

Tandem Reload Analyzer Manual

Abstract

This manual describes how to use Tandem Reload Analyzer (TRA) to identify fragmented Enscribe files and NonStop SQL tables and to determine the files and tables that will benefit from an online reorganization.

Product Version

Reload Analyzer D42

Supported Releases

This manual supports D42.00 and all subsequent releases until otherwise indicated in a new edition.

Part Number	Published	Release ID
129830	November 1996	D43.00

Document History

Part Number	Product Version	Published
129830	Reload Analyzer D42	November 1996

New editions incorporate any updates issued since the previous edition.

A plus sign (+) after a release ID indicates that this manual describes function added to the base release, either by an interim product modification (IPM) or by a new product version on a .99 site update tape (SUT).

Ordering Information

For manual ordering information: domestic U.S. customers, call 1-800-243-6886; international customers, contact your local sales representative.

Document Disclaimer

Information contained in a manual is subject to change without notice. Please check with your authorized Tandem representative to make sure you have the most recent information.

Export Statement

Export of the information contained in this manual may require authorization from the U.S. Department of Commerce.

Examples

Examples and sample programs are for illustration only and may not be suited for your particular purpose. Tandem does not warrant, guarantee, or make any representations regarding the use or the results of the use of any examples or sample programs in any documentation. You should verify the applicability of any example or sample program before placing the software into productive use.

U.S. Government Customers

FOR U.S. GOVERNMENT CUSTOMERS REGARDING THIS DOCUMENTATION AND THE ASSOCIATED SOFTWARE:

These notices shall be marked on any reproduction of this data, in whole or in part.

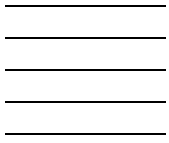
NOTICE: Notwithstanding any other lease or license that may pertain to, or accompany the delivery of, this computer software, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Section 52.227-19 of the FARS Computer Software—Restricted Rights clause.

RESTRICTED RIGHTS NOTICE: Use, duplication, or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013.

RESTRICTED RIGHTS LEGEND: Use, duplication or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the rights in Technical Data and Computer Software clause in DAR 7-104.9(a). This computer software is submitted with “restricted rights.” Use, duplication or disclosure is subject to the restrictions as set forth in NASA FAR SUP 18-52p227-79 (April 1985) “Commercial Computer Software—Restricted Rights (April 1985).” If the contract contains the Clause at 18-52p227-74 “Rights in Data General” then the “Alternate III” clause applies.

U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

Unpublished — All rights reserved under the Copyright Laws of the United States.



New and Changed Information

This is the second edition of the *Tandem Reload Analyzer Manual*. This section identifies the information that has been added, deleted, or revised since the previous edition.

New Information

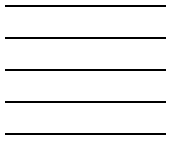
This version of the manual documents the following new features:

New Information	Section
<i>fupin</i> and THRESHOLD <i>value</i> options were added to the Run command.	2
A FUP obey file containing RELOAD commands can be generated automatically.	2
Six new messages have been added.	Appendix A

Changed Information

This version of the manual includes the following changes:

Changed Information	Section
This version of the manual was produced using FrameMaker.	All
Titles of the manuals you can read for more information have been updated to match published Tandem CD Read and TIM titles.	1 and 2
Examples and figures have been updated.	1 and 2
Cause, Effect, and Recovery texts have been expanded.	Appendix A
One message that Reload Analyzer no longer uses was removed.	Appendix A



Contents

New and Changed Information iii

About This Manual vii

Notation Conventions xi

1. Tandem Reload Analyzer Overview

Key-Sequenced Files 1-1

File Disorganization 1-3

Data Chains 1-7

Summary 1-7

2. Using Tandem Reload Analyzer

System Requirements 2-1

Installing Reload Analyzer 2-1

Running Reload Analyzer 2-1

 Reloading the Database 2-2

 Interrupting Reload Analyzer 2-3

 Status Indicator 2-3

Understanding the Output 2-3

 General Information 2-6

 Block Information 2-7

 Data Chain Information 2-9

Output After Reloading 2-12

When to Reload 2-13

A. Error Messages

Glossary

Index

Examples

Example 2-1. Sample FUP RELOAD Command File 2-3

Example 2-2. Sample Reload Analyzer Output 2-5

Example 2-3. General Information Format 2-6

Example 2-4. Block Information Format 2-7

Example 2-5. Data Chain Information Format 2-9

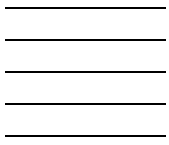
Example 2-6. Output from SALESDB After Reloading 2-12

Figures

- Figure i. Documentation Road Map viii
- Figure 1-1. Key-Sequenced File Structure 1-2
- Figure 1-2. An Organized File 1-4
- Figure 1-3. A Disorganized File 1-6

Tables

- Table 2-1. Values in Chain Length Histogram 2-10



About This Manual

This manual describes Tandem Reload Analyzer, a database management productivity tool that analyzes audited, key-sequenced Enscribe files or NonStop SQL (Structured Query Language) objects (tables or indexes only). Tandem Reload Analyzer (Reload Analyzer) helps you determine whether a file or NonStop SQL object needs to be reloaded.

Audience

This manual is intended for database administrators or others who maintain Enscribe or NonStop SQL database files.

Audience Prerequisites

Before using this manual, you should be familiar with the following:

- File Utility Program (FUP)
- Tandem Advanced Command Language (TACL)
- Peruse
- Audited, key-sequenced Enscribe files or NonStop SQL objects

Manual Organization

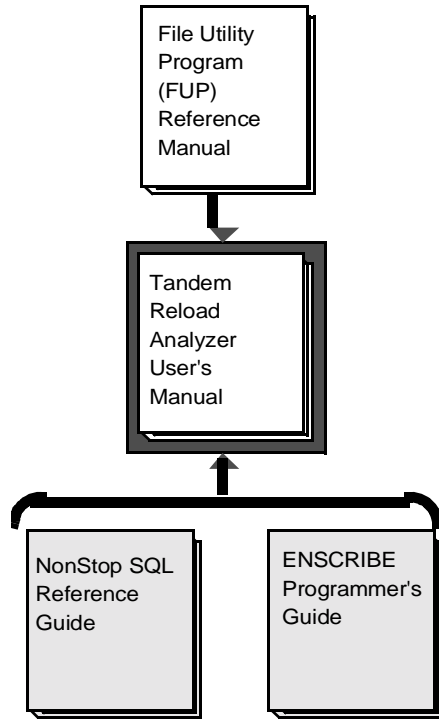
This manual consists of the following sections and appendixes:

Section	Contents
1	Gives an overview of Reload Analyzer.
2	Describes Reload Analyzer hardware and software requirements and installation procedures and how to use Reload Analyzer.
Appendix A	Explains each Reload Analyzer error message and how to recover from the condition causing it.

Documentation Road Map

Figure i shows how this manual is related to other Tandem manuals. The flowchart shows the order in which to read these manuals.

Figure i. Documentation Road Map



Legend

- Required reading
- Recommended reading

006

To Order Manuals

To order any Tandem manual, call 1-800-243-6886.

