

Adabas

Concepts and Facilities

Manual Order Number: ADA741-006IBB

This document applies to Adabas Version 7.4 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the address on the back cover or to the following e-mail address:

Documentation@softwareag.com

© December 2002, Software AG

All rights reserved

Printed in the Federal Republic of Germany

Software AG and/or all Software AG products are either trademarks or registered trademarks of Software AG. Other products and company names mentioned herein may be the trademarks of their respective owners.

TABLE OF CONTENTS

ABOUT THIS MANUAL	1
1. ADABAS IS	3
Operational Highlights	3
High Availability	3
Storage Space Optimization	3
Performance	3
Fault Tolerance	3
Operating Environment	4
Supported Data Models	4
Operating Structure	6
Nucleus, I/O Buffer, and Threads	6
Data Storage, Associator, and Work	7
Utilities, User Programs, and TP Monitors	8
Executing Adabas	10
Session Types	10
Storage Areas	10
Modes of Operation	10
ADARUN Start-up Parameters	12
Session Control	12
2. ADABAS DESIGN	13
Adabas Entities	13
Adabas Limits	13
Adabas Space Management	14
Database Components	14
Data Storage	14
Associator	18
Work	21
Other Components	21

Adabas Concepts and Facilities

Database Files	22
System Files	22
Coupled Files	22
Structuring Files to Enhance Performance	23
Records and Field Definitions	26
Record Structure	27
Field Levels	27
Field Names	28
Field Length and Data Format	28
Field Options	29
Special Field and Descriptor Attributes	33
3. USING ADABAS	37
Accessing a Database from Programs	37
Direct Call Interface	37
Complex Searches	40
Access Methods	41
Using Triggers and Stored Procedures	45
Universal Encoding Support (UES)	46
Maintaining Database Integrity	47
Transaction Logic	47
Competitive Updating	48
Timeout Controls	50
Backout, Recovery, and Restart	51
Extended Error Handling and Message Buffering	53
4. ADABAS UTILITIES	55
Initial Design and Load Operations	55
ADACMP : Compress / Decompress	55
ADALOD : Loader	57
ADAULD : Unload	58

Table of Contents

Backup / Restore / Recovery Routines	59
ADAPLP : Protection Log / Work Print	59
ADARAI : Recovery Aid	59
ADARES : Restart	60
ADASAV : Save / Restore Database or Files	61
ADASEL : Select Protection Data	62
Database Modification Routines	62
ADACDC : Changed-Data Capture	62
ADACNV : Database Conversion	63
ADADBS : Database Services	63
ADADEF : Define a Database	67
ADAFRM : Format Datasets	67
ADAINV : Invert	68
ADAORD : Reorder	69
ADAZAP : Modify Physical Database Blocks	70
Audit / Control / Tuning Procedures	70
ADAACK : Check Address Converter	70
ADADCK : Check Data Storage	70
ADAICK : Check Index and Address Converter	71
ADAMER : ADAM Estimation	71
ADAREP : Report	72
ADAVAl : Validate the Database	73
ADAPRI : Print Selected Adabas Blocks	73
5. ADABAS SECURITY	75
Data Encryption	75
Multiclient Files	76
Adabas Security and ADASCR	76
Access/Update Level Protection	76
Value Level Protection	77
Adabas Interface to SAF-based Packages	77
Adabas SAF Security (ADASAF)	78

Adabas Concepts and Facilities

Related Security Options	81
Adabas Online System Security	81
Natural Security	81
Using the SAF Repository to Secure Software AG Products	81
Entire Security SAF Gateway	81
Entire Net-Work SAF Security (NETSAF)	83
6. OPTIONAL EXTENSIONS	85
Adabas Online System	85
Adabas Caching Facility	87
Adabas Delta Save Facility	88
Adabas Fastpath	89
Adabas Vista	90
Adabas Transaction Manager	92
Adabas Review	92
The Hub Server	93
The Interface Client	94
Interface Calls	94
Example Client/Server Environment	95
Adabas Statistics Facility	96
Data Collection Program	96
Data Evaluation Programs	97
Adabas Parallel Services	99
Adabas Cluster Services	100
Cluster Services With Other Adabas Products	100
Advantages of Using Entire Net-Work	101
Adabas Text Retrieval	103
Adabas Bridges	104
Adabas Bridge for VSAM	104
Adabas Bridge for DL/I (and IMS/DB)	107
Entire Transaction Propagator	110

Table of Contents

Entire Net-Work Multisystem Processing Tool 111
Adabas Native SQL 114
Adabas SQL Server 115
Natural Application Development Environment 117
Predict Data Dictionary System 119

GLOSSARY OF TERMS 121

INDEX 131

ABOUT THIS MANUAL

This manual provides a technical introduction to the principles and functions of Adabas, Software AG's adaptable database management system. It provides an overview of Adabas for those who require a basic understanding of Adabas operating environments, design, use, and optional extensions.

Unit	Content
Chapter 1	presents a system overview and discusses the supported operating environments.
Chapter 2	describes the structures used by Adabas and their functions and interactions.
Chapter 3	talks about accessing the database and maintaining its integrity.
Chapter 4	briefly describes the Adabas utilities.
Chapter 5	describes the security features available with Adabas.
Chapter 6	briefly describes the optional extensions available from Software AG to add functionality to the Adabas core product.
Appendix A	is a glossary of the Adabas terms used in this manual.

ADABAS IS . . .

Adabas, the adaptable database, is a high-performance, multithreaded, database management system for mainframe platforms where database performance is a critical factor. It is interoperable, scalable, and portable across multiple, heterogeneous platforms including mainframe, midrange, and PC.

Operational Highlights

High Availability

Adabas is designed for operation 7 days a week and 24 hours a day. Space is managed dynamically (see page 14), files can be loaded and unloaded, backed up and restored, and system performance can be analyzed without interrupting the active database.

Storage Space Optimization

Adabas stores data in compressed form to reduce space requirements. Since modern databases are measured in gigabytes (1000 megabytes) or even terabytes (1000 gigabytes), the savings in disk space can be considerable. Reduced space requirements also mean that the input/output (I/O) system is more efficient.

Performance

Performance is the key factor of Adabas, which includes a number of features to enhance it. For instance, a number of set-up parameters are available for fine-tuning the database operating environment, and many of these can be modified while the database is active.

Fault Tolerance

Adabas recovers automatically after an abnormal database or system termination. Each time an Adabas database is started, an automatic check is initiated to determine whether the database previously terminated “cleanly” or an active transaction was interrupted. If a transaction was interrupted, Adabas automatically resets all changes of the uncompleted transaction so that the database is consistent.

Operating Environment

Adabas 7.4 for mainframes supports the following operating environments:

- SNI's BS2000
- IBM's OS/390, z/OS, VSE/ESA, VM/ESA, and z/VM
- Fujitsu's OS IV/F4 MSP

Mainframe Adabas can be used in distributed environments with Adabas on

- Digital's VAX and Alpha AXP under OpenVMS
- IBM's AS/400 under OS/400
- Wang under VS
- IBM's PC and compatibles under OS/2 or Windows/NT
- a variety of supported UNIX platforms

As the telecommunication interface, mainframe Adabas supports TP monitors such as Software AG's Com-plete; and other popular monitors such as TSO, CICS, IMS/DC, AITM/DC, TIAM, UTM, and Shadow.

Software AG's multisystem processing tool Entire Net-Work provides the benefits of distributed processing by allowing you to communicate across a network with Adabas and other service tasks.

Support for network access methods is implemented in the form of line drivers. Mainframe Entire Net-Work provides drivers for VTAM, IUCV, DCAM, CTCA (channel-to-channel adapter), TCP/IP, and XCF.

Supported Data Models

Adabas is a "relational-like" database in that

- it stores information in tables in which rows represent individual data **records** and columns represent **fields**; and
- separate Adabas files can be linked logically by a common field.

Adabas differs from true relational databases in that it

- stores many data relationships physically, resulting in fewer demands on CPU resources than true relational databases, which create all relationships logically at runtime.
- supports repeating groups of fields.

Adabas separates data relationships, management, and retrieval from the actual physical data and stores the physical data independently. It provides flexible access techniques and performs both simple and complex searches quickly and efficiently. The independence of the data from the program minimizes the need to reprogram when the database structure changes.

Logical data relationships can be created as needed. Adabas can accommodate any representational and access requirements dictated by the user environment. Each individual corporate user can decide how to view data in the system, and can alter data relationships dynamically—without altering the database or existing programs.

In contrast to systems that require a single model for all data, Adabas allows you to choose any structure your application requires. You can access the same data using your choice of data-model perspective:

- relational including nested relational (tables within tables)
- entity relationship, with proven ability to support structural objects
- hierarchical; network
- geographical
- text

Support for an object-oriented data model is currently being developed.

These data models can be combined within a single business solution; multiple solutions can view Adabas data using different data models.

As new requirements develop, Adabas evolves in both scope and complexity without redesign of the database or reprogramming of application systems. For example, field and access keys may be added to an Adabas file at any time without reloading or reorganizing the file.

Operating Structure

Figure 1-1 shows the operating structure of the Adabas system.

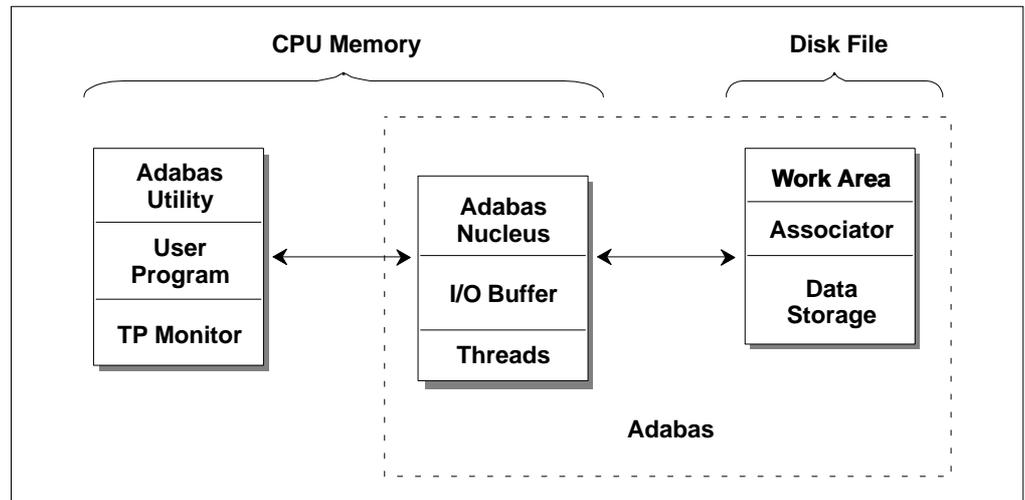


Figure 1–1: The Adabas System

Nucleus, I/O Buffer, and Threads

The Adabas nucleus and input/output (I/O) buffer are loaded into main memory at startup. The “nucleus” is a set of programs that drives Adabas, coordinates all work, and translates user program statements into Adabas commands. All programs access Adabas files through the nucleus. All database activities such as data access and update are managed by the Adabas nucleus. In most cases, a single nucleus is used to manage a single physical database.

Note:

For information about running multiple nuclei against a single physical database under a single operating system image (Adabas Parallel Services), see page 99; or under multiple OS/390 or z/OS images (Adabas Cluster Services), see page 100.

The Adabas “I/O” buffer area, which can be resized for each Adabas session, contains the most frequently used data and data relationships; it helps to minimize physical input/output (I/O) activity and thus saves computer time. It contains blocks read from the database and blocks to be written to the database:

- For blocks read from the database, a “buffer algorithm” ensures that the most frequently accessed blocks stay in memory. When a block from the database is needed, the buffer content is checked to determine if the block is already in memory, thus avoiding unnecessary reads.
- Multiple updates are accumulated in a block before being written (“flushed”) to the database.

Adabas provides multithreaded processing to maximize throughput. If I/O activity suspends command processing in an active thread, Adabas automatically switches to another thread. The user may set the number of 8-kilobyte threads to be used for an Adabas session up to a maximum of 250.

