into small computer system interface (SCSI) commands for passing on to the optical drives.\(^1\) The controller receives fixed block data from the optical media, which it formats into 3390-2 track images, and then selects and buffers the requested data for sending to the host system. Performance is improved by loading an entire simulated track of data into controller memory for any future requests.

**Accessor**

The 3995-153 can hold 144 optical cartridges, which the accessor selects for mounting and places into one of the optical drives. The process of picking a cartridge, loading it into a drive, and spinning it up to operational rotation speed takes roughly 10 seconds, assuming there is a free optical drive.

**Optical drives**

There are four optical drives, which can read or write both single and double density optical cartridges. The controller always tries to keep one drive available for mount requests.

---

\(^1\) The 3995 Model 153 is channel attached and has integrated controller functions, thus reducing its cost.
Data is stored on the cartridges in 1024-byte sectors in a spiral pattern, similar to vinyl records except that the start of the disk is at the center. The cartridges rotate at fixed speed. Therefore, the data transfer rate of the cartridge is significantly greater on the outside of the disk surface.

The controller contains two 400MB hard disks, which buffer any write requests, reducing the need for the cartridge to be mounted. Any write to the optical platter is delayed until the data server is idle, or the appropriate cartridge needs to be loaded for another request. This process is similar in concept to DASD fast write. When the hard disks become full, the delayed write function is bypassed.

The actual write to the optical media consists of three steps:

1. Erase previous contents.
2. Write new contents.
3. Check.

Up to 144 cartridges can be placed into a 3995-153, each with a formatted capacity of nearly 1.3GB. Three cartridges are used to transparently map to one 3390-2 volume, with 4,365 emulated cylinders. The emulated volume appears to the system as 3390-2 DASD; no additional software is required. By using all 144 cartridges, the 3995-153 can emulate 48 volumes of 3390-2, which amounts to about 76GB more than a single, fully configured 3390-9.

The 3995 Model 113 can be used in conjunction with the 3995-153. The 3995-113 offers the same storage capacity and configuration as the 3995-153 except that it
has no controller. The 3995-153 handles all control operations for the 3995-113. The two used in combination allow eight optical drives to be used at one time. The fully configured storage capacity of each unit is 178GB, giving a combined capacity of 356GB, or approximately 16 times the capacity of a fully configured 3390-B2C. The cost per gigabyte is one-sixth of equivalent 3390-3 DASD, or one-third of equivalent 3390-9.

The use of optical storage delivers slower performance (see Table 1 on page 9). Response times can be as low as 60 milliseconds for access to a mounted cartridge, but delays of 10 to 20 seconds may occur for unmounted cartridges. Data transfer rates of 1.2 to 2.2 megabytes per second for reads are typical. Use of larger block sizes improves data transfer rates.

The 3995-153 is suitable for thousands of accesses per day, compared with millions of accesses for the 3390 models 1, 2, and 3. This type of performance is low compared with conventional 3390 technology, but the cost of storage is significantly lower.

Three main factors affect performance of the 3995 Model 153:

- The time for the accessor to mount optical cartridges and get them spinning at full speed
- The slow effective data transfer rate when compared to traditional DASD
- The small number of physical drives compared with the number the system emulates, that is, 4 versus 48.

Figure 6 on page 14 demonstrates the average mount response times for random retrievals from unmounted cartridges. The figures were modeled and include:

- Waiting for the accessor
- Moving the disk from the cartridge slot to the drive
- Inserting the disk to the drive
- Spinning the disk up to speed
- Reading a 4KB record
- Transferring to the host
- Spinning the disk down
- Ejecting the disk from the drive
- Returning the disk to the cartridge storage slot.

The figures show that heavily loading the subsystem can drastically degrade response times. For this reason it is important to choose the correct type of data to place on this type of device. Storing a large number of small files, each of which is accessed independently, on optical storage is likely to result in mount requests for many different cartridges.
The 3995-153/113 should not be considered as a replacement for conventional DASD, but rather as a slower, very high volume device with DASD characteristics. The value for unit capacity per actuator in Table 1 on page 9 indicates that the device performance is rather restricted in cases where parallel access to data is prevalent. Like the 3390-9, the 3995 costs less per gigabyte compared to the 3990-2 and 3990-3 and requires a fraction of the floor space.

2.2.3 Summary

The introduction of the 3390 Model 9 and the 3995 Model 153 completes the range of requirements for storage products. It is now possible to store data on a device that is best suited to its needs. Table 2 on page 15 is a comparison of characteristics and recommendations for use of the different storage technologies. The cost associated with direct access to online data has been reduced, if the frequency and speed of access to the data can be low. Chapter 3, “Data Storage Selection Based on Access Requirements” on page 17 discusses this subject in more detail.
### Table 2. Media Suitability Matrix

<table>
<thead>
<tr>
<th>Usage</th>
<th>DASD</th>
<th>Optical</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Short response times required</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Small amounts of data per access</td>
<td>H</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>Large files</td>
<td>M</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>High data rates</td>
<td>H</td>
<td>M</td>
<td>H</td>
</tr>
</tbody>
</table>

**Note:** L means low, M means medium, and H means high cost or degree of suitability of the storage medium for the particular usage characteristic.
Chapter 3. Data Storage Selection Based on Access Requirements

One key to getting a system to meet performance requirements is to identify the demands on each type of data in the system. The data includes application data, system data, and application code. All such data needs to be placed on the storage medium that best suits the access requirements.

This chapter discusses various classes of data, focusing on types that might be used in support of an Information Warehouse environment. It identifies the best storage medium for each class of data, with special emphasis on data that could reside on 3390 Model 9 or 3995 Model 153 storage devices.

Considering various access needs of each class of data, the requirements are then mapped to the various devices that are available. Use of the DFSMS products for different types of data is then discussed.

3.1 Identifying Classes of Data

Many Information Warehouse environments have a wide variety of data set access requirements, ranging from very fast and frequently used data, to data that is seldom used but must always be available.

In determining the best device types for different types of data, we need to classify the various data sets a typical installation may have:

**System code**
MVS and associated subsystems (such as DB2, VTAM, IMS, and CICS)

**System logs**
Any logs used by the system to operate the system. Examples include SMF data sets and DB2 or IMS recovery logs.

**Application data**
Any data stored by an application. Such data includes sequential or partitioned data sets but excludes data that is stored under control of some database or similar product (see “DB2 data” below). Application data might include extract files, control files, and temporary work files.

**Application code**
Typically application source code (such as C, PL/I, and COBOL) and object code.

**DB2 data**
Covers a wide variety of data. See 3.1.1, “Classes of DB2 Data” on page 18 for a more detailed analysis and description.

**Migrated data**
Data that has been moved from one storage medium to another of lower cost. Data that is infrequently used is moved onto slower devices, freeing valuable space on devices that can be best used by frequently accessed data. A request to access a piece of migrated data may occur at any time, possibly triggering its placement onto a faster device for ready access.

**Other data**
Just as DB2 data set requirements vary, so too can requirements for other types of data such as VSAM and IMS-DB.
3.1.1 Classes of DB2 Data

As mentioned previously, DB2 data covers a wide spectrum of data. Each class potentially has different storage requirements as described below.

Catalog data
The DB2 catalog consists of tables of data about everything defined to DB2. The DB2 catalog is contained in a system database (DSNDB06). When you create, alter, or drop any object, DB2 inserts, updates, or deletes rows of the catalog that describe the object and tell how the object relates to other objects. The tablespaces used for the catalog tables are contained in VSAM linear data sets for which fast access and high availability are vital.

Directory data
The DB2 directory contains information that it uses during normal operation. The directory consists of a set of internal tables stored in a system database (DSNDB01). Each is contained in a VSAM linear data set for which fast access and high availability are vital. The directory holds:

- SCT02 - the skeleton cursor tablespace (SKCT)
- SPT01 - the skeleton package tablespace
- SYSLGRNG - the log range tablespace
- SYSUTIL - the system utilities tablespace
- DBD01 - the database descriptor (DBD) tablesapce.

Bootstrap data set (BSDS)
This data set records information about log data sets and DB2 system checkpoints. The BSDS is a VSAM key sequenced data set (the only KSDS associated with DB2). The BSDS is very critical (DB2 requires dual copies), but very small; therefore it should be placed on the most reliable fast device.

Active log data sets
DB2 uses active log data sets to store all changes and significant events as they occur. DB2 uses this information for recovery purposes. Access to the active logs should be as fast as possible.

Archive log data sets
When the active log is full, DB2 copies the contents of the active log to the archive logs. These could be on many types of storage devices, as long as they are available when needed for recovery.

Application tablespaces
DB2 tablespaces are allocated a VSAM data set (or collection of VSAM data sets) to hold one or more DB2 tables. Application tablespaces are created on storage devices according to the performance goals of the application. If the application requires exceptionally good performance, application tablespace data sets should reside on dedicated storage devices or even be assigned to a separate buffer pool in order to minimize any delay in processing requests.

Application indexes
DB2 indexes are allocated a VSAM data set, or collection of VSAM data sets, to hold a DB2 index to provide faster access to the corresponding table
residing in a tablespace. The storage medium for the index should be chosen based on performance requirements of the applications that use the corresponding table.

**End-user tablespaces**

End users use these tablespaces to hold various temporary tables. These would not ordinarily be used as part of an application, so performance is not usually as critical, nor is the amount of data as large.

**Temporary work file tablespaces**

These are used for operations such as sorting or joining. These are all defined in database DSNDB07. Contention may be reduced by allocating multiple temporary work file tablespaces on separate DASD volumes on different channels or control unit paths.

**Image copy data sets**

Image copy data sets are used purely for recovery purposes. Once written, they are rarely read except in recovery situations. Consequently the performance requirements will depend on the time available to actually take image copies. Alternatively, DFSMSdss* concurrent copy may be used as described in 3.2.2, “Implementation of DFSMS and DB2” on page 25.

If absolutely everything were in real storage, our worries about performance would largely be over, barring inefficient code and the limitations on the addressing capabilities of the processor. Unfortunately this is not possible, partly because of real storage costs, so we use high performance DASD to store data and code. This DASD is expensive, and the amount of data to be stored increases every year, so cost constraints force us to find ways in which we can archive data for which we have no immediate need.

Tape is an acceptable cost versus availability compromise, but the amount of information that we require at our fingertips is constantly growing.

Any data set or class of data set will have some performance criteria addressed at design time. These criteria generally define the lower limits allowed in accessing the data. No user will complain if the response time to a transaction is immediate, so long as this is not detrimental to the access of the rest of the data. Upper limits are set to assist in the placement of similarly performing data sets.

The boundaries exist largely for performance management and run time prediction purposes. If the system meets its performance objectives based on worst case access times, there would normally be no reason to expect the system to perform badly as a whole. The graphical representation of boundaries emphasizes the data sets that are critical to the business process.

Figure 7 on page 20 is an example of a data set response time range graph. Compared with Figure 3 on page 8, the two may have some similarities that can be used for rules of thumb for device type selection. The mapping of particular data sets can be done as in Figure 7 on page 20 and analysis of requirements done as part of application design.
Figure 7. Data Set Response Time Range

Figure 7 may appear to imply that the larger the tablespace, the slower the response time may be, and that end-user tablespaces generally have a lower requirement for fast access. This may sometimes be true in particular situations, but there are many cases where application tablespaces may be large and still require fast response times. It is also true that end-user tablespaces may require fast access. The mapping of the requirements to the physical hardware is somewhat of a balance.

3.1.2 Mapping Classes to Devices

Having discussed the various types of data sets that may be found within an Information Warehouse system, it is now possible to categorize each for placement in a suitability matrix, as shown in Table 3 on page 21.
Based on device characteristics of the available storage devices and their suitability for the various types of data sets, a storage strategy can be defined to satisfy the requirements of the application. The strategy should consider the storage devices available and the requirements on the system, such as data volumes for each type of data, data set access frequencies over time, performance critical data sets, and backup procedures. This process could be used to identify areas where desired performance may be difficult to attain, and where new or additional hardware may be required. Definition of the strategy will indicate which types of data sets are to be on which volumes.

Traditionally, the implementation of a storage strategy has taken considerable planning and effort, such as placing data sets on specific DASD volumes, deciding whether the data sets should be eligible for migration by DFSMShsm, and defining the types of backup required. This process has been simplified with the introduction and development of System Managed Storage.

### Table 3. Data Set and Device Suitability Matrix

<table>
<thead>
<tr>
<th>Data Set Type</th>
<th>Tape</th>
<th>3995 Model 153</th>
<th>3390 Model 9</th>
<th>3390 Model 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB2 Data Sets</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSIDS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Catalog</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Directory</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Active Log</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Archive Log</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes¹</td>
</tr>
<tr>
<td>Application table space</td>
<td>No</td>
<td>Yes²</td>
<td>Yes²</td>
<td>Yes</td>
</tr>
<tr>
<td>Application index space</td>
<td>No</td>
<td>Yes²</td>
<td>Yes²</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporary work file table space</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>End-user table space</td>
<td>No</td>
<td>Yes²</td>
<td>Yes²</td>
<td>Yes</td>
</tr>
<tr>
<td>Image copy data sets</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Non-DB2 Data Sets</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System code</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>System logs</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Application data</td>
<td>Yes³</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Application code</td>
<td>No</td>
<td>Yes²</td>
<td>Yes²</td>
<td>Yes</td>
</tr>
<tr>
<td>Migrated data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No¹</td>
</tr>
</tbody>
</table>

**Notes:**

¹ Although the 3390-3 is capable of storing this type of data, it may not be advisable because of high volumes of this type of data and contention on the device for other types of data.

² This type of data can have varying access requirements. The access requirements of a particular data item may mean that this type of device is too slow in certain instances.

³ If the data sets are used for input or output records for a batch application, for example.
3.2 Data Facility Storage Management Subsystem

This section discusses the use of Data Facility Storage Management Subsystem (DFSMS) focusing on the management of DB2 data sets. For a full discussion of DB2 implementation with SMS see: DB2 Usage in the DFSMS Environment Parts 1 and 2.

The section starts with a quick introduction to DFSMS and some of the terminology used. Throughout this text the terms SMS (the System Managed Storage functional capabilities) and DFSMS (the family of products) are used to describe the functions made available by:

- DFSMS - Data Facility Storage Management Subsystem
- DFSMSdfp* - Data Facilities functional component
- DFSMSdss - Data Set Services functional component
- DFSMShsm - Hierarchical Storage Management functional component.

3.2.1 Introduction to System Managed Storage

The need for effective storage management increases with the volume of data stored. This is particularly true for large-volume informational databases. The task of managing the data storage can be handled using System Managed Storage (SMS). The system determines:

- Data placement
- Data availability
- Performance targets
- Space reclamation.

The aim is to match the logical needs of the data to the physical characteristics of storage devices as transparently as possible to the users of the system.

Traditionally the database administrator (DBA) had to know details about device geometries, capacities, and speeds in order to manage the information flow between the storage devices and the applications. There typically were frequent dialogs between the DBA and the storage administrators. With DFSMS, the logical and physical aspects of data storage are separated, leaving the DBA to concentrate on the logical data needs, and the storage administrator on the physical needs.

For the DBA, the implementation of a storage policy is specified at CREATE TABLESPACE and CREATE INDEX. The USING block of the CREATE TABLESPACE and CREATE INDEX statements determines whether the DBA is going to let DB2 allocate the necessary data sets, or whether data sets defined by the DBA are to be used. See Figure 8 on page 23.
If the DBA specifies a catalog name in the VCAT option, DB2 uses the VSAM data set specified to contain the tablespace. The DBA may or may not choose to allow DFSMS to control allocation of the VSAM data set. If a DB2 storage group (STOGROUP) is specified, DB2 tries to allocate the tablespace on one of the volumes in that STOGROUP. The DB2 storage group is defined using the CREATE STOGROUP statement. Within the VOLUMES parameter of CREATE STOGROUP, the DBA can specify a list of DASD volumes to be used for the DB2 storage group. Alternatively, the DBA could specify '*' in the VOLUMES parameter, in which case DFSMS will select a volume for allocation of the data set on DB2’s behalf.

The storage management configuration is created and maintained through Interactive Storage Management Facility (ISMF), which allows the definition of storage management policies. ISMF controls the definition of classes and DFSMS storage groups, and the specification of automatic class selection (ACS) routines.

### 3.2.1.1 Classes and Storage Groups

Classes and DFSMS storage group definitions define the installation’s storage management policies to the system. Classes are assigned to data sets, which are allocated on volume pools called storage groups. There are three types of classes and a DFSMS storage group definition:

- **Data class**
  
  Data classes contain allocation defaults to simplify and standardize the allocation of new data sets. Their use saves having to define all of the attributes associated with a data set each time a new data set is created.

  The data class can be explicitly specified at data set creation, or it can be selected implicitly by the data class automatic class selection (ACS) routines based on data set name formats. The attributes of a data class may be overridden with JCL keywords.

  Within an installation, data classes should be created for each collection of data sets that have similar allocation requirements.

- **Storage class**

  Storage classes allow the specification of different levels of performance and availability for data sets. Using a storage class causes DFSMS to attempt to find a volume that most closely matches the requirements.
Within an installation, storage classes should be defined for each group of data sets that have similar performance and availability objectives. These can be defined to take advantage of hardware features, such as dual copy, where such facilities are required. The storage classes should be set up to match the devices available to the installation.

Key features potentially of benefit to DB2 include:

- Dual copy - a feature of the 3990 controller, a recommended way of preventing data loss from hardware failure
- Cache - a feature of the 3990 controller, particularly suitable for repeated access to data
- DASD fast write - a feature of the 3990 controller which allows faster writing of data by writing directly to the controller’s cache
- Guaranteed space - allows the DBA to specify to DFSMS that a specific volume is to be used if volume serial is specified. When volume serial is not specified, this feature allows preassignment of space on multiple volumes.

**Management class**

Management classes allow the specification of different levels of migration, backup, and data set retention. DFSMSHsm uses the management classes to manage the storage of the data. If no class is defined, DFSMSHsm uses the default class defined to DFSMS for all DFSMS data sets.

Management classes should be defined for each collection of data sets in an installation that have similar migration, backup, and retention requirements. One can also prevent automated migration, backup, or deletion by management class specification.

**DFSMS Storage Group**

The use of DFSMS storage groups simplifies the management of individual devices by pooling them together; DFSMS storage groups typically contain similar devices. The storage group ACS routine selects a storage group to which to allocate the data set, and from the list of volumes in this storage group, selects an appropriate volume.

A volume can be assigned to only one DFSMS storage group. A data set can be in only one storage group but can span multiple volumes of a storage group.

For VSAM, the entire base cluster and all alternate indexes must be in the same DFSMS storage group. (This does not have much relevance to DB2 as DB2 does not use VSAM indexes except for the BSDS.) There is no requirement that all data sets associated with a DB2 tablespace or index be in the same storage group. The DFSMS storage group can include any DASD device type that MVS supports. This includes the 3995 units described in 2.2.2.
“3995 Model 153” on page 10. All of the volumes in a DFSMS storage group must have the same device geometry, that is, bytes per track and tracks per cylinder. Therefore, one DB2 tablespace partition could potentially cover multiple volumes of storage with the same device geometry. However, if the tablespace is partitioned, some partitions may be on devices with different geometries than those used for other partitions, because each partition is a separate data set. These data sets could also be in different storage groups. One can also enable or disable DFSMSHsm operations for space management and availability at the DFSMS storage group level.

### 3.2.1.2 Automatic Class Selection Routines

ACS routines determine DFSMS classes and storage groups for all new data set allocations and for all allocations that occur as a result of converting, copying, recalling, restoring, or moving data sets. ACS routines use filter lists (which are defined using a CLIST-like language) to assign storage management attributes to data sets. There are ACS routines for each type of DFSMS class, and one for identifying the DFSMS storage group. They are executed in the following sequence:

1. Data class
2. Storage class
3. Management class
4. Storage group.

Whether or not a data set gets placed under DFSMS control is determined by the storage class ACS routine. If a valid storage class is identified at allocation time, the data set will be managed by DFSMS and must be assigned to a DFSMS storage group. Use of the data class and management class are optional.

### 3.2.2 Implementation of DFSMS and DB2

Having introduced some of the functions of DFSMS and some of the performance requirements of various DB2 data sets, it is possible to suggest some implementation options. Certain options are applicable to more than one type of data set. In such cases, one is discussed and the other types of data sets with similar requirements are identified.

#### 3.2.2.1 Application Tablespace

It is important to stress that there can be considerable differences in the performance and availability requirements of application tablespaces. Some tablespaces require constant availability with guaranteed response times, and others may be accessed infrequently with little requirement for fast access.

The points discussed below are applicable to medium to high volume databases with high performance and high availability in mind. It is assumed that no part of the data is inactive for long periods of time. The term *application table* is used here to describe the tablespace and any associated DB2 indexes that relate to the table.

The application tables should be assigned to a DFSMS management class that prevents use of DFSMSHsm migration, automatic backup, or deletion. All backup activities should be managed by the DBA through the use of DB2 or other utilities. DFSMSdss may be invoked for concurrent copy operations. See
Chapter 6 of the *DB2 Version 3 Administration Guide* for information on recovery from copies made using facilities other than the DB2 Image Copy utility.

In cases where tables are critical to the performance of the system and are regularly joined, it may be advisable to use DBA-defined data sets to prevent SMS from placing two performance-critical data sets on the same volume. This should be the exception rather than the rule. In most cases SMS can be used to place the data sets on an appropriate volume on the basis of performance criteria set by the storage class.

Where fast access is needed all the time, a storage class can be used that will allow SMS to utilize 3990 cache. The 3990 Model 6 with DFSMS Version 1 Release 2 has shown considerable improvement in response time when cache and fast write features are used. In cases where protection from hardware failure is absolutely critical to the success of the application, dual copy functions can be requested; however, if no dual copy volumes are available, the SMS allocation may fail. The use of DASD fast write should also be considered if it is appropriate for the application and performance is critical.

Data sets used for application tablespaces should conform to a defined naming standard that will allow the ACS routines to identify each type of tablespace and set the appropriate storage class automatically.

These options should also apply to the DB2 Catalog and the DB2 Directory. The performance and availability requirements for these data sets are even greater than for an application tablespace.

### 3.2.2.2 BSDS and Active Logs

As described previously, the BSDS contains information about the DB2 logs, system checkpoints, and buffer pools. The active log contains data changes and events within DB2 as they occur. Both data sets are critical to the recovery of DB2 in the event of failure.

The performance of the log is critical to the performance of the DB2 system, and the availability of the BSDS and active logs are important to DB2. Neither should be capable of being archived, backed up, or deleted by DFSMS.