Credits

Many people contributed to the development of Tandem Reload Analyzer and this manual. Special thanks to the following people: Buddy Wilbanks, Chavoit Li, Bridgett Low, Melissa Wibom, Dave McBride, Jennifer Curci, Pam Muraca, John Cosgrave, Paul Denzinger, David O'Dette, Gary Smith, and Mike McDonald.

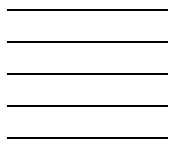
Your Comments Invited

After using this manual, please take a moment to send us your comments. You can do this by returning a Reader Comment Card or by sending an Internet mail message.

A Reader Comment Card is located at the back of printed manuals and as a separate file on the Tandem CD Read disc. You can either FAX or mail the card to us. The FAX number and mailing address are provided on the card.

Also provided on the Reader Comment Card is an Internet mail address. When you send an Internet mail message to us, we immediately acknowledge receipt of your message. A detailed response to your message is sent as soon as possible. Be sure to include your name, company name, address, and phone number in your message. If your comments are specific to a particular manual, also include the part number and title of the manual.

Many of the improvements you see in Tandem manuals are a result of suggestions from our customers. Please take this opportunity to help us improve future manuals.



Notation Conventions

General Syntax Notation

The following list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters. Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

```
file-name
```

[] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [ \system-name. ] $terminal-name
```

```
INT[ ERRUPTS ]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list may be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
LIGHTS [ ON           ]  
        [ OFF         ]  
        [ SMOOTH [ num ] ]
```

```
K [ X | D ] address-1
```

Punctuation. Parentheses, commas, semicolons, and other symbols not previously described must be entered as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;
```

```
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must enter as shown. For example:

```
"[ repetition-constant-list ]"
```

Item Spacing. Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In the following example, there are no spaces permitted between the period and any other items:

```
$process-name.#su-name
```

Notation for Messages

The following list summarizes the notation conventions for the presentation of displayed messages in this manual.

Nonitalic text. Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

Backup Up.

lowercase italic letters. Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

p-register

process-name

1

Tandem Reload Analyzer Overview

The File Utility Program (FUP) RELOAD command physically reorganizes audited, key-sequenced Enscribe files and NonStop SQL objects (tables and indexes) that have become fragmented or disorganized. Because a FUP RELOAD is not a procedure that you should use indiscriminately, determining when to reload is difficult. Tandem Reload Analyzer is a database management tool that helps you determine whether a key-sequenced Enscribe file or key-sequenced NonStop SQL object needs to be reorganized.

Note. Throughout this manual, the term *file* refers to any NonStop SQL object (table or index) or key-sequenced Enscribe file. Even though Reload Analyzer runs against any file or NonStop SQL object, FUP RELOAD will not reorganize a nonaudited Enscribe file or nonaudited NonStop SQL object.

When you analyze a file with Tandem Reload Analyzer (Reload Analyzer), it provides various key information. The information includes the total blocks read in the file, minimum and maximum number of records, average slack in the data blocks, average percentage of slack, and the space used in the data and index blocks. This information is the same as the information given by the FUP INFO *filename*, STAT command. Reload Analyzer, however, gives the information faster than the FUP INFO *filename*, STAT command and one other key piece of information—the number of contiguous data blocks. When you have all this information about a file, you can easily decide whether to reload a key-sequenced Enscribe file or key-sequenced NonStop SQL object.

To help you understand why a FUP RELOAD is necessary for a file, the next section describes key-sequenced file structure. You will also learn how a file becomes disorganized. If you want more information, see the *ENSCRIBE Programmer's Guide*.

Key-Sequenced Files

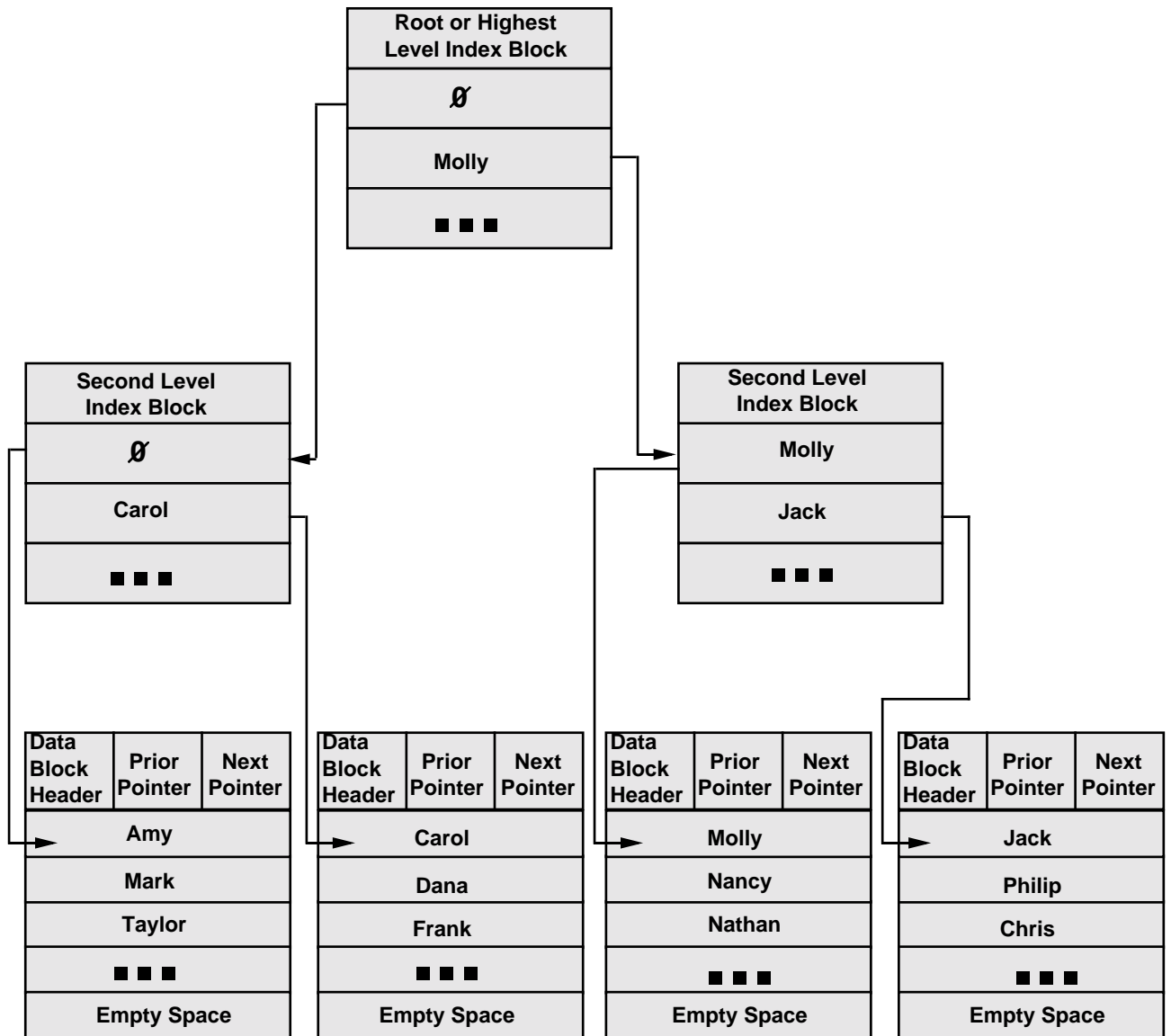
A key-sequenced file consists of a set of variable-length records. Each record in a key-sequenced file is uniquely identified by the value of its **primary key**. The primary key is a unique field in the record and cannot be changed when a record is updated. Records in the file are stored in ascending primary key order. In the database in Figure 1-1, the person's name is the primary key. The disk process keeps the order of the records intact by using a combination of different block types to maintain the file.

A key-sequenced file is physically organized as one or more bitmap blocks (Disk Process 2 files only) and a tree structure of index and data blocks. This tree structure is illustrated in Figure 1-1.

The bitmap blocks organize the free space of Disk Process 2 (DP2) structured files. The index blocks have pointers that point to where the data is stored, and the data blocks contain the records. In the first record in any index, the record's primary key is null. This means that any record before the primary key for the second record will be in the data block pointed to by the null record. In Figure 1-1, the records for Amy, Mark, and Taylor are in the first data block. The records for Carol, Dana, and Frank are in the second data

block. The last record in the index block that points to the first data block is null. The second record in the index block is Carol. That record points to the second data block.

Figure 1-1. Key-Sequenced File Structure



Each data block contains a header plus one or more data records, depending on the record size and data block size. For each data block, an index block entry contains the value of the primary-key field for the first record in the data block and the address of that data block.

The data blocks in Figure 1-1 have a certain percentage of empty space or **slack**. When you load data into a key-sequenced file, you can specify how much slack (empty space for future growth) is needed.

When a new record is inserted into a key-sequenced file, the position of the record is determined by the value of its primary-key field. If the block where the new record will

be inserted is full, the block splits. This means that the disk process creates a new data block, moves part of the data from the old block into the new block, and gives the index block a pointer to the new data block. Block splits can be costly because they consume CPU (central processing unit) cycles and increase the time needed to complete the input/output (I/O) requests.

Index blocks function much like data blocks in that they contain records. Those records are actually pointers to respective data blocks. These index blocks are the mechanism for key-sequenced data retrieval.

When an index block becomes full, it splits, a new index block is created, and some of the pointers are moved from the old index block to the new one. The first time this occurs in a file, the disk process must generate a new level of indexes. It does this by allocating a higher-level index block containing the low key and pointer to the two lower-level index blocks (which in turn point to many data blocks). The disk process must do this again each time the “root” (highest-level) block is split.

The DP2 disk process sometimes performs a three-way block split, creating two new blocks and distributing the original block's data or pointers (plus the new record or pointer) among all three.

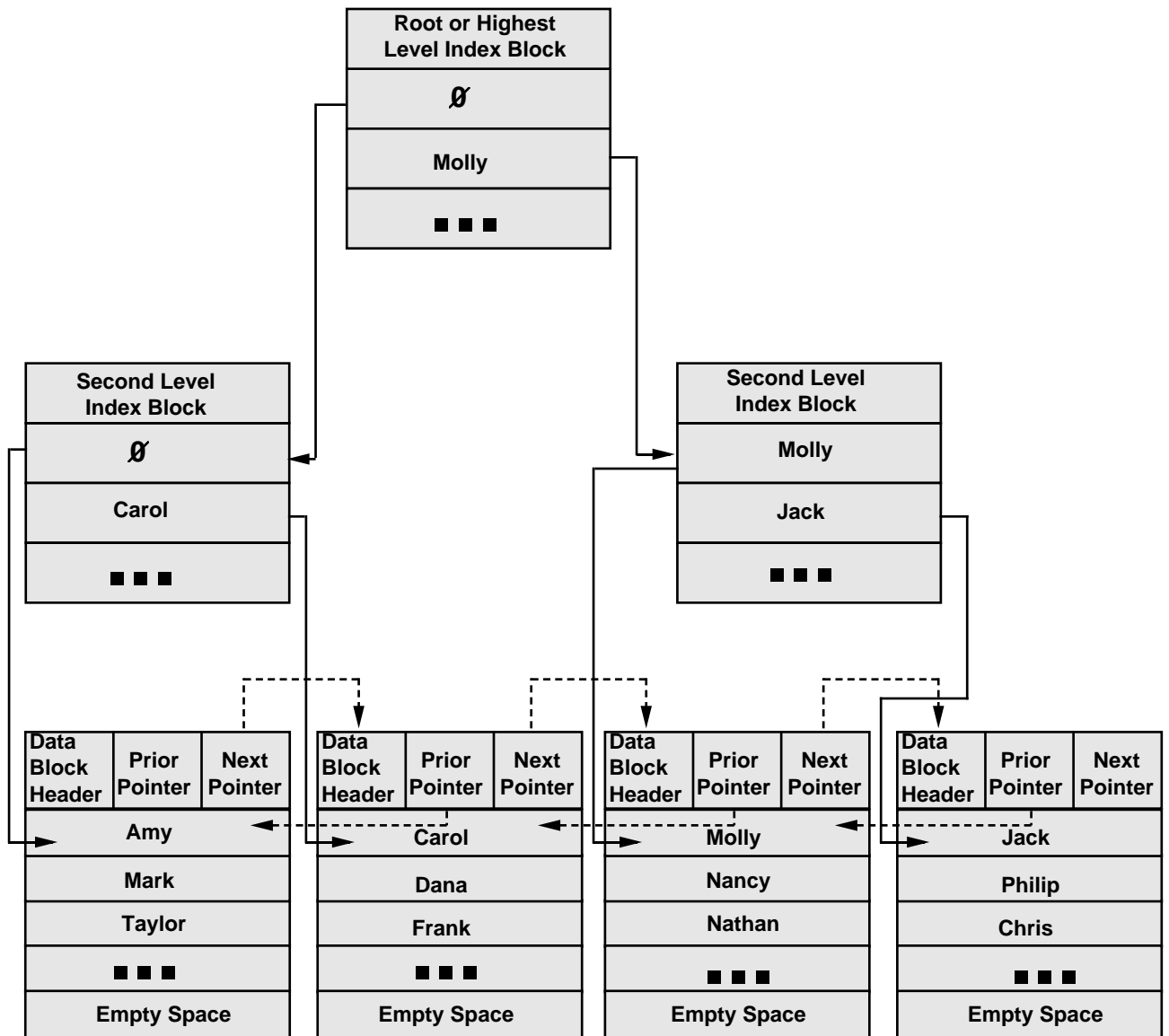
As online transactions and batch jobs take place, both index and data blocks can split. Eventually, this process causes the file to become fragmented or disorganized. The next section explains how this happens and shows the difference between an organized and disorganized file.

File Disorganization

Figure 1-2 shows an **organized file**. This file is organized because:

- The data blocks are physically contiguous on the disk and are arranged according to the primary record keys.
- The index and data blocks are uniformly filled with data and have little or no wasted disk space.
- The file has the fewest possible index levels.

Figure 1-2. An Organized File



The data blocks are both physically and logically contiguous on disk. The next pointer from each data block points to the adjacent data block. In Figure 1-2, the dashed lines with arrowheads point from the block to its adjacent block. The next and previous pointers are the only differences between Figure 1-2 and Figure 1-1.

Note that the records within the data and index blocks are in logical order by the primary record key. In this example, the first name is the primary record key. The first primary record key in the second data block comes logically after the last record in the first data block. The first primary record of the third block also follows the second block.

The index and data blocks have a uniform amount of empty space and number of records. Each data block shown has three records. The file also has the fewest possible index levels. This file has a root index level and only one index level below the root

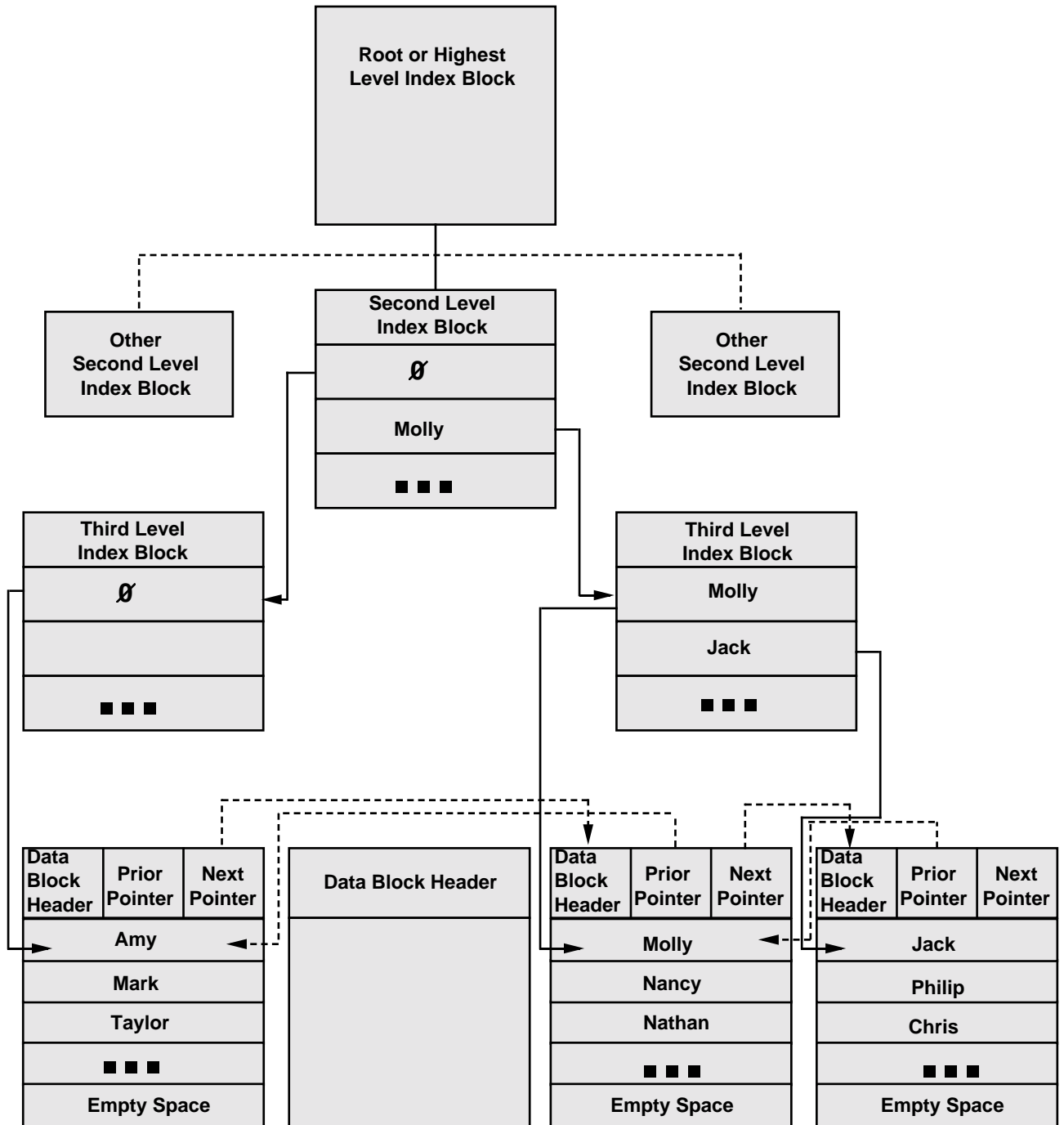
index. Looking at the disorganized file (Figure 1-3), you will see that the file has more than one index level below the root index.

A file does not have to be misused to become disorganized. Over the life of a file, these operations can cause it to become disorganized, namely:

- Random inserts that cause block splits
- Deletions of the last record in the data block that cause the block to collapse
- Updates with record length changes that cause the record to no longer fit in its current data block

In Figure 1-2, suppose all records in the data block starting with Carol's record are deleted, causing it to collapse. When the last record in that block is deleted, DP2 removes the block from the chain. That data block becomes a free or empty data block. Because the free data block is no longer in the chain, the pointer from the index block is deleted to that data block. Figure 1-3 shows what happens to the file. Note the empty or free block in the file.

Figure 1-3. A Disorganized File



The data blocks in Figure 1-3 are physically adjacent but are not logically contiguous. That is, the next pointer from the first data block does not point to its adjacent block, but points to a nonadjacent data block instead. The file lacks physical continuity and proper space utilization. Physical discontinuity occurs when the physical order of the data blocks does not match the blocks' logical order.

This file also suffers from improper space utilization. This occurs when the index and data blocks do not contain the optimal number of records. Notice that the amount of free space in the blocks is not consistent throughout the file. For example, the second block has no records. When the records from that block were deleted, it became empty.

Remember that a condition of an organized file is to have the fewest possible index levels. If you compare Figure 1-3 and Figure 1-2, you will notice that there is one more index level in Figure 1-3. (The dashed line above the second level index block points to the other index blocks on this index level.) The more index levels, the longer the disk access time and the disk space required.

When a file is disorganized, the data blocks are not contiguous or not physically adjacent. Disorganized files cannot take advantage of certain performance features, such as bulk reads for sequential prefetch or bulk writes for sequential updates in files. Consequently, sequential performance slows down. To improve application performance and disk space utilization, the file must be reorganized.

Determining when disk space is not properly utilized requires the right database management tool. Reload Analyzer gives you this information.

Data Chains

One key piece of information Reload Analyzer gives about a file is the number of contiguous data blocks. The number of contiguous data blocks helps you determine whether a file needs to be reorganized.

Reload Analyzer counts the data blocks in contiguous order and reports the number of data chains. A **data chain** is a set of physically and logically contiguous data blocks in a file.

You can determine whether a file is disorganized by how many data chains a file has and how long each chain is. An organized file will have one long data chain. If a file has a few long chains, it is better organized than a file with many short chains. The number of data chains and the length of each chain are key factors in deciding whether to reorganize a file. Section 2, “Using Tandem Reload Analyzer,” explains how the program gives this information.

Summary

You now know what a key-sequenced file is, the difference between an organized and disorganized file, and what a data chain is. From here, turn to Section 2, “Using Tandem Reload Analyzer,” to learn how to run the program and analyze the output.

2

Using Tandem Reload Analyzer

This section explains what you need to install Reload Analyzer, run it, and analyze the output. Once you understand the output, you can decide whether reorganizing a file is necessary.

System Requirements

Reload Analyzer requires the Tandem operating system, version D20 or later, including the following standard system software:

- File Utility Program (FUP)
- Tandem Advanced Command Language (TACL)
- Peruse

Installing Reload Analyzer

Reload Analyzer is installed from the Site Update Tape (SUT).

During the normal installation process, the REPSUBSYS phase moves Reload Analyzer to the installation subvolume (ISV), named ZTRA. The object module, TRA, can then be moved using the FUP DUP command to the subvolume from where it will execute. This subvolume is usually \$system.system.

Running Reload Analyzer

To run Reload Analyzer, enter the following syntax at the TACL prompt:

```
l> tra / IN infile [,OUT reptfile] / [fupin][:THRESHOLD  
value]
```

IN *infile*

names the file being analyzed. The file can be a single file or an EDIT file (type 101) that contains a list of files to be analyzed. These files must meet the Tandem file naming conventions.

If you have a partitioned file, you must analyze each partition separately. Reload Analyzer analyzes only the partition you specify. Therefore, you must run Reload Analyzer against each partition. The easiest way to run Reload Analyzer against multiple partitions is to create an EDIT file with the file names of all the partitions you want to analyze. Reload Analyzer will provide output for each partition.

Since Reload Analyzer opens a file for unstructured read-only access, it will not damage or interfere with a file, for example, by changing the file or record locks or damaging the file's contents. In addition, Reload Analyzer does not interfere with other operations on the system. Although Reload Analyzer does not interfere with operations, it does generate I/O; so you might want to run it during off hours.

OUT *reptfile*

is the designated output device for displaying the output. It may be a spooler, disk file, or process. If the disk file already exists, Reload Analyzer appends the new data to the file. If the specified disk file does not exist, Reload Analyzer creates one. If the specified spooler or process does not exist, Reload Analyzer abends. If you do not enter anything, the output is sent to the home terminal.

fupin

specifies the name of the FUP obey file to be created. *fupin* must be a valid Tandem file name. FUPIN is the default file name. Specify this option to automatically create a FUP obey file containing the RELOAD commands for files with a fragmentation ratio less than the threshold value. If both *fupin* and THRESHOLD *value* are omitted from the Run command, the FUP obey file is not created.

THRESHOLD *value*

specifies the acceptable fragmentation ratio of the file being inspected. *value* must be an integer between 1 and 99, inclusive. The default is 20%. Specify this option to automatically create a FUP obey file containing the RELOAD commands for files with a fragmentation ratio less than the threshold value. If both *fupin* and THRESHOLD *value* are omitted from the Run command, the FUP obey file is not created.

Note. You can use all TACL command-line options except NOWAIT with Reload Analyzer. The NOWAIT option will interfere with the status indicator when the output is directed to the home terminal. If you use the NOWAIT option with Reload Analyzer and direct the output from another program to the home terminal, that program's output and the status indicator will be interspersed on the home terminal. This makes it hard to distinguish the output. For more information about the TACL command-line options, see the *Tandem Advanced Command Language (TACL) Reference Manual*.

Reloading the Database

When you specify *fupin*, THRESHOLD *value*, or both in the Run command, Reload Analyzer automatically creates a file FUP obey file containing the FUP RELOAD commands for files with a fragmentation ratio less than the specified or default threshold value. The fragmentation ratio is the percentage of total blocks in the average chain, which is found by dividing the average number of blocks in a chain by the total number of blocks in the file.

When Reload Analyzer is finished running, enter RUN FUP/IN *fupin*/ to reload the database. Example 2-1 shows a sample file containing FUP RELOAD commands.

Example 2-1. Sample FUP RELOAD Command File