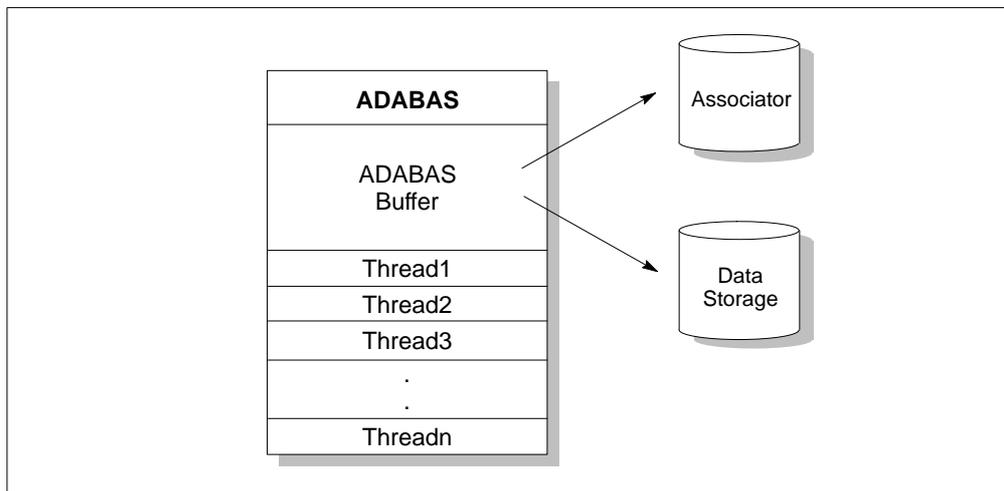


Figure 1–2: Adabas Multithread Processing

Data Storage, Associator, and Work

The Data Storage, Associator, and Work components are physical disk areas:

- Data Storage contains raw data, generally in compressed form.

- The Associator contains information about data relationships.
- The Work area contains the data protection area and temporary storage for intermediate results during complex search operations or distributed transaction processing.

See the chapter **Adabas Design** starting on page 13 for more information about these database components.

Utilities, User Programs, and TP Monitors

Utilities

Database services such as loading or deleting files are handled by an integrated set of online and batch-mode “utility” programs. Most utilities can be run in parallel with normal database activity to preclude interruption of daily production.

Adabas utilities provide initial design and load operations, backup/restore/recovery routines, database modification routines, and audit/control/tuning procedures. See the **Utilities** chapter starting on page 55 for a brief explanation of each utility.

User Programs

The nucleus is called using a batch or online “user program” written in

- Natural, Software AG’s fourth-generation application development environment, or some other fourth-generation language; or
- Assembler, or a third-generation programming language such as FORTRAN, COBOL, or PL/I (the REXX/VSE interpreter language is also supported) that uses the powerful and flexible Adabas direct call interface. Each Adabas call is accompanied by a parameter list identifying buffers defined in the user program that are used to transfer information to and from Adabas.

Note:

See page 114 for information about Adabas Native SQL, a precompiler for Ada, COBOL, FORTRAN, and PL/I programs; and page 115 for information about Adabas SQL Server, an SQL interface to Adabas.

Data from VSAM, DL/I, IMS/DB, SESAM, or TOTAL database structures can be transferred to and stored in Adabas using Adabas “bridge” products. The original, unmodified programs continue operating with their original data access commands while the bridge products intercept the data access commands and translate them to Adabas direct calls. See page 104 for more information about Adabas bridges.

Special User Programs: Triggers and Stored Procedures

The Adabas triggers and stored procedures facility, an integral part of Adabas, can be used with Natural (see page 117) to write and manage triggers and stored procedures in the Adabas server environment. “Triggers” are systematically used programs that are started automatically based on an event; they can be used to ensure referential integrity, for instance. “Stored procedures” are programs used by a number of different clients that are executed by Adabas as a result of a special user call. Storing these programs in an Adabas file on the server reduces the amount of data traffic to and from the server. See page 45 for more information about Adabas triggers and stored procedures.

TP Monitors, Adalinks, and the Adabas API

Since most systems do not allow a standard call to Adabas, Software AG provides an application program interface (API) to translate calls issued by an application program into a form that can be handled by Adabas. The Adabas API is available across all supported mainframe platforms for both batch and online operations.

Online operations are controlled by teleprocessing (TP) monitors, which serve as telecommunication interfaces to Adabas. Supported TP monitors are listed on page 4. Software AG provides versions of the Adabas API that are specific to particular TP monitors. “Adalink” is a generic term that refers to the portion of the API that is specific to a particular TP monitor.

Batch applications are supported in both single-user and multiuser mode (see page 10 for a discussion of these modes). The Adabas batch API uses a standard calling convention that is supported by all major programming languages through their CALL mechanisms. Most mainframe operating systems allow batch application modules to be linked either with the batch API or with ADAUSER.

Software AG strongly recommends linking batch application programs with the Adabas version-independent module ADAUSER. The ADAUSER module can optionally be linked with the Adabas API. ADAUSER provides upward compatibility with Adabas releases and a degree of isolation from future changes to the API or to mechanisms that handle interregion communication between the user and the nucleus (see page 11).

A client running under IBM’s OpenEdition can access Adabas. An OpenEdition application containing calls to Adabas can be linked with either the batch API or ADAUSER.

Executing Adabas

Session Types

Three types of sessions can be identified for Adabas:

- The “Adabas session” starts when the nucleus is invoked and ends when the nucleus is terminated. An Adabas nucleus is invoked using job control specific to a particular operating system that contains Adabas start-up or “ADARUN” parameters.
- A “user” is either a batch mode program or a person using a terminal. A “user session” can occur only during an Adabas session; that is, when the Adabas nucleus is active. A user session is a sequence of Adabas calls optionally starting with an open user session (OP) command and ending with a close user session (CL) command.
- A “utility session” is executed in batch, or online using the Adabas Online System (see page 85). Some utilities require the Adabas nucleus to be active; others do not. ADARUN start-up parameters are also used for executing utilities.

Storage Areas

The Adabas nucleus and each user program or Adabas utility is executed in a separate storage area defined by the operating system. The name of the storage area depends on the operating system:

BS2000	task
OS/390 or z/OS	address space, data space, hiperspace, 64-bit virtual space
VM/ESA or z/VM	virtual machine
VSE/ESA	partition, address space, data space

For consistency and simplification, Adabas manuals refer to all non-VM areas (task, address space, partition, etc.) as “regions”. VM areas are called “virtual machines”.

Modes of Operation

Adabas supports two modes of operation: single-user and multiuser.

Single-user mode is in effect when a user program (or Adabas utility) is executed in the same partition/region as the Adabas nucleus.

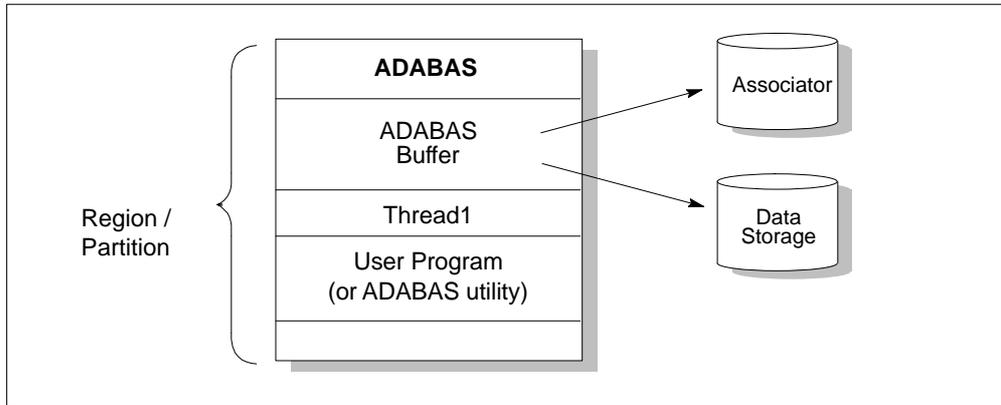


Figure 1-3: Single-User Mode

Multuser mode is in effect when the Adabas nucleus is located in a separate partition/region. It is the most efficient and therefore the recommended mode of operation.

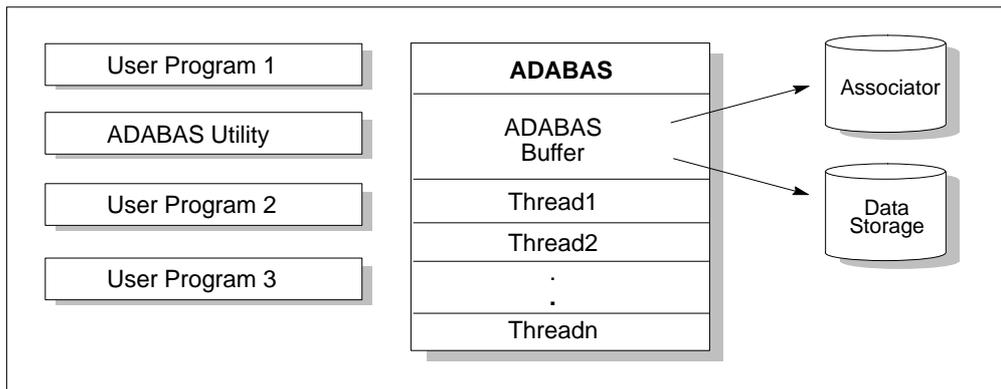


Figure 1-4: Multiuser Mode

When using Adabas in multiuser mode, interregion communication is handled by Adabas in a manner that takes optimum advantage of the communications facilities offered by the various operating systems.

For single-user mode, the appropriate Adabas nucleus JCL must be included with the job control for the utility or user program.

ADARUN Start-up Parameters

The ADARUN control statement defines and starts the Adabas operating environment. The ADARUN control statement also starts Adabas utilities. ADARUN

- loads the ADAIOR module, which performs all database I/O and other operating-system-dependent functions;
- interprets the ADARUN parameter statements; then loads and modifies the appropriate Adabas nucleus or utility modules according to the ADARUN parameter settings; and
- transfers control to Adabas.

The ADARUN statement, normally a series of entries each specifying one or more ADARUN parameter settings, is specified in the DDCARD (OS/390, z/OS, MSP, VM/ESA, z/VM, or BS2000) or VSE/ESA CARD dataset.

Session Control

Adabas provides several ways to monitor and control Adabas, user, and utility sessions:

- Adabas “operator commands” can be entered from the operator console during an Adabas session or during utility operation.
- The ADADBS OPERCOM utility function can issue operator commands to the Adabas nucleus. Adabas then issues a message to the operator confirming the command execution.
- For those using Adabas Online System (demo or full version), you may be able to execute functions corresponding to operator commands while an Adabas session is active using menu options or direct commands.

Operator commands can be used to terminate an Adabas or user session; display nucleus or utility information; log commands; and change Adabas operating parameters or conditions.

Adabas “direct call commands” can also be used to open and close a user session. See page 37 for more information about direct call commands.

ADABAS DESIGN

Database systems often involve complex data structures and data handling procedures that can be designed and used only by persons with extensive knowledge and experience. Adabas has a remarkably simple structure by comparison, yet it provides significant advantages for operational efficiency, ease of design, definition, and database evolution.

Adabas Entities

In Adabas, a “field” is the smallest logical unit of information (e.g., current salary) that may be defined and referenced by the user. A “record” is a collection of related fields that make up a complete unit of information (e.g., all the payroll data for a single employee). A “file” is a group of related records that have the same format (with some exceptions; see page 24). A “database” is a group of related files.

Adabas Limits

The table below shows the maximum number that mainframe Adabas supports for each entity:

Entity	Maximum
Databases	65,535
Blocks per database	2,147,483,646 using 4-byte RABNs
Files per database	the lower of 5,000 or the Associator block size minus one
Records per file	4,294,967,294 using 4-byte ISNs
Fields per record	926
Uncompressed record length	depends on the operating system
Compressed record length	Data Storage block size

Adabas Space Management

The disk storage space allocated to a single Adabas database is segmented into “logical” Adabas files. A certain part of the overall space within the database is allocated to each logical file. When the space is filled with records from the file, Adabas automatically allocates more space to the file from the common free space pool. This dynamic space allocation, together with the dynamic recovery of released space, allows Adabas databases to run without intervention for long periods of time.