DB2 maintains dual copies of these data sets. Both data sets are allocated only once, when DB2 is installed, so there is little benefit from the DFSMS functions, except that DFSMS can prevent other data sets from being allocated on these volumes.

### 3.2.2.3 DB2 Archive Logs

The archive logs play an important role in recovery from failure. In many installations archive data sets end up on tape. These data sets may be allocated directly to tape, or they can be placed on DASD and be eligible for migration to tape by DFSMShsm. If archive logs are allocated to DASD, there is no benefit from use of DASD fast write or cache.
3.2.2.4 Image Copy Data Sets
The storage requirements for image copies are similar to those for archive logs. Once written, these data sets rarely need to be accessed except in cases of tablespace recovery.

Image copy data sets do not require the high performance features of the 3990 controllers; because they are backups of existing tablespaces, they do not require further backups or dual copy facilities. They may be eligible for migration to tape or some other medium almost as soon as they are created, reducing the demand on faster DASD.

Management of image copy data sets can be made easier by choosing a naming standard that allows DFSMS to recognize them and take appropriate action.

3.2.2.5 End-User Tablespaces
End-user tablespaces are typically less critical to the running of applications and to the installation’s business than either application databases or the DB2 system databases. Generally this type of data does not require the high performance and high availability of data used by online applications or the DB2 system itself. This is not to say that end users should be given old, slow DASD; merely that response times are often not as critical as operational transactions.

End-user tablespaces may be allocated under DFSMS control with no need for dual copy or DASD fast write. There may be special cases where cache may be required, but this would be exceptional.

Because end-user data may go through periods of inactivity, the data sets should be capable of being migrated by DFSMSshsm when inactive for a defined period. Housekeeping activities could be put in place to “force” migration of user tablespaces to DFSMSshsm Migration Level 1 periodically, for example, daily. This can be done by stopping the user tablespaces and issuing HMIGRATEs for the data sets through some tool. (See Appendix A, “An Example of Partition Wrapping” on page 107.) The time to recall the data from DASD will not inconvenience the user a great deal. If the data stays inactive for a longer period, the data can safely go to tape. When the user tries to access the data there may be a short delay before the data is returned. In certain circumstances the time taken to recall the data may be longer than the defined timeout period specified for the system. In these cases the user would get a -904 (UNSUCCESSFUL EXECUTION CAUSED BY UNAVAILABLE RESOURCE) return code. User education can teach users to wait a few minutes before trying again. Deletion after long periods of inactivity should not be used.

As with application tablespaces, the use of a naming standard will simplify the management of end-user tablespace data sets.

3.2.3 Conclusions
The use of DFSMS in a DB2 environment can reduce the amount of administration required by the DBA to create and manage data sets. Some special activities may be required to guarantee the performance of critical tablespaces used by applications, but as long as sufficient DASD is available to hold the data needed online, there should be few performance issues.

A dedicated DB2 pool could be used for application tablespaces and DB2 catalog and directory data. However, catalog and directory data should not be mixed with application data or other data that could create potential I/O contention.
problems. Non-DB2 data can be mixed with data sets requiring slower access times.

As stated previously in 1.4, “Information Warehouse Framework and Storage Management” on page 5, DFSMS (and its hierarchical storage management component in particular) could be useful in economically managing large amounts of informational data—particularly reconciled detailed data—whose access patterns change over time. (See also Figure 14 on page 44 in Chapter 6, “Data Rolling or Wrapping” on page 43.)
Chapter 4. DataPropagator Relational

For many installations, inquiring about the current state of the business is complicated by the fact that the operational records that contain the information are under constant change. If access using some decision support system were allowed, the needs of the DSS user (usually read only) could interfere with the needs of the operational applications (usually requiring some updating) which could cause deadlock situations. There is also the problem of mismatch in the data design. This problem is described in more detail in Chapter 1, “Information Warehouse Framework” on page 1.

Several approaches have been used to address these problems. These approaches often eliminate the need for direct user access to operational data by using data replication techniques on the operational system to provide an informational system.

The operational applications can be written to automatically initiate a process that feeds the informational systems. This approach may be feasible if designed into the system at the outset, but it is usually not feasible where the operational system has been running and growing for years without taking into account the informational needs. An alternative is to use a product such as DataRefresher (formerly DXT*) to “take a picture” of the data periodically. The resulting copy or replica is then used to update or refresh informational tables. Time needs to be made available in the operational schedule for this processing to be done. Unless the operational systems maintain a change history, it is very difficult to get a record of every change that has been applied to the system.

The requirement for processing change history data may be addressed through the use of an integrated data replication tool such as DataPropagator Relational (DPropR) or DataPropagator NonRelational.

This chapter describes the function of DPropR at a summary level to enable the reader to understand the various components of the product and its role within the Information Warehouse framework. It is not intended to be a full functional and technical description because such information can be found elsewhere.

The role of DataPropagator NonRelational is discussed in 4.3, “DataPropagator NonRelational” on page 35.

4.1 Product Function

DataPropagator Relational (DPropR) is designed to address the copy needs in the relational environment, both for operational and informational applications. Operational data is typically under constant change, making copies of the data both out of date and difficult to schedule. Data in the informational system is generally more stable, because data is periodically taken from operational files or databases, and various enhancements are made to enrich it into a form suitable for end-user access.

For a decision maker, data must be available in the right format, at the right time. The data must not change under the feet of the person who is analyzing it. It must be possible to run a query twice and get the same answer. Data
replication, as part of a comprehensive copy management solution, offers capabilities that address these needs.

DPropR fulfills the data replication role of the copy management function within the Information Warehouse framework. The requirements of data replication are:

- Support data currency requirements
- Let the data be tailored as it is copied
- Promote interoperability with other copy tools
- Recover automatically from system and communication errors
- Be automated and easy to use.

DPropR supports all of these requirements and provides the flexibility required to manage resources at a system level.

### 4.2 Implementation

DPropR is a set of easy-to-use, automated copy tools that copy relational data from DB2 to DB2, from DB2 to DB2/2*, and from DB2/2 to DB2/2 (see Figure 9 on page 31).

Support will also be provided for DB2/6000 and OS/400 database platforms. DPropR has an open architecture that facilitates tool interoperability.

A DataHub* workstation, as a central control point for data across the whole organization, is used to define, synchronize, automate, and manage DPropR copy operations.

The data can be tailored as it is copied; detailed, summarized, aggregated, and derived data is delivered to the target location. Tailoring is defined by SQL expressions, specified to DPropR when the copy subscription is defined. When using an MVS host as the source of the copy operations, you can replace the data in the target location (Refresh) or apply only the changes since the last full copy (Update).2 The Update capability also allows you to create a complete history of changes.

---

2 When the source data is in DB2/2, only Refresh processing is possible, as there is no CAPTURE component for DB2/2 as of this date. CAPTURE/MVS is the component that reads the DB2 recovery log and records changes.
There are three logical components to the DataPropagator Relational product:

**CAPTURE**
Captures changes in the base table and puts them onto a staging table.

**APPLY**
Pulls data out of either the base table or a staging table, possibly enhances the data, and applies it to the target copies.

**ADMINISTRATION**
Runs on the DataHub control point to support the copy specifications and control functions.

Each component runs independently. The components rely on Distributed Relational Database Architecture* (DRDA*) connectivity between the source system and the DataHub control point. The DPropR administration facility causes DataHub to issue SQL statements across a DRDA link to update control tables for the CAPTURE and APPLY components. The APPLY component uses a DRDA connection to the source system to select the data.

### 4.2.1 Capture and Data Staging

When DB2 makes any changes to DB2 tables it puts into the log the minimum information needed to recover from an error situation. Specification of `DATA CAPTURE CHANGES` on the `DB2 CREATE TABLE` SQL statement causes DB2 to expand the amount of information it logs for changes to that table.

The CAPTURE component periodically scans the DB2 log using the DB2 Instrumentation Facility Interface, looking for information about requested tables.
The CAPTURE component can reconstruct “before” images, “after” images, or both, depending on the administrator’s specifications.

Selection criteria can be set up by the administrator so that each change to the base table inserts a row into a CHANGES table. The CHANGE table has a history of changes that have been applied to the base table with time stamps. This copy of the changes is referred to as a staging table.

To ensure that only committed changes are captured, a UNIT OF WORK table tracks the commit points. This table may be joined with the CHANGES table to get an accurate picture of committed changes. See Figure 10 which shows the relationship of the tables used by DPropR to the ADMINISTRATION, CAPTURE, and APPLY components.

The CAPTURE component maintains the staging tables on the basis of user-specified parameters and completion of the APPLY operations to ensure that:

- All requested changes to the base (source) table are logged
- The APPLY component copies consistent data
- Once copied successfully by the apply components, change data is deleted from the staging tables so the tables do not continue to expand.
4.2.2 Applying the Changes

The APPLY component is installed wherever a copy of the source data is needed. If the source data is on DB2 (MVS), a Refresh or Update copy can be established on the same MVS system on which the CAPTURE is running, on a different MVS system, or on a system with OS/2* and DB2/2. If the source data is on DB2/2, a Refresh copy can be established on the same OS/2 system or another OS/2 system.

The APPLY component runs at user-specified intervals, depending on the application and business need. The APPLY component can read data from the base table, the CHANGES table, or a combination of the CHANGES and UNIT OF WORK tables.

The APPLY function is used to:

- Select a subset of rows to be applied
- Select a subset of columns to be applied
- Derive new columns based on aggregation and summarization
- Provide summaries of the changes rather than the base table
- Run any dynamic SQL statement preceding and/or following the copy, perhaps to remove indexes before running an update.

The APPLY component also supports append and replace logic, thus allowing a historical record of changes or a point in time picture of changes to be maintained.

The choice of updating or refreshing the target data depends on the business need as well as the volatility of the data when compared to the copy frequency. The size of the base table should also be considered.

The APPLY component provides a comprehensive range of enhancement facilities to support variations in copy requirements.

4.2.3 Administration

DataHub provides a workstation-based graphical front end for the DPropR administration functions. DataHub allows the administrator to:

- Control CAPTURE operations, such as:
  - Column selection
  - Before and after image selection
  - Consistency controls (through the use of the UNIT OF WORK table).
- Control APPLY specifications, such as:
  - How often APPLY should run
  - Whether to replace or append the data
  - Requirements for data enhancement.

The administration functions can be divided into two main areas:

<table>
<thead>
<tr>
<th>Registration</th>
<th>Defining copy sources: base table, staging table, other approved data format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscription</td>
<td>Defining the copy table, including copy frequency and data enhancements.</td>
</tr>
</tbody>
</table>
4.2.4 Update and Refresh Considerations

This section provides some guidelines for deciding whether to do Update or Refresh processing.

The decision about whether to update existing data with changes from the source tables or do a total refresh is dependent on the business need, the amount of change, and the opportunity for getting data from the data source, as illustrated in Figure 11.

![Figure 11. Refresh and Update Considerations](image)

The business need should determine the type of data to be extracted from the operational system. Users may want to have access to a complete log of all changes that have been applied to the operational system over a period of time, or just an end of period picture of the operational system. These user requirements will influence whether refresh or update processing is requested.

The decision is also affected by the amount of change in the operational source system. The amount of change is comprised of the percentage of rows changed in any period and/or the number of changes that have occurred. Certain types of applications will update a large proportion of data items during a processing period. Others may update a small subset of information but do it many times.

Depending on the amount of change and the user requirements, it may be more efficient in terms of CPU and elapsed times to refresh the data rather than update it. The refresh processing could be done by DPropR or by Data Refresher. The choice of method is to some extent dependent on the opportunity, because use of Data Refresher will require the operational system to be stopped while extraction is taking place. If there is opportunity to stop the
operational system, Data Refresher will be more efficient in refreshing large amounts of data.

This publication is oriented toward handling and implementing large informational databases, with particular focus on the problems of storing historical data. In some cases the large volume of data involved in refresh processing, especially the size of extract files, may mean that refresh processing will be too costly. Update processing, specifically adding before and after images to a large informational table, will consume far fewer resources than a full refresh in those cases.

4.3 DataPropagator NonRelational

DataPropagator NonRelational performs functions similar to DPropR and supports IMS/ESA\* DB (IMS-DB) databases as well as DB2.

IMS-DB customers have a considerable investment in their IMS-DB databases, which often support their core business applications. DataPropagator NonRelational allows installations to migrate to relational databases over a period of time by maintaining consistency between an IMS-DB database and a DB2 copy of that database. DPropR may then be used to propagate changes to the DB2 copy across the enterprise, as described above. The installation may realize the benefits which are available to a relational database user using decision support systems and other tools.

4.4 Summary

DataPropagator Relational is a new addition to the Information Warehouse list of products. Its chief attraction is that operational applications do not need to be stopped to allow data to be extracted because DPropR reads the DB2 log and interferes very minimally with operational applications.

The amount of information that needs to be read from the DB2 log depends on the size and degree of changes in the operational applications. Using DPropR to capture changes to a table that changes many millions of times per day is debatable. Few users would be capable of analyzing many millions of rows of data unless some form of aggregation were applied beforehand. The APPLY component would need to process all of these changes, which could potentially take a significant amount of CPU resource depending on whether the Update or Refresh option is chosen.

The reliance on DataHub to control the facilities offered by DPropR means that installation of the products and getting started requires more planning than, for example, a simple Interactive System Productivity Facility (ISPF) dialog running on an MVS system. This is more than compensated for, however, by the effectiveness of the graphical user interface, which is capable of controlling many systems running DPropR.

The goal of 24 hours, 7 days a week for operational systems is now a step closer because of reduced interference from accesses to operational data by users of informational applications such as DSSs. The replication and copy management tools that build the informational data are designed to support that goal.
Chapter 5. The Role of Data Compression

When large quantities of DB2 data need to be stored, the cost of physically storing the data can represent a considerable proportion of the total system cost. Large amounts of DB2 data may mean that a significant proportion of all DASD is used directly or indirectly by DB2.

One of the challenges in providing a large-volume Information Warehouse is handling the volume of DB2 data in an effective manner. Implementing a tiered approach to providing information for users may well cause a degree of data duplication, further increasing the demand for storage devices.

![Figure 12. We Always Need to Store More Data Than We Have Space For](image)

When costs increase, DASD space reduction analysis inevitably occurs; large tables are often singled out for reduction of the amount of data being stored. This is often achieved by reducing the amount of time data is held in the table, by removal of unwanted columns, by the filtering of rows inserted into the table, or some combination of these.

DB2 data compression reduces data storage costs with hardware and software approaches which are described below.

5.1 DB2 Data Compression

The objective of data compression is to allow an installation to compress a tablespace (or partition) to reduce DASD usage with a minimum of CPU overhead.
With DB2 Version 3, when creating a tablespace it is possible to specify COMPRESS YES or NO (NO is the default). ALTER TABLESPACE is also enhanced to support COMPRESS YES/NO.3

Compression is done at the row level and is invoked after any DB2 exit routines. Decompression occurs before any DB2 exit routines are applied while the data is being read. See Figure 13. Once compressed, a page may contain up to 255 rows (previously 127).

Installations with ES/9000 511- or 711-based processors can use a facility (Hardware Assisted Data Compression) that provides an instruction to perform compression. Other installations can use software emulation within DB2, but this approach requires more CPU time.

Data compression is achieved through a Lempel Ziv algorithm, which uses two static dictionaries (one for reading, one for writing) for each tablespace or tablespace partition. An average compression percentage of 50% may be achieved, although compression ratios can vary from 80% to 5%, depending on the data types and value distribution.

Figure 13. The Compression and Decompression Life Cycle

---

3 Once a tablespace is compressed, it must be moved to a noncompressed tablespace to take out the compressed rows. This decompression may be done with UNLOAD/RELOAD.
5.1.1 Dictionaries

The dictionary critically influences the compression ratio. The larger the dictionary, the better the compression ratio. The dictionary size is a minimum of 2 pages to a maximum of 17 pages. The dictionary for a tablespace is anchored in the page header, after the space map, but before the data pages for that tablespace.

Dictionaries are built by the LOAD and REORG utilities, which have been extended to support some additional parameters. One of these parameters is \( n \), the number of rows to be sampled when building the dictionary.

When loading data with the REPLACE or RESUME NO parameters, the first \( n \) rows are used to build a dictionary tree. This dictionary tree defaults to 4096 nodes of 8 bytes each to hold dictionary information. The dictionary tree is translated into a dictionary when the tree is full, or when the data RELOAD phase is exhausted (that is, when there are a small number of rows to load). Because compression is best suited to large tables, this latter case should rarely occur.

The dictionary is validated once it has been closed.

The use of LOAD REPLACE KEEPDICTIONARY avoids the CPU time needed to build a new dictionary.

When reorganizing data, the dictionary tree is built from the first \( n \) rows unloaded from the table. The REORG utility samples the remainder of the data for improvements to the dictionary. The sampling rate depends on the number of rows (\( n \)) used to build the initial dictionary tree, specified in the parameters. The dictionary tree is converted into a dictionary at the end of the unload phase.

The use of REORG KEEPDICTIONARY may save CPU time used to expand the unload records. It will also avoid the CPU time needed to build a new dictionary and compress the records.

A row is stored in its compressed form only if the COMPRESS option on the partition or tablespace is YES, a dictionary is available, and the compressed row is shorter than the decompressed row.

Data in the DB2 log and in the buffer pools is in compressed form. The DSN1COPY and DSN1PRNT utilities keep the data compressed, as does a REORG UNLOAD CONTINUE KEEPDICTIONARY.

5.1.2 Compression Statistics

At the end of the LOAD and REORG utilities, compression information is put into the DB2 catalog tables and the SYSOUT data sets. The catalog information can be updated with the RUNSTATS utility. The values include the overhead for dictionary pages.

The PAGESAVE column of SYSTABLEPART contains the percentage of pages saved:

\[
\%PAGESAVE = 1 - \frac{\text{compressed bytes}}{\text{decompressed bytes}}
\]
The PCTROWCOMP in SYSTABLES contains the percentage of rows that are compressed per table. The PCTROWCOMP in SYSTABSTATS contains the percentage of rows that are compressed per partition.

The DSN1COMP utility can be used to estimate the gains to be made from DB2 compression. Input can be from:

- DB2 image copy data sets
- VSAM data sets that contain DB2 tablespaces or tablespace partitions
- Sequential data sets containing the data that is to be loaded into a DB2 tablespace (or tablespace partition). This may be the output of an unload.

The utility builds dictionary information to estimate the percentage of bytes saved and the percentage of DB2 pages saved.

5.2 Benefits and Considerations

Compression is particularly recommended for large tables, especially where the 64GB limit may nearly be reached, or in cases where it may be desirable to compress less active partitions. In these cases a 50% to 80% DASD saving could possibly be achieved, depending on the data type and characteristics. Compression also results in a reduced I/O rate when retrieving the compressed data and can improve performance when recovering from an image copy.

Use of a compression routine for space savings can result in increased CPU utilization, particularly if Hardware Assisted Data Compression is not used.

Use of DSN1COMP to identify candidate tablespaces or partitions is recommended, because the compression ratio for tablespaces can vary greatly. A good compression ratio is 80%; a poor one could result in negligible space savings while still using additional CPU resources. Therefore, the benefits of space savings must be evaluated against the requirement for additional CPU time. A realistic estimate of the overhead of compression with the use of Hardware Assisted Data Compression would be 3% to 5%. A 10% space saving could be used as a minimum rule of thumb for compression to pay for itself. The degree of normalization influences the compression ratio; less normalized data may compress better.

Do not use LOAD for compressing tablespaces containing multiple tables. LOAD only examines the first \( n \) rows and hence bases compression on a dictionary based on the first table. REORG would be preferable in this case because it continues to sample data until it has sampled all of the data.

There will be up to 255 rows per page when a tablespace is compressed, resulting in fewer I/Os, a higher hit ratio for buffer pools, more rows locked by a page lock, and an increased CPU time for a tablespace scan. There is the advantage that more rows will be available in the buffer pools.

The DB2 optimizer considers compression of tablespaces and partitions when determining access path selection.

Writing DB2 data involves searching a tree of dictionary terms, similar to searching a dictionary for a word’s definition rather than for the word itself. Reading only requires a dictionary reference, as you would look up a word in a dictionary. On average, reading will be about four times faster than writing.
It is not recommended that all tablespaces be defined to be compressed. Compression has an overhead, even when done by hardware. The fastest performance can be achieved for tablespaces that can remain resident in memory without compression because there will be minimal CPU overhead.
This chapter describes a method in which potentially large volumes of mostly historical information can be stored without becoming a drain on DASD and CPU resources.

Some of the concepts described in this section are based on the information in DB2 Version 2 Release 2 Design Guidelines for High Performance. Although that book was written in 1989, and thus some performance measurement results are out of date, the concepts discussed remain valid.

## 6.1 The Problem

For various reasons, many installations need to store information that has been collected over time. Some of these reasons might include:

- Legal or taxation requirements
- Company policy
- Auditing by accounting or computer staff
- Trend analysis by marketing, planning, or supervisory staff
- Anticipating potential new business requirements that require history data to be available at installation
- Correlation of data from other sources.

Data of this form that has been collected over a period of time and contains references to events at a point in time is called **time series data**. This data is often referenced by the date of the events, and access might focus on an individual data item or a period of history. Generally the frequency of access to data decreases as data ages, but there are certain periods where historical information will be accessed regularly for reports, such as monthly, quarterly, and annual summarizations.

Figure 14 on page 44 shows how data accesses to an individual data item or collection vary over time. Depending on the type of data being stored, the decrease in access frequencies may be gradual as shown in Figure 14, or quite acute. Order information may get many accesses immediately after the placement of the order and then decrease as the order is built and shipped. The data relating to the order may be used as input to improvement processes that cause the data to be periodically accessed over some time, such as quarterly measurement reports. These reports provide measurement of the order fulfillment process over time.

Alternatively, order costing information may be referenced frequently while charges for the order are being calculated but infrequently once the bill has been sent and paid.

There is a requirement in many organizations to store information with access patterns similar to the order and billing data over a potentially large timescale without special actions being necessary by the data requester.
This problem is compounded by the fact that it is often difficult to determine when data may be safely deleted without any repercussions. Data is a valuable resource for an organization to have. To throw it away can be costly.

Some installations have stored this information as hard copy or as image data, but ideally it needs to be in a format that will allow some form of automated processing at a later date. This may be achieved by storing it in historical tables that are added to and updated by some periodic housekeeping activities. In an Information Warehouse environment, this detailed information can be used for multiple purposes. It can be used as derived or reconciled informational tables for end user access, or it can be used to build additional derived or reconciled information stores.