```
-- Tandem Reload Analyzer - T9516D40 (03JUN96) Start time - 18:36:48

-- This file contains the FUP RELOAD commands for the files that meet
-- the criteria you specified. Use this information and your knowledge
-- of your systems and applications to help decide whether a file
-- should be reloaded. Remember that an online reload causes performance
-- degradation and creates a large amount of audit trail information for
-- TMF-audited files or tables.
--
-- Review the FUP RELOAD commands for accuracy before running them.
--
-- TRA used the THRESHOLD value of 25 while generating this file.

-- \scube.$data.insurnc.salesdb
-- Total blocks read          303
-- Number of chains           203
-- Avg blocks/chain           1
-- % of total blks in avg chain < 1%
RELOAD \scube.$data.insurnc.salesdb, RATE 40
```

Interrupting Reload Analyzer

To interrupt the Reload Analyzer process, press the Break key. Pressing Break returns control to the command interpreter. However, the process continues to execute in the background. Interrupting the process is safe and will not damage the file being analyzed.

If you execute another program before Reload Analyzer finishes, Reload Analyzer and the other program will both try to write to the home terminal. Because both programs are writing output to the home terminal, you can have trouble distinguishing the Reload Analyzer's output from the other program's output.

Note. If the file is empty and you press Break, the process will not be aborted.

Status Indicator

The status indicator, a period (.), tells you that Reload Analyzer is working. The status indicator is displayed on the home terminal when you direct the output to the home terminal. For each 1000 blocks that Reload Analyzer reads, another period is displayed, with ten periods per line.

Understanding the Output

This subsection describes the output information Reload Analyzer gives about the file being analyzed. This information cannot tell you exactly when a file needs to be reorganized, but it can help you draw conclusions about whether to reorganize a file. In this section, you will learn about each part of the output and see how to draw conclusions from the output.

When Reload Analyzer analyzes a file, two things can happen. If the file is not found or a security error happens, Reload Analyzer displays a banner line with the starting time stamp, the reason the process could not continue, and an error message. (See Appendix A, “Error Messages,” for more information.) If no error occurs, the process starting time stamp is shown. After that, Reload Analyzer displays three sections of output followed by the completion time stamp. This subsection shows a sample of the output and explains each part.

Suppose you analyze the key-sequenced Enscribe file SALESDB. The following syntax analyzes the file SALESDB and sends the output to the spooler:

```
3> tra / IN SALESDB, OUT $spooler.loc /
```

Example 2-2 shows the Reload Analyzer output for SALESDB. At the top is the process starting time and the name of the file that was analyzed. The completion time is at the end of the output. How long it takes to read the file is determined by the size of the file and your system configuration. To calculate how long it took to analyze the file, find the difference between the output starting and ending time.

The use of bulk I/O greatly enhances the performance of the Reload Analyzer. If you are familiar with the FUP INFO *filename*, STAT command, you will notice the performance improvements with Reload Analyzer.

The output is grouped into three sections:

- General Information
- Block Information
- Data Chain Information

Example 2-2. Sample Reload Analyzer Output

Tandem Reload Analyzer (07JUN92) Start time - 16:47.35

\$DATA.INSURNC.SALESDB

General Information:

Enscribe	Block 512
Type K	IBlock 512
Code 300	Rec 64
Ext 17, 32	Extents Allocated 3
Maxextents 16	Buffered
Owner 20, 33	Audited
EOF 155136	Security (RWEF) 0000

Block Information:

Total blocks read	303	Free blocks	0
Index blocks read	23	Bitmap blocks	1
Data blocks read	279	Unrecognizable	0

Level	Total Blocks	Total Recs	Avg # Recs	Min Recs	Max Recs	Avg Slack	Avg% Slack
3	1	2	2	2	2	444	86
2	2	20	10	9	11	216	42
1	20	279	13	6	30	183	35
DATA	279	1572	5	2	8	164	32

Data Chain Information:

Number of chains 203
 Longest chain 15
 Chain length histogram

1	191
2 - 10	9
11 - 25	3
26 - 50	0
51 - 100	0
> 100	0

Avg blocks/chain 1
 % of total blks in avg chain < 1%

completed at - 16:49.48

Reload Analyzer is stopping

General Information

The General Information section shows detailed information on the file characteristics. The information is derived from the FUP INFO *filename*, DETAIL command format for NonStop SQL objects and key-sequenced Enscribe files. Example 2-3 shows the format of the General Information section.

Example 2-3. General Information Format

Object type	Block <i>block length</i>
Type <i>file type</i>	IBlock <i>length</i>
Code <i>file code</i>	Rec <i>record length</i>
Ext <i>pri-num</i> PAGES, <i>sec-num</i> PAGES	Extents Allocated: <i>num-ext</i>
Maxextents <i>maextents</i>	Buffered
Owner <i>group-id,owner-id</i>	Audited
EOF <i>eof</i>	Security (RWE) : <i>rwep</i>

Headers and variables that appear this output section include:

<i>object type</i>	indicates whether the file is a NonStop SQL table, NonStop SQL index, or Enscribe file.
<i>Block</i>	indicates the length of a block.
<i>Type</i>	indicates the file organization. The only identifier is K for key-sequenced because Reload Analyzer only runs against key-sequenced files.
<i>IBlock</i>	is the length of an index block for an Enscribe file.
<i>Code</i>	is the file code. The default file code of 0 is not displayed. File codes 100 through 999 refer to specific types of files and are reserved by the system. For information about these codes, see the <i>File Utility Program (FUP) Reference Manual</i> .
<i>Rec</i>	indicates the maximum exploded record length for the file.
<i>Ext, PAGES</i>	lists the sizes of the primary (<i>pri-num</i>) and secondary (<i>sec-num</i>) extents that can be allocated.
<i>Extents Allocated</i>	is the number of extents currently allocated for the file.
<i>Maxextents</i>	lists the maximum number of extents set for the file.
<i>Buffered</i>	writes to the file are buffered.
<i>Owner</i>	is the user ID of the file's owner, given as: <i>group-num, user-num</i>
<i>Audited</i>	shows whether or not the file is audited.
<i>EOF</i>	is the end-of-file pointer containing the relative byte address of the byte following the last significant data byte.

security is the security assigned to the file given as:

r security level for read
w security level for write
e security level for execute
p security level for purge

Values for *rwep* are:

* SAFEGUARD protected
 - local super ID only
 O owner only (local)
 G member of owner's group (local)
 A any user (local)
 U member of owner's user class—that is, owner only (local or remote)
 C member of owner's community—that is, member of owner's group (local or remote)
 N any user (local or remote)

Block Information

The Block Information section reports the number of index and data blocks, the data blocks read, any free or bitmap blocks, and any unrecognizable blocks.

Following these block counts, Reload Analyzer provides a table similar to the output from a FUP INFO *filename*, STAT command, except for two additional values:

- Minimum records per block at this level
- Maximum records per block at this level

Example 2-4 shows the format of the Block Information section.

Example 2-4. Block Information Format

	Total blocks read	<i>num</i>		Free blocks	<i>num</i>		
	Index blocks read	<i>num</i>		Bitmap blocks	<i>num</i>		
	Data blocks read	<i>num</i>		Unrecognizable	<i>num</i>		
Level	Total	Total	Avg #	Min	Max	Avg	Avg%
<i>num</i>	Blocks	Recs	Recs	Recs	Recs	Slack	Slack
	<i>num</i>	<i>num</i>	<i>num</i>	<i>num</i>	<i>num</i>	<i>num</i>	<i>num</i>

The variables that appear in Example 2-4 are:

<i>Total blocks read</i>	is the total of index blocks, data blocks, free blocks, bitmap blocks, and unrecognizable blocks.
<i>Free blocks</i>	is the total unused blocks. A file with 0 free blocks is unusual.
<i>Index blocks read</i>	is the total index blocks.
<i>Bitmap blocks</i>	is the total bitmap blocks. Bitmap blocks are used to keep track of block usage in a file.
<i>Data blocks read</i>	is the total data blocks.
<i>Unrecognizable</i>	is the total unrecognizable blocks. This number should be 0. If this value is not 0, the unrecognizable block header has structural damage. Run the FILCHECK utility to determine which block header is damaged. If the block header is damaged or you are not familiar with FILCHECK, contact your Tandem representative for assistance.
<i>Level</i>	indicates the tree level of the entry. Values for <i>level</i> are: DATA indicates that each entry in this row is for the data level. A one (1) or greater indicates that the entry is for an index level. One (1) is the lowest index level.
<i>Total Blocks</i>	is the total number of blocks in use at the indicated level.
<i>Total Recs</i>	is the total number of records at the indicated level. At the DATA level, <i>t-recs</i> is the total number of data records in the file.
<i>Avg # Recs</i>	is the average number of records for each block at the indicated level.
<i>Min Recs</i>	is the minimum number of records for each block at the indicated level.
<i>Max Recs</i>	is the maximum number of records for each block at the indicated level.
<i>Avg Slack</i>	is the average number of unused bytes for each block at the indicated level.
<i>Avg % Slack</i>	is the average percentage of unused bytes for each block at the indicated level.

Data Chain Information

The Data Chain Information section is the most significant. This section shows the number of data chains, a histogram of the chains, and the average blocks in each chain. A data chain is the number of contiguous data blocks.

Example 2-5 shows the format of the Data Chain Information section.

Example 2-5. Data Chain Information Format