The distribution of database space across disk drives can be controlled by “physically” segmenting it into multiple independent datasets. When all physical database space is filled, more datasets can be allocated dynamically, or the size of existing datasets can be increased so that new physical files can be loaded without reorganizing the entire database.

Database Components

To support the separation of data and access structures, the Adabas nucleus uses three database components:

- Data Storage for compressed data
- Associator for data management and retrieval
- Work, a scratch area for complex search criteria, etc.

Data Storage

Data Storage is divided into “blocks”, each identified by a 3- or 4-byte relative Adabas block number or “RABN” that identifies the block’s physical location relative to the beginning of the component. Data Storage blocks contain one or more physical records and a padding area to absorb the expansion of records in the block.

A logical identifier stored in the first four bytes of each physical record is the only control information stored in the data block. This internal sequence number or “ISN” uniquely identifies each record and never changes. When a record is added, it is assigned an ISN equal to the highest existing ISN plus one. When a record is deleted, its ISN is reused only if you instruct Adabas to do so. Reusing ISNs reduces system overhead during some searches and is recommended for files with records that are frequently added and deleted.

For each file, between 1–90 percent (default 10%) of each block can be allocated as padding based on the amount and type of updating expected. This reserved space permits records to expand without migrating to another block and thus helps to minimize system overhead.

RABN	Records (identified by ISNs)			Padding Area
1	0071	0595	0221	[Hatched Area]
2	0222	0991	2021	
3	0300	0401	0532	

Figure 2–1: Adabas Data Storage Blocks

Free Space and Space Reusage

If records become too large for their blocks, they migrate to new locations. When a record migrates or is deleted, free space is opened in the data block between the last record and the padding area. Figure 2–2 shows free space created when the record with ISN 0401 becomes too large for the block and migrates to another block:

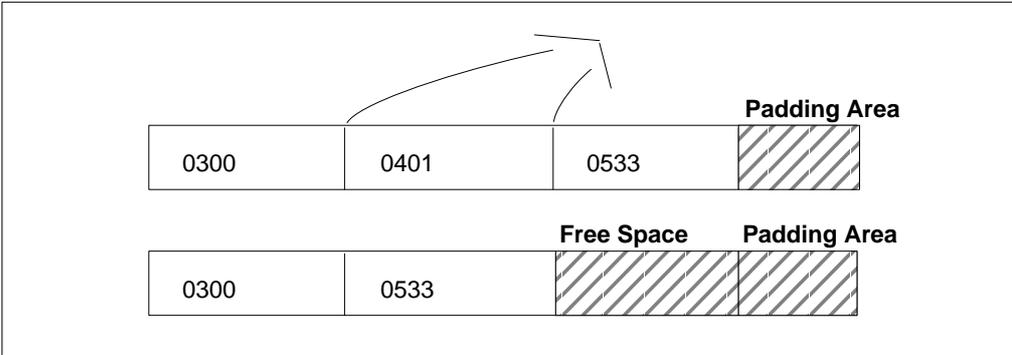


Figure 2–2: Free Data Storage Space Created by Record Migration

You can instruct Adabas to reuse free space. Reusing space saves computer time, since Adabas then reads fewer physical blocks during searches. It is recommended for all files.

Compression

Data compression significantly reduces the amount of storage required. It also permits the transmission of more information per physical transfer, resulting in greater I/O efficiency.

Adabas retains data records in compressed form. Four compression options are supported:

- default compression;
- null suppression;
- fixed format; and
- forward or “prefix” index compression.

The first three options define and execute compression at the field level; the fourth option can be implemented at the file or the database level, in which case specific files can be set differently; the file-level setting overrides the database setting. The null suppression and fixed format options are added as field options and are discussed on page 30. The forward index compression option is set using the ADALOD utility and can be changed using the ADAORD utility.

“Default compression” deletes trailing blanks in alphanumeric fields and leading zeros in binary fields. An inclusive length byte (ILB) at the beginning of the field indicates the total number of stored bytes, including the ILB. Thus, if “Susan” is entered in a “first-name” field defined with a 20-character length and default compression, its stored size will be six bytes: five bytes for the letters of the name, plus one byte for the ILB. In addition, empty fields in a record are not stored; an empty field is replaced by a one-byte empty field counter (EFC). Adabas can store up to 63 contiguous empty fields in a single hexadecimal byte.

Many Adabas files require only 50% to 60% of the space used for the raw data. Even with the addition of approximately 25% for the access structures stored in the Associator, Adabas storage requirements are still less than those required for traditional file storage or for DBMSs that do not use data compression.

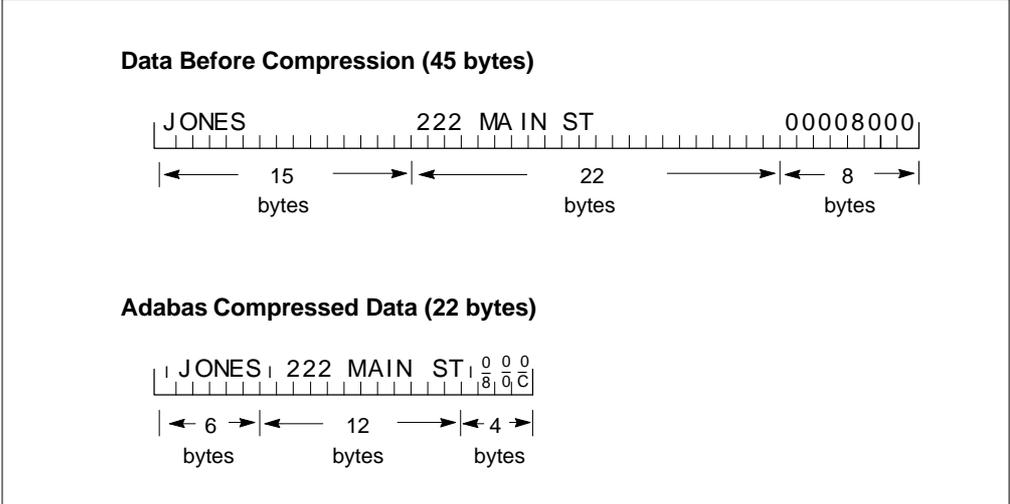


Figure 2-3: An Example of Adabas Default Compression

Forward (or ‘front’ or ‘prefix’) index compression removes redundant prefix information from index values. Within one index block, the first value is stored in full length. For all subsequent values, the prefix that is common with the predecessor is compressed. An index value is represented by

<l,p,value>

—where

- p is the number of bytes that are identical to the prefix of the preceding value.
- l is the exclusive length of the remaining value including the p-byte.

For example:

Before Compression	After Compression
ABCDE	6 0 ABCDE
ABCDEF	2 5 F
ABCGGG	4 3 GGG
ABCGGH	2 5 H

The decision to compress index values is based on the similarity of index values and the size of the file:

- the more similar the index values, the better the compression results.
- small files are not good candidates because the absolute amount of space saved would be small whereas large files are good candidates for index compression.

Even in a worst case scenario where the index values for a file do not compress well, a compressed index will not require more index blocks than an uncompressed index.

Associator

The Associator is an organizational unit used for storing the structures required to access data in Data Storage. It contains

- a control block for the database as a whole and control blocks for each file;
- all tables needed to control and maintain the database including a field definition table or “FDT” (see page 26) for each file and coupling lists for physically coupled files (see page 22);
- an inverted list for each descriptor in each file of the database and an address converter for each file.

Inverted Lists

An inverted list, which is used to resolve Adabas search commands and read records in logical sequence, is built and maintained for each field in an Adabas file that is designated as a key field or “descriptor” (see page 29). It is called an “inverted” list because it is organized by descriptor value rather than by ISN. The list comprises the normal index (NI) and as many as 14 upper indexes (UI).

The normal index (NI) of the inverted list for a particular descriptor has an entry for each value. The entry contains the value itself, the number of records in which the value occurs, and the ISNs of those records.

To increase search efficiency, upper index (UI) levels are automatically created by Adabas as required, each level to manage the next lower level index. The first level UI, like the NI it manages, contains entries for only one descriptor in each index block. All other UI levels contain entries for all descriptors in each index block. UIs require a minimal amount of space: two blocks is the minimum.

Note:
The Adabas direct access method (ADAM) facility permits the retrieval of records directly from Data Storage without accessing the inverted lists. The Data Storage block number in which a record is located is calculated using a randomizing algorithm based on the ADAM key of the record. The use of ADAM is completely transparent to application programs and query and report writer facilities. See page 44 for more information.

Figure 2-4 shows a typical normal index for the descriptor “city” in a customer file.

Value	Count	ISNs
London	27	3 . . .
New York	61	96 . . .
Zurich	31	2 6 23 76 . . .
. . .		

Figure 2-4: A Normal Index

The example indicates that there are 31 records with the “city” Zurich (the ISNs of these records are 2,6,23,76...).

Address Converter

The address converter determines the physical location of a record. It is an index that maps the logical identifier of a record (that is, the ISN) to the relative Adabas block number (RABN) of the Data Storage block where the record is stored.

The address converter contains a list of RABNs in ISN order. Only the RABNs are actually stored in the address converter; the ISNs are identified by their relative position.

Figure 2-5 shows the relationship between an inverted list, the address converter, and Data Storage. For example, to determine the physical location of the record whose ISN is 6, Adabas uses the ISN as an index into the address converter. The sixth entry in the address converter is 2. Therefore, ISN 6 is located in physical block 2 in Data Storage for this file.

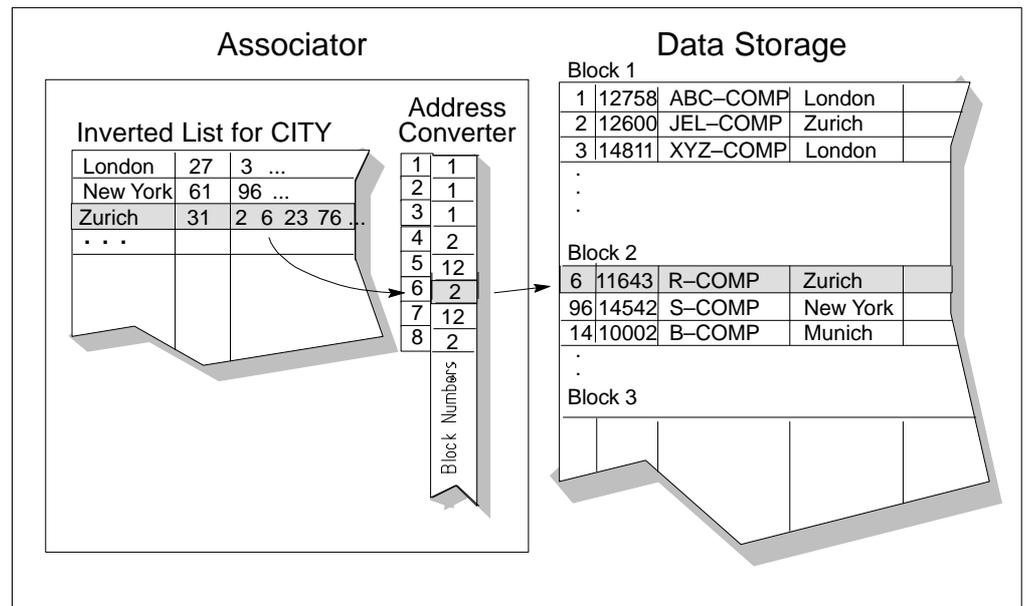


Figure 2-5: Adabas Access Technique

When a record moves or is deleted, Adabas updates the address converter automatically and transparently.

Since the ISN for a record never changes, and its physical block address is stored only in the address converter entry, the record itself may be moved in Data Storage with only one update to the address converter required and with no extension to the access path of the record.

Even if a record has many descriptors defined, the inverted list for each descriptor need not be modified because it contains ISNs.

This process explains how Adabas is able to perform simple and complex searches quickly and efficiently without storing pointer information in Data Storage.

Work

The Work area stores information in four parts:

Part Stores . . .

- 1 data protection information required by the routines for autorestart and autobackout. See page 51 for more information.
- 2 intermediate results (ISN lists) of search commands.
- 3 final results (ISN lists) of search commands.
- 4 data related to two-phase commit processing.