Because of the large volumes associated with storing historical data, and the fact that users will want to access the data through some form of date processing, careful thought needs to go into how the data is stored.

The large volume of data presented to the user means that any ad hoc query will potentially have long run times. Something must be done to allow the user to select time periods to reduce the amount of data accessed. Including a date column in the index helps, but sometimes the ideal access path is not chosen, causing a scan of large volumes of data.

If you maintain history data with a date field as the primary key, you could experience “index creeping,” where key values are constantly increasing, resulting in large numbers of page splits and large amounts of empty space in the index. See Figure 15 on page 45. You may also get “hot spots” in the index and data, where the data is always being accessed.
Once data has been held for a period of time, much of it is no longer needed and may be deleted. Depending on the particular circumstances, large numbers of data and index pages may be affected by any SQL-based delete activity across the whole tablespace. This can generate potentially large amounts of I/O and logging activity.

6.2 Partition Wrapping: A Possible Solution

To reduce the problems associated with holding large time-based data tables with irregular access patterns, a process called “partition wrapping” is used. As its name suggests, partitioned tablespaces are used, together with a method of reusing partitions as the data is deleted and new data is added.

6.2.1 Implementation of Partition Wrapping

If the data is stored in date sequence in the table, we could partition the table so that we have one partition per month, or one partition per quarter. This partitioning could be done by using a date field as the clustering index, but this would cause the index creeping problem described above.

![Diagram of partition wrapping](image)

Figure 15. Possible Data and Index Implementations

It is not possible to ALTER the partitioning index clustering information, so it is not possible to periodically change the date values to match the data. It is also unsuitable to frequently unload, drop, redefine, and reload the table.

An alternative would be to define a key that naturally “cycles” rather than constantly increases. Month is an example of a cycling key. The proper date
field could be included in the index, but not as the first column. This approach at first appears to be ideal, but it has a few major disadvantages.

Assuming there was data for more than one year in the table, as shown in Figure 15 on page 45, a user accessing data for Feb 94 will be accessing data and index pages shared with Feb 93, thus increasing the number of pages read to retrieve Feb 94 data. A request to access six months of data may result in accessing 50% of the data pages when only 25% are needed. This approach also prevents the use of techniques to place older data on slower devices.

A solution in this case is to add an additional column to each row of the table that has no business meaning but is used to control the partitioning of the table. The value relates to a time period that is used in conjunction with a reference table used to control placement of data into the partitions of the historical table.

The reference table (Figure 16 on page 47) might have columns as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PN</strong></td>
<td>The partition field value or partition number. The PN is added to each row of the table and is defined as the primary partitioning key of the table, thus determining the physical partition in which the row will be placed. There will be one row in the reference table for each value of PN. Because of the number of rows to be held in the large historical table, the PN value should be small. SMALLINT or CHAR(2) would be suitable data types.</td>
</tr>
<tr>
<td><strong>Period Label</strong></td>
<td>This column indicates the period to which the row in the reference table relates and hence the associated partition value in the history table. Typical values might take the form yyyy-mm or nQyy, where yy is the year, n is the quarter, and mm is the month.</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>This column is used to achieve the partition wrapping requirement. There is always the same number of distinct ID values stored in the table as there are physical table partitions. The actual value used depends on the specific implementation, but the approach is always the same. The general idea is that some ID value is always thought of as the CURRENT TIME PERIOD, say 0. Depending on the specific implementation, 1 may relate to the previous period or the next period. Assuming that 1 relates to the previous period, it could be said that data with an ID of 1 is one month or quarter or year old, and that data with an ID of 2 is two months or quarters or years old. The column name could even be called AGE. Alternatively, -1 could relate to the previous period, -2 to the period before that, indicating current month minus two.</td>
</tr>
</tbody>
</table>

Each column of the reference table could be indexed.
The solution described above is best illustrated by an example.

6.2.2 Example: Electronic Equipment Manufacturer

Consider a major, international, electronic equipment manufacturer shipping products throughout the world. The marketing of the products is controlled largely by individual country locations that place the orders on a centrally run system.

In any day more than 10,000 orders may be placed and 10,000 orders fulfilled. Orders may also be updated to change configurations, requested date, and so forth. There may be up to 30,000 order changes in any one day.

The company wants to track the progress of all orders from the moment they are placed, keeping any changes to an order until the date it is fulfilled plus an additional three months. This detail of order history is needed to monitor order quality and the degree of change to orders involving key products. Except for the time required to update the data in the information store from an extract from the operational systems, the information store is available to end users at all times.

A periodic housekeeping process extracts the last recorded state of fulfilled orders older than three months. This data is then loaded into a historical table containing 33+ months of fulfilled orders. The information is stored on a partitioned table that has 12 partitions, each partition containing three months of data:
The partition wrapping reference table is used to determine which partition to load the data into next and which partition relates to which time period for the user views. The user views are created referencing particular values of ID, as illustrated in Figure 17.

```
CREATE VIEW ORDER_HIST_LAST
(
    PERIOD_LABEL ,
    ORDER_NUM ,
    CNTL_STMP ,
    PN ,
    ID
)
AS SELECT
    B.PERIOD_LABEL ,
    A.ORDER_NUM ,
    A.CNTL_STMP ,
    B.PN ,
    B.ID
FROM
    ORDER_HIST_FULL A
    , REFERENCE_F B
WHERE
    B.ID = 0 AND
    A.PN = B.PN
;

CREATE VIEW ORDER_HIST_P2
(
    PERIOD_LABEL ,
    ORDER_NUM ,
    CNTL_STMP ,
    PN ,
    ID
)
AS SELECT
    B.PERIOD_LABEL ,
    A.ORDER_NUM ,
    A.CNTL_STMP ,
    B.PN ,
    B.ID
FROM
    ORDER_HIST_FULL A
    , REFERENCE_F B
WHERE
    B.ID = 1 AND
    A.PN = B.PN
;
```

*Figure 17. Sample View Creation*

It could be argued that the user should be spared implementation details such as ID and PARTNO. They are included here for illustration, but given appropriate education, users will select the right view to get the data in which they are interested.

A general-purpose view could be set up to cover all periods held on the history table. These views could include all of the columns from the reference data table that could be useful to the end user in clarifying which period the data is from. Of course, access to the full order history going back 33 months would be needed only infrequently. The majority of queries from customers or representatives would normally be for open orders or orders that have just been fulfilled. Output from a last history period view might be as follows:
SELECT * FROM ORDER_HIST_LAST

---------+--------+---------+---------+---------+---------+----
PERIOD_LABEL ORDER_NUM CNTL_STMP PN ID
---------+--------+---------+---------+---------+---------+----
4Q 93    B1234567890A 1993-10-26-19.55.02.454552 9 0
4Q 93    B1234567890A 1993-11-26-19.51.03.015054 9 0
4Q 93    B1234567890A 1993-12-26-19.54.10.448612 9 0

Alternatively the PN and ID columns could be excluded, totally hiding the
implementation from the user.

Note: This example is covered in more detail in Appendix A, “An Example of
Partition Wrapping” on page 107.

6.2.3 View Creation Considerations

Views can be created for current period, last period, and other time intervals
potentially covering many months. When creating the views it is inadvisable to
include specific dates, either in the view name or the view definition. Such
inclusion would require additional processes to periodically change the details at
the end of every period. Authorizations on views would need to be regranted;
synonyms created by users would be dropped; application plans would be
marked as invalid, forcing a bind at the next invocation. A suitable naming
convention might be ORDER_HIST_CURR, ORDER_HIST_P1, ORDER_HIST_P2.

Experience has shown that users are interested in accessing data in the simplest
and quickest way possible, so a full view can be made available to users to
reduce the need for UNIONS. Appropriate indexing is highly desirable to
maintain performance.

6.2.4 Table Maintenance

The implementation of partition wrapping has one handicap: some additional
periodic maintenance to maintain the ID column.

At the end of the period, the ID pointers need to be incremented and/or
decremented (or reset to zero in one case) to point to the next “available”
partition. The term “available” is used even though the oldest partition is not
empty but is to be overwritten by newer data. This overwriting can be done by
using the LOAD utility to LOAD REPLACE into the next partition (see Figure 45
on page 115).

The periodic activity is summarized as follows (assuming there is one partition
for each period of information held):

1. Process the following SQL against the reference table, possibly using a
simple piece of application code, or SPUFI input records:

   UPDATE REFERENCE SET ID = ID + 1

   UPDATE REFERENCE SET ID = 0 WHERE ID = 12

   UPDATE REFERENCE SET PERIOD = 'implementation defined'
       representation of new period'
       WHERE ID = 0
2. Unload the data from the operational table by means of a view. The view is defined as a join of the operational table with the wrapping reference table, requesting the PN from the REFERENCE table as the first column, and a where clause specifying REFERENCE.ID=0. The result is a load file where each line contains the partition number to which the data is to be loaded.

3. Using some routine determine the partition number by reading the first two bytes of the first line of the load file. Alternatively, issue:

   SELECT PN FROM REFERENCE WHERE ID = 0

   Use this number to update a standard set of LOAD cards to tell the load utility into which partition to load the data.

4. Load REPLACE the data into the information table.

Note: An alternative approach to steps 1, 2, and 3 is to produce a conventional unload file, then invoke a small application program to produce an unload file of the correct format (see Appendix B.1, “Adjusting the Partition Number” on page 124).

The reference table can potentially be shared across several time series tables. If the timing of the housekeeping activities on these tables does not match, an additional column could be added to the table to identify the tables to which a particular reference row relates.

This maintenance activity is made simpler if the table is set up with the same number of partitions as there are years, months, or days to be stored.

If, as in the electronic equipment manufacturer example, a partition is used to store information relating to a quarter, another partition may be needed in cases where the data is put into the historical table on a monthly basis.

For example, if the business requirement is to hold data for at least the past 12 quarters (36 months), an additional (thirteenth) partition would be needed to collect information for the new quarter. If we assume that we are about to load data (not LOAD REPLACE in this instance) relating to Jan 1994, its natural place would be in partition 10, overwriting the information for Jan 1991, because that (according to the reference table) is the next partition to use. This would potentially lead to a problem. The view is defined solely through the partition number, but this would give a mix of 1994 and 1991 data. We do not want to require additional complex date logic in the view definition, because that would require views (and hence authorizations) to be modified monthly.

For this reason, as well as simplicity, it is advisable to have the same number of partitions as there are periods to be represented. This approach eases the periodic data management activities.

---

4 This unload view could be replaced by the output from Data Refresher, possibly using a field exit to determine the correct partition number to use.

5 See page 116 for an example of code to handle an empty unload.
6.3 Benefits of Partition Wrapping

There are various benefits of using a partition wrapping approach to historical data. These benefits are in addition to the reduction of wasted space and processing time that might result from using nonpartitioned tables or partitioned tables that do not use this technique of separating records by date.

The physical size and characteristics of the tablespace partitions can be altered individually, with no effect on any others. You could start with one partition that is large enough to hold the data put into it and allocate all other partitions with a nominal 4K size. This will allow the size of the tablespace to grow over a potentially long period, spreading the acquisition cost of the storage media over a longer period.

If it is recognized that old data is likely to be used infrequently, you could compress and even migrate it to slower, cheaper storage media. Figure 18 on page 52 illustrates a typical situation with historical data, where the older data is less frequently referenced. Migration could be done by the HMIGRATE facility of DFSMShsm or by some data management function that moves the data, such as a utility.

The compression could be done by DB2, and the migration by DFSMS after the tablespace has not been referenced for some specified period. It may be desirable to daily ‘force’ a migration to DFSMShsm migration level 1 or even migration level 2 devices (typically tape) to save on DB2 DASD pool space. This migration can be done by stopping individual tablespaces during the DB2 or system housekeeping slot, then issuing the HMIGRATE command against the data sets containing the older data. An example of code to stop the database and cause DFSMShsm to migrate the tablespace (either partitioned or not) can be found in B.2, “REXX Example to HMIGRATE Partitions” on page 130.
If a user executes some SQL that accesses these migrated data sets, a DFSMShsm RECALL would be initiated. Depending on the locally defined RECALL and DELAY parameters on the Operator Functions panel (DSNTIPO) and the time to retrieve the appropriate data set, DB2 may issue an SQLCODE -904 (RESOURCE UNAVAILABLE) message for the data set. This message often goes away when the SQL is attempted a second time. Users should be informed of this possibility.

Perhaps a better solution would be to move older partitions to a slower device that DB2 can directly access, such as the 3995 Model 153 or the 3390 Model 9. The cost of storage is cheaper, but the access times can be longer. You can build jobs that move the VSAM data sets containing the older data based on a report produced from the REFERENCE table. You can then refine the REXX EXEC provided in B.2, “REXX Example to HMIGRATE Partitions” on page 130 to capture the volumes occupied by table spaces, and then generate a job to invoke the DFSMS utilities to move data to or from 3390-3 DASD and 3390-9 DASD or 3995 optical devices. A sample job is provided in B.3, “Job to Move Tablespace” on page 133. The tablespace can be moved after periodic maintenance has been applied.

Moving the data allows an installation to balance the importance (and therefore access frequency) of the data with the cost of storing it. Reports can be run to determine how often each data set is accessed each day. The statistics could be used to highlight cases where a data set is being used more frequently than expected.
DFSMS I/O statistics reports can be used for this purpose. SMF type 42 records, subtype 6 for SMS-managed data sets and subtype 5 for non-SMS-managed data sets, can be used to collect information on individual data sets. However, DB2’s buffer pools will mask the frequency of access to a particular tablespace.

This information can also be obtained from a DB2 Performance Monitor I/O Activity Report, by starting a Performance Class 4 trace. The trace should be done only when trying to identify the reason for poor performance.

### 6.4 Summary

The use of the partition wrapping concept allows:

- Grouping of data into period-based “buckets”
- Faster loading and deletion of data
- Running of utilities against very large tables
- Reduced run time of these utilities (because only one partition is affected)
- User choice of which view to use depending on his or her needs. The need for a scan of all historical data is reduced.
- Ability to direct DFSMShsm to migrate unused partitions if unreferenced for a period of time, but only if they are placed on DFSMS volumes, and the tablespace has been stopped.
Chapter 7. Data Archive Retrieval Manager

For various reasons, many installations need to store information they have collected over time. The objective of Data Archive Retrieval Manager (DARM) is to provide an automated facility for separating aged data from active data on DB2 tables, efficiently storing that data over a long interval, and selectively retrieving it on demand.

As data is collected from operational applications, the amount of storage required also grows, but not all data is continually active (see Figure 19).

![Data Usage Chart]

*Figure 19. Data Access over Time*

Some data will ultimately reach a stage where it is never updated again and is infrequently referenced. This type of data might be part of a reconciled data store that is input to a data summarization or aggregation process. Once the summarization is complete, the data may never be referenced again, or only infrequently, to rebuild the aggregation in cases of error in some of the input data (see Figure 20 on page 56). This type of data might be too primitive or too voluminous for useful access by end users. Keeping such data on fast DASD will only occupy valuable space that can be best utilized by other data.

DARM allows an installation to control the size of DB2 tables, helping to maintain acceptable application performance levels and separate frequent and infrequently accessed data.
7.1 Functions

Data Archive Retrieval Manager provides four major functions working together to achieve a total solution (see Figure 21 on page 57):

**Archive**

The process of identifying and transferring rows of data from production databases to the archive based on specified selection criteria.

**Storage management**

Several programs to permit management of the data sets containing archived table data.

**Retrieve**

A process for recovering and selectively copying archived data from the archive to designated tables.

**Archive maintenance utility**

Processes for defining archive characteristics, interrogating archive status, and managing various performance factors related to each archive. Other maintenance actions against the archive definition are also supported.
7.1.1 Archive

The archiving (removal) of aged data (row-by-row) from the active data store is desirable and often imperative to relieve demand on the highest performance DASD devices (see Figure 22 on page 58). At times the older data can be completely discarded because the application, organization, and government no longer have any need for it. Often, however, some of the information must be retained in a retrievable form. The application or the organization may have some future need (currently defined or undefined) or desire to use selected portions of it. Definition of which portions to retrieve may occur only when the retrieval is imminent. Some data is required to be retained to satisfy legal or procedural demands of a government. In this instance, the information must be stored for a specified time. Retrievals again will probably be defined at the time they are to be performed.
It is important that each application can select and determine which of its data is eligible to be archived or discarded. DARM is designed to implement such rules. These rules can be as simple as a single WHERE clause of an SQL statement or as complex as a program.

DARM can manage archiving of individual tables or systems of tables that are connected or related to one another. Thus data rows from one table may become eligible for archive, depending on archive eligibility of a data row in another table. DARM is not constrained by referential integrity relationships but is supportive of them.

The archive process supervises the selection of eligible data rows from the one or more tables that participate in an archive and their transfer to the archive. This transfer is usually a MOVE, where the original data is deleted following inclusion in the archive. DARM also supports COPY, where the original data remains where it was. Copying is desirable when the archive is to serve as a point-in-time image of some portion of the data that has not become inactive. Both the MOVE and COPY operations must currently be between objects on the same DB2 subsystem.

The archive process can also serve to DELETE data that should be removed from the tables. The delete can be done by entire rows, or by exclusion of specific columns within a row as the remainder of the row is moved to the archive. A REFER capability allows linking of two tables through a third, which is not otherwise participating in the archive.

The archive process is sensitive to the quantity of data being written to the archive and supports division of this data into manageable units. You may
specify the maximum size for an archive. DARM will use the same archive (possibly over repeated runs) until the specified archive is full. It also invokes the storage management function of DUMP to transfer archived data to other media (for example, tape) to free expensive DASD space for reuse.

7.1.2 Storage Management

The storage method for archived data is of interest because it must be stable over a period of years and even tens of years. However, if the tools to retrieve data do not exist or function at the time the data must be retrieved, storage has been for nothing. DARM uses DB2 tables for this storage because they have been defined as a stable format. Migration of the archived data to other media (such as magnetic tape) is managed by additional stable mechanisms such as DFSMShsm and DFSMSdss.

The DARM storage management process provides installations with the ability to control the residence of archive data. Its principal purposes are to move freshly archived data from DASD to tape, provide duplicate copies of those tapes, and recover (COPY) data from tape to DASD for retrieval processing. It is also possible to make additional copies of dumped data and to exchange such a secondary copy with its primary when the primary has become damaged or is otherwise unusable. Specific tapes by volume serial may be used, or the system will accept any volume mounted by the operator.

7.1.3 Retrieve

The exact details of the retrieval process are often not known until retrieval is about to take place, because of the varying purposes of retrieving the data, such as recovery and auditing. For this reason the tool used to retrieve the data is SQL, with benefits such as stability, ease of use, and thorough documentation.

To support data retrievals, indexes to the archived data are supported. These indexes will assist in the detection of interesting information and its location within the greater body of archived data. DARM provides for the creation and maintenance of application-defined indexes to assist in the location and efficient recovery of archived data. An Interactive System Productivity Facility (ISPF) dialog is provided to assist in retrieval of data.

The retrieval process collects retrieve requests from individual users, optimizes available requests for processing at installation-specified intervals, and copies individual rows (or selected columns from rows) to a destination table chosen by the user. If the archive data being sought has been transferred to an alternative medium (such as tape), the retrieve process invokes the DFSMS storage management processes to recover that data to DASD for access.

Individual users can devise their own retrieval requests through an interactive interface supplied with DARM. A selection among available archives, tables, and columns is made to customize each retrieval to the current requirements of the user. The retrieval is conditioned upon SQL WHERE clauses that the user provides. Access to the retrieval dialog should be tightly controlled to restrict the amount of data being brought back.

After retrieval, the retrieved data may be used as any other DB2 table, allowing analysis with any tool capable of accessing DB2 data.
Because of the large volumes of data that can potentially be held in the archive, some staging process may be needed to manage them. There may not be enough DASD defined to allow the full archive of data to be brought back, so small retrievals may be necessary.

### 7.1.4 Archive Maintenance Utility

The Archive Maintenance Utility program collects parameters that constitute the archive definitions such as:

- Name of the archive
- Production tables
- Other specifications, such as parameters used to tune the performance of the archive and to determine when automated functions are to be scheduled.

All such information is in the definition of the archive maintained by the utility. The utility also can be used to observe archive status and to manipulate selected information about each archive. A basic security system is implemented to limit the access to this information among selected users.

The ability to export and import archive definitions is supported, together with removal of aged and discardable portions of the archive.

### 7.2 Capacity and Performance

The capacity of any single archive is limited only by the amount of space it is permitted to occupy. Additional magnetic tapes can be acquired for storage of archived data. The real limit becomes the ability to manage the media and its physical storage, for example, the need to periodically recopy each tape to assure that the data is readable and refreshed.

DARM insists on maintaining a minimum of two copies of any dumped data under its control. The use of optical devices could be considered for this purpose because they require no periodic upkeep to maintain their data integrity.

Performance of DARM and its impact on other activities within an installation is an important consideration. It is possible for an archive to interfere with production activities, particularly in a nearly continuous availability environment. For this reason, DARM is operated as a low priority activity within the MVS system and consumes resources only to the extent that other concurrently active tasks are not using them.

The archive administrator is given access to certain tuning parameters to provide some control over lock contention and event frequencies. Some lockouts and contention have been observed where DARM creates new archive tables and prepares new plans to archive to those tables. This contention is no more than would be experienced by any other, similar, long-running application program. Because of the manner in which the COMMIT interval has been established, little effort is required to resume an interrupted process.

SQL is used heavily by DARM and consequently DB2 can do large amounts of logging when DARM is active. This should be considered when determining the best time to perform archiving.
7.3 Recovery and Stability

Data Archive Retrieval Manager reliably stores the user’s data by exploiting features of DB2. Movement of data from the production tables to the archive occurs within a common commit interval to allow recovery of the data in the event of a failure through execution of DB2 backout processes. DB2 image copies of table data (production and archive) and the system logs are available to support the recovery of lost or damaged data in other circumstances.

As stated above, when the archived data is moved to alternative media (usually tape), DARM insists on making two copies. This is normally sufficient to prevent accidental destruction. It is advised that the two copies be stored separately to prevent their destruction through a single agent or event.

The Resource Access Control Facility (RACF) and the access privileges granted to DB2 users are used to limit the availability of the archived data, and the programs and facilities of DARM, to selected individuals or groups. No additional levels of security are enforced by DARM.

Each component of DARM uses functions of other program products as much as possible. For example, all structured data movement is accomplished with SQL as implemented in DB2. The DUMP and RECOVER of archive data sets are accomplished with DFSMSdss, and these activities are secured through RACF. The terminal interface to the Archive Manager uses ISPF facilities.