```
Number of chains num
Longest chain num
Chain length histogram num
  Histogram ranges num
Average blocks per chain num
% of total blocks in average chain num
```

Headers and variables that appear in Example 2-5 are:

<i>Number of chains</i>	is the total number of data chains.
<i>Longest chain</i>	is the number of data blocks in the longest chain. An organized file will have one chain with a chain length equal to the number of data blocks.
<i>Chain length histogram</i>	shows the number of data chains and the length of those chains. For example, the file SALESDB has 191 chains 1 block long. Nine chains have 2 to 10 data blocks in each chain, and 3 chains have 11 to 25 blocks in each chain. The histogram has six different ranges that show the length of a chain.
<i>Histogram ranges</i>	Reload Analyzer uses different histogram ranges, depending on the size of the file you are analyzing. Table 2-1 shows the value ranges used in a chain length histogram. If the file is less than 100 MB, Reload Analyzer displays the data chains using the ranges on the left side of Table 2-1. The right side of Table 2-1 is the histogram range for the files greater than 100 MB. For example, with a file greater than 100 MB, Reload Analyzer shows the number of chains that are 1 to 25 blocks long, the number of chains that are 25 to 50 blocks long, the number of chains that are 51 to 100 blocks long, and so forth.
<i>Average blocks per chain</i>	is the average number of data blocks in each chain. To calculate the average blocks per chain, the total data blocks are divided by the total number of data chains.
<i>% of total blocks in average chain</i>	is the percentage of total blocks in the average chain. In an organized file, the file will have one long chain. This number is calculated by taking the average blocks in a chain and dividing it by the total data blocks.

Table 2-1. Values in Chain Length Histogram

File < 100 MB	File > 100 MB
1	1 - 25
2 - 10	26 - 50
11 - 25	51 - 100
26 - 50	101 - 250
51 - 100	251 - 500
> 100	> 500

Counting Data Chains

A data chain is a set of logically contiguous data blocks. Reload Analyzer counts the data chains in a file by:

- Opening a file for unstructured read-only access using bulk I/O
- Counting data and index block types
- Analyzing the length of the data block chains as well as the frequency of the chaining

Understanding Data Chain Information

The length and the number of chains are the most important factors when deciding whether to reload a file. One long chain in a file is better than many short chains. If the file has many small chains, the file probably should be reloaded.

The following example shows the Data Chain Information sections of two files. This first example is the Data Chain Information section of File A.

Data Chain Information:			
Number of chains			100
Longest chain			2
Chain length histogram			
1			0
2 - 10		100	
11 - 25			0
26 - 50			0
51 - 100			0
> 100			0
Avg blocks/chain			2
% of total blks in avg chain			1%

File A has one hundred chains, each 2 blocks long. The average number of data blocks per chain is 2, which is 200 total blocks divided by the number of chains, 100. The percentage of total blocks in the average chain, the last value in the Data Chain Information section, is 1 percent. That percentage comes from dividing the average blocks per chain by the total data blocks ($2/200=1\%$).

Now look at the example for File B.

Data Chain Information:			
Number of chains			2
Longest chain			100
Chain length histogram			
	1		0
	2 -10		0
	11 -25		0
	26 - 50		0
	51 - 100		2
	> 100		0
Avg blocks/chain			100
% of total blks in avg chain			50%

File B has two chains, each 100 blocks long. In File B, the average blocks per chain is 100, which is the total data blocks divided by the number of chains (200/2). The percentage of total blocks in the average chain is 50 percent, which is the average blocks per chain divided by the total data blocks (100/200=50%).

An organized file has more long chains than short ones. If a file has just one long chain, the percentage of total blocks in the average chain will be 100 percent. If a file has one long chain or more long chains than short ones, reloading the file is probably not necessary. Apply this principle to files A and B.

File A has many short chains, and the percentage of total blocks in the average chain is 1 percent. In File B, the percentage of total blocks in an average chain is 50 percent. When the percentage of total blocks in the average chain is low (it was 1 percent in File A), the file is disorganized. The closer the percentage of total blocks in the average chain is to 100 percent, the more organized the file. That is true for File B.

When you analyze a file, you should take a copy of the output and go through it like we just did with files A and B. Pay close attention to two parts of the output:

- The average blocks per chain
- The percentage of total blocks in the average chain

Reload Analyzer cannot tell you whether or not to reload a file. Instead, you must take the output, analyze it, and consider the conditions on your system to determine whether it is advantageous to reload a file. Reload Analyzer only gives you only the information you need to make an informed decision about reloading. Whether you reload a file and when you do so is up to you.

After you reload a file, you should see how the file has changed. Run Reload Analyzer against the reloaded file to see how the output has changed.

Output After Reloading

Example 2-6 shows the Reload Analyzer output for SALESDB after the file was reloaded. Note that the values, especially in the Data Chain Information section, have changed. This section compares this output with the output before the file was reloaded.

Example 2-6. Output from SALESDB After Reloading

```
Tandem Reload Analyzer (07JUN92)      Start time - 17:02.35
$DATA.INSURNC.SALESDB1

General Information:

    Enscribe                               Block 512                <-- Unchanged
    Type K                                 IBlock 512
    Code 300                               Rec 64
    Ext 17, 32                             Extents Allocated 3
    Maxextents 16                           Buffered
    Owner 20, 33                            Audited
    EOF 106496                              Security (RWE) 0000

Block Information:

    Total blocks read      208              Free blocks              0
    Index blocks read     10                Bitmap blocks           1
    Data blocks read      197              Unrecognizable         0

Level      Total      Total      Avg #      Min      Max      Avg      Avg
           Blocks    Recs      Recs      Recs    Recs    Slack   Slack
  2         1         9         9         9       9       239    46
  1         9        197        21        14      34       29     5
  DATA    197      1572        7         4       8       33     6

Data Chain Information:

Number of chains          1                <-- Previously 203
Longest chain            197              <-- Previously 15
Chain length histogram
   1                      0
  2 - 10                  0
 11 - 25                  0
 26 - 50                  0
 51 - 100                 0
  > 100                  1
Avg blocks/chain          197                <-- Previously 1
% of total blks in avg chain < 100%      <-- Previously 1%
```

completed at - 17:05.48

Reload Analyzer is stopping.

The values in the General Information section have not changed. In the Block Information section, the number of index blocks and data blocks has decreased significantly. The file now has 197 data blocks compared to 279 before and 10 index blocks compared to 23 before. The file had three index levels before the reload and now has two index levels.

Look at the Data Chain Information section. One chain is 197 data blocks long. An organized file like this one will have one long chain. The average blocks per chain is 197

because the file has only one chain. The percentage of total blocks in the average chain is 100 percent. This percentage is calculated by taking average blocks per chain divided by the total data blocks. (The average blocks per chain is 197 divided by 197 total data blocks equals 100 percent.) A percentage of 100 signifies an organized file.

Before you reload a file, consider the following. The file in Example 2-6, SALESDB, is a small file, and reloading it probably will not affect the system. This might not be true with a large file. An online FUP RELOAD can take time, degrade OLTP performance, and generate large quantities of audit. Before reloading a file, you should consider the impact on the system. For more information on how to reload, see the Reload Analyzer support note.

When to Reload

Reload Analyzer provides information to help you decide whether to reload a file, but it cannot determine whether a file should be reloaded because there are no universally accepted criteria for this decision. The criteria you need to weigh include the cost of application performance degradation versus the cost of reloading and the technical confidence level of the person reloading the file.

The most common reason for considering a reload is degradation of application performance. For example, if performance has gradually worsened for an application that sequentially reads a frequently updated file, reloading the file is likely to help. In other cases, it might be less clear whether file disorganization is affecting your application performance.

You need to weigh the benefits of reloading the file against the costs of reloading. The cost of reloading a file is significant and includes the following:

- Performance degradation of processes that are using the file while the file is being reloaded
- Rapid generation of large amounts of Transaction Management Facility (TMF) audit data

The extreme cases of good and bad file organization are easy to identify:

- If the average chain contains 100 percent of the blocks, the file's organization is ideal.
- If the average chain contains 1 per cent of the blocks, the file is maximally disorganized.

Between these extremes exists a large gray area. If a file's average chain contains 20 percent of the blocks and the application performance has degraded moderately in the last month, it is difficult to state that the file definitely should be reloaded. You should use Reload Analyzer's output to help you decide whether the expected performance improvement (probably moderate in this case) is worth the cost in TMF audit generation and performance degradation during the reload.

A Error Messages

This appendix gives each Tandem Reload Analyzer (Reload Analyzer) error message, what caused the error, and how to recover from it.