Other Components

Sort and Temp Areas

Certain Adabas utilities (ADAINV, ADALOD) require two additional datasets, sort and temp, for sorting and intermediate storage of data. Certain functions of other utilities require the temp dataset for intermediate storage.

The size of the temp and sort datasets varies according to the utility function to be executed. These datasets can be allocated during the job and then released, or permanent datasets can be allocated and reused.

Logs

Adabas uses the following optional logs:

- The “command log” (CLOG) records information from the control block of each Adabas command that is issued. The CLOG provides an audit trail and can be used for debugging and for monitoring the use of resources. Single, dual, or multiple (2–8) datasets can be used (multiple datasets are recommended).
- The “protection log” (PLOG) records before- and after-images of records and other elements when changes are made to the database. It is used to recover the database (up to the last completed transaction or “ET”) after restart. Single, dual, or multiple (2–8) datasets can be used (multiple datasets are recommended).
- The “recovery log” (RLOG) records additional information that the Adabas Recovery Aid uses to construct a recovery job stream. See the ADARAI utility discussion on page 59 for more information.

Database Files

Each database contains system files and data files. A data file is generally created for each record structure required; that is, for each set of related fields identified.

Files are loaded into the database using the ADALOD utility. A file number must be unique in the database and not greater than the maximum file number defined for the database in the MAXFILES parameter. For a checkpoint, security, system file, or physically coupled file, the number cannot be greater than 255; other files including a trigger file can have two-byte file numbers. File numbers are assigned by the user in any sequence.

System Files

Adabas uses certain files to store system information. Using the ADALOD utility's FILE parameter, you can identify an Adabas **system** file as one of the following:

CHECKPOINT	Adabas checkpoint file
SECURITY	Adabas security file
SYSFILE	Adabas system file
TRIGGER	Adabas trigger file

Coupled Files

File coupling allows you to select, using a single search command, records from one file that are related (coupled) to records containing specified values in a second file.

Physical Coupling

Any two files with file numbers 255 or lower may be physically coupled if a common "descriptor" (see page 29) with identical format and length definitions is present in both files. A single file may be coupled with up to 18 other files, but only one coupling relationship may exist between any two files at any one time. A file may not be coupled to itself.

When files are coupled, coupling lists are created in the Associator for each file being coupled. File coupling is bidirectional rather than hierarchical in that two coupling lists are created for each coupling relationship with each list containing the ISNs that are coupled to the other file.

Once the physical coupling lists have been created, any key field in either file may be used within a search criteria.

Physical coupling may add a considerable amount of overhead if the files involved are frequently updated. The coupling lists must be updated if a record in either of the files is added or deleted, or if the descriptor used as the basis for the coupling is updated in either file.

Physical coupling may be useful for information retrieval systems in which

- files seldom change;
- the additional overhead of the coupling lists is insignificant compared with the increased ease of formulating queries; or
- files are small and primarily query-oriented.

Logical or “Soft” Coupling

Multiple files may also be queried by specifying the field to be used for interfile linkage in the search criteria. Adabas then performs all necessary search, read, and internal list matching operations.

This technique is called logical or “soft” coupling because it does not require the files to be physically coupled. Although logical coupling requires read commands, it is normally more efficient because it avoids the increased overhead of coupling lists.

Structuring Files to Enhance Performance

An Adabas database with one file for each record type supports any application functions required of it and is the easiest to manipulate for interactive queries, but it may not yield the best performance:

- As the number of Adabas files increases, the number of Adabas calls increases. Each Adabas call requires interpretation, validation and, in multiuser mode, supervisor call (SVC) and queuing overhead.
- In addition to the input/output (I/O) operations necessary for accessing at least one index, address converter, and Data Storage block from each file, the “one file per record type” structure requires buffer pool space. If sufficient buffer space is not available, blocks are overwritten that may be needed for a later request.

The number of Adabas files used by critical programs can be reduced by

- using multiple-value fields and periodic groups (see page 27);
- linking physical files into a single logical (expanded) file;
- including more than one type of record in an Adabas file;
- including records for more than one category of user in an Adabas (multiclient) file; and
- controlling data duplication and the resulting high resource usage.

Expanded Files

If you have a large number of records of a single type, you may need to spread the records over multiple physical files.

To reduce the number of files accessed, Adabas allows you to link multiple physical files containing records of the same format together as a single logical file. This file structure is called an “expanded file” and the physical files comprising it are the “component files”. An expanded file can comprise up to 128 component files, each with a unique range of logical ISNs. An expanded file cannot exceed 4,294,967,294 records.

Note:

Since Adabas now supports larger file sizes and a greater number of Adabas physical files and databases, the need for expanded files has, in most cases, been removed.

Although an application program addresses the logical file (the address of the file is the number of the expanded file’s base component or “anchor” file), Adabas selects the correct component file based on the data in a field defined as the “criterion” field. The data in this field has characteristics unique to records in only one component file. When an application updates the expanded file, Adabas looks at the data in the criterion field in the record to be written to determine which component file to update. When reading expanded file data, Adabas uses the logical ISN as the key to finding the correct component file.

Multiple Record Types in One File

Multiple record types can be defined within a single physical record; each record type is a logical record composed of a subset of the fields defined for the file. Fields that do not belong to a given type are null-suppressed.

Record types can be identified to Adabas by

- defining a record type field with values to differentiate one type from another; or
- using values of an existing field to differentiate type; for example, to differentiate two types, a value of zero for a field common to both types might identify one type and any nonzero value for the same field might identify the other type.

Multiclient Files

Records for multiple users or groups of users can be stored in a single Adabas physical file defined as “multiclient”. The multiclient feature divides the physical file into multiple logical files by attaching an internal owner ID to each record.

The owner ID is assigned to a user ID. A user ID can have only one owner ID, but an owner ID can belong to more than one user. Each user can access only the subset of records that is associated with the user’s owner ID.

Note:

For any installed external security package such as RACF, CA-ACF2, or CA-Top Secret, a user is still identified by either Natural ETID or LOGON ID.

All database requests to multiclient files are handled by the Adabas nucleus.

Controlled Data Redundancy

“Physical” redundancy increases storage requirements but may also enhance performance and decrease complexity. For example, if a database stores customer and order information in a customer-orders file and product descriptions in an inventory file, and a program that generates invoices requires product descriptions in addition to customer-order data, it might enhance performance to store a duplicate copy of the product descriptions in the customer-orders file.

“Logical” redundancy also increases storage demands while decreasing complexity. It involves storing in one file the results of a process on data in another file; thus, the duplicate data is implied by the content of another file, rather than being physically stored in two places.

Physical and logical redundancy cause update programs to run more slowly. The duplicate updates required when changes in one file affect records in another file may degrade performance severely. Redundancy should be used only for static data or data that is updated rarely. You can control data redundancy by using multiple-value fields, periodic groups, and multiple record types within a file.

Records and Field Definitions

In Adabas, the record structure and the content of each field in a physical file are described in a field definition table or “FDT”, which is stored in the Associator. There is one FDT for each database file. The FDT is used by Adabas during the execution of Adabas commands to determine the logical structure and characteristics of any given field (or group) in the file.

The FDT lists the fields of the file in physical record order; provides a “quick index” to the file’s records; and defines the file’s fields, sub/superfields, and descriptors including collation, sub-/super-/hyper- and phonetic. A minimum of one and a maximum of 926 field definitions may be specified.

Information about each field includes the level, name, length, format, options, and special field and descriptor attributes.

FIELD DESCRIPTION TABLE								
LEVEL	I	I	I	I	I	I	I	PARENT OF
	I	I	I	I	I	I	I	
----	I	I	I	I	I	I	I	----
1	I	AA	I	8	I	A	I DE, UQ	I
1	I	AB	I		I		I	I
2	I	AC	I	20	I	A	I NU	I
2	I	AE	I	20	I	A	I DE	I SUPERDE, PHONDE
2	I	AD	I	20	I	A	I NU	I
1	I	AF	I	1	I	A	I FI	I
1	I	AG	I	1	I	A	I FI	I
1	I	AH	I	6	I	U	I DE	I
1	I	A2	I		I		I	I
1	I	AO	I	6	I	A	I DE	I SUBDE, SUPERDE
1	I	AQ	I		I		I PE	I
2	I	AR	I	3	I	A	I NU	I SUPERDE
2	I	AS	I	5	I	P	I NU	I SUPERDE
1	I	A3	I		I		I	I
2	I	AU	I	2	I	U	I	I SUPERDE
2	I	AV	I	2	I	U	I NU	I SUPERDE

Figure 2–6: Field Definition Table

Record Structure

The order of the fields listed in the FDT determines the structure of the record and the efficiency of retrieval. The following factors should be considered when ordering fields:

- Fields that will be accessed frequently should be ordered first in the FDT. This technique reduces CPU time because Adabas does not have to read the whole record when retrieving a field.
- Fields that will frequently be accessed together should be assigned to a “group” field.
- Fields that will **always** be accessed together should be defined as a single field. This technique may inhibit compression and query language use; however, it decreases processing time by providing more efficient internal processing and shorter format buffers.
- If appropriate, fields that will frequently be empty should be ordered together in the FDT and set to use default compression or null suppression.
- Numeric fields should be loaded in the format in which they will be used most often.

Field Levels

When two or more consecutive fields in the FDT are frequently accessed together, you can reference them together by defining a group field. Other than its level and Adabas short name, a group field has no attributes defined. It immediately precedes its member fields in the FDT. A higher field “level” number is used to assign the member fields to the group field. Adabas supports up to seven field levels. User programs can access each member field individually, or all member fields together by referencing the group field.

For example, in Figure 2–6 on page 26, field AB is defined as a group field and assigned to level 1. Fields AC, AE, and AD are assigned to level 2, indicating that they belong to group field AB. The next field, AF, is assigned to level 1, indicating that it is not part of the AB group. User programs can access AC, AE, and AD individually, or together by referencing the group field AB.

A group field can be assigned as a “periodic” group field if it is comprised of fields that can have more than one value (for example, group field AQ in Figure 2–6); see page 31.

Field Names

A field is identified to Adabas by a two-character Adabas “short” name that must begin with an alphabetic character and can be followed by a numeral or letter (the combinations E0–E9 are reserved and special characters are not allowed) and must be unique within a file. Adabas assigns short names to fields automatically, although you can choose to assign them yourself. Adabas uses the short names internally and actually accesses fields by their short names.

Field Length and Data Format

Field values are fixed or variable in length and can be in alphanumeric, binary, fixed-point, floating-point, packed/unpacked decimal, or wide character formats.

The length (expressed in bytes) and format (expressed as a one-character code) of a field define the standards (defaults) to be used by Adabas during command processing. They are used when the field is read/updated unless the user specifies an override.

If standard length is zero for a field, the field is assumed to be a variable-length field. Standard format must be specified for a field. The format specified determines the type of default compression to be performed on the field.

The maximum field lengths that may be specified depend on the “format” value:

Format	Format Description	Maximum Length
A	Alphanumeric (left-justified) : see also the long alphanumeric (LA) option on page 31	253 bytes
B	Binary (right-justified, unsigned/positive)	126 bytes
F	Fixed point (right-justified, signed, positive value in normal form; negative value in two’s complement form)	4 bytes (always exactly 2 or 4 bytes)
G	Floating point (normalized form, signed)	8 bytes (always exactly 4 or 8 bytes)
P	Packed decimal (right-justified, signed)	15 bytes
U	Unpacked decimal (right-justified, signed)	29 bytes
W	Wide character (left-justified) : see also the long alphanumeric (LA) option on page 31	253 bytes

Field Options

Field options are specified using two-character codes, which may be specified in any order, separated by a comma.