Because of the high use of program product function and the amount of interfacing between DARM and DB2 and DFSMS, certain code components need to be placed in the Authorized Program Facility (APF) libraries. This requirement may limit the speed at which archiving algorithms can be updated.

7.4 An Implementation

Data Archive Retrieval Manager is used in the TEAMS Central Information System (CINFO) (see 8.1, “TEAMS Central Information System (CINFO) Description” on page 65) to perform an archival function for tables that build a history of changes made in the operational system. Some of these tables may receive many changes in a day, so they could grow by as much as 500,000 rows on a peak day. The information is used to feed basic user tables in support of a decision support system.

The business requires that information must be available to users going back at least 13 months, and in certain cases up to two years, to satisfy trend analysis and monitoring of production build cycle time.

The data is also needed to anticipate potential new user requirements. There are situations that might require construction of a new user table that is "primed," that is, contains historical information at the outset. These situations involve looking back over time to build a picture of the state of the business as it was when the start of the history is required. Then from this stage, the process involves summarizing, aggregating, or deriving the data as necessary to produce a table that meets the user requirements.

Because of the volumes involved, the process requires careful planning and potentially large amounts of DASD space for retrieving the data into DB2 tables one portion at a time, performing data enhancement on each.
If DARM had not been used to control the copies of operational data, development of some other method of archiving using custom-developed applications would have been required. This would have cost far more than installing and deploying DARM. Use of DARM thus saves money and the development and maintenance of code.

Currently MOVE archiving is carried out every day on certain large tables only, and will be applied to user tables in the future. The archives are all date dependent—archived when the expiration date is greater than today’s date plus six days. All archives move data (and delete) from the production tables and write to tape. Data goes first to an intermediate table that is dumped to tape when full. The intermediate tablespace’s VSAM cluster is then deleted. There is some overhead from use of DARM, but the perception is that it does not consume undue resources. However, there is a lot of logging because of the amount of SQL being executed on:

- The table being deleted from
- The intermediate table that DB2 inserts the records to
- The work tables.

Performance is generally good in the TEAMS CINFO environment. Specific performance testing has not been done on DARM during its archival process. Examination of some job statistics shows that between 5MB and 7MB per minute archival rate is easily achievable, with cases of 20MB+ per minute being achieved in some cases. Larger record lengths generally process more bytes per minute because of commit frequency.

The output reports contain a lot of information about the progress being made to help keep track of what has been archived. This is not at a data level but at a row and date level. The checkpoint and restart ability has proved useful. There have been some cases where other activities have tried to perform operations that have locked DARM out of DB2. Without the checkpointing, data would have been lost. Problems can be experienced running two or more archives in parallel due to the creation of new DB2 tables and application plans.

This is a typical example of the selection criteria passed to DARM:

```sql
SELECT
    OCL_CODE ,
    ORDER_REF_NUM ,
    ORDER_LINE_TYPE ,
    PT_BUS_ID ,
    TYPE_OF_PROD_TYPE ,
    CNTL_EXPRY_MNTH ,
    CNTL_START_STMP
FROM JBQ.JBQT0840
WHERE
    CURRENT_DATE - DATE(CNTL_EXPRY_STMP) >= 6 DAYS
```

The WHERE clause identifies the rows to archive and the selected columns identify these rows for checkpointing. This does not mean that only these columns are archived.

The retrieval process has been tested but has not needed to be exercised fully to date. It is expected that it will be used frequently in the near future. It is currently used as an audit trail. All retrieval is done by operations staff at the...
present time. It is not practical to allow general user use because of space
limitations and performance considerations.

Over time, various tables that are the subject of archiving have required minor
table format changes. This has not caused a problem with DARM as retrieval of
data is not to the original table, but to another table.

The use of DARM has provided a successful way of managing table sizes, while
still allowing for future needs. The only caveats are that it can be complex to set
up because of the need to place some of the DARM code into the APF libraries
to allow that code to call the routines of DFSMS directly. Care must also be
taken when defining the names of the application plans and databases DARM
will use to ensure that it runs without contention against itself or other
applications, particularly against the catalog.

7.5 Identifying the Need for Data Archive Retrieval Manager

If you answer YES to many, some, or just one of the following questions, DARM
might be worth considering:

1. Do you have an application where the amount of data is growing and shows
   no signs of peaking?
2. Do you have an application where you frequently need to add extents to
tables?
3. Do you have problems where you cannot REORG some tables because of
   their size?
4. Is the image copy job stream a significant part of your application load?
5. Is the “batch window” requirement too long?
6. Is performance too slow because of the table size?
7. Do you have cases where a lot of the data in the tables has no present
   participation in the business?
8. Does your application periodically discard data that you would much rather
   retain?
9. Do you print historical reports to paper or microfiche and then store that
   report forever?
10. Are there complaints about not being able to find something that was stored
    (no index)?
11. Do you have legal requirements to save data for years after its
    organizational usefulness has expired?
12. Are there other reasons for keeping data for a long time?
13. Have you implemented some application-specific processes to save aged
    data, and do you subsequently spend significant resources on maintenance
    and management of those processes?
14. Would you like a tool to manage and maintain the processes and data?
15. Are your databases polluted with “dead” data that must remain available?
16. Do you feel uncomfortable if you throw anything away that someone might
    later need?
This chapter describes an actual decision support system, illustrating some of the components of the Information Warehouse Architecture and techniques used to maintain user data.

8.1 TEAMS Central Information System (CINFO) Description

IBM Information Solutions Limited has built and maintains a decision support system called TEAMS Central Information System (CINFO) at the company Headquarters in Portsmouth, United Kingdom.

This decision support system provides the informational layer for a centrally run supply management system, supporting countries and sources of supply throughout Europe, the Middle East, and Africa. The informational system and operational system combined form the largest application development project undertaken anywhere in IBM Europe, and one of the largest order processing systems anywhere in the world, handling orders from 100 countries.

The informational system is maintained using a set of custom-designed components and standard product offerings together to provide end users with the data they need to run the applications.

8.1.1 Technical Architecture

The MVS-based system coresides with the operational system on an ES/9000 Model 962 processor. The major system software components are as follows:

- IMS-DC as the transaction manager
- DB2 as the database manager
- Query Management Facility (QMF*) as the decision support tool, with some use of the Distributed Data Facility (DDF) for remote data access
- Operations Planning and Control (OPC), used to manage batch processing
- TSO/ISPF used for end-user dialog control.

8.1.2 Design and Implementation

An overall system architecture was developed, defining the technical architecture, functional components to be developed, and the needs of an informational system. The chief purposes of the informational system are for decision support, auditing changes, and error recovery in the operational systems. A history of changes is not kept in the operational systems.

Data modeling sessions were run to determine a definitive business model upon which the operational systems were built.

The informational systems architecture was designed and developed based on An Architecture for a Business and Information System, by B. Devlin and P. Murphy, IBM Systems Journal, Vol. 27, Number 1 (1988). The architectural infrastructure and components were built to support two distinct levels of data as part of a centralized information store:

- The Business Data Warehouse (BDW) layer supporting staging tables held in a nearly normalized form
• The Process Data Warehouse (PDW), also known as the Client Data Warehouse, intended to be a set of tables built to support the user decision making process.

### 8.1.3 Business Data Warehouse Layer

The BDW (Figure 23) consists of more than 100 business-related and implementation control tables, occupying nearly 10GB of DASD. The tables contain the latest recorded status of the operational applications, complete with a limited change history.

![Figure 23. CINFO BDW Component](image)

The BDW tables are implemented as nearly normalized DB2 tables, which are used as a staging area for extracts driving the building of user tables. The BDW tables contain a limited history of changes sufficient to ensure that the extract process has had time to reflect the changes in the user tables before the old data is archived. The volume of data held fluctuates according to the volume of orders on the backlog. At the time of writing 15 million rows of information were stored in this layer.

The BDW is controlled by two major components, the interface to macros and the table handlers.
8.1.3.1 Interface to Macros
The interface to macros (ITM) component is responsible for handling the information passed from the operational systems.

The operational systems send business transactions (BTXs) across an Intersystems Coupling (ISC) link. The BTXs represent a business change in the operational system. The transaction may consist of several changes to several operational tables, each with a time stamp value indicating the precise time the change was made. (Designing the operational systems in this manner—providing timestamped records of changes—offered a distinct advantage in development of the informational systems.)

The ITM uses its DB2 control tables and definitions to:
1. Take the transactions from the IMS queue, examine the BTX identifier
2. Decompose the BTX into a set of component row identifiers (RIFs) representing a change or group of changes to a single table
3. Replicate and/or pass one or more RIFs to the appropriate table handlers, and handle error situations, depending on referential integrity constraints and table partitioning considerations.

The ITM is “cloned” a number of times. Each cloned ITM looks after transactions passed across each ISC link running in parallel.

8.1.3.2 Table Handlers
A table handler is a piece of application code running as an IMS message processing program (MPP). A table handler is responsible for taking a single table instruction passed as a RIF and applying the instruction to a BDW DB2 table. Some tables have one table handler per partition. A table handler also maintains the various time stamps on each table row to indicate when the event occurred in the operational system, the time the event was logged by the BDW, and when the event expired (replaced by a newer row in the BDW).

Partitioning indexes allow a reasonable degree of parallelism to be achieved when executing the table handlers.

Depending on the transaction workload, the BDW can represent the latest status of data in the operational system, but in practice there is a minor time lag (typically a few seconds) in the sending of transactions over the ISC link. The design and simplicity of the table handlers are such that they can nearly always outperform the ISC link and the operational system.

In an effort to reduce DASD costs, some user tables have been defined as views over the BDW tables. Such definitions were never anticipated as part of the data design and no coding was included in the table handlers to allow for locking due to user access. For this reason, some IMS queues are stopped during the day.

Data Archive Retrieval Manager is used to control the amount of history held, as described in 7.4, “An Implementation” on page 61.

Custom-developed applications are used to drive an extract process to feed the Process Data Warehouse from the BDW.
8.1.4 Process Data Warehouse

The Process Data Warehouse (PDW) consists of 100 tables and more than 200 views (some of which point to BDW tables), all used for decision support (see Figure 24). The total size is currently approaching 40GB and will peak at about 50GB. The PDW contains about 65 million rows of data. Some of this data is passed from operational systems and other data is from a legacy system. The data from the legacy system was migrated at cutover to production to provide up to two years of history for users in certain business areas.

![Diagram of PDW Component](image)

Figure 24. CINFO PDW Component

Access to the PDW and all ancillary applications is through a flexible menu system that presents users with only those application options they are authorized to use. Menu items can be PUBLIC, controlled by RACF, or controlled by authorization of the user ID.

The menu system supports:

- Development and sharing of predefined “canned” reports
- Running and distribution of canned and tailored reports
- Data dictionary facility
- Various tools for sending the produced reports over the computer network
- User access to dialog
- User reference information
- Online application news and problem raising facility
- Access to application dialogues for user administration and operational control
- Enablers for access to any application that can run under TSO.

QMF is provided as an option within the menu system, allowing users to have direct access to views. The views and access capabilities are based on the
perceived user requirements before cutover to production and are discussed in 8.1.5, "Implementation of User Tables" on page 72.

Users are provided with a starter set of QMF reports developed centrally on their behalf. Once the QMF reports are written, they can be scheduled to run on a regular basis using the Data Distribution Manager as discussed in 8.1.4.2, "Data Distribution Manager."

Various miscellaneous applications were developed or modified to support the CINFO environment. They do not directly address interoperability in an Information Warehouse sense but allow a high degree of automation in handling interfaces from other systems and sending information to other systems.

8.1.4.1 File Receipt Manager
The File Receipt Manager runs on an hourly basis and automatically receives data sets that have been sent from external sources. Control tables are used to store information about files that have been received from other systems. Actions to be applied to the received files can be determined using values in the control tables, allowing triggering of batch processes to perform some follow-on action.

The File Receipt Manager has been implemented to receive information from country organizations for incorporation into centrally maintained tables. The data may then be queried by all users.

8.1.4.2 Data Distribution Manager
The Data Distribution Manager runs QMF reports in batch. With this application it is not necessary to update a batch reporting suite each time a new report needs to be developed. A dialog component registers requests, accepts parameters, and provides a trial reporting facility to ensure that reports work correctly before they are added to the table of available requests. The table of available requests is read by a batch process to identify which QMF procedures to run.

The Data Distribution Manager is capable of invoking any QMF procedure and is currently used to:

- Submit minor batch processes to update tables through QMF queries rather than having to develop application programs
- Run QMF reports on a user’s behalf to get a status of the business and send the output over the network to single or multiple users
- Run reports to provide flat extract files for sending to applications.

The loading of the PDW tables is controlled by the Data Enhancement Manager component, which takes data from the BDW, enriches it, and places it on the PDW for user access.

8.1.4.3 Data Enhancement Manager
The Data Enhancement Manager (DEM), a set of application components, is responsible for ensuring that the user PDW tables are updated when the corequisite BDW tables have changed. The DEM was custom built to support updating of the user tables (see Figure 25 on page 70).

One of the aims of the CINFO system, and the design rationale of the Data Enhancement Manager, is to provide nearly 24 hour availability for any user data table. The reality is that a user is prevented from accessing a table only when
that table is being updated with the latest data extracted from the BDW, not while other tables are being updated as part of a nightly batch shutdown.

The disadvantage of having nearly continuous availability is that a logged-on user could lock out batch applications from performing updates. One of the key components of DEM is the check and bounce facility, which prevents such lockouts (see 8.1.4.5, “Loading Application” on page 71).

Essentially, the PDW tables are updated by extracting the changed details from the BDW tables and applying those changes in various ways to the PDW tables.

Data Enhancement Manager is controlled by control tables. The control tables can be manipulated to limit the amount of time spent updating the PDW tables, the number of parallel tasks that are allowed to be initiated, the priority of various tables in preference to others, and much more.

Because of the number of tables and to assist process management, DEM logically groups together the BDW tables. The grouping can be by priority, expected run times, or logical business function.

Each group defined to DEM is responsible for a collection of BDW tables and the corresponding PDW tables they feed. Each group is completely independent of any other and may be scheduled whenever data update is required by its PDW tables.

![Figure 25. Example of a DEM Configuration](image-url)

The processing of groups can be staggered to maximize performance. Each group selectively stops table handlers before extracting information from the BDW and restarts when all extracts for the group are complete.
The typical run frequency for a group is daily, after a peak processing period has occurred in the operational systems.

Processing for each group is split into two stages, which are implemented as two applications under control of OPC. The first application controls the extraction from BDW tables, the second the loading of information to the PDW tables.

Information about the BDW and PDW tables are held in DEM control tables. The components interrogate these tables to provide four types of triggers that are used to initiate the execution of an extract-then-load process to update the PDW:

- At a point in time
- On user request
- On a file receipt
- On a data change in BDW (autocascade).

8.1.4.4 Extract Application

The extract application uses control tables that are updated by the table handlers (to identify whether a BDW table has changed) and other control tables to determine whether an extract is necessary. This application submits dummy jobs to OPC to initiate the extraction from the BDW tables.

This application:

1. Stops the table handlers for the group of tables being extracted
2. Determines which PDW tables in the group should be updated
3. Submits triggers to start extract applications to perform the updates

The extract applications call a program that produces an extract file of a defined format depending on whether the PDW table is to receive a delta update or a full refresh. The extract program processes an SQL SELECT statement held in a data set, which for a delta update references a host variable containing the last run time stamp. Control information is placed in the file to control run numbers and record counts.

The BDW tables hold various control fields for each row indicating, for example, whether the row was an insert, update, or delete; the time stamp used to label the change in the operational systems; and the time the row was loaded to the BDW. These fields were added for auditing and to aid the extract process.

4. Starts the table handlers for the group of tables when all extracts are complete.

8.1.4.5 Loading Application

The loading application begins on successful completion of all of the extracts for a DEM group. The loading may be performed by a DB2 load program (for refresh operations) or by a generated delta processing program that reads and applies the extract data set and handles checkpoint restart according to various parameters.

The check and bounce facility is used to identify any users who hold locks that conflict with update processing. The facility will inform users of a PDW table that the table is to be updated and that they should stop using it. Following a series of warnings, if the users are still using the table, they will be canceled from the
system. Once check and bounce is complete, all user contention should be removed and the table update is ready to run.

The entire extract and load process is illustrated in Figure 26.

![Figure 26. Extract and Load Process](image)

The delta processing program applies inserts, updates, and deletes to the PDW tables one line at a time as indicated by the extract file (see Figure 27 on page 73). Some exit routines have been built into the update routine to allow for some individual cases.

Once the update is complete, check and bounce is called again to inform users who were warned by the first run that the table they were using is available to them again.

### 8.1.5 Implementation of User Tables

This section describes various methods of maintaining history data. The implementation of the PDW tables was based on the perceived needs of the end users through a representative. This was because with such a large system it is very difficult for a user to specify precise requirements in advance without some initial use of the system. As with most informational systems, many requirements are not defined or even known in advance but evolve with experience. For this reason very little true data enhancement is done. Most data is essentially in a raw or reconciled data format, with a few instances of summarization and derivation.

The implementation of the physical tables has varied in accordance with user requirements, which can be categorized as follows:
Keep Everything: All changes applied in the BDW layer are applied to the PDW version. This change application is implemented in two ways; the method used depends on the data volumes:

- **Two-table method**
  The two-table method is used for larger volume PDW tables. It involves two tables:
  - Current table, which contains the latest picture of the business. The delta processing program reads the extract file. If it finds an insert operation, it adds the row to the Current table. If it finds an update operation, it fetches the previous version and updates it. For a deletion the row is removed.
  - Activity table, which is used to store all activity that has been applied to the Current table within the past $n$ months. All rows read from the extract file are inserted to the Activity table.

This implementation effectively means that duplicate information is kept. The Current table provides faster access for those users who are not concerned with historical data. The Activity table allows a user to go back to a period of time and show changes to a data item up to the current time without needing to perform a UNION and then sort large tables.

Views are created over the Current table to provide for a current view, and over the Activity table for a historical view.

- **Single-table method**
  The single-table method stores data as one large table. The delta processing program reads the extract file as before. If it finds an insert operation, it adds the row to the table. If it finds an update operation, it

Figure 27. Processing an Extract File
fetches the previous version of a business key, expires it through use of time
stamps, and adds the new updated row. For a deletion, the last row is
expired, and a new row is added as being deleted, but with no expiration
date set.

The advantage of the single-table method is that users do not need to do a
UNION for a complete history of a data entity. Indexing is added to assist in
the selection of current data through a view. The delta processing program
provides little provision for partitioning to group together old data.

**Keep Current:** This method is comparable to the two-table method except that
only the Current table is maintained. It may also be implemented by a DB2 load
process to improve system performance. Typically, this type of PDW is used
where the volume of activity data is either too great or is of no benefit to the
user community.

**Keep Recent:** This method of storing the data is a modification of the Keep
Everything method implemented as two tables. The only difference is that a
housekeeping process removes detailed activity once entities have been
“deleted” for a certain amount of time.

**Snapshots:** The snapshot method is used where users require information for
potentially long periods of time, but not at the activity level. Monthly unloads are
taken from the Current or Activity tables to provide latest status information.
This information is then loaded into partitioned tables using either a DB2 load or
a PL/I filtering process.

Some use is made of partition wrapping, but not to the level of specialization
discussed in Chapter 6, “Data Rolling or Wrapping” on page 43.

## 8.2 Conclusion

The CINFO set of applications is still very much in its youth, since it has been
operating for less than a year. The majority of problems have occurred in the
design of the user tables. Work is currently under way to provide more
summarization of data and to modify the table designs to join information from
many BDW tables into one PDW table. This type of evolutionary change happens
with many decision support systems in their infancy, because as users attempt
to write queries they find potential improvements that are not identified at the
design stage.

Products such as DataPropagator Relational are being considered in preference
to some table handlers for some new low volume tables.

Generally the architecture of CINFO has proved to be a good base on which to
add support for new applications and incorporate new tables. The provision of
two layers (BDW and PDW) does mean that there is a degree of data duplication.
However, this duplication is necessary to provide flexibility in the design of the
user PDW tables.

Like the Information Warehouse Architecture, CINFO has:

- **Applications**—to help decision makers analyze and present the information
  needed (the flexible menu system and Data Distribution Manager)

- **Tools**—to assist users in accessing the information, and to assist in the copy
  management processes (the Data Enhancement Manager)
• **Access enablers**—such as SQL, to provide users with an efficient way to get data (through use of QMF)

• **Organization asset data**—operational and informational data within an organization (in this case the BDW and PDW layers)

• **Infrastructure**—those components necessary to implement the system, such as the database and transaction managers (in this case DB2, IMS-DC, and OPC).

### 8.3 The Future

Since the design of the CINFO applications there have been various product announcements that would have altered their implementation. These products and features are being considered for future use. Some examples include:

**DPropR**

This product would have removed much of the need for table handlers and the ITM components. The provision of these two components took as much as 25% of the development effort.

Development opportunities are being considered to use DPropR for new applications that will be added to the CINFO applications in the near future.

**DB2 compression**

Nearly all PDW tables and the majority of the BDW tables are being evaluated to determine the benefit of using data compression. A potential 50% DASD saving is expected.

**Advanced storage techniques**

The use of optical storage devices and partition wrapping is being considered for new applications. These advanced techniques will reduce the need to purchase more 3390 hardware despite the growth in data volumes held.
Chapter 9. Database Design Considerations

This chapter addresses logical structuring of data in informational databases, focusing primarily on the need for multiple levels of detail and summary data (multiple or varying degrees of granularity). Some of the different types of summary or aggregate tables that can be used are described.

9.1 Summaries or Aggregates

It is usually desirable to maintain summaries of the raw or atomic detailed informational data for reasons pertaining both to function (as needed by the users) and performance.

If a telephone company were to store a log of all telephone calls, many millions of rows would be generated each day. The telephone company rarely needs access to the row or record pertaining to an individual call for analysis. Decisions about how to market services—perhaps based on which area codes are most frequently called from which other area codes, or the number of operator-assisted calls against unlisted numbers—rarely require access to every call made by each customer, provided the summary data is sufficient to satisfy the analysis.

The initial summary may be very “light,” that is, just a short step above the raw data. For example, one could summarize raw call data into a table that contains certain information for each hour of the day (see Table 4).