```
filename is not an edit file.
```

Cause. An Edit file name was not entered when you were running Reload Analyzer.

Effect. Reload Analyzer abends.

Recovery. Make sure that the file name you enter is an Edit file. Reload Analyzer can tell if the file you entered is an Edit file name or single file name.

```
filename Edit file Error: nnn.
```

Cause. A file-system error occurred while Reload Analyzer was trying to analyze an Edit file.

Effect. Reload Analyzer abends.

Recovery. Refer to the *Guardian Procedure Errors and Messages Manual* for information about the error number indicated by *nnn*.

```
filename failed on Edit file Open, error nnn.
```

Cause. A file-system error occurred when Reload Analyzer tried to open an Edit file.

Effect. Reload Analyzer abends.

Recovery. Refer to the *Guardian Procedure Errors and Messages Manual* for information about the error number indicated by *nnn*.

```
filename failed on Open, error nnn.
```

Cause. A file-system error was found in the input file name being analyzed.

Effect. Reload Analyzer skips the specified file and continues to run.

Recovery. Refer to the *Guardian Procedure Errors and Messages Manual* for information about the error number indicated by *nnn*.

```
filename is not a key-sequenced file or SQL table.
```

Cause. The input file is not a key-sequenced file or NonStop SQL table.

Effect. Reload Analyzer does not start.

Recovery. Enter a key-sequenced file or NonStop SQL table as input. Tandem Reload Analyzer only analyzes key-sequenced files or NonStop SQL tables.

```
outfile failed on Open, error nnn.
```

Cause. A file-system error occurred when Reload Analyzer tried to write to the designated output file.

Effect. Reload Analyzer does not start.

Recovery. Refer to the *Guardian Procedure Errors and Messages Manual* for information about the error number indicated by *nnn*.

```
filename does not exist, error nnn.
```

Cause. The specified file does not exist in the list of files for batch processing.

Effect. Reload Analyzer skips the specified file and continues to run.

Recovery. Specify a file that exists in the file list.

```
filename security violation, error nnn.
```

Cause. The specified file is not secured for read access.

Effect. Reload Analyzer skips the specified file and continues to run.

Recovery. Have the file resecured for read access.

```
fupin filename is invalid.
```

Cause. The *fupin* file name specified in the command line is an invalid Guardian file name.

Effect. Reload Analyzer does not start.

Recovery. Specify a valid Guardian file name as the *fupin* file name in the command line. A valid Guardian file name has the form `\system.$volume.subvolume.filename`.

Invalid invocation syntax.

Cause. The Reload Analyzer Run command was entered incorrectly.

Effect. Reload Analyzer displays the online help for the Run command and stops.

Recovery. Start Reload Analyzer using the correct syntax. See the online help or manual for more information about the command line syntax.

infile must be an existing disk file.

Cause. The specified input file is not an existing disk file.

Effect. Reload Analyzer does not start.

Recovery. Specify an existing key-sequenced file or Edit file as the input file.

fupin file must be a disk file.

Cause. The specified *fupin* file is not a disk file.

Effect. Reload Analyzer does not start.

Recovery. Specify an existing Guardian disk file or the name of the file you would like Reload Analyzer to create. The file name must have the form *\system.\$volume.subvolume.filename*. If you do not specify a name, Reload Analyzer will, by default, place a file named FUPIN in the system, volume, and subvolume where Reload Analyzer is running. If both *fupin* and the THRESHOLD option are omitted from the Run command, the *fupin* file is not created.

Glossary

data chain. A set of physically and logically consecutive data blocks in a file or table.

disorganized file. A file or table where the physical order of the data blocks does not match their logical order, and the index and data blocks do not contain the optimal number of records.

Disk Process 2 (DP2). The disk process used by NonStop SQL.

Enscribe. A Tandem software facility that provides high-level access to, and manipulation of, records in a relational database.

File Utility Program (FUP). A Tandem utility that lets you create, purge, and manage disk files.

Guardian 90 operating system. The operating system that is a standard feature of the Tandem NonStop computer systems.

index block. A table of references kept in some sequence and accessed to obtain the addresses of data items.

Installation Subvolume (ISV). The subvolume on which this version of the Tandem Reload Analyzer is stored.

key-sequenced file. A file that consists of a set of variable length records.

NonStop SQL. A relational database management system that promotes efficient online access to large distributed databases.

object. A database entity that is created, manipulated, or dropped by the NonStop SQL statements and that is described in an SQL catalog.

Online Transaction Processing (OLTP). Data processing that involves immediate access and updates to a database while the user waits for the results. The database is always up to date.

organized file. The data blocks in the file or table are in contiguous order, and the index and data blocks have little or no wasted disk space.

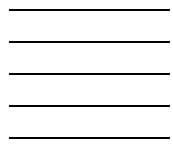
primary key. A unique field in a record in a key-sequenced file.

RELOAD command. A FUP command that physically reorganizes a key-sequenced file or SQL object (table or index only) online while allowing shared read/write access to the file or object.

slack. The amount of free space in data or index blocks.

spooler. A set of programs that acts as an interface between users (and user applications) and the print devices of a system.

Transaction Monitoring Facility (TMF). A database protection subsystem incorporated into the Guardian 90 operating system. TMF helps maintain the consistency and integrity of a database that is updated by concurrent transactions and protects data against damage caused by system media failures.



Index

B

Bitmap blocks 1-1, 2-8

Block

length 2-6

slack 2-8

Block Information section 2-7, 2-12

Blocks

bitmap 1-1, 2-8

data 2-8

data, definition 1-3

index, definition 1-3, Glossary-1

read 2-8

slack Glossary-1

splits 1-3

Break key 2-3

C

C30 Disk Process (DP2) 1-7

Code, file 2-6

D

Data blocks 2-8

See Blocks 1-3

Data Chain Information section 2-9, 2-12

analysis 2-10

Data chains 2-9

definition 1-7

examples 2-10

DETAIL, FUP INFO 2-6

Device, output 2-2

Disk Process 2 (DP2) Glossary-1

E

Extents 2-6

F

File

code 2-6

disorganized Glossary-1

definition 1-5

identifying 1-7

Edit, analyzing 2-1

extents 2-6

key-sequenced 2-1, Glossary-1

organized Glossary-1

definition 1-3

identifying 1-4

owner 2-6

partitioned 2-1

reading 2-9

reloading 2-12

type 2-6

FUP

DETAIL command 2-6

RELOAD 2-13

RELOAD command 2-11

STATISTICS command 2-7

I

Index block

count 2-8

length 2-6

Installation 2-1

K

Key-sequenced file 1-1, 2-1, Glossary-1

M

Maximum records 2-8

Minimum records 2-8

O

Object type 2-6

Output 2-2, 2-3

- Block Information section 2-7

- Data Chain Information section 2-9

- example 2-4, 2-12

- General Information section 2-6

- reload 2-12

Owner, file 2-6

P

Period

- See Status indicator 2-3

Primary key 1-1, 1-3, 1-4

R

Records

- count 2-8

- length 2-6

- order by primary key 1-4

Reload Analyzer

- install 2-1

- interrupt 2-3

- output 2-3

- run 2-1

S

Slack space 2-8

STATISTICS, FUP command 2-7

Status indicator 2-3

System requirements 2-1