Code	Option	Page
DE	Field is to be a descriptor (key).	29
FI	Field is to have a fixed storage length; values are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.	30
LA	An alphanumeric or wide-character, variable-length field may contain a value up to 16,381 bytes long.	31
MU	Field may contain up to 191 values in a single record.	31
NC	Field may contain a null value that satisfies the SQL interpretation of a field having no value; that is, the field's value is not defined (not counted).	33
NN	Field defined with NC option must always have a value defined; it cannot contain an SQL null (not null).	33
NU	Null values occurring in the field are to be suppressed.	30
NV	An alphanumeric or wide-character field is to be processed in the record buffer without being converted.	31
PE	This group field is to define consecutive fields (which may include one or more MU fields) in the FDT that repeat together (up to 191 times) in a record.	31
UQ	Field is to be a unique descriptor; that is, for each record in the file, the descriptor must have a different value.	29
XI	For this field, the occurrence (index) number is to be excluded from the unique descriptor (UQ) option set for a periodic group (PE).	29

Descriptor Options DE, UQ, and XI

A “descriptor” is a search key. The DE option indicates that the field is to be a descriptor. The UQ option can only be specified if DE is also specified; it indicates that the DE field is to have a different (i.e., unique) value for each record in the file. Entries are made in the Associator's inverted list for DE fields, adding disk space and processing overhead requirements.

Any field can be used within a selection criterion. When a field that is used extensively as a search criterion is defined as a descriptor (key), the selection process is considerably faster since Adabas is able to access the descriptor's values directly from the inverted list without reading any records from Data Storage.

A descriptor field can be used as a sort key in a search command, as a way of controlling a logical sequential read process (ascending or descending values), or as the basis for file coupling.

Any field and any number of fields in a file can be defined as descriptors. When a multiple-value field or a field in a periodic group is defined as a descriptor, multiple key values are generated for the record. Key searches may be limited to particular occurrences of a periodic group.

The XI option is used for unique descriptors in periodic groups to exclude the occurrence (index) number from the definition of uniqueness.

Because the inverted list requires disk space and update overhead, the descriptor option should be used judiciously, particularly if the file is large and the field that is being considered as a descriptor is updated frequently. For instance, the inverted list for a periodic group used as a descriptor may be very large because each occurrence is stored.

A descriptor may be defined at the time a file is created, or later by using an Adabas utility. Because the definition of a descriptor is independent of and has no effect on the record structure, descriptors may be created or deleted at any time without the need for database restructuring or reorganization.

Note, however, that if a descriptor field is not ordered first in the record structure and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to reorder the field first for each record of the file.

A portion of a field may be defined as a "subdescriptor"; combinations of fields or portions thereof may be defined as a "superdescriptor"; a user-supplied algorithm may be the basis of a "collation descriptor" or "hyperdescriptor"; and a "sounds-like" encoding algorithm may be the basis of a "phonetic descriptor", which may be customized for specific language requirements. See page 33 for more information.

Data Compression Options FI and NU

Default data compression is described on page 16. At the field level, additional compression can be specified (null suppression option) or all compression can be disabled (fixed storage option).

“Null suppression” (NU) differs from default compression in that searches on descriptor fields defined with null suppression do **not** return records in which the descriptor field is empty.

Fields defined as “fixed format” (FI) do not include a length byte and are not compressed. This option actually saves storage space for one-byte fields or fields that are nearly always full (e.g., a field containing the social security number).

Encoding Conversion Option NV

Alphanumeric (A) or wide-character (W) format fields with the NV option are processed in the record buffer without being converted to or from the user.

The field has the characteristics of the file encoding; that is, the default blank

- for A fields is always the EBCDIC blank (X'40'), and
- for W fields is always the blank in the file encoding for W format.

The NV option is used for fields containing data that cannot be converted meaningfully or should not be converted because the application expects the data exactly as it is stored.

The field length for NV fields is byte-swapped if the user architecture is byte-swapped.

Long Alpha Option LA

The long alphanumeric (LA) option can only be specified for variable-length alphanumeric or wide-character fields; i.e., “A”- or “W”-format fields having a length of zero. With the LA option, such an alphanumeric or wide-character field can contain a value up to 16,381 bytes long.

An alpha or wide field with the LA option is compressed in the same way as an alpha or wide field without the option. The maximum length that a field with LA option can actually have is restricted by the block size where the compressed record is stored.

MU and PE Options and Field Types

Adabas supports two basic field types: elementary fields and multiple-value fields. An “elementary” field has only one value per record. A “multiple-value” (MU) field can have up to 191 values, or occurrences, in a single record. Each multiple-value field has a “binary occurrence counter” (BOC) that stores the number of occurrences.

A “periodic” (PE) group field defines consecutive fields in the FDT that repeat together in a record. Like the members of a non-periodic group field, PE members immediately follow the PE group field, have a higher level number than the PE field, and can be accessed both individually and as a group. Each PE has a BOC that stores the number of occurrences.

A periodic group may be repeated up to 191 times per record and may contain one or more multiple-value fields. Occurrences or values that are not used require no storage space.

Adabas thus supports four field types:

	Single Value per Record	Multiple Values per Record
Single Field	Elementary	MU
Multiple Fields	Group	PE

Figure 2–7 illustrates the four field types in a single record structure.

CUSTOMER NUMBER	FIRST NAME	LAST NAME	PE BOC	MU BOC	STREET	CITY	STATE	ZIP
19811	Laura	Cagnetti	3	1	118 Glade	Erie	PA	16509
	ELEMENTARY FIELD			2	271 Larue	Cincinnati	OH	45211
					P.O. Box 88			
	GROUP FIELD (NAME)			2	733 Hall	Easton	PA	19014
					P.O. Box 7			
					MULTIPLE VALUE FIELD			
					PERIODIC GROUP (ADDRESS)			

Figure 2–7: The Adabas Field Types

A PE cannot be nested within another PE. Nesting an MU within a PE, as shown in Figure 2–7, is permitted but complicates programming by introducing a two-dimensional array. It also has implications for data access: when Adabas accesses the periodic group, it returns only the first occurrence of the MU for each occurrence of the PE returned.

The unique characteristic of the periodic group and the reason for choosing the periodic group structure is its ability to maintain the order of occurrences. If a periodic group originally contains three occurrences and the first or second occurrence is later deleted, those occurrences are set to nulls; the third occurrence remains in the third position. This contrasts with the way leading null entries are handled in multiple-value fields. The individual values in a multiple-value field do not retain positional integrity if one of the values is removed.

SQL Compatibility Options NC and NN

Special data definition options are included in Adabas to accommodate Software AG's mainframe Adabas SQL Server (ESQ) and other structured query language (SQL) database query languages that require SQL-compatible null representation.

A field designated with the NC (not counted) option may contain a null value that satisfies the SQL interpretation of a field having no value. An NC field containing a null means that **no field value has been entered**; that is, the field's value is not defined.

This undefined state differs from a null value assigned to a non-NC field for which no value has been specified: a non-NC field's null means the value in the field is either zero or blank, depending on the field's format.

The NN (not null) option can be specified only for NC-defined fields. It indicates that an NC field must always have a value defined; it cannot contain an SQL null. This ensures that the field cannot be left undefined when a record is either created or updated. The field value may be zero or blank, however.

Special Field and Descriptor Attributes

Parent Of

The FDT indicates whether a field is a "parent" field for a collation descriptor, sub/superfield, sub/superdescriptor, hyperdescriptor, or phonetic descriptor as described in the following section.

Special Descriptors

Information about any special fields and descriptors (collation descriptors, subdescriptors, subfields, superdescriptors, superfields, phonetic descriptors, and hyperdescriptors) in the file is maintained in the special descriptor table or "SDT" part of the FDT.

SPECIAL DESCRIPTOR TABLE

TYPE	NAME	LENGTH	FORMAT	OPTIONS	STRUCTURE
SUPER	H1	4	B	DE, NU	AU (1 - 2)
SUB	S1	4	A	DE	AV (1 - 2)
SUPER	S2	26	A	DE	AO (1 - 4)
SUPER	S3	12	A	DE, NU, PE	AO (1 - 6)
PHON	PH				AE (1 - 20)
COL	Y1	20	W	DE	AR (1 - 3)
COL	Y2	12	A	DE, NU, PE	AS (1 - 9)
					PH = PHON (AE)
					CDX 8, PA
					CDX 1, AR

Figure 2–8: Special Descriptor Table

Along with the name, length, format, and specified options of each special field and descriptor, this table provides the following information:

Column	Explanation
TYPE	COL Collation descriptor HYPER Hyperdescriptor PHON Phonetic descriptor SUB Subfield/subdescriptor SUPER Superfield/superdescriptor
STRUCTURE	The component fields and field bytes of the sub-, super-, or hyperdescriptor. Phonetic descriptors show the equivalent alphanumeric elementary fields. Collation descriptors show the associated collation descriptor user-exit and the name of the parent field.

Collation Descriptor

An alphanumeric or wide-character field can be defined as a parent field of a “collation” descriptor. A collation descriptor is used to sort field values in a special user-defined sequence. The LF command reports the collation descriptor field information.

A collation descriptor is assigned a collation descriptor user exit (1–8) which encodes the collation descriptor value and decodes it back to the original field value. The ADARUN parameter CDXnn is used to specify collation descriptor user exits.

Hyperdescriptor

The hyperdescriptor option can be used to generate descriptor values based on a user-supplied algorithm. Up to 31 different hyperdescriptors can be defined for a single physical Adabas database. Each hyperdescriptor must be named by an appropriate HEXnn ADARUN statement parameter in the job where it is used.

With hyperdescriptors, “fuzzy” matching is possible; i.e., retrieving data based on **similar** rather than on **exact** search criteria. Hyperdescriptors allow multiple virtual indexes, meaning that several different search index entries can be made for a single data field.

Hyperdescriptors can be used to implement “n”-component superdescriptors, derived keys, or other key constructs. Using hyperdescriptors, it is possible to develop applications that are simpler and more flexible than applications based on a strictly normalized relational structure.

One application area for hyperdescriptors is name processing. For example, the name SCHROEDER could be stored not only with the index SCHROEDER itself, but also with the “virtual” indexes SCHRODER, SCHRADER, SCHRÖDER or any other variation of the name. Thus, although only the name SCHROEDER is physically stored in the data area of the database, multiple search indexes exist to the data. If, subsequently, a search is made for the name SCHRODER, the record SCHROEDER will be found.

A more sophisticated application area for hyperdescriptors is fingerprint matching, in which typical characteristics of fingerprints can form the basis of a fuzzy matching algorithm; i.e., the original fingerprint is stored in the database, but any number of search indexes can be made to the fingerprint, based on an algorithm that allows small-scale deviations from the original.

Phonetic Descriptor

A “phonetic” descriptor may be defined and used to search for all records that contain similar phonetic values. The phonetic value of a descriptor is determined by an internal algorithm based on the first 20 bytes of the field value with only alphabetic values being considered (numeric values, special characters and blanks are ignored).

Subfield / Superfield

A portion of a field (“subfield”) or any combination of fields (“superfield”) may be defined as an elementary field (see page 31). Subfields and superfields may be used for read operations only. They may only be changed by updating the original fields.

Subdescriptor

A “subdescriptor” is part of a single field used as a descriptor. The field from which the subdescriptor is derived may or may not be an elementary descriptor (see page 29). If a search criteria involves a range of values contained in the first “n” bytes of an alphanumeric field or the last “n” bytes of a numeric field, a subdescriptor may be defined using only the relevant bytes of the field. A subdescriptor allows you to increase the efficiency of a search by specifying a single value rather than a range of values.

For example, if the first two bytes of a five-byte field refer to a geographical region and you want to retrieve all records for region 11 without using a subdescriptor, you would have to search for all records in the range 11000–11999. If you define a subdescriptor comprising the first two bytes of the field, you could search for all records with 11 in the subdescriptor.

Superdescriptor

A “superdescriptor” combines all or parts of 2–20 fields. The fields from which the superdescriptor is derived may or may not be elementary descriptors. When search criteria involve values for a combination of fields, using a superdescriptor is more efficient than using a combination of several elementary descriptors.

For example, to search for customers by last name within regions, you could create a superdescriptor by combining the first two bytes (i.e., the geographical region indicator) of the five-byte customer number field and the entire customer last name field.

USING ADABAS

Generally, Adabas uses between 10% and 50% of the data processing resources (disk storage, CPU time, elapsed processing time) used by other database management systems. Since fewer hardware facilities are used, more can be accomplished with less. Very large online applications using several terabytes of data have been successfully implemented, with thousands of terminal workstations, and with the response times and the cost of much smaller systems.