<table>
<thead>
<tr>
<th>Field Description</th>
<th>Key</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area code from which calls originated</td>
<td>Y</td>
<td>814</td>
</tr>
<tr>
<td>Prefix from which calls originated</td>
<td>Y</td>
<td>505</td>
</tr>
<tr>
<td>Year, month, day</td>
<td>Y</td>
<td>940327</td>
</tr>
<tr>
<td>Starting hour of hourly interval</td>
<td>Y</td>
<td>07:00</td>
</tr>
<tr>
<td>Area code called</td>
<td>Y</td>
<td>841</td>
</tr>
<tr>
<td>Total number of calls</td>
<td></td>
<td>10511</td>
</tr>
<tr>
<td>Average length of call (seconds)</td>
<td></td>
<td>130</td>
</tr>
<tr>
<td>Maximum length (seconds)</td>
<td></td>
<td>1941</td>
</tr>
<tr>
<td>Number of calls billed at full rate</td>
<td></td>
<td>6509</td>
</tr>
<tr>
<td>Number falling under a discount or other plan</td>
<td></td>
<td>4002</td>
</tr>
<tr>
<td>Number from home phones</td>
<td></td>
<td>4798</td>
</tr>
<tr>
<td>Number from business phones</td>
<td></td>
<td>1687</td>
</tr>
<tr>
<td>Number from phone booths</td>
<td></td>
<td>3979</td>
</tr>
<tr>
<td>Number from cellular</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td>Other data of interest</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Even the very light summarization in Table 4 could answer the vast majority of questions from managers and business professionals. One would still want the atomic detailed data available in some form, but access to large amounts of
atomic detailed data to answer a single question should be the exception rather than the rule. Additional summaries could be developed from the raw data, or "heavier" (higher level) summaries could be built from the lightly summarized table.

There are some obvious advantages to providing summarized data for analysis purposes:

- Improved resource utilization—if the summarized data is sufficient to answer all but the most obscure questions, the detailed data can be stored on cheaper media, meaning that storage costs can be reduced.
- Improved response—if the summarized data can be stored on faster media, the queries against such data will benefit. Another contributing factor to better response is that fewer rows or records need to be accessed (sometimes far fewer).
- Easier aggregation—if some level of summary were requested in the first place, and the aggregated data explicitly stored, queries or requests are easier to formulate and faster to execute.
- Easier "drill-down" access to data—"drill-down" means going from a higher level of summary to successively lower (more detailed) levels (see 9.1.2, "Drill-Down Example" on page 79). Multiple-level summaries may be designed to support specific drill-down query requirements.
- Manageability for the analyst—providing the analyst with less data helps the analyst to more easily understand the scope and meaning of the data. If a user is unsure about the data content, it could be feasible for that user to look at all of the summarized data to understand it. It may not be feasible to permit more than a few people to examine all of the unsummarized data at once, because of the large quantity of such data.

### 9.1.1 Multiple Levels of Summary

Very often it is desirable to have more than one level of summary data (see Table 5).

<table>
<thead>
<tr>
<th>Level of Summary</th>
<th>Example</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (atomic detailed)</td>
<td>Transaction history</td>
<td>Type (deposit/withdraw/payment/...), date, time, account no., amount, ATM # or teller, ...</td>
</tr>
<tr>
<td>First</td>
<td>Weekly summary</td>
<td>Account no., account type, no. deposits, total deposit amount, no. withdraws, total withdraw amount, opening balance, closing balance, year, week, ...</td>
</tr>
<tr>
<td>Second</td>
<td>Yearly analysis</td>
<td>Account no., year, no. deposits, average deposit, no. withdraws, average withdraw, ...</td>
</tr>
</tbody>
</table>

Note that the lower and higher level summaries need not contain the same type of data. The higher level summary could in fact contain information not in the lower level summary; if this were the case, the higher level summary would have to be built (at least in part) from the atomic detailed informational data rather than from the lower level summary.
9.1.2 Drill-Down Example

As stated above, a key functional requirement in many cases is the support of "drill-down" analysis (see Figure 28).

Figure 28. Example of Drill-Down Analysis

The initial focus of the analysis is on one particular product, X1234, and its sales for 1990 in the highest level of summary. The analyst then wants more detail for that year. It appears that June, July, and August were somewhat "exceptional," so the analyst wants to analyze one or possibly all three months in more detail. That is characteristic of the drill-down process: starting from one or a few pieces of data at the highest level and proceeding successively down through lower levels of summary.

In the absence of summary data (or where there is only one lightly summarized level), a "roll-up" process must occur before the drill-down data can be shown. A roll-up is in a way the opposite of a drill-down; it is aggregation of the required summary data at specified levels. If summary data is not explicitly stored, the roll-up must be done at the time each request is issued. To support drill-down, either the requester must specify, or the information system must be intelligent enough to understand, that the roll-up should be done for each level that potentially will be requested—not just for what is initially wanted, namely the highest level.
9.2 Summary Tables

This section discusses six types of summary tables:

- Simple cumulative
- Multilevel cumulative
- Rolling
- Simple direct
- Continuous
- Sample.

These may be considered basic or canonical forms; variations on or combinations of them are possible and may be useful in some situations. A simple example of each type is presented below, with a discussion of advantages and considerations.

9.2.1 Simple Cumulative Summary

A simple cumulative summary is a one-level summarization of the raw or atomic data (see Figure 29).

---

Figure 29. Simple Cumulative Summary

The significant work of Mr. W. H. Inmon in the development of this terminology and classification is gratefully acknowledged. Mr. Inmon published several trade and technical press articles regarding forms of informational data (and differences between informational and operational data) in the late 1980s and early 1990s. Mr. Inmon’s book, *Building the Data Warehouse* (QED Information Sciences, Inc., 1992, ISBN 0-89435-404-3) includes (among other things) extensive descriptions and examples of different types of informational (or data warehouse) databases. The project coordinator for this redbook worked with Mr. Inmon in late 1992 on a series of IBM seminars on Information Warehouse concepts and products, and the insights gained from that work have contributed significantly to this chapter.
The example in Figure 29 might apply to a financial institution. It is very similar to the first (weekly) summary in Table 5 on page 78. The summarization criteria are by account, by day. Each account may have several transactions per day, which are summarized into just one record for each day.

Simple cumulative summaries have the advantage that they are very easy to maintain. Any summary can significantly reduce I/Os (or row accesses) for queries satisfiable with the summary data. For example, suppose the summary in Figure 29 on page 80 covered a four-year period, and there were 10,000 transactions over that time (not unusual for a commercial customer). A query that asked for the average deposit over the entire period could be done with 1,461 row accesses (the number of days in four years) rather than 10,000.

The simple cumulative summary aids summarization at higher levels. If one wanted a summary report or to generate a higher level summary table (for instance, averages by week or month) either could be generated from the daily summary data rather than the raw transaction history. The simple cumulative summary does not directly support drill-down analysis, except from (in this situation) the daily data to the unsummarized data (a trivial case).

The simple cumulative summary cannot be used to answer a question about a specific event (for example, “Did Joe Jones deposit $98.17 on March 17?”). This fact is true of all of the summaries discussed in this section.

**9.2.2 Multilevel Cumulative Summary**

The multilevel cumulative summary (Figure 30) is a variation of and starts with a simple cumulative summary.

![Figure 30. Multilevel Cumulative Summary](image)
The first two levels in the illustration—atomic (unsummarized) data and the daily summary—could be the same as those in Figure 29 on page 80. Monthly and yearly summary data have been added. At each level, data in the previous level could be omitted. Data could be added, but that would require generation of data from a summary that is two or more levels removed (more detailed), or even from the unsummarized detail records.

Obviously some additional maintenance is required compared to the simple cumulative summary. But the advantages can be significant. I/Os, or row accesses, may be significantly reduced for queries satisfiable with the summary, compared to operating on the unsummarized data or even the lowest level summary data. A determination of average monthly balance over the four-year period could be done with 48 accesses rather than 10,000; yearly information can be accessed with 4 rather than 10,000 accesses.

The roll-up, or summarization to increasing degrees, is inherently available to users, as it is done by the refresh process, and the results are explicitly stored in the tables for that level. Even better, the drill-down process is directly and easily supported at all available levels.

As with any summary, the summarized data cannot be used to answer questions about a specific event. However, by using the summarized data, a user who needs access to specific raw or unsummarized data may be aided in specifying what is required. Put another way, the summary may help the user determine what to ask for from the detailed data.

The different levels of a multilevel cumulative summary may be summarized or aggregated by some criteria other than time period. For example, the first level may be summarized by time period (say, daily), but other levels may be summarized and aggregated by organizational unit (department, location, division, line of business, entire corporation).

### 9.2.3 Rolling Summary

A rolling summary is a special case of a multilevel cumulative summary. If we call the unsummarized data "level zero," the first summary "level 1," the next "level 2," and so on, a rolling summary is a multilevel summary where data is stored at level N - 1 for only the most recent or selected periods or instances that are stored at level N, the next more summarized level. Figure 31 on page 83 illustrates the concept.
The rolling summary entails some additional complexity of maintenance compared to the previous cases, in that deletion of data at more detailed levels is dependent on several time periods (or organizational levels, or whatever the aggregation criterion is).

However, there can be a significant reduction in I/Os for queries satisfiable with the summary. In addition, there may be a significant space savings compared with the full multilevel cumulative summary and even compared with the simple cumulative summary. For example, we store 31 to 62 days (to allow for the "next" month, which is not yet complete), 12 to 24 months, and four or five years, for a total of between 47 and 91 summary rows (over and above the 10,000 unsummarized rows). This compares to 1,461 rows for the (daily) simple cumulative summary and 1,513 rows for the full multilevel cumulative summary.

Summarization at multiple levels is inherent and done by the refresh process that maintains the summary data. Data rows to support drill-down exist at each level, but only for the most recent or selected data. The level of detail is reduced (granularity is higher) for all except the most recent or selected data.

A rolling summary may be the ideal solution where requests for data at more detailed levels are almost always for the more recent data. The functional capabilities of the full multilevel cumulative summary, such as inherent roll-up and drill-down support, are provided for such requests at a fraction of the data storage costs of the full multilevel summary. More detailed summaries for less recent data may still be derived as needed from the unsummarized data at some cost in performance. If these requests are comparatively infrequent, that cost may be acceptable.
Again, the summary cannot be used to answer a question about a specific event.

**9.2.4 Simple Direct Summary**

At first glance, the simple direct summary does not look like a summary at all, and often it does not contain aggregated data such as sums, averages, and counts. Its main use is for status tracking over time. Figure 32 shows how the simple direct summary table on the bottom, labeled “Informational Data,” is derived from operational master data and transactions.

![Simple Direct Table Diagram]

Let’s assume that we want to track the address changes of customers. (Actually, in a financial institution, there are many interesting things we may want to track in this manner, such as credit rating.) Different types of transactions occur, most of which do not affect the customer’s address. But if an address change (or account opening or closing) occurs in a given month, a row is written to the simple direct table for that month. The customer master data would also be updated. The simple direct row could be written at the time of the transaction or constructed after the fact from a transaction history.

Simple direct summary tables could be used for reporting on history (changes over time). They could be used for existence checks or simple data retrieval, as

---

7 The representation of the month (or time period) may be explicit, or implicit by the use of separate tables for each month. If “continuous summary” data is also being kept, as described in 9.2.5, “Continuous Summary” on page 85, possibly only one or two months of simple direct data would be needed, and one or two small tables could be used, named (as in the example) for the month. The simple direct data can be discarded, once reflected into the continuous summary. But if a continuous summary does not exist, the month should be explicitly stored in the simple direct data.
an alternative to accessing the operational customer master data, thus reducing interference with operational processing. Maintenance is very easy, and the simple direct data is normally never updated, except to correct an error.

9.2.5 Continuous Summary

A continuous summary is an aggregation of data from simple direct summaries. Its main use, like the simple direct summary, is for status tracking over time. As Figure 33 shows, it can be developed from two or more sets of simple direct records. The tables labeled “January Customers” and “February Customers” are combined to develop the “Customer History” table that reflects both months.

<table>
<thead>
<tr>
<th>JANUARY CUSTOMERS</th>
<th>FEBRUARY CUSTOMERS</th>
<th>(SIMPLE DIRECT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZZI, JEN</td>
<td>AKLEY, VI</td>
<td>363 COURT</td>
</tr>
<tr>
<td>BROWN, BOB</td>
<td>AZZI, JEN</td>
<td>801 LASUEN</td>
</tr>
<tr>
<td>CHOI, LE</td>
<td>BROWN, BOB</td>
<td>616 DOLORES</td>
</tr>
<tr>
<td>DANA, DAN</td>
<td>CHOI, LE</td>
<td>501 YEWOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128 TORREY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 ALVARADO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CUSTOMER HISTORY (ADDRESSES)</th>
<th>(CONTINUOUS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKLEY, VI</td>
<td>363 COURT</td>
</tr>
<tr>
<td>AZZI, JEN</td>
<td>801 LASUEN</td>
</tr>
<tr>
<td>BROWN, BOB</td>
<td>616 DOLORES</td>
</tr>
<tr>
<td>CHOI, LE</td>
<td>501 YEWOO</td>
</tr>
<tr>
<td>CHOI, LE</td>
<td>128 TORREY</td>
</tr>
<tr>
<td>DANA, DAN</td>
<td>12 WATSON</td>
</tr>
<tr>
<td>EVANS, JANET</td>
<td>6 ALVARADO</td>
</tr>
</tbody>
</table>

Figure 33. Continuous Summary Table Derived from Two Simple Direct Tables

In this very simple example, our financial institution opened in January and attracted (only) four customers who opened accounts. In the month of February, customers Azzi, Brown, and Choy came in or did some type of business. Ms. Azzi and Mr. Brown did not report a change of address, so the system infers that there has been no change. Ms. Choy changed addresses. So, in the continuous summary for her, there will be two records, one for January and one for February through the present day. (For this example it is assumed that the present day is just after February—say, 00:00:00 on March 1.)

The two new customers, Ms. Akley and Ms. Evans, were not customers in January and (absent other data) one cannot infer anything about their addresses. So the continuous summary records indicate “February through present.” Notice that Dan Dana never was heard from in February; therefore, nothing is inferred about his address for that month. Absent any other data, the continuous summary can only reflect that he lived at 12 Watson in January.
A continuous summary may also be developed from a previous continuous summary, augmented by additional simple direct records. Consider the example in Figure 34 on page 86.

<table>
<thead>
<tr>
<th>HISTORY THRU FEB. (CONTINUOUS)</th>
<th>MARCH CUSTOMERS (SIMPLE DIRECT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZZI, VI</td>
<td>363 COURT FEB-PRES</td>
</tr>
<tr>
<td>AZZI, JEN</td>
<td>801 LASUEN JAN-PRES</td>
</tr>
<tr>
<td>BROWN, BOB</td>
<td>616 DOLORES JAN-PRES</td>
</tr>
<tr>
<td>CHOY, LE</td>
<td>501 YEEOO JAN-JAN</td>
</tr>
<tr>
<td>CHOY, LE</td>
<td>128 TORREY FEB-PRES</td>
</tr>
<tr>
<td>EVANS, JANET</td>
<td>6 ALVARADO FEB-PRES</td>
</tr>
</tbody>
</table>

**NEW CUSTOMER HISTORY**

| AKLEY, VI | 363 COURT FEB-PRES |
| AZZI, JEN | 801 LASUEN JAN-PRES |
| BROWN, BOB | 616 DOLORES JAN-FEB |
| BROWN, BOB | 888 OCEAN MAR-PRES |
| CHOY, LE | 501 YEEOO JAN-JAN |
| CHOY, LE | 128 TORREY FEB-PRES |
| DANA, DAN | 12 WATSON JAN-JAN |
| EVANS, JANET | 6 ALVARADO FEB-FEB |

Figure 34. Continuous Summary Table Derived from Previous Summary and Additional Simple Direct Data

The table in the upper left is the continuous summary, reflecting January through February, from the previous example. Additional simple direct records now exist for the month of March. No new customers opened accounts, and of the four that transacted business, one (Mr. Brown) reported an address change. The "New Customer History" continuous summary reflects the additional knowledge of customer addresses. Note that there is no additional history for Dana after January, and none for Evans after February, because they did no business in March.8

Continuous summary data can be used to recreate status at a point in time in addition to being very useful for status tracking. Consider the "New Customer History." Is it possible to determine where everyone who was represented in the table lived in February? Looking solely at the table at the bottom of Figure 34, we want those names and addresses that we know to have been valid in February. We see that the following records qualify:

- Akley at Court
- Azzi at Lasuen

---

8 No claim is made that this is necessarily a valid business process. More likely, the financial institution would infer that the address remained the same, unless a statement or other correspondence were returned by the postal service due to "not at this address." The example is simply to illustrate the interrelationship of the data.
• Brown at Dolores (and not at Ocean because he did not move there until March)
• Choy at Torrey (and not at Yee Woo)
• Evans at Alvarado.

These are the exact five rows that comprise the February simple direct summary in Figure 33 on page 85.

Maintenance procedures should take into consideration the size of the simple direct data that is to be used to update the continuous summary (or create a new one) and the size of the existing continuous summary if one exists. There will be insertions to reflect activity in the current month (new customers or address changes in this case) and updates to change the "to" date, from "present" to the value of the previous period ("February" in this case) for those subject occurrences (customers, here) that are not represented in the current simple direct data. The latter, however, can be done with a set-level update (using a ‘NOT IN’ clause, for example).

A design alternative is to always use an explicit period (month, here) instead of some indicator that means “present.” The “present” indicator works very well if the majority of the subject occurrences in the continuous summary do in fact have a corresponding simple direct record, representing activity in a given period. This would probably be the case here—most bank customers have at least one transaction in a given month, and address changes would affect a small percentage of the total. If the number of simple direct occurrences in a given period were small compared to the total number of continuous summary records, explicitly storing both “from” and “to” dates in all cases may be preferable to use of the “present” indicator.

Another alternative is to use one date—“latest reported date”—instead of two. This has the advantage that there is one less column to store and maintain. However, to do interval processing, such as determining how long Bob Brown lived at Dolores, requires access to two continuous summary rows, rather than the one that is required if both “from” and “to” dates are stored.

9.2.6 Sample

A sample is not really a summary in that the data is not usually aggregated. However, it can be very useful for certain queries and for testing. Figure 35 shows a sample that is derived from the transactions of the mythical financial institution that has been used throughout this section.

<table>
<thead>
<tr>
<th>TRANSACTIONS:</th>
<th>WD</th>
<th>PMT</th>
<th>DEP</th>
<th>PMT</th>
<th>DEP</th>
<th>DEP</th>
<th>WD ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE:</td>
<td>WD</td>
<td>DEP</td>
<td>PMT</td>
<td>DEP</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 35. Sample of Unsummarized Transaction History Data
Properly constructed samples can be very useful in providing reasonably accurate answers to statistical questions. For example, a business analyst may say “I would like to know the percentage of our customers who have written an overdraft over the last three years, and I would be satisfied if the answer were within plus or minus 1% with 95% confidence.” The size of the sample that could satisfy this request might be an order of magnitude smaller than the original table. If this were a frequent type of request, the cost of maintaining the sample might be much more than offset by the performance gain and resource savings afforded by executing the query against the sample.

Testing of procedures (such as queries, reports, spreadsheets, and charts) that support decision support processing and executive information systems may also be done against sample data. Such testing is analogous to the common practice of testing operational applications against small subsets of the operational data.

If many of the information requests are satisfiable with sample data, scarce and expensive online storage could be saved by moving part or all of the unsummarized data to less expensive media.

The sample need not be random. One may wish to include or exclude subject occurrences with certain attributes (for example, the analyst wants the above percentage to exclude commercial accounts). One may want to build the sample for a particular region or part of the organization and send that sample data to users located there who are interested in analyzing just that part of the entire collection of data. (Such users might also or instead be interested in working with a full subset containing data for their region or organization.)

The sample in general cannot be used to answer a question about a specific event. Additionally, the sample is not normally useful for further summarization—you cannot accurately summarize from the sample data.

In the design of sample tables, the most important consideration is the statistical confidence level desired by the analysts who will use it. That, and the size of the tables from which the sample will be drawn, will determine the sample size required. Other important considerations are the refresh requirements—how large should the sample be, how often should it be refreshed, and how bias should be eliminated as much as possible for a random sample.

9.3 Concluding Remarks

This chapter has presented the basic structures of some types of summary tables. These basic forms are often varied or combined. For example, it might be useful to have a multilevel cumulative summary by organization unit (division, location, department) that in turn is “rolling-summarized” by time period. Multiple levels of summary also provide the opportunity to implement a data placement strategy according to the relative frequency of access. This is discussed in Chapter 3, “Data Storage Selection Based on Access Requirements” on page 17. The ideal situation is where the higher levels of the summary—which normally have fewer records, sometimes far fewer—are the most frequently accessed and placed on the optimum storage device.

It should be anticipated that not all user requirements can be known in advance and that changes to both the unsummarized and the summary data will be required over time. But a rich supply of historical data with summaries to aid
usability and performance can provide an information asset of great value for
management decisions. The capability of Data Archive Retrieval Manager to
archive data and selectively retrieve data allows detailed historical data to
support the summarization process while managing the cost to store the data.

DataRefresher, DataPropagator NonRelational, or DataPropagator Relational
(DPropR) may be used to produce point-in-time copy (target) tables.
Point-in-time tables are useful for extraction from operational to informational
data stores. DPropR may also be used to produce other types of informational
copy data, including:

- Base aggregate tables (of which simple and multilevel cumulative
  summaries are examples)
- Change aggregate tables (of which continuous summaries are an example)
- Subsets, through the use of SQL predicates in the DPropR subscription
definition
- Derived data, through the use of added column definitions and grouping
  specifications in the subscription definition.

The use of these and other constructs, alone or in combination, is consistent
with the types of data described in Information Warehouse Architecture I
(GC26-3244). The types of data described here may be classified according to
the terminology used in the Information Warehouse Architecture, as follows:

- Operational data: real time data
- Atomic detailed informational data: reconciled data (but could have
  additional derived attributes)
- Simple cumulative summary: derived data (one form)
- Multilevel cumulative and rolling summaries: derived data (other forms)
- Simple direct summary: change data (basic)
- Continuous summary: change data (accumulated)
- Sample: reconciled data (but could have additional derived attributes). If not
  sampled from real time data but from reconciled data (atomic detailed
  informational), could be considered a form of derived data.

Atomic detailed (unsummarized) informational data is extracted from data
representing operational entities and events and is then further transformed,
derived, and aggregated. These additional types of data often are of the types
described in this chapter. Some considerations relevant to the transformation
from operational to informational data are the subject of Chapter 10, "Table
Design Considerations" on page 91.
Chapter 10. Table Design Considerations

This chapter addresses the design of individual informational tables, in particular, how the design is derived, but also differs, from the design of related operational data. The primary focus is on the design of the unsummarized informational rows. The chapter also addresses some issues related to summarization and indexing based on the preferred access to the data.

10.1 Design Steps for Informational Tables

The derivation of basic, initial informational data from operational data is basically a four-step process:

1. Remove purely operational data (described on page 91)
2. Add one or more elements of time (described on page 93)
3. Add derived data (described on page 94)
4. Add artifacts of relationships (described on page 96)

This section describes each of these steps and presents a very simple example. Subsequent sections of the chapter suggest additional enhancements to informational data to aid decision support analysis.

10.1.1 Step 1: Remove Purely Operational Data

“Purely operational data” is data that managers or business analysts are very unlikely to ever need or even request. The data may be very relevant for operational applications but is considered to be of no interest, at any time, even at the unsummarized level, for informational users or applications.

In our example, shown in Figure 36 on page 92, there are, on the operational side, two tables (or files) that maintain data on orders and the line items (individual parts) that comprise an order. The order master row (or record) is the parent and the line item row is the dependent in logical database design terms, as there is a one-to-many relationship—one order may have many line items. In fact, on the operational side, it may be very desirable to define a referential integrity constraint on the relationship.
The database designer has determined that the user requirements boil down to two: (1) tracking of supplier performance, defined as the interval from the time orders were placed until they were completely received and (2) tracking of which suppliers received the most orders in terms of dollar volume. Both of these analyses could be very useful in improving the business, providing, for example:

- Exception reminders to suppliers who were habitually tardy and thus affecting cycle times in our organization
- Reports of total purchase dollar volumes to support negotiations for improved terms and conditions from our most frequently used suppliers.

The designer decides that, in the context of the known user requirements, the following data will not be represented in the informational tables:

- From the order master table, DATE_REQUIRED, which specifies to the supplier the desired date of shipment or receipt of the order
- From the line item table:
  - DELIV_NAME, the name of the person (in the Receiving department) to whom the items should be delivered and who should sign for them
  - DELIV_PHONE, the above person’s phone number.

This decision is obviously a judgment call. One might consider whether comparison of DATE_REQUIRED with the date on which the items were received is a better index of supplier performance. It really depends on what the data element means and how it is used in the business processes. One should expect that some data will probably have to be added as user requirements change.
The result is that on the informational side, nothing is yet ruled out except these three columns (or elements). The remaining steps address what is to be done with the other columns. However, one can be quite sure that the order and supplier identifications are needed in the informational table and other columns as well.

10.1.2 Step 2: Add One or More Elements of Time

Step 2 in initial informational table design is the addition or incorporation of one or more columns that indicate a time or time interval. In our example (Figure 37) we see that two additional columns are defined in the informational table: ORDER-DATE and FINAL_DELIV.

![Figure 37. Addition of Data Elements to Provide Time Basis](image)

ORDER_DATE can come directly from the operational Order Master record. It may be desirable to change the format of the data. For example, the operational ORDER_DATE field could be stored in character format or even as an integer, particularly if the operational data is in an older nonrelational system that does not provide explicit support for DATE and TIME data types. But otherwise ORDER_DATE in the informational ORDER_HIST table has the same content and semantics as ORDER_DATE in the operational SUPPLIER_ORDER file.

The FINAL_DELIV column is the date on which the last line item for this particular order was received. So it is an aggregation of all of the occurrences of DELIV_DATE for those records in the operational LINE_ITEM file for which ORDERN (containing the order number) matches ORDERN in the SUPPLIER_ORDER file. If the operational data is in a relational table, this aggregation is easily accomplished with one set-level SQL statement:
SELECT MAX(DELIV_DATE) FROM LINE_ITEM WHERE ORDERN = :HVORDERN;

where the host variable in this case contains the order number for the particular row in ORDER_HIST that is being built. Again one could use a DATE data type for FINAL_DELIV regardless of the format of the operational data.

10.1.3 Step 3: Add Derived Data

The addition of derived data is one of the most important steps in terms of impact on usability of the informational database table. In the example (Figure 38) we need at least a column to provide the total dollar amount of the order.

```
OPERATIONAL:
SUPPLIER_ORDER
  | ORDERN | SUPPLIER_ID | ORDER_DATE | ...
LINE_ITEM
  | ORDERN | PARTN | AMT | DELIV_DATE | ...

INFORMATIONAL:
ORDER_HIST
  | ORDERN | SUPPLIER_ID | ORDER_DATE | FINAL_DELIV | TOTAL_ORDER | ...
```

Figure 38. Addition of Derived Data to the Informational Table

The column is called TOTAL_ORDER. The derivation is from the AMT fields for all line items for a particular order. We assume that the operational LINE_ITEM records have multiplied the quantity ordered (which is undoubtedly stored elsewhere in LINE_ITEM) by the unit price from the item master record. If this is not the case, the program that creates the informational row would have to do this calculation in addition to aggregating the resulting amounts. In any event, once again, if the operational data is already in a relational database, the aggregation is quite simple:

```
SELECT SUM(AMT) FROM LINE_ITEM WHERE ORDERN = :HVORDERN;
```

As stated above, the choice of derived data is very important in terms of making the informational data as usable as possible. This is one situation where it is probably better to “overdesign” the informational table, providing more derived
data than is immediately obvious from the requirements. Table designers should try to anticipate the future needs of the managers and analysts and reflect those needs in the design. Good ongoing communications between the table designer and the prospective users may lead to consideration of additional “soft” requirements: data about which the user says, “I think we might need that sometime, but I’m not sure.”

Provision of a rich variety of derived or aggregated data is one way of minimizing the need for access to the unsummarized, detailed data—which should be a high-priority goal of informational database design. The larger the amount of informational data, the higher should be this priority.

At this point it is also good to consider derived data for which there may never be a requirement from an end-user viewpoint (for DSS or EIS processing, for example), but which may be useful for maintenance of the summary table itself.

As an example, consider a situation where the number of rows at level N - 1 used (aggregated) to derive the data for a given row at level N is variable. This can occur in two situations:

- The aggregation criterion is based on something that is inherently variable. For example, if level N - 1 represents departments and level N represents divisions in an organization, divisions can in fact have varying numbers of departments.
- Even when the ratio of records in the two levels is constant, such as when level N - 1 represents months and level N represents years, there could be variations during the time that the informational data is growing to encompass the complete desired horizon (say, five years).

Where this variability exists, it is highly desirable to store the count, in the level N row, of the number of rows at level N - 1 that correspond to the level N row. One usage of this count is in recalculating averages. In the case where the informational database is growing, one can then calculate a new average for some column in the level N - 1 rows without actually retrieving anything from level N - 1. The recalculations are done with the data in the (one) new row that is being added at level N - 1 and the existing average and count in the row at level N:

\[
\begin{align*}
\text{NEW\_COUNT} &= \text{OLD\_COUNT} + 1 \\
\text{NEW\_AVERAGE} &= \left(\frac{\text{OLD\_AVERAGE} \times \text{OLD\_COUNT} + \text{NEW\_DATA\_VALUE}}{\text{NEW\_COUNT}}\right)
\end{align*}
\]

If the OLD\_COUNT were not explicitly stored, one would have to do a SELECT COUNT(*) FROM the table at level N - 1 WHERE the period or instance matches the row at level N. This would involve retrieval of several index records at the very least; absent an index, it could involve a scan of all of the data at level N - 1.

This is but one example of explicitly storing data (the count) that is of little or no interest to users but could improve both implementation and performance of refresh and maintenance procedures.
10.1.4 Step 4: Add Artifacts of Relationships

The users notice at this point that although the SUPPLIER_ID is provided in the ORDER_HIST table, not much other information about that supplier is there. Let’s assume that there is also a file on the operational side that contains data about the suppliers, called SUPPLIER. It is also reasonable to assume a primary key—foreign key relationship between SUPPLIER.SUPPLIER_ID and SUPPLIER_ORDER.SUPPLIER_ID. See the top part of Figure 39.

Let’s assume that there is also a file on the operational side that contains data about the suppliers, called SUPPLIER. It is also reasonable to assume a primary key—foreign key relationship between SUPPLIER.SUPPLIER_ID and SUPPLIER_ORDER.SUPPLIER_ID. See the top part of Figure 39.

![Figure 39. Addition of Artifacts of Relationships to Informational Table](image)

What the users want is other data about the supplier, such as name and phone number. At this point there is a temptation to simply relate the SUPPLIER_ID in ORDER_HIST back to a particular row (record) in the operational SUPPLIER data. This would be an invalid design in almost all cases!

Consider what SUPPLIER_ID in ORDER_HIST really means. It means that at a particular period (specifically between ORDER_DATE and FINAL_DELIV) the supplier represented by SUPPLIER_ID was responsible for filling the order represented by ORDERN. It does not imply anything about the current state of that supplier. The supplier could have changed name, address, phone number; moved out of the country; been acquired by another company; or gone out of business. The problem with trying to relate informational data to operational data goes back to the fundamentals of the definitions: operational data represents what is true today; informational data represents facts that are or were true at some point in time. So it is not normally valid to attempt to relate the two types of data, and it is even less valid to attempt to define some sort of integrity constraint between them.
One very simple alternative is to explicitly store such data about the supplier in the ORDER_HIST table. This would certainly satisfy the user requirement and provide a straightforward design—everything in one table. The data would come from the operational SUPPLIER table at the time of creation of the ORDER_HIST row (when FINAL_DELIV is finally known). Presumably the operational data is correct for a supplier that just that day shipped something to the organization. The design appears to (and does) violate second-normal form, but that is not nearly as much of an issue in informational applications because of the rarity of updates to such data. The basic tradeoff is between simplicity of the queries and space requirements.

The explicitly stored data is termed “artifacts of relationships.” The analogy to archaeology is apt because just as a piece of an arch or column from ancient Rome does not represent the Rome of today, the artifacts of relationships generally represent the state of the subject (for example, supplier) as it was some time in the past. The representation of the state of the subject today and the relationships that exist today are in the operational data.

An alternative design is to implement an additional table on the informational side, to track supplier history, for example. The two informational tables in Figure 40 have at least the same information content as the one informational table ORDER_HIST in Figure 39 on page 96.

![Figure 40. Alternative: Use Two Informational Tables instead of One](image)

There is no longer a need to store data in ORDER_HIST that pertains only to suppliers. SUPPLIER_HIST has a time dimension (unlike the SUPPLIER operational file), and a new row is added each time anything changes about a supplier (such as name, address, or phone) and the TO_DATE of the previous row is the day previous to the FROM_DATE of the new row that reflects the
change. But the old data could be kept at least as long as the ORDER_HIST rows are kept. Retrieval of the supplier data for a particular order is slightly more complex in that a two-table join is needed:

```sql
SELECT O.ORDER#, O.SUPPLIER_ID, S.S_NAME, S.S_PHONE, ...
FROM ORDER_HIST O, SUPPLIER_HIST S
WHERE O.SUPPLIER_ID = S.SUPPLIER_ID
AND O.ORDER_DATE >= S.FROM_DATE
AND (O.ORDER_DATE <= S.TO_DATE OR S.TO_DATE IS NULL)
```

The above example assumes that setting TO_DATE to NULL in a particular SUPPLIER_HIST row means that that row is the most recent and shows the present state of the supplier.

There are advantages to this alternative:

- Space requirements should be reduced because second normal form is not violated; supplier data is not stored multiple times in ORDER_HIST rows.
- If SUPPLIER_HIST data is updated, only one row would be affected (again a consequence of second normal form). But as stated in the discussion of second normal form on page 97, update frequency is not as much of an issue as it is for operational data.
- If only the ORDER_HIST data is required, the retrieved rows are smaller.

Some considerations are:

- A join is required to obtain supplier data other than SUPPLIER_ID.
- If a design philosophy of prohibiting denormalization is adopted as a rigid rule, a small initial informational database design could proliferate into one involving many tables. This is contrary to the rule of thumb that for informational data and decision support systems one should start small and add information incrementally. It mostly depends on how much interrelationship exists between the initial highest priority subject areas and other related subjects. A compromise might involve a mixture of related informational tables and a few artifact columns in denormalized tables, eliminating many denormalization problems while at the same time keeping the initial design manageable.

In the example, the use of two tables is probably preferable. Two tables still represent a very simple design, and the number of rows in the SUPPLIER_HIST table is not likely to be large—hundreds or thousands, not millions or billions.

Relating the two tables is not that difficult, which brings up the point: is there any role for referential integrity? The answer is usually no for the following reasons:

- The validity of SUPPLIER_NO in the ORDER_HIST data has already been established on the operational side. The ORDER_HIST data was derived from completed operational SUPPLIER_ORDER records. The integrity checking should be done between the two operational tables: is the SUPPLIER_ORDER.SUPPLIER_ID value equal to the value of SUPPLIER.SUPPLIER_ID in a row in SUPPLIER?
- There can be multiple rows in SUPPLIER_HIST with the same value for SUPPLIER_ID. SUPPLIER_HIST.SUPPLIER_ID by itself is not a candidate primary key; one would need that column plus either FROM_DATE or TO_DATE (or both) as a unique identifier. What columns in ORDER_HIST would then comprise a candidate foreign key? One could compare these
columns in SUPPLIER_HIST with the combination of SUPPLIER_ID and ORDER_DATE in ORDER_HIST, but it is not an equality relationship; the rows are related if ORDER_DATE is between FROM_DATE and TO_DATE (or greater than FROM_DATE and TO_DATE is null).

- The business rules are different. On the operational side it could be reasonable to want ORDER records to be deleted if the corresponding SUPPLIER record is deleted. It is doubtful that on the informational side one would want deletions of ORDER_HIST rows upon deletion of a SUPPLIER_HIST row. The two informational tables may have different time horizons. Users may be more tolerant of “missing” rows in SUPPLIER_HIST for older ORDER_HIST rows that are accessed only for statistical purposes.

10.2 Summary Tables and Indexing

In some cases, it may not be obvious exactly how the summary tables can be designed to meet all, or even most, of the users’ initial requirements, let alone anticipate unstated or future requirements. Often this is because the unsummarized data itself is complex. There may be many, many columns in the unsummarized data, recording many facts about an event. The aggregation criteria may not be clear, or the needs of some users may appear to conflict with the needs of others. Often no one summary table can satisfy all needs. It may be a necessity to implement and maintain two or more very different summaries at the same time. The designer must consider whether an index over the unsummarized data (or over the lowest level of summary data) may be more advantageous than a particular summary (or level of summary).

10.2.1 Summaries: Evaluating Alternatives

Consider the example shown in Figure 41 on page 100. This is a typical history table that might be found in the retail industry (although such tables in reality often contain far more data than the 18 columns in this design).
The designer has chosen the combination of these columns as the unique identifier (primary key):

- Credit card type (MasterCard, Visa, Discover, or store card)
- Credit card number
- Date and time.

Note that for customers who pay by check or cash, other informational tables would be used. For example, instead of CCTYPE and CCARD, one would store the Bank ID number and the checking account number. The credit transaction authorization code (CCAUTH) could be replaced by the check number or the check authorization number.

Note also that there is more than one candidate key. CCAUTH may in fact be unique, depending on the credit card company’s authorization mechanism, although it is not very useful by itself. REGION, STORE, DEPT, REGISTER, DATE and TIME would also be unique (assuming it is impossible for one register to ring up two sales simultaneously). Other combinations may be possible.

The designer has split the transaction history data into two related informational tables to adhere to first normal form while allowing for multiple items on a single purchase. From now on, we use the term “CUSTOMER ID” as a shorthand substitute for CCTYPE concatenated with CCARD; this shorthand is not necessarily part of the database design. The detail table has “CUSTOMER ID,” DATE and TIME as a foreign key, and these columns concatenated to Stock Keeping Unit Number (SKU) as its primary key.
This is a case where, if the retail chain is larger than just a few store locations, there will undoubtedly be a need for summary tables, regardless of the database technology used. The challenge is to design summary tables that meet the users’ needs but are not so large as to approximate the size of the unsummarized data, nor so many in number that both the maintenance and end-user environments become difficult and overly costly.

The tables in Figure 42 basically summarize, by region and store, the activity for each product on a given day.

<table>
<thead>
<tr>
<th>INDEXES</th>
<th>REGION</th>
<th>STORE</th>
<th>PRODLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>REGION</td>
<td>STORE</td>
<td>DATE</td>
</tr>
<tr>
<td>DETAIL</td>
<td>&quot;CUSTOMER ID&quot;</td>
<td>...</td>
<td>SALESTAX</td>
</tr>
<tr>
<td></td>
<td>&quot;CUSTOMER ID&quot;</td>
<td>...</td>
<td>QTY</td>
</tr>
</tbody>
</table>

*Figure 42. Summary Table and Indexes. Addresses requirements for store volume or product volume tracking.*

The PRODLINE column is derived from a table (which could be operational or informational) that classifies the many thousands of SKUs sold by the chain into product lines (for example, tires, cologne, kitchen hand utensils, shoes). One could store this column in the unsummarized rows as they are inserted, but the designer here chose to put it in each summary row as it is created.

There are indexes that assist in analysis of volumes by store and by product line or product within product line. This summary could be very useful to answer such questions as:

- What was the effect of the promotion that ran from May 15 through May 23 in all stores in a particular region, on tire sales?
- What was the effect of the same promotion on sales of all products in a particular store?
- How do stores in one region compare to stores in another for a particular product or product line?
Suppose, however, that other departments have completely different requirements. Assume that the retail chain has a department (and manager) responsible for telemarketing, and another department (and manager) for catalog distribution—and the chain has several specialized catalogs. Now the subject of this type of analysis is the customer, and a summary such as that shown in Figure 43 may be useful.

Figure 43. Alternative Summary Table and Index. Addresses requirements for analysis of customer activity.

This type of summary could be used for targeting customers for specific mailings (such as a catalog for products carried by a specific department, for example, housewares), focused telemarketing based on purchase history, and measuring overall promotional effectiveness. This summary does not show customer data other than the CUSTOMER ID, but if such data were available from operational or other informational sources, it could be incorporated into this informational summary. Such information as address would be necessary for promotions targeted by region. Demographic data would be useful for targeted mailings, such as a toy catalog sent only to customers known to have young children.

The challenge is that a summary designed to meet the needs of one group of users may be unsuitable for others. The two summaries above appear to have little in common. Each is targeted to specific needs, and neither is very useful to users for whom it was not targeted. So the summary oriented toward stores and products has no information about specific customers, and the customer summary has only very gross information about products and none about specific stores.

One could always implement both summaries and have two sets of procedures for generating and maintaining the summaries. This is often the best solution if maintenance time and DASD space are not a problem, and performance is satisfactory. However, it may be advisable to consider a design that meets most of the needs with a single summary, perhaps with reduced level of detail, if:

- The summaries grow quite large.
• The number of rows searched by users of the summary table is not an order of magnitude less than an equivalent search of, and aggregation from, the unsummarized data.
• The number of such specialized summaries grows to the point where it becomes difficult for end users to select which one is most useful, and maintenance procedures become very time consuming.
• The summaries are so specialized that many new users must access the unsummarized data to get the information they require.

A possible design that is a compromise between the needs of the users analyzing store and product results and users requiring analyses oriented toward customers is shown in Figure 44.

![Figure 44. Summary to Accommodate Different Types of Users (but with reduced detail)](image)

The design of the summary in Figure 44 combines the essential features of the previous two examples of summary tables into one. There is reduced level of detail (higher granularity) in that:
• Aggregations are at the product line level, instead of SKU as in the summary shown in Figure 42 on page 101.
• The data are aggregated over weekly rather than daily time periods.

This design allows for analyses by store within region, product, and customer. One cannot analyze as precisely as in the previous examples events or phenomena that occurred on specific days, as the summarization is by week rather than day. The purpose of that design choice was to reduce the number of rows in the summary. However, in cases other than this example oriented toward retail sales, it may be quite feasible to summarize very lightly without changing the time basis as was done here. It depends on whether changing
some other aggregation criterion would reduce significantly the ratio of summarized rows to unsummarized rows. In the example, the designer determined that a summary based on customer/prodline/day would produce a table of nearly as many rows as a table based on customer/SKU/day, which equates to the unsummarized data. Therefore, it was decided that the summary should be on a weekly rather than daily basis.

10.2.2 Indexes for Access

With informational tables, indexes are often needed to improve performance, as with operational data. In fact, indexes may be more important for informational tables because some can be very large. Indexes are useful for the following capabilities:

- **Uniqueness enforcement**, as for operational data.
- **Clustering**, to aid retrieval of sets of data. This capability could be more important for informational data. For example, a telemarketer might want to see the entire purchase history of a customer (assuming the rows are clustered by CUSTOMER ID) in order to make an informed call. This history display is analogous to a reporting function on operational data, but the accesses to informational data are usually less predictable.
- **Retrieval in different sequences**. Especially if summary requirements are consolidated as in Figure 44 on page 103, additional indexes may be needed on the summary tables to satisfy multiple user requirements with acceptable performance.
- **Aid for the summarization process itself**. As the process of building a summary row involves aggregation from a related set of unsummarized rows, an index would aid this process in the same way as described above under clustering.
- **Possible use in lieu of a summary table**. One might want to avoid altogether the creation and maintenance of a summary table if:
  - The aggregation criterion being considered is such that the summary would not be that much smaller than the unsummarized table.
  - The frequency of requests is low to medium compared with the frequency of requests based on other criteria.
  - The total amount of data satisfying such requests represents a comparatively small amount of the unsummarized data.

In such cases, one should consider indexing the unsummarized data rather than implementing a summary table. For example, suppose that in the example shown in Figure 44 on page 103, requests by DEPT are relatively infrequent, and, when one is made, it is for one or a few departments rather than for a “dump” report on all departments (which would require a full table scan). In this case, it might be better to omit DEPT as an aggregation criterion for the summary table and provide an index by DEPT over the unsummarized table.

It is recommended that one be somewhat more liberal in the creation of indexes over informational data than one would be in the creation of indexes over

---

10 The assumption is that departments are uniformly identified (numbered) across all outlets in the retail chain, so that analyses across regions and store locations by department are possible and meaningful. Also, the assumption is that all or most store locations have all or most departments.
operational data. Many of the issues of index maintenance overhead in an environment of heavy update and delete activity are much less applicable to informational tables.

10.2.3 Other Considerations

Consideration should also be given to creation of “creative” subsets of the unsummarized data, and in some cases of the summary tables as well. Subsets are similar to samples as discussed in 9.2.6, “Sample” on page 87, but play a somewhat different role. A subset might consist of the “top ten” or “most active 100” subjects of a particular type.11 These subsets are unlike samples, because they represent every subject occurrence that qualifies for the selection criterion used. Therefore, aggregations may be done and meaningful conclusions may be drawn from such aggregations, for example: “Our top 100 accounts had an average credit limit of $9218 and a total of only two late payments over the last year.”