Accessing a Database from Programs

Adabas access is field-oriented: user programs access and retrieve only the fields they need. User program statements invoke Adabas search and retrieval operations automatically.

Direct Call Interface

Adabas provides a powerful and flexible set of direct call commands for performing database operations. Adabas direct call commands provide a direct interface to the Adabas database when Natural or another fourth-generation database language is not being used.

The commands can be categorized by function:

- database query
- read (Data Storage or Associator)
- database modification
- logical transaction processing
- special commands

Database Query Commands (Sx)

Database query commands (S1/S4, S2, S5) search for and return the ISNs of specified records or record groups according to specified search criteria. Other commands in this category (S8, S9) sort the resulting ISN lists in preparation for later operations.

The ISN lists resulting from any Sx command may be saved on the Adabas Work dataset for later retrieval during the user session.

In most cases, these commands do not actually read the database; ISNs are read directly from the Associator's inverted lists. Options allow the ISN's record to be placed in hold status to prevent it from being updated by other programs until the record is released; if desired, additional field values contained in the first ISN's record can be read from Data Storage.

Read Commands (Lx)

The L1 through L6 commands are used to read actual records from Data Storage. Depending on the specified command and its options, records are read individually

- in the sequence in which they are stored;
- in the order of an ISN list created by one of the database query commands; or
- in logical sequence according to a user-specified descriptor.

See page 41 for more information about sequential access methods.

A hold option allows the database records to be locked until released by a separate command or at transaction end.

The L9 and LF commands read information directly from the Associator inverted lists or file definition tables (FDTs), returning either the inverted list values for a specified descriptor or the field definitions for a specified file in the database.

Database Modification Commands (A1, E1, N1/N2)

Database modification commands (A1, E1, and N1/N2) change, delete, or add database records and update the related Associator lists accordingly. ISNs can be assigned to new records either by the user or by Adabas.

The inverted lists and the address converter are automatically maintained by Adabas. When the user supplies a new value for a descriptor, either in a new record or as part of a record update, Adabas performs all necessary maintenance of the inverted lists. When a new record is added or a record is deleted, the address converter is appropriately updated. These operations are completely transparent to the user.

Logical Transaction Control Commands (ET/BT)

An Adabas logical transaction defines the logical start (BT) and end (ET) of the database operation being performed. If the user operation or Adabas itself terminates abnormally, these commands provide the capability for restarting a user, beginning with the last unsuccessfully processed transaction. ET/BT commands

- define the transaction start and end;
- restore pretransaction conditions if a situation occurs that prevents successful completion of the transaction; and
- read program-specified user data written during the transaction sequence.

Programs that use these commands are called ET logic programs. Although not required, Software AG recommends that you use ET logic. See page 47 for more information about transaction logic.

Special Commands

Special commands perform many of the “housekeeping” functions required for maintaining the Adabas database environment. Commands in this group

- open (OP) and close (CL) a user session (but do not control a transaction);
- write data protection information, user data, and checkpoints (C1, C3, C5); and
- set (HI) and release (RI) record hold status.

In addition, the RC command releases one or more command IDs currently assigned to a user, or deletes one or all global format IDs.

The RE command reads user data previously stored in an Adabas system file by C3, CL, or ET commands.

Complex Searches

In many large database systems, the time required to process complex searches is often excessive. Adabas efficiently solves this problem. Many Adabas applications are currently in use with up to 150 complex selection criteria created dynamically. Data is retrieved immediately from files with more than 50 million records.

Multifile Searching

It is often necessary to perform a multifile search in order to resolve an inquiry. Multifile searching can be accomplished using multiple search commands, physically coupled files, or soft file-coupling. See page 22 for information about coupled files.

Multiple search commands may be used in which a value retrieved in one command is used as the search value for the next command. This process is not restricted to two files.

Multi-index Searching

“Fuzzy” matching (i.e., retrieving data based on **similar** rather than on **exact** search criteria) can be implemented using hyperdescriptors. Hyperdescriptors allow multiple virtual indexes, meaning that several different search index entries can be made for a single data field. See page 35 for information about hyperdescriptors.

Access Methods

Adabas supports both sequential and random access methods. Different calls use different Adabas access paths and components; the most efficient method depends on the kind of information you want and the number of records you need to retrieve.

Sequential Access

Physical sequence retrieves every record in a file in the order the records are stored in Data Storage. You can limit the fields within each record for which values are to be returned. You can also specify a “start” ISN: the sequential pass then begins at the first record physically located after the record identified by the specified ISN.

Adabas bypasses the Associator and goes directly to Data Storage, reading the first data block (or the first record following a specified ISN) and continuing in consecutive sequence until the last block is read. Physical sequence is the fastest way to process a large volume of records.

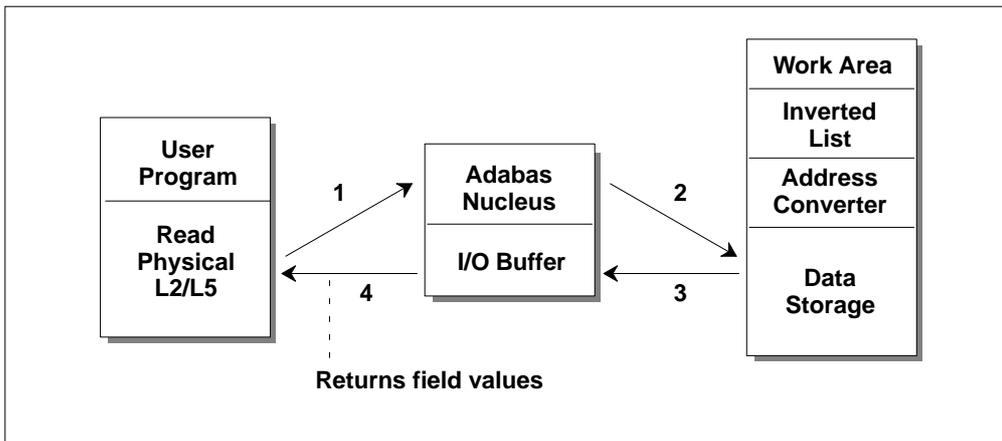


Figure 3-1: Read in Physical Sequence (L2/L5)

ISN sequence retrieves records in ISN order. Adabas uses database query commands (Sx) to build and sort ISN lists, which can then be read using L1/L4 commands with the GET NEXT option. When reading, Adabas uses the address converter to find the RABNs of each ISN in the list and then reads and returns the records from Data Storage.

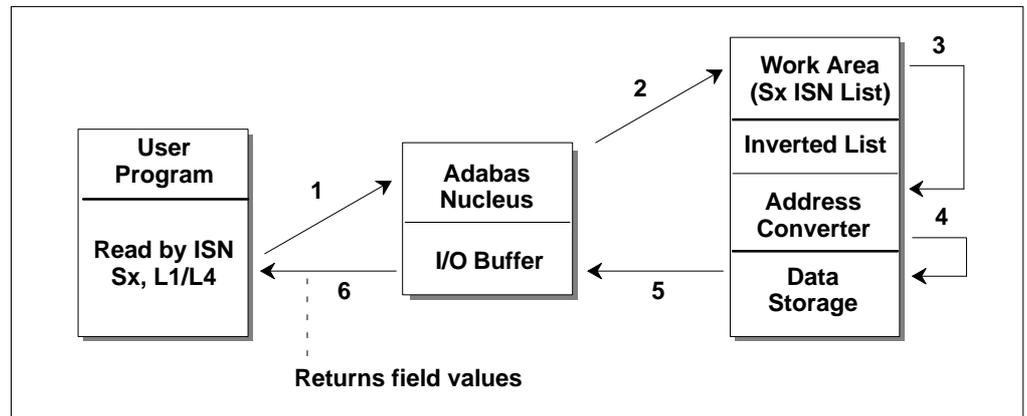


Figure 3-2: Read in ISN Sequence (L1/L4)

Logical sequence retrieves records by descriptor value. Adabas finds the value(s) in the inverted list, uses the address converter to find the RABNs of the ISNs related to the value, and retrieves the records from Data Storage.

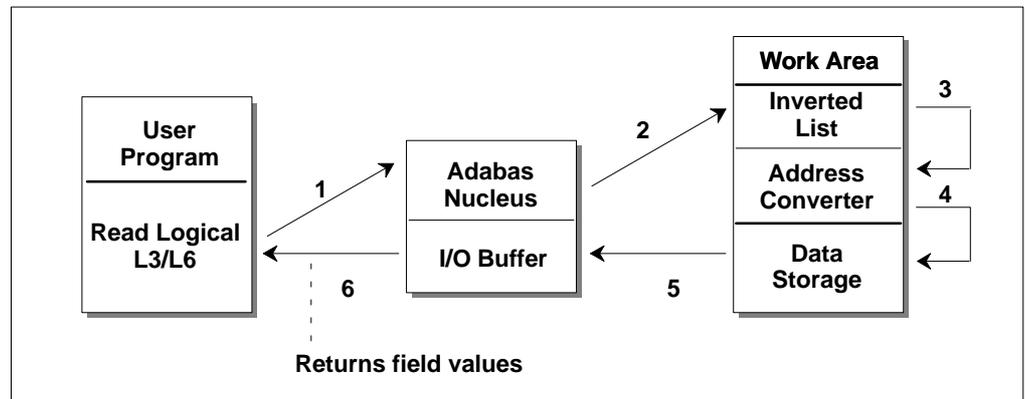


Figure 3-3: Read in Logical Sequence (L3/L6)

Reading in logical sequence retrieves all the records related to a single value or a range of values of the specified descriptor. It returns the records sorted in ascending/descending order by descriptor value, and in ascending/descending ISN order within each value. You can specify a starting and ending value. You can also specify the field(s) for which values are returned. Read logical is useful when you want the returned records sorted on a particular field.

Adabas provides a special read command (L9) to determine the range of values present for a descriptor, and the number of records that contain each value. Such a retrieval is called a “histogram”. The L9 command does not require any access to data records, only to the inverted lists stored in the Associator.

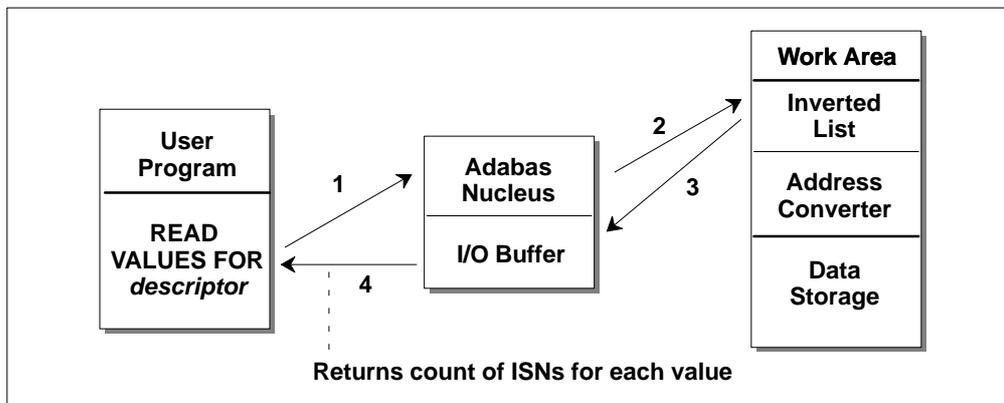


Figure 3-4: Read Descriptor Values (L9)

Random Access

Adabas uses the S1/S2/S4 commands to select records that satisfy a **search criterion**: the count of the records found and a list of their ISNs is returned. The S1/S4 commands return the ISNs in ascending sequence; the S2 command allows you to specify a sort sequence for the returned ISNs.

The search criterion may comprise

- one or more fields in a single file;
- fields contained in two or more physically coupled files; or
- search, read, and internal list matching based on the soft coupling feature.

A search criterion may contain one or more fields that are not defined as descriptors. If nondescriptors are used, Adabas performs read operations to determine which records to return to the user. If only descriptors are used within the search criterion, Adabas resolves the query by using the Associator inverted lists; no read operation is required.

Random Access Using the Adabas Direct Access Method (ADAM)

Note:

The ADAM feature is available on mainframe platforms only.

The Adabas direct access method (ADAM) improves random access performance on a particular descriptor field in a file. ADAM uses the field value to compute the relative block address (RABN) for record storage. The ADAM descriptor may also be used like any other descriptor within a selection criterion, and for logical sequential processing as well. This option may be selected for any given file when the file is loaded into the database.

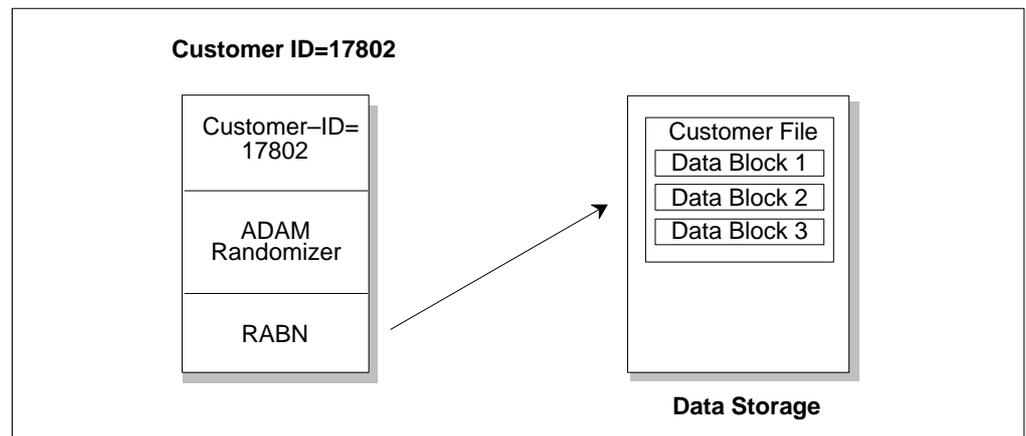


Figure 3-5: Adabas Direct Access Method (ADAM)

Using Triggers and Stored Procedures

The Adabas triggers and stored procedures facility is an integral part of Adabas; however, Natural (see page 117) is required to use the facility. The online Trigger Maintenance facility can be accessed using the selectable unit Adabas Online System (see page 85).

A procedure is a Natural subprogram that is written and tested using standard Natural facilities.

- A **trigger** is a procedure that is executed automatically by Adabas when a specified set of criteria is met. The set of criteria is determined for each command sent to Adabas and is based on the target file number and optionally the command type and/or field. The command type refers to the find, read, store, update, and delete commands. The field must be in the corresponding format buffer of the command.
- A **stored procedure** is executed by Adabas, but is invoked directly by a special user call from any of a number of applications that use it. Storing programs that are used by multiple clients in an Adabas file on the server reduces the amount of data traffic to and from the server.

The same types of parameters are passed to the subprogram whether it is a trigger or a stored procedure.

The Adabas facility for triggers and stored procedures allows you to implement and maintain both types of procedures. It resides within Adabas and provides an extension to an application. It can be used to

- implement various security and auditing features for an application; and
- provide a consistent, central environment where data can be verified or manipulated, either manually by the application or automatically by Adabas when triggers are defined.

Universal Encoding Support (UES)

Adabas provides facilities for converting data bidirectionally between ASCII and EBCDIC architectures. This makes it possible to support access to the mainframe database through the TCP/IP protocol from web-based applications or from PC-based applications such as Natural for Windows.

Adabas supports a wide range of character sets or “code pages” to handle the world’s languages. Character encoding and data conversion take place within Adabas using Unicode as the default encoding for both storage (file encoding) and user presentation (user encoding).

The client application can specify a special encoding and communicate it to the Adabas nucleus at session open (OP command). The LNKUES/ADALNK converts Adabas buffer data depending on the architecture of the caller. A number of utilities provide for special encoding and architecture settings.

Adabas supports double-byte character sets (DBCS) and multiple-byte character sets (MBCS) characteristic of Asian languages. It uses two field formats: alphanumeric and wide-character.

- alphanumeric fields are extended to support wide-character data by defining encoding keys on both the database and file levels: the file level encoding takes precedence over the database encoding. The encoding specifies the format in which the data is to be stored. It is also used as the default format in which data is exchanged with a local user.
- wide-character fields are similar to alphanumeric fields in that encoding keys are defined on both the database and file levels: the file encoding takes precedence over the database encoding. It differs from alphanumeric field encoding in that
 - if no encoding is specified, the default Unicode encoding is used.
 - the “internal” encoding specifies the format in which the data is stored.
 - the “user” encoding specifies the default format for data presented to the user.

To ensure round-trip compatibility between architectures and encodings, Adabas uses a file encoding that holds the superset of all characters defined in the “default user” and any specific “user” encodings. For wide-character fields, such a file encoding defaults to the universal character set encoding Unicode.

Collation encoding is defined for a descriptor field. Values for this encoding are obtained algorithmically by calling a collation exit programmed to produce a culturally correct sorted key; that is, a dictionary order. Collation encoding may be defined for both alphanumeric and wide-character fields; the collation encoding/exit is defined on the file level for alpha and/or wide descriptor fields.

Maintaining Database Integrity

Adabas provides facilities to ensure the logical consistency of data in a competitive updating environment and when a user or Adabas session is interrupted.

Facilities are available for both online and traditional batch update. For online, transaction-oriented processing, Adabas ensures that the database is free of incomplete transactions. For batch mode updating, Adabas ensures restart in the event of failure by writing checkpoints and backing out/regenerating updates.

Transaction Logic

Adabas data protection, recovery, and user restart are based on the concept of a “logical transaction”: the smallest unit of work (as defined by the user) that must be performed in its entirety to ensure that the information contained in the database is logically consistent.

A logical transaction may comprise one or more Adabas commands that together perform the database read/update required to complete a logical unit of work. A logical transaction begins with the first command that places a record in hold status and ends when an ET (end transaction), BT (back out transaction), CL (close), or OP (open) command is issued for the same user.

The OP (open) or RE (read ET data) commands can be used to retrieve user restart data stored by the C3, CL, or ET command. This data is also written to the Adabas data protection log with each checkpoint written by the transaction and can be read using the ADASEL utility.

The ET command must be issued at the end of each logical transaction. Successful execution of an ET command ensures that all the updates performed during the transaction are physically applied to the database, regardless of subsequent user or Adabas session interruption.

Updates performed during transactions for which ET commands are not successfully executed are backed out, either manually by issuing the BT command or automatically by the autobackout routine (see page 52).

Distributed Transaction Processing

Adabas incorporates nucleus functions to support the execution of “global” database transactions in distributed environments; that is, across multiple local or remote databases and/or system images in parallel.

A two-phase commit protocol ensures that all database management systems (DBMSs) that participate in processing the global transaction (that is, resource managers or RMs) either commit or roll back their local parts of a transaction as a whole. In the first phase, the coordinating component (a transaction manager or TM) prepares all involved RMs for the commit. Only when the first phase is successful does the TM instruct the RMs to commit (second phase).

Adabas functions in this scenario as an “RM”. Adabas Transaction Manager, a selectable unit (see page 92), functions as the coordinator within operating system images and, with the help of Entire Net-Work, across system images.

The new protocol also integrates Adabas with other DBMSs. It is transparent to existing application systems and to Natural.

Included in Adabas is a CICS-controlled interface that conforms to the CICS Resource Manager Interface (RMI). It issues the appropriate Adabas commands to coordinate with the two-phase commit protocol.

Competitive Updating

Competitive updating is in effect when two or more users (in multiuser mode) are updating the same Adabas file(s). The Adabas facilities used to ensure data integrity in a competitive updating environment include record hold/release, avoidance of resource deadlock, and exclusive control updating.

Record Hold and Release

The Adabas record hold facility ensures that a record will not be updated by more than one user at a time. A user can put the record in “hold” status (that is, the ISN of the record is put in the hold queue) using the commands S4 (find with hold), L4/L5/L6 (read with hold), A1/E1 (update/delete) with the hold option specified, N1/N2 (add record with hold), or HI (hold record).

If a record requested for hold status is already being held by another user or utility, the user issuing the record hold command is put in “wait” status until the record becomes available, at which time Adabas reactivates the command. You may request that a response code be returned if you do not want to be put in “wait” status.

Records in “hold” status can be accessed (found and read) by users who do not seek to hold the record.

Records in “hold” status are released by issuing the ET or CL commands. Options are available with ET and BT commands to release records selectively. The CL command releases all records in “hold” status for the issuing user.

Avoiding Resource Deadlock

Resource deadlock occurs when two users are each waiting for a record held by the other user. Adabas protects against such a user deadlock situation by detecting the potential deadlock and returning a response code to the second user after putting the first user in “wait” status.

Exclusive Control Updating

Users who use logical transaction commands (ET/BT) are called ET logic users.

Alternatively, a user can request exclusive control of one or more Adabas files for the duration of the user session. If the file or files for which exclusive control is requested are not already opened for update by another user or utility, exclusive control is granted and the user becomes an exclusive update (EXU) user. Adabas treats EXU users as non-ET logic users.

Adabas does not place an ISN in “hold” status for EXU users. Adabas disables hold logic processing for files being updated under exclusive file control.

Instead of using ET commands, EXU users can request checkpoints to act as reference points; for example, updates applied after a checkpoint can be removed.

Timeout Controls

Adabas times out

- transactions that exceed a specified limit; and
- users who are inactive for a specified amount of time.

Transaction Time Limit

Adabas provides a transaction duration time limit for ET logic users. The time limit is set with the ADARUN TT parameter; an override for a specific user can be set using the OP command.

If a transaction exceeds the prescribed limit, Adabas generates a BT (back out transaction) command to remove all the updates performed during the transaction and release all held records. The user can then either repeat the backed out transaction from the beginning or begin another transaction.

Non-Activity Time Limit

All users are subject to a non-activity time limit; different limits can be set for different user types and for specific users within each user type.

If a user exceeds the prescribed limit and the user is

- an ET logic user, Adabas backs out the current transaction, releases all held records and command IDs, and deletes the user's file list.
- an EXU user, Adabas deletes the user's file list and releases all command IDs. The user loses his EXU user status and becomes an access-only user.
- an access-only user, Adabas deletes the user's file list.

Backout, Recovery, and Restart

Backout, recovery, and restart may be required when a user or Adabas session is interrupted due to a timeout (see page 50); a program error when Adabas determines that a transaction cannot be completed successfully; an Adabas, hardware, or operating system failure; or a power failure.

A “user session” is a sequence of Adabas calls optionally starting with an open (OP) command and ending with a close (CL) command. A “user” is either a batch mode program or a person using a terminal. The uniqueness of each user is assured by the user ID, a machine, an address space, and a terminal ID.

An “Adabas session” starts when Adabas is activated and continues until Adabas is terminated. During this time, the Adabas nucleus creates a sequence of protection entries in exact historical sequence reflecting all modifications made in the database. The sequence of protection entries is written to the Work dataset (part 1) and to a protection log in blocks. Each block contains the nucleus session number, a unique block number, and a time stamp.

User Program Error

A user program that is processing a transaction can detect that the transaction cannot be completed successfully. In this case, a BT (back out transaction) command is used to remove or “back out” the incomplete transaction.

If a user program error causes logical damage to the database, it may be necessary to recover the affected files using the ADASAV and ADARES utilities.

Adabas, Hardware, or Operating System Failure

After any failure that causes the Adabas nucleus to terminate abnormally, an automatic procedure is executed when Adabas is reactivated to bring the database to a physically and logically valid status. All partially executed update commands are reset; all incomplete transactions are backed out.

The automatic procedure comprises three steps: repair the database, autorestart, and autobackout:

- Database repair modifies the database to the status it would have had if a buffer flush had just been completed at the time of failure. That is, all blocks in the database are at a status that enables the nucleus to perform normally.
- Autorestart backs out updates of single update commands that were partially executed when the system failed. It resolves internal inconsistencies in the database and ensures physical integrity.
- Autobackout, which is performed only for ET logic users, backs out updates of user transactions that were partially executed when the system failed. Adabas performs an internal BT (back out transaction) followed by autorestart, and then informs the user that the last transaction has been backed out.

The autobackout routine is executed at the end of an ET session that was terminated with HALT. It is also executed automatically at the beginning of the next Adabas session to remove any updates performed within transactions that did not complete successfully.

After autobackout execution, the database contains updates only from logically complete transactions.

Note:

ET users can manually back out an incomplete transaction at any time by issuing the BT (back out transaction) command. See page 47.

If an Adabas, hardware, or operating system failure results in physical damage to the database, it may be necessary to recreate the database using the ADASAV and ADARES utilities.

Power Failure

Depending on the hardware, a power failure during an I/O operation may damage the Adabas blocks that were being processed. This damage cannot be detected during autorestart and therefore can result in problems later such as unexpected response codes or lost database updates.

If the ADARUN IGNDIB=YES parameter is set, the autorestart routine checks whether a buffer flush was active when the session interruption occurred. If a buffer flush was in process, the autorestart shuts down and Adabas alerts the user to the potential problem and includes a list of the files being updated when the buffer flush was in process. The DBA must then determine whether a power failure occurred.

If the cause of a session interruption

- is a power failure, Software AG strongly recommends recovering the affected files using the ADASAV and ADARES utilities.
- is **definitely not** a power failure and the integrity of the information on the output hardware can be guaranteed, the database can be reactivated immediately. Database recovery is not necessary.

Extended Error Handling and Message Buffering

The error handling and message buffering facility helps implement 24X7 operations by analyzing and recovering from certain types of errors automatically with little or no DBA intervention. It also generates additional information so that the error can be diagnosed by the user and by Software AG.

A wrap-around message buffer collects Adabas messages for later review by Adabas Online System in case online access to the console or to DDPRINT messages becomes unavailable. The buffer aids problem analysis and performance tuning.

The error handling functions of the facility can be invoked from the operator console or from Adabas Online System.

User exits and hyperexits that are essential to the operation of the Adabas nucleus can be marked as critical (the default) or not:

- a **critical** user exit is not affected by the error handling and message buffering facility: an abnormal termination in it causes the Adabas nucleus to terminate abnormally as well.
- for a **“notcritical”** user exit, the facility maintains an active Adabas nucleus, optionally refrains from invoking that exit, takes a dump of the nucleus at the point when the exit failed, and issues messages to the system log to inform the DBA of the problem. The DBA can then examine the diagnostic information, use it to resolve the problem, then load and reactivate the corrected exit.

The extensions (plug-in routines or “PINs”) analyze and, in some cases, determine the cause of an ABEND while allowing the nucleus to continue processing. Each PIN service routine handles a predefined condition when encountered, allowing the Adabas nucleus to

- remain active when it otherwise would terminate abnormally; or
- print extended error diagnostics as an aid to error recovery.

While the PIN is executing, most Adabas functionality is available to it as the registers at the time of the abnormal event are available. The PIN determines whether it is safe to allow the nucleus to continue processing and prints appropriate messages to notify the DBA.

Based on its execution, a PIN can either transfer control to the Adabas nucleus so that it can resume normal processing—usually with a response code—or it can return control to the error handling and message buffering facility, allowing the Adabas nucleus to terminate abnormally.

A PIN can also be used to format an intelligent dump in a number of circumstances to help debug a particular response or ABEND code.

A special PIN routine user exit can be used to obtain additional information about response codes and ABENDs. The user exit allows you to specify particular response codes or response code/subcode combinations to be monitored. Once you have modified the user exit, you can reload it and make your changes effective without bringing the database down.

ADABAS UTILITIES

Database services such as loading or deleting files are handled by an integrated set of online and batch-mode “utilities”. Most utilities can be run in parallel with normal database activity to preclude interruption of daily production. See the *Adabas Utilities Manual* for more information.

Adabas utilities address initial design and load operations, backup/restore/recovery routines, database modification routines, and audit/control/tuning procedures.

Note:

See page 85 for information about Adabas Online System, a menu-driven, interactive DBA tool.

Initial Design and Load Operations

ADACMP : Compress / Decompress

ADACMP COMPRESS is used to edit and compress data records to be loaded into the database using ADALOD; ADACMP DECOMPRESS is used to decompress individual files for data structure or field definition changes, or for use as input to non-Adabas programs.

COMPRESS

Input

ADACMP input data must be in a sequential dataset/file. Indexed sequential and VSAM input cannot be used. The records may be fixed, variable, or of undefined length. The maximum input record length permitted depends on the operating system. The maximum compressed record length is restricted by the Data Storage block size in use and the maximum compressed record length set for the file (see the MAXRECL parameter, ADALOD utility). The input records can be in either blocked or unblocked format.

It is possible to omit the input dataset if the parameter NUMREC=0 is supplied.

The logical structure and characteristics of the data for input to COMPRESS are described with field definitions statements (FNDEF to define fields or groups of fields; SUBFN and SUPFN to define sub- or superfields, respectively; COLDE, HYPDE, PHONDE, SUBDE, and SUPDE to define various types of descriptors). Field definitions are used to create the Adabas field definition table (FDT).

By default, input data records are processed in the order of the field definition statements. The `FORMAT` parameter allows you to change the order of field processing or skip fields.

To support universal encoding (UES), parameters allow you to specify the data architecture and user encodings of the input and the desired file and user encodings to use during compression.

Processing

ADACMP COMPRESS edits and compresses the data records.

Editing includes checking each field defined with a “packed” (P) or “unpacked” (U) format to ensure that the field value is numeric and in the correct format. Any record that contains invalid data is written to the ADACMP error dataset and is not written to the compressed dataset. Adabas user exit 6 can be used to specify additional editing to be performed during ADACMP COMPRESS processing. See the *Adabas DBA Reference Manual* for information about user exits.

Compression includes removing trailing blanks from alphanumeric fields; removing leading zeros from numeric fields; removing trailing zeros in floating-point format fields; and packing numeric unpacked fields. Fields with the fixed (FI) option are not compressed, and empty fields located at the end of the record are neither stored nor compressed. Null value fields are processed differently depending on options being used. SQL null value processing is supported.

If universal encoding support (UES) parameters have been specified, compression includes converting the input to the specified encoding for compressed files.

Output

Processed data records are written out together with the file definition information to a sequential dataset with the “variable blocked” record format. This dataset, or several such datasets from multiple ADACMP executions, can be used as input to the ADALOD utility. The dataset can be used as input to ADALOD even if it contains no records, meaning that no records were provided on the input dataset or all records were rejected during editing.

The ADACMP processing report indicates the approximate amount of space required in Data Storage for the compressed records by device type (specified with the `DEVICE` parameter) and for Data Storage padding factors between 5 and 30 percent. The compression rate is computed based on the real amount of data used as input to the compression routine.

DECOMPRESS

ADACMP DECOMPRESS accepts as input data records from existing Adabas files, either directly without separate file unloading, or already unloaded with the ADAULD utility. If a file is directly decompressed, it is unloaded without FDT information as part of the decompression process, which can save time when decompressing larger files.

Direct decompression of multiclient files can be limited to records for a specific user only when a valid owner ID (ETID parameter) is specified.

The FORMAT parameter may be used to decompress the record to a format other than that specified by the FDT. This is particularly useful when the FDT of an existing file is to be changed.

If universal encoding support (UES) is used, the encoding characteristics for the decompressed file are passed in the header of the compressed sequential input. Parameters allow you to overwrite these encoding characteristics.

Processed data records are written to a sequential dataset with the “variable blocked” record format. Rejected data records are written to the error dataset.

ADALOD : Loader

The ADALOD LOAD function loads a file into the database. Compressed records produced by the ADACMP or ADAULD utility may be used as input. A parameter specifies whether the file index is loaded in compressed or uncompressed form.

ADALOD loads each compressed record into Data Storage, builds the address converter for the file, and enters the field definitions for the file into the field definition table (FDT). ADALOD also extracts the values for all descriptors in the file together with the ISNs of all records in which the value is present, to an intermediate dataset. This dataset is then sorted into value/ISN sequence and entered into the Associator inverted lists.

The ADALOD UPDATE function is used to add or delete a large number of records to/from an Adabas file. The UPDATE function requires considerably less processing time than the repetitive execution of the Adabas add/delete record commands. Records to be added may be the compressed records produced by the ADACMP or ADAULD utility. The ISNs of records to be deleted can be provided either in an input dataset or by using control statements.

Records may be added and other records deleted during a single execution of ADALOD.

ADAULD : Unload

The ADAULD utility unloads an Adabas file from the database or from a save tape.

Adabas files are unloaded from a database to

- permit the data to be processed by a non-Adabas program. In this case, the file must also be decompressed after unloading using the DECOMPRESS function of the ADACMP utility.
- create one or more test files, all of which contain the same data. This procedure requires that a file be unloaded, and then reloaded as a test file with a different file number.
- change the field definition table (FDT). This requires that the file be unloaded, decompressed, compressed using the modified field definitions, and reloaded. If the ADADBS utility is used to add field definitions to a file, the file does not need to be unloaded first.

The sequence in which the records are unloaded may be

physical the order in which they are physically positioned within Data Storage.

logical a sequence controlled by the values of a user-specified descriptor.

ISN ascending ISN sequence.

The unloaded record output is in compressed format. The output records have the same format as the records produced by the ADACMP utility.

Adabas files may be unloaded from a qualified database or file save tape to

- include a file from a save tape in one or another test environment.
- move a file from a save tape with one blocksize to a database with another.

Backup / Restore / Recovery Routines

ADAPLP : Protection Log / Work Print

The ADAPLP utility prints data protection records contained on the Adabas Work dataset or the Adabas data protection log. You can specify whether to print

ALL	all protection records—the default
ASSO	just Associator protection records
DATA	just Data Storage protection records
C1	records resulting from Adabas C1 commands
C5	records resulting from Adabas C5 commands
EEKZ	records written at completion of a nucleus buffer flush
ET	records resulting from Adabas ET commands
REPR	Work dataset records used by autorestart to repair the index
SAVO	records resulting from online SAVE database/file operations
VEKZ	records written at completion of update commands

The number of protection records printed can be reduced even more by specifying a file, ISN, or RABN.

ADARAI : Recovery Aid

The Adabas Recovery Aid utility ADARAI can be used to automate and optimize database recovery. See also the restart/recovery information in the *Adabas Operations Manual*.

ADARAI supports all Adabas-compatible tape management systems.

The ADARAI utility prepares the recovery log file (RLOG), which records the information about datasets, utility parameters, and protection logs needed to build the recovery job control statements. ADARAI lists the information contained in the RLOG, creates the job control statements to recover the database, and disables ADARAI logging.

Information is stored on the RLOG by generations. A “generation” includes all activity between consecutive ADASAV SAVE/RESTORE (database) and/or RESTORE GCB operations. The first generation includes the first ADASAV SAVE/RESTORE (database) or RESTORE GCB operation and extends to (but excludes) the second.

Minimally, the RLOG retains the number of generations specified by the MINGENS parameter during the ADARAI PREPARE step. However, a maximum of 32 generations will be stored on the RLOG if there is enough space available.

Systems using the Recovery Aid feature require a recovery log (RLOG) dataset DD/RLOGR1, which must first be formatted with the ADAFRM utility and then defined using the ADARAI utility.

ADARES : Restart

The ADARES utility performs functions related to database recovery:

- **BACKOUT** removes all the updates applied between two checkpoints. The checkpoints used are normally the result of a non-synchronized checkpoint command (C1) but may also be synchronized checkpoints. The complete database may be included in the back-out process, or backout may be limited to selected files.
- **CLCOPY** copies a command log dataset from disk to a sequential dataset. This function is necessary only if dual or multiple command logging is in effect for an Adabas session.
- **COPY** copies a sequential Adabas protection log dataset. This function should be executed if the Adabas session in which the sequential protection log dataset was created was terminated abnormally.
- **MERGE CLOG** manually merges command log datasets resulting from individual nucleus **CLCOPY** executions into a single command log for a cluster of nuclei.
- **PLCOPY** copies a protection log dataset from disk to a sequential dataset. This function is necessary only if dual or multiple protection logging is in effect for an Adabas session.
- **REGENERATE** reapplies all the updates performed between two user-specified checkpoints. The checkpoints specified may be the result of a non-synchronized checkpoint command (C1) but may also be synchronized checkpoints. The **REGENERATE** function may process all files or be limited to one or more files. It is most often used after the database (or one or more files) has been restored to a previous status with the **RESTORE** or **RESTONL