Again it is recommended that one be liberal in designing such adjunct tables and the indexes on the underlying detailed data to support them.

10.3 Conclusion

This chapter and Chapter 9, “Database Design Considerations” on page 77 have presented some suggestions for logical design of informational databases and the tables that comprise them, along with some simple examples. The following is a summary of what has been suggested in the form of a series of general design steps:

1. Select the subjects to be represented by data in the informational tables based on initial business and information technology requirements and priorities.
2. Determine the best sources of data from the operational files.
3. Design the columns of the initial, unsummarized informational tables according to the four-step process described in 10.1, “Design Steps for Informational Tables” on page 91, considering user requirements.
4. Design the first level of summary tables over the unsummarized informational data in such a way that most initial user requirements can be satisfied with requests to summary tables rather than the unsummarized data. This approach will usually lead to summaries of a high level of detail (low granularity, light aggregation).
5. If the number of summaries is high, and/or one or more summaries is less than half of an order of magnitude smaller than the unsummarized table (for example, if the summary contains greater than one-fourth of the number of rows in the unsummarized table), consider consolidating two or more summaries into one, as described in 10.2.1, “Summaries: Evaluating Alternatives” on page 99.
6. If the result is again a summary that is too large, consider changing an aggregation criterion to reduce it. For example, summarize by day instead of hour, or week instead of day. This analysis necessitates balancing the

11 Apologies to David Letterman!
needs of users and a design that can provide acceptable performance at reasonable cost.

The absolute ideal result of steps 4, 5, and 6 would be a single simple cumulative summary that is of very low granularity (very little loss of detail compared to the unsummarized data), satisfying the vast majority of user requirements (which should be a consequence of little loss of detail), and providing a basis for all or nearly all future requirements for summaries. If DASD space permits, such a summary should be implemented for testing and performance comparison if for no other reason.

7. Design additional levels of cumulative or rolling summary data, and simple direct, continuous, and sample tables as indicated by evolving user requirements, as described in 9.2, “Summary Tables” on page 80.

8. Use indexes (liberally) where appropriate as suggested in 10.2.2, “Indexes for Access” on page 104.

9. Consider other subset and adjunct tables as suggested in 10.2.3, “Other Considerations” on page 105.
Appendix A. An Example of Partition Wrapping

This example expands on the material presented in 6.2.2, “Example: Electronic Equipment Manufacturer” on page 47. It assumes that it is March 1994 and the housekeeping activities are just about to be run for the ORDER_HIST_FULL table. Three months of orders are to be put on the table for the months of January, February, and March.

The data definition statements below are for the DB2 table that holds the order history data using partition wrapping techniques. The DDL creates a tablespace, table, and associated partitioning index. The PRIQTY, SECQTY, and column details are all sample values. In reality the volumes would be much larger, and there would be far more columns in the table.

CREATE TABLESPACE REDBOOK2
  IN QMFDCINF
  USING STOGROUP JBRG0001
  PRIQTY 1
  SECQTY 1
  ERASE NO
  FREEPAGE 15
  PCTFREE 0
  NUMPARTS 12 ( 
    PART 1
      USING STOGROUP JBRG0001
      PRIQTY 10
      SECQTY 10
      ERASE NO
      FREEPAGE 15
      PCTFREE 0 ,
    PART 2
      USING STOGROUP JBRG0001
      PRIQTY 10
      SECQTY 10
      ERASE NO
      FREEPAGE 15
      PCTFREE 0 ,
    PART 3
      USING STOGROUP JBRG0001
      PRIQTY 10
      SECQTY 10
      ERASE NO
      FREEPAGE 15
      PCTFREE 0 ,
    PART 4
      USING STOGROUP JBRG0001
      PRIQTY 10
      SECQTY 10
      ERASE NO
      FREEPAGE 15
      PCTFREE 0 ,
PART 5
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 0,
PART 6
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 0,
PART 7
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 0,
PART 8
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 0,
PART 9
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 0,
PART 10
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 0,
PART 11
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 0,
PART 12
USING STOGROUP JBRG0001
PRIQTY 10
SECQTY 10
ERASE NO
FREEPAGE 15
PCTFREE 0 )
BUFFERPOOL BPO
LOCKSIZE ANY
CLOSE NO
;

CREATE TABLE ORDER_HIST_FULL
(
  PARTNO CHARACTER(3) NOT NULL,
  ORDER_NUM CHARACTER(12) NOT NULL,
  CNTL_STMP TIMESTAMP NOT NULL
) IN QMFDCLNF.REDBOOK2
AUDIT NONE
;
COMMENT ON TABLE ORDER_HIST_FULL IS
'SAMPLE ORDER TABLE FOR HISTORY'
;
CREATE INDEX ORDER1_HIST_F
ON ORDER_HIST_FULL
(
  PARTNO ASC,
  ORDER_NUM ASC
)
USING STOGROUP JBRG0001
PRIQTY 1
SECQTY 1
ERASE NO
FREEPAGE 15
PCTFREE 11
CLUSTER
(
  PART 1 VALUES ('P01', '999999999999')
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 5,
  PART 2 VALUES ('P02', '999999999999')
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 5,
  PART 3 VALUES ('P03', '999999999999')
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 5,
  PART 4 VALUES ('P04', '999999999999')
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 5,
  PART 5 VALUES ('P05', '999999999999')
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 5,
  PART 6 VALUES ('P06', '999999999999')
  USING STOGROUP JBRG0001
  PRIQTY 10
  SECQTY 10
  ERASE NO
  FREEPAGE 15
  PCTFREE 5,
  PART 7 VALUES ('P07', '999999999999')
  USING STOGROUP JBRG0001

110 IW Storage Guidelines and Considerations
PRIQTY 10
SECQTY 10
ERASE NO
FREEPAGE 15
PCTFREE 5,
PART 8 VALUES (′P08′, ′999999999999′)
USING STOGROUP JBRG0001
    PRIQTY 10
    SECQTY 10
    ERASE NO
FREEPAGE 15
PCTFREE 5,
PART 9 VALUES (′P09′, ′999999999999′)
USING STOGROUP JBRG0001
    PRIQTY 10
    SECQTY 10
    ERASE NO
FREEPAGE 15
PCTFREE 5,
PART 10 VALUES (′P10′, ′999999999999′)
USING STOGROUP JBRG0001
    PRIQTY 10
    SECQTY 10
    ERASE NO
FREEPAGE 15
PCTFREE 5,
PART 11 VALUES (′P11′, ′999999999999′)
USING STOGROUP JBRG0001
    PRIQTY 10
    SECQTY 10
    ERASE NO
FREEPAGE 15
PCTFREE 5,
PART 12 VALUES (′P12′, ′999999999999′)
USING STOGROUP JBRG0001
    PRIQTY 10
    SECQTY 10
    ERASE NO
FREEPAGE 15
PCTFREE 5)
SUBPAGES 1
BUFFERPOOL BP0
CLOSE NO
;
Having defined the table, some sample data can be loaded to match the examples specified. The following data was initially loaded.

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>ORDER_NUM</th>
<th>CNTL_STMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>4567890AB123</td>
<td>1991-10-24-21.12.00.000000</td>
</tr>
<tr>
<td>P01</td>
<td>4567890AB123</td>
<td>1991-11-24-21.12.00.000000</td>
</tr>
<tr>
<td>P01</td>
<td>4567890AB123</td>
<td>1991-12-24-21.12.00.000000</td>
</tr>
<tr>
<td>P02</td>
<td>567890AB1234</td>
<td>1992-01-21-20.01.00.000000</td>
</tr>
<tr>
<td>P02</td>
<td>567890AB1234</td>
<td>1992-02-21-20.01.00.000000</td>
</tr>
<tr>
<td>P02</td>
<td>567890AB1234</td>
<td>1992-03-21-20.01.00.000000</td>
</tr>
<tr>
<td>P03</td>
<td>67890AB12345</td>
<td>1992-04-20-20.10.00.000000</td>
</tr>
<tr>
<td>P03</td>
<td>67890AB12345</td>
<td>1992-05-20-20.10.00.000000</td>
</tr>
<tr>
<td>P04</td>
<td>7890AB123456</td>
<td>1992-06-23-13.40.00.000000</td>
</tr>
<tr>
<td>P04</td>
<td>7890AB123456</td>
<td>1992-07-23-13.40.00.000000</td>
</tr>
<tr>
<td>P04</td>
<td>7890AB123456</td>
<td>1992-08-23-13.40.00.000000</td>
</tr>
<tr>
<td>P05</td>
<td>890AB1234567</td>
<td>1992-09-22-13.40.00.000000</td>
</tr>
<tr>
<td>P05</td>
<td>890AB1234567</td>
<td>1992-10-22-13.40.00.000000</td>
</tr>
<tr>
<td>P05</td>
<td>890AB1234567</td>
<td>1992-11-22-13.40.00.000000</td>
</tr>
<tr>
<td>P06</td>
<td>90AB12345678</td>
<td>1993-01-24-19.50.00.000000</td>
</tr>
<tr>
<td>P06</td>
<td>90AB12345678</td>
<td>1993-02-24-19.50.00.000000</td>
</tr>
<tr>
<td>P06</td>
<td>90AB12345678</td>
<td>1993-03-24-19.50.00.000000</td>
</tr>
<tr>
<td>P07</td>
<td>0AB123456789</td>
<td>1993-04-20-20.17.00.000000</td>
</tr>
<tr>
<td>P07</td>
<td>0AB123456789</td>
<td>1993-05-20-20.17.00.000000</td>
</tr>
<tr>
<td>P07</td>
<td>0AB123456789</td>
<td>1993-06-20-20.17.00.000000</td>
</tr>
<tr>
<td>P08</td>
<td>AB1234567890</td>
<td>1993-07-23-20.00.000000</td>
</tr>
<tr>
<td>P08</td>
<td>AB1234567890</td>
<td>1993-08-23-20.00.000000</td>
</tr>
<tr>
<td>P08</td>
<td>AB1234567890</td>
<td>1993-09-23-20.00.000000</td>
</tr>
<tr>
<td>P09</td>
<td>B1234567890A</td>
<td>1993-10-26-19.55.00.000000</td>
</tr>
<tr>
<td>P09</td>
<td>B1234567890A</td>
<td>1993-11-26-19.55.00.000000</td>
</tr>
<tr>
<td>P09</td>
<td>B1234567890A</td>
<td>1993-12-26-19.55.00.000000</td>
</tr>
<tr>
<td>P10</td>
<td>1234567890AB</td>
<td>1991-01-25-20.15.00.000000</td>
</tr>
<tr>
<td>P10</td>
<td>1234567890AB</td>
<td>1991-02-25-20.15.00.000000</td>
</tr>
<tr>
<td>P10</td>
<td>1234567890AB</td>
<td>1991-03-25-20.15.00.000000</td>
</tr>
<tr>
<td>P11</td>
<td>234567890AB1</td>
<td>1991-04-22-13.29.00.000000</td>
</tr>
<tr>
<td>P11</td>
<td>234567890AB1</td>
<td>1991-05-22-13.29.00.000000</td>
</tr>
<tr>
<td>P11</td>
<td>234567890AB1</td>
<td>1991-06-22-13.29.00.000000</td>
</tr>
<tr>
<td>P12</td>
<td>34567890AB12</td>
<td>1991-07-20-12.55.00.000000</td>
</tr>
<tr>
<td>P12</td>
<td>34567890AB12</td>
<td>1991-08-20-12.55.00.000000</td>
</tr>
<tr>
<td>P12</td>
<td>34567890AB12</td>
<td>1991-09-20-12.55.00.000000</td>
</tr>
</tbody>
</table>
The following DDL was used to create the REFERENCE table for use in partition wrapping the ORDER_HIST_FULL table:

```
CREATE TABLESPACE REDBOOK1
    IN QMFDCINF
    USING STOGROUP JBRG0001
    PRIQTY 1
    SECQTY 1
    ERASE NO
    FREEPAGE 15
    PCTFREE 5
    BUFFERPOOL BP0
    LOCKSIZE ANY
    CLOSE NO
;
CREATE TABLE REFERENCE_F
(  PARTNO CHARACTER(3) NOT NULL,
  PERIOD_LABEL CHARACTER(10) NOT NULL,
  ID SMALLINT NOT NULL
)  IN QMFDCINF.REDBOOK1
    AUDIT NONE
;
COMMENT ON TABLE REFERENCE_F IS
    'REFERENCE TABLE FOR ORDER HISTORY FULL'
;
CREATE UNIQUE INDEX REFERENCE1
    ON REFERENCE_F
(  ID ASC,
  PARTNO ASC
)  USING STOGROUP JBRG0001
    PRIQTY 1
    SECQTY 1
    ERASE NO
    FREEPAGE 15
    PCTFREE 11
    CLUSTER
    SUBPAGES 1
    BUFFERPOOL BP0
    CLOSE NO
;
```
Having created the REFERENCE_F table, data was loaded to match the REFERENCE table in Figure 16 on page 47:

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>PERIOD_LABEL</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>4Q 91</td>
<td>8</td>
</tr>
<tr>
<td>P02</td>
<td>1Q 92</td>
<td>7</td>
</tr>
<tr>
<td>P03</td>
<td>2Q 92</td>
<td>6</td>
</tr>
<tr>
<td>P04</td>
<td>3Q 92</td>
<td>5</td>
</tr>
<tr>
<td>P05</td>
<td>4Q 92</td>
<td>4</td>
</tr>
<tr>
<td>P06</td>
<td>1Q 93</td>
<td>3</td>
</tr>
<tr>
<td>P07</td>
<td>2Q 93</td>
<td>2</td>
</tr>
<tr>
<td>P08</td>
<td>3Q 93</td>
<td>1</td>
</tr>
<tr>
<td>P09</td>
<td>4Q 93</td>
<td>0</td>
</tr>
<tr>
<td>P10</td>
<td>1Q 91</td>
<td>11</td>
</tr>
<tr>
<td>P11</td>
<td>2Q 91</td>
<td>10</td>
</tr>
<tr>
<td>P12</td>
<td>3Q 91</td>
<td>9</td>
</tr>
</tbody>
</table>

Based on this data, selecting from the ORDER_HIST_LAST view (see Figure 17 on page 48, that is, selecting the last three months of data written to the history table), the following report would be produced:

```
SELECT * FROM ORDER_HIST_LAST
---------+--------+---------+---------+---------+---------+----
PERIOD_LABEL ORDER_NUM CNTL_STMP PN ID
---------+--------+---------+---------+---------+---------+----
4Q 93    B1234567890A 1993-10-26-19.55.02.455452 P09 0
4Q 93    B1234567890A 1993-11-26-19.51.03.015054 P09 0
4Q 93    B1234567890A 1993-12-26-19.54.10.448612 P09 0
```

As part of the housekeeping work the following SQL was applied to the reference table:

```
UPDATE JIR.REFERENCE_F
SET ID = ID + 1
;
UPDATE JIR.REFERENCE_F
SET ID = 0
WHERE ID=12
;
UPDATE JIR.REFERENCE_F
SET PERIOD_LABEL = '1 Q 94'
WHERE ID=0
;
```
An extract has now been taken of January, February, and March. The data was unloaded from a detailed order table containing all activity for orders. The extract was done with a view that joins with the reference table with an ID = 0, to get the right partition number on the load file, which looks like this:

<table>
<thead>
<tr>
<th>Part</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P10NEW DATA</td>
<td>1994-01-25-20.15.00.000000</td>
</tr>
<tr>
<td>P10NEW DATA</td>
<td>1994-02-25-20.15.00.000000</td>
</tr>
<tr>
<td>P10NEW DATA</td>
<td>1994-03-25-20.15.00.000000</td>
</tr>
</tbody>
</table>

This data was loaded using the LOAD control statements shown in Figure 45:

```
LOAD DATA LOG NO RESUME YES INDDN SYSREC00 INTO TABLE JIR.ORDER_HIST_FULL PART 10 REPLACE
  (PARTNO            POSITION( 1 )
   CHAR( 3 ),
   ORDER_NUM        POSITION( 4 )
   CHAR( 12 ),
   CNTL_STMP        POSITION( 16 )
   TIMESTAMP EXTERNAL( 26 )
)
```

*Figure 45. Load Cards for LOAD Replacing a Partition*
The LOAD CARDS were produced by editing a standard set provided as a temporary file as part of the JCL PROC. The edit was performed by the following routine, which calls an edit macro:

/* REXX */
******************************************************************************
/* Name : JBREDITC */
/* */
/* Function : Invoke the ISPF editor for a data set in */
/* batch. */
/* */
/* Process : */
/* This CLIST invokes the ISPF editor to edit the */
/* file of DB2 load utility control statements */
/* produced by the DB2 unload utility. */
/* */
/* No parameters are passed into this routine */
/* since it determines names of files used in the */
/* JCL proc. */
/* */
/* The CLIST JBREDITD is invoked to perform the */
/* actual changes to the data set. */
/* */
/* */
/* Change Activity */
/* */
/* Version Date Author Comments */
/* 1.01 Adrian Houselander */
/* */
******************************************************************************
parse upper arg a1
trace 'o'
ok.8 = 'Y'
rxvclid = SYSVAR(SYSICMD)
userid = SYSVAR(SYSUID)
call issue("ISPEXEC CONTROL ERRORS RETURN")
ok.8 = 'Y'
******************************************************************************
******************************************************************************
/*-----------------------------------------------------------*/
/* Determine the name of the extract and load cards data sets*/
/* to determine what load cards to edit. It is assumed the */
/* JCL PROC calling this routine uses the following DDNAMEs */
/* EXTRACT - the name of the unload file */
/* LOADCD - the name of a standard set of LOAD CARDS */
/*-----------------------------------------------------------*/
cc = listdsi(extract file)
say sysdsname
ext_dsname = sysdsname
cc = listdsi(loadcds file)
say sysdsname
dsnname = sysdsname
call Readfil
if cc = 2 then
    call partdet
else
    partno = substr(line,1,3)
end
say PARTNO
call issue("ISPEXEC VPUT PARTNO SHARED")
call issue("ISPEXEC EDIT DATASET("dsname") MACRO(jbreditd)")
call issue("execio 0 DISKR EXTRACT (FINIS")
exit
/*- PARTDET ------------------------------------------------*/
/* Determine the partition to be used, or abend */
PARTDET:
/* Allocations to run SPUFI are in the PROC. */
/* Run spufi to determine the partition number that needs */
/* to be load replaced using following query. */
/* SELECT */
/* 'ZZZHERE' , */
/* PARTITION_NUMBER */
/* FROM JBR.JBRTCWRP A -- wrapping REFERENCE table */
/* WHERE */
/* A.ID = 1; */
/* */
"MAKEBUF"
buff = rc
Queue "RUN PROGRAM(RXVSBAP) PLAN(RXVSBAP) PARM('/I(S)')".
" LIBRARY('JB#LD.PROD.LOAD')"
Queue "END"
Address TSO "DSN SYSTEM(DB2A)"
"DROPBUF" buff
Now read the output file to find the partition number to use. SPUFI repeats the QUERY in the listing, so we need to find the second instance of ZZZHERE.

```plaintext
call readspu
a = index(results, 'ZZZHERE')
do while (a = 0) & (cc = 0)
  call readspu
  a = index(results, 'ZZZHERE')
end

call readspu
a = index(results, 'ZZZHERE')
do while (a = 0) & (cc = 0)
  call readspu
  a = index(results, 'ZZZHERE')
end
if cc ≠ 0 then
  do
    say ext_dsname||'|' is an empty file. This program is unable to generate a partition number for the load cards' say 'since no partition number could be found on JBRTWCWRP.' out.1 = 8
    call issue("EXECIO 1 DISKW JBRERROR (FINIS STEM OUT.")
    exit
  end
results = substr(results, a, length(results))
parse var results garbage '|' partno rest return
```

/* Read the infile data set. */
/*- READFIL -----------------------------------------------*/
READFIL:
call issue("EXECIO 1 DISKR EXTRACT (FINIS STEM LINE.")
if cc = 0 then
  do
    line = substr(line.1', 1, 72)
    say line
  end
return
/*- READSPU -----------------------------------------------*/
/* */
/* Read the SPUFI report. */
/*---------------------------------------------------------*/
READSPU:
call issue ("EXECIO 1 DISKR Sysprint (STEM PRNT.")
if cc = 0 then
   do
      results = substr(PRNT.1,2,133)
      say results
   end
return
/*- ISSUE -----------------------------------------------*/
/* */
/* This common routine processes a command contained in the */
/* string passed in. If the result of this proc produces a */
/* code which is not one indic by an OK.n value, then write */
/* error messages to the default output data set and return */
/* from the exec. */
/*----------------------------------------------------------*/
ISSUE:
   ARG issuecmd
   address TSO issuecmd
   ok.0 = 'Y'
   ok.2 = 'Y'
   cc = rc
   if ok.cc ^= 'Y' then
      do
         say SYSVAR(SYSICMD) 'got a return code of ' cc,
         'when attempting' issuecmd,
         ' the REXX exec has been terminated.'
         out.1 = cc
         ADDRESS TSO "EXECIO 1 DISKW JBRERROR (FINIS STEM OUT.)"
         EXIT CC
      end
   else
      return cc
/* */
/*---------------------------------------------------------*/
**REXX**

ISREDIT MACRO

 /**************************************************************************
 /* Name : JBREDITD */
 /* */
 /* Function : Edit a data set in batch. */
 /* */
 /* */
 /* Process : */
 /* */
 /* This CLIST uses the ISPF editor to change the */
 /* file of DB2 load utility control statements */
 /* produced by the DB2 unload utility used to take */
 /* monthly unloads of PDW tables. */
 /* */
 /* The CLIST is designed to produce utility control */
 /* statements that will replace the contents of one */
 /* partition of a partitioned tablespace. */
 /* */
 /* This CLIST is invoked by CLIST JBREDITC which */
 /* initially invokes the ISPF editor for the data set */
 /* and determines the value of the partition number */
 /* concerned. This partition number is obtained by */
 /* this CLIST via ISPEXEC VGET. */
 /* */
 /* The load parameters are changed from default */
 /* values to 'LOG NO RESUME YES'. */
 /* The target table name is changed from the view */
 /* name used to unload the current table, */
 /* JBRT5xxx.UNLOAD, to the snapshot table name */
 /* JBRT7xxx. */
 /* */
 /* Change Activity */
 /* Version Date Author Comments */
 /* 1.00 */
 /* */
 /**************************************************************************

parse upper arg a1
trace 'o'
call issue("ISPEXEC VGET partno SHARED")
ok.8 = 'Y'
rxvclid = SYSVAR(SYSICMD)
userid = SYSVAR(SYSUID)
ok.8 = ' '
/**************************************************************************

ok.4 = 'Y'
call issue("ISREDIT CHANGE 'LOG NO' || ,
   'LOG NO RESUME YES DISCARDS 1' ALL")
call issue("ISREDIT CHANGE 'UNLOAD' 'HIST_FULL' ALL")
call issue("ISREDIT LINE AFTER 2 = 'PART' ||,
   partno ||' REPLACE'")
call issue("ISEDIT END")

exit
ISSUE:
ARG issuecmd
address TSO issuecmd
ok.0 = 'Y'
cc = rc
if ok.cc ^= 'Y' then
  do
    if SYSVAR(SYSENV) = 'FORE' then
      do
        RXVCLID = SYSVAR(SYSICMD)
        address ISPEXEC "DISPLAY PANEL(RXVSERR)"
      end
    else
      do
        say SYSVAR(SYSICMD) 'got a return code of ' cc ,
        'when attempting' issuecmd ,
        'the REXX exec has been terminated.'
      end
    exit
  end
else
  return cc
/* */
/*----------------------------------------------------------*/

The edit macro changed the LOAD cards, which were used to load the table. The ORDER_HIST_LAST view now produces the following result:

```
SELECT * FROM ORDER_HIST_LAST
---------+---------+------+---------+---------+---------+----
PERIOD_LABEL ORDER_NUM CNTL_STMP PN ID
---------+---------+------+---------+---------+---------+----
1Q 94 NEW DATA 1994-01-25-20.15.00.000000 P10 0
1Q 94 NEW DATA 1994-02-25-20.15.00.000000 P10 0
1Q 94 NEW DATA 1994-03-25-20.15.00.000000 P10 0
```

This proves that the data loading worked as expected.
Appendix B. Additional Coding Examples

This appendix shows coding examples for additional functions for moving and maintaining data in an environment with large DB2 tables. The examples do the following:

- Adjust the partition number that is appended to records that are subsequently loaded into DB2 by the LOAD utility (see B.1, “Adjusting the Partition Number” on page 124)
- Migrate partitions of DB2 tables (see B.2, “REXX Example to HMIGRATE Partitions” on page 130)
- Move DB2 tables or partitions to another volume (see B.3, “Job to Move Tablespace” on page 133).
B.1 Adjusting the Partition Number

The following piece of code can be used as a basis for building a LOAD file converter to convert an ordinary unload file into a file suitable for loading into a partition wrapping history table:

JBR263P:
/*----------------------------------------------------------*/
/*----------------------------------------------------------*/
/* */
/* FUNCTION ; This program adds the partition number onto */
/* an extract file */
/* */
/* */
/* DESCRIPTION; Each record is read and the current */
/* partition number is assigned to the record */
/* for later use with the DB2 utility. */
/* */
/* */
/*----------------------------------------------------------*/
PROC OPTIONS(MAIN) REORDER; /* NO PARAMETERS */
/*----------------------------------------------------------*/
/* STANDARD DECLARES */
/*----------------------------------------------------------*/
DEFAULT RANGE(*) STATIC;
DCL PLIXOPT CHAR(48) VAR EXT
   INIT('ISASIZE(36K),HEAP(4K,4K,ANYWHERE)');
DCL INTERNAL CHAR(21) INIT('IBM INTERNAL USE ONLY');
/*-----------------------------------------------------------*/
/* TABLE DECLARATIONS */
/*-----------------------------------------------------------*/
EXEC SQL DECLARE JBRTCWRP TABLE
   ( TABLE CHAR(4) NOT NULL,
     PARTITION_NUMBER SMALLINT NOT NULL,
     PERIOD CHAR(10) NOT NULL,
     ID SMALLINT NOT NULL
   );
/*-----------------------------------------------------------*/
/*-----------------------------------------------*/
/*                      FILE DECLARATIONS           */
/*-----------------------------------------------*/

DCL INDATA FILE RECORD INPUT ENV (FB RECSIZE(359) TOTAL);
DCL OUTDATA FILE RECORD OUTPUT ENV (FB RECSIZE(364) TOTAL);
DCL SYSPRINT FILE STREAM;

/*-----------------------------------------------*/
/*                      DECLARATIONS              */
/*-----------------------------------------------*/

DCL CHAR2 CHAR (2) INIT(' ');  
DCL P1 POINTER;  
DCL P2 POINTER;  
DCL WRITE_0181 FIXED BIN(31) INIT(0);  
DCL READ_0181 FIXED BIN(31) INIT(0);  
DCL CAT1 FIXED BIN(31) INIT(0);  
DCL CAT2 FIXED BIN(31) INIT(0);  
DCL EOF_INDATA BIT(1) INIT('0');  
DCL TRAILER BIT(1) INIT('0');  
DCL PART_NO FIXED BIN(15) INIT(0);  
DCL PIC99 PIC '99' DEF CHAR2;  
DCL INSTR CHAR(359);  
DCL OUTSTR CHAR(364);  
DCL PLIDUMP BUILTIN;  
DCL ADDR BUILTIN;  
DCL NULL BUILTIN;  
DCL INDEX BUILTIN;  
DCL SUBSTR BUILTIN;  
DCL DATE BUILTIN;  
DCL TIME BUILTIN;

/*-----------------------------------------------*/
/*                      RECORD STRUCTURES          */
/*-----------------------------------------------*/

DCL 1 IN_181 BASED (P1) UNALIGNED,  
%INCLUDE JBRD263R ;;

DCL 1 OUT_181 BASED (P2) UNALIGNED,  
%INCLUDE JBRD263S ;;
ON ERROR
BEGIN REORDER;
ON ERROR
SYSTEM;
CALL PLIDUMP('SHAFTB');
SIGNAL ERROR;
END;

ON ENDFILE(INDATA) EOF_INDATA = '1'B;

DCL (RXVC92P) EXTERNAL ENTRY; /* SQL ERROR HANDLING MODULE */

EXEC SQL INCLUDE SQLCA;
EXEC SQL WHENEVER SQLERROR GOTO SQLERR;
EXEC SQL WHENEVER SQLWARNING GOTO SQLERR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

P1 = ADDR(INSTR);
P2 = ADDR(OUTSTR);

IF SUBSTR(DATE,5,2) = '01' THEN DO;
EXEC SQL
UPDATE JBRTCWRP
SET ID = ID -1
WHERE TABLE = '0181';
IF SQLCODE = 0 THEN DO;
    PUT PAGE FILE(SYSPRINT)
    EDIT('UPDATE OF JBRTCWRP TABLE 0181 FAILED ID = ID - 1 ')
         (SKIP(1),A(50));
    SIGNAL ERROR;
END;

EXEC SQL
    UPDATE JBRTCWRP
    SET PERIOD = CHAR(CURRENT DATE)
    WHERE TABLE = '0181'
    AND ID = 1;

IF SQLCODE = 0 THEN DO;
    PUT PAGE FILE(SYSPRINT)
    EDIT('UPDATE OF PERIOD FIELD IN JBRTCWRP FAILED ')
         (SKIP(1),A(50));
    SIGNAL ERROR;
END;

EXEC SQL
    UPDATE JBRTCWRP
    SET ID = 23
    WHERE TABLE = '0181'
    AND ID = -1;

IF SQLCODE = 0 THEN DO;
    PUT PAGE FILE(SYSPRINT)
    EDIT('UPDATE OF ID FIELD TO 23 IN JBRTCWRP FAILED ')
         (SKIP(1),A(50));
    SIGNAL ERROR;
END;

END;

/*****************************************************************************/
/* FIND PARTITION NUMBERS */
/*****************************************************************************/
EXEC SQL
    SELECT
        PARTITION_NUMBER
    INTO :PART_NO
    FROM JBRTCWRP
    WHERE TABLE = '0181'
    AND ID = 1;

IF SQLCODE = 0 THEN DO;
    PUT PAGE FILE(SYSPRINT)
    EDIT('SELECT INTO PART_NO FROM JBRTCWRP FAILED ')
         (SKIP(1),A(50));
    SIGNAL ERROR;
END;
OPEN FILE (INDATA) INPUT;
OPEN FILE (OUTDATA) OUTPUT;

READ FILE (INDATA) INTO (INSTR);
IF (EOF_INDATA)
THEN DO;
   PUT PAGE FILE(SYSPRINT)
       EDIT(′EMPTY INPUT FILE INDATA"
           (SKIP(1),A(50));
   SIGNAL ERROR;
END;
READ_0181 = READ_0181 + 1;
/* LOOP THROUGH FILE, COPYING DATA UNTIL THE END */
DO WHILE (¬EOF_INDATA);
   READ_0181 = READ_0181 + 1;
   /* CALL ROUTINE TO COPY INPUT RECORD STRUCTURE TO OUTPUT */
   CALL COPY_AC( PART_NO , /* PARTITION NUMBER */
                 IN_181 , /* INPUT RECORD LAYOUT */
                 OUT_181 /* OUTPUT RECORD LAYOUT */
             ) ;
   WRITE FILE(OUTSTR) FROM(OUTSTR) ;
END;

CLOSE FILE (INDATA) ;
CLOSE FILE (OUTDATA) ;

PUT FILE (SYSPRINT)
   EDIT(′PROGRAM RUNTIME STATISTICS′,
       ′NUMBER INPUT RECORDS = ′, READ_0181,
       ′PARTITION NUMBER USED = ′, PART_NO,
       ′PROGRAM ENDED AT ′, TIME, ′ ON ′, DATE)
( SKIP(2), A(34),
  SKIP(1), A(29), F(7),
  SKIP(1), A(29), F(7),
  SKIP(1), A(29), F(7),
  SKIP(3), A(25), P′99.99.99.999′,A(4),A(6));
GOTO MYEND;
/*--------------------------------------------------------*/
/*SQL ERROR ROUTINE */
/*--------------------------------------------------------*/
SQLERR:
    CALL RXVC92P(SQLCA, 'JBR263P');
    EXEC SQL WHENEVER NOT FOUND CONTINUE;
    EXEC SQL WHENEVER SQLWARNING CONTINUE;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK;
    SIGNAL ERROR;
/*****************************/
/** END OF MODULE */
/*****************************/
MYEND:
END; /* PROGRAM */
B.2 REXX Example to HMIGRATE Partitions

The following is a REXX example to issue the HMIGRATE command against tablespaces (parameters are high-level VCAT, database name, and tablespace name):

```rexx
/* REXX */
/* ********************************************************* */
/* Name : TBMIGRAT */
/* Function : Migrate Partitions of a Partitioned tablespace */
/* to either save DB2 POOL space or for further */
/* action such as data moving. */
/* Process : */
/* Identify data sets which need to be migrated */
/* Stop the tablespace */
/* Migrate the data sets */
/* NB: This may take some time according to the */
/* work being performed by DFHSM, so optionally */
/* check to see if the data sets have been */
/* migrated before the next step. */
/* Start the tablespace */
/* */
/* According to the data set migration path for the data sets */
/* in question the data sets may get moved to DASD, TAPE or */
/* optical devices. */
/* */
/* Change Activity */
/* Version Date Author Comments */
/* 1.00 20/02/94 A Houselander */
/* */
/* ********************************************************* */
parse upper arg hlvcat dbname tsname
trace 'o'
rxvcld = SYSVAR(SYSICMD)
userid = SYSVAR(SYSUID)
/* *************** MAIN BODY STARTS HERE *********************** */
cc = 0
quote = ''''
linestodo = 64
call finddsn
call stopdb
call migts
call startdb
exit
/*- FINDDSN --------------------------------------------------*/
/* */
/* Find data sets associated with a tablespace */
/*- ************************************************************/
FINDDSN:
ok.4 = 'Y'
ok.12 = 'Y'
```

IW Storage Guidelines and Considerations
hlq = strip(hlvcat,b)
dbname = strip(dbname,b)
tsname = strip(tsname,b)

/* Form up a LISTC command to determine the data sets assoc */
/* with the requested tablespace, then issue the command. */
/* output from the LISTC command is trapped onto a STEM for */
/* processing. */

cmd = 'listc en('quote||hlq||'.'||dbname')||
      dbname.'tsname'.I0001.*'quote')'
cc = outtrap('line.',linestodo,concat)
call issue(cmd)
count = 0

cc = outtrap(off)

/* Work through the stem and store the partitions seperately */
do i = 1 to line.0
   a = index(line.i,hlq)
   if a /*= 0 then
      do
         count = count + 1
         dsname = strip(substr(line.i,a,length(line.i)),b)
         partit.count = dsname
      end
   end
partit.0 = count
return

/*- STOPDB --------------------------------------------------*/
/* */
/* Stop the requested tablespace so we can migrate the data */
/*--------------------------------------------------------*/
STOPDB:
  "MAKEBUF"
  buff = rc
  Queue "-STOP DB("dbname") SPACE="tsname""
  Queue "END"
  address TSO "DSN SYSTEM(DD1A)"
  "DROPBUF" buff
return
/*- STARTDB -------------------------------------------------*/
/* */
/* Start the requested tspace so users can access the data */
/*----------------------------------------------------------*/
STARTDB:
"MAKEBUF"
buff = rc
Queue "-START DB(“dbname”) SPACENAM(“tsname”)"
Queue "END"
address TSO “DSN SYSTEM(DD1A)"
"DROPBUF" buff
return

/*- MIGTS --------------------------------------------------*/
/* */
/* Migrate the data sets associated with a tablespace */
/*-----------------------------------------------------*/
MIGTS:
do i = 1 to partit.0
  cmd = ’HMIGRATE ’quote||partit.i||quote
  call issue(cmd)
end
return

/*- ISSUE ------------------------------------------------*/
/* */
/* This common routine processes a command contained in the */
/* string passed in. If the result of this proc produces a */
/* code which is not one indic by an OK.n value, then write */
/* error messages to the default output data set and return */
/* from the exec. */
/*--------------------------------------------------------*/
ISSUE:
ARG issuecmd
address TSO issuecmd
ok.0 = ’Y’
cc = rc
if ok.cc ^= ’Y’ then
do
  say SYSVAR(SYSICMD) ’got a return code of ’ cc ,
    ’ when attempting ’ issuecmd ,
    ’ the REXX exec has been terminated.’
  exit
end
else
  return cc
/* */
This is a sample job to move a tablespace, or partition of a tablespace, to another volume:

```
//***********************************************************
//*** JOB TO MOVE DATA SETS OFF ONE VOLUME TO ANOTHER ***
//***********************************************************
/*JOBPARM L=99,SYSAFF=* 
//COPY EXEC PGM=ADRDSSU,REGION=4096K
//SYSPRINT DD SYSOUT=* 
//IN1 DD UNIT=3390,DISP=SHR,VOL=SER=D1DA97 * 3390-3 vol 
//OUT1 DD UNIT=3390,DISP=SHR,VOL=SER=D1DA95 * 3995 volume 
COPY DATASET( - 
    INCLUDE(JBRRD.DSNDDBC.JBRD0001.JBRS0100.I00001.A0001)) - 
    INDDNAME(IN1) - 
    OUTDDNAME(OT1) - 
    TGTGDS(SOURCE) - 
    DELETE CATALOG ALLEXCP ALLDATA(*) TGTALLOC(SOURCE) 
/* 
```
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS</td>
<td>automatic class selection (for DFSMS)</td>
</tr>
<tr>
<td>APF</td>
<td>Authorized Program Facility (in MVS)</td>
</tr>
<tr>
<td>BDW</td>
<td>Business Data Warehouse (part of CINFO set of applications)</td>
</tr>
<tr>
<td>BTX</td>
<td>Business Transaction (within CINFO set of applications)</td>
</tr>
<tr>
<td>DARM</td>
<td>Data Archive Retrieval Manager</td>
</tr>
<tr>
<td>DBA</td>
<td>database administrator</td>
</tr>
<tr>
<td>DBD</td>
<td>database descriptor (in DB2)</td>
</tr>
<tr>
<td>DDF</td>
<td>Distributed Data Facility (component of DB2)</td>
</tr>
<tr>
<td>DSS</td>
<td>decision support system</td>
</tr>
<tr>
<td>ECKD</td>
<td>extended count-key-data (recording method)</td>
</tr>
<tr>
<td>EIS</td>
<td>executive information system</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>IMS</td>
<td>Information Management System</td>
</tr>
<tr>
<td>ISC</td>
<td>Intersystems Coupling (component of IMS)</td>
</tr>
<tr>
<td>ISMF</td>
<td>Interactive Storage Management Facility</td>
</tr>
<tr>
<td>ISPF</td>
<td>Interactive System Productivity Facility</td>
</tr>
<tr>
<td>ITM</td>
<td>interface to macros (function within CINFO set of applications)</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>MPP</td>
<td>message processing program (in IMS)</td>
</tr>
<tr>
<td>OPC</td>
<td>Operations Planning and Control</td>
</tr>
<tr>
<td>PDW</td>
<td>Process Data Warehouse (in CINFO set of applications)</td>
</tr>
<tr>
<td>QMF</td>
<td>Query Management Facility</td>
</tr>
<tr>
<td>RACF</td>
<td>Resource Access Control Facility</td>
</tr>
<tr>
<td>RIF</td>
<td>row identifier (in CINFO set of applications)</td>
</tr>
<tr>
<td>SCSI</td>
<td>Small Computer Systems Interface</td>
</tr>
<tr>
<td>SKCT</td>
<td>skeleton cursor table (in DB2)</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
</tbody>
</table>
indexes (continued)
indexing and summary tables (continued)
consideration of size and cost 101
effect on granularity 103
evaluating alternatives (examples) 100, 101, 102, 103
index as alternative to summary 99
indexes, importance of 104
multiple summaries 102
single summary - compromise 102
user requirements 99
maintenance issues less applicable 104
uniqueness enforcement 104
indexing and summary tables 99
changing aggregation criteria 104
complexity of unsummarized data 99
conflicting user requirements 99, 102
consideration of size and cost 101
effect on granularity 103
evaluating alternatives (examples) 100, 101, 102, 103
index as alternative to summary 99
indexes, importance of 104
maintenance issues less applicable 104
multiple summaries 102
single summary - compromise 102
user requirements 99
indexspace 18
application, description of 18
information catalog 2
information store 2
data content 3
feeding from operational 29
Information Warehouse 1
components 4
definition 4
description 1
implementation of 65
information catalog 2
Information Warehouse and SMS 22
rationale 1
role of DataPropagator Relational 29
solution 2
user access 2
Information Warehouse Architecture and types of summaries 89
informational 1
applications 2
databases 1
feeding from operational 29
informational data 77
atomic detailed 77
reference to - the exception rather than the rule 78
derivation from operational 91
L
logs 17
DB2 active logs 18
DB2 archive logs 18
DFHSM definitions 26
SMS definitions 26
M
meta-data (in Information Warehouse Architecture) 3
multilevel cumulative summary 81
multiple levels of summary 78
O
operational 2
applications 2
feeding informational 29
P
partition wrapping 45
see wrapping 45
R
real-time data (in Information Warehouse Architecture) 3
reconciled data (in Information Warehouse Architecture) 4
response 19
data set response time range 19
roll-up 79, 82
needed before drill-down 79
rolling summary 82
S
sample 87
simple cumulative summary 80
simple direct summary 84
subsets, creative 105
aggregation from 105
indexes in support of 105
summaries 77
advantages 78
combinations of summaries 88
data placement strategy 88
design changes over time 88
light summarization 77
multiple levels 78
telephone call example 77
types of 80
variations of summaries 88
why needed 77
summary tables, types of 80
combinations of summaries 88
continuous summary 85
aggregation from previous continuous summary 86
summary tables, types of (continued)
continuous summary (continued)
  aggregation of simple direct summaries 85
design alternatives 87
example (illustration) 85
interval processing 87
maintenance considerations 87
recreate status at point in time 86
set-level update, applicability of 87
status tracking 85
data placement strategy 88
design changes over time 88
multilevel cumulative summary 81
  adding data not in previous level 82
  aggregation criteria 82
different aggregation criteria 82
  drill-down, direct support of 82
  omitting data from previous level 82
  roll-up, direct support of 82
rolling summary 82
  complexity of maintenance, additional 83
  DASD space, possible saving of 83
  drill-down, support of 83
  granularity, effect on 83
  I/Os, possible reduction of 83
  skewed reference pattern, benefits 83
sample 87
  advantages compared to accessing
    unsummarized data 88
  confidence level, importance of 88
  example (illustration) 87
  inapplicability to further summarization 88
  nonrandom 88
  not an aggregation 87
  refresh requirements 88
  role in testing 88
  sample size 88
  statistical questions, applicability to 88
  storage cost, potential savings 88
simple cumulative summary 80
  drill-down 81
  I/O, reduction of 81
  maintenance, ease of 81
simple direct summary 84
  alternative to access to operational data 85
  changes over time, reporting 84
  ease of maintenance 85
  example (illustration) 84
  explicit or implicit time representation 84
  status tracking 84
  when to generate record 84
  variations of summaries 88
System Managed Storage 21
ACS routines 25
classes 23
description 22
ISMF 23
the need 22

T
table design considerations 91
derivation from operational data 91
design steps (summary) 105
design steps for informational tables 91
ideal result of design of summaries 106
operational data, derivation from 91
tablespace 18
  application, description of 18
  compression of 39
  creation 22
  end-user, description of 19
  SMS definitions 25
time series data 43
types of summaries, relation to Information
  Warehouse Architecture 89
types of summary tables 80

U
user interface 2

V
variations of summaries 88

W
wrapping 45
  benefits 51
  example 47
  example of coding 115
  implementation 46
  maintenance 49
  views 49
Your feedback is very important to help us maintain the quality of ITSO Bulletins. Please fill out this questionnaire and return it using one of the following methods:

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction  
Organization of the book  
Accuracy of the information  
Relevance of the information  
Completeness of the information  
Value of illustrations  
Grammar/punctuation/spelling  
Ease of reading and understanding  
Ease of finding information  
Level of technical detail  
Print quality

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:
   Do you provide billable services for 20% or more of your time?  
   Yes___ No___
   Are you in a Services Organization?  
   Yes___ No___

b) Are you working in the USA?  
   Yes___ No___

c) Was the Bulletin published in time for your needs?  
   Yes___ No___

d) Did this Bulletin meet your needs?  
   Yes___ No___

If no, please explain:

What other topics would you like to see in this Bulletin?

What other Technical Bulletins would you like to see published?

Comments/Suggestions:  ( THANK YOU FOR YOUR FEEDBACK! )

Name
Address
Company or Organization
Phone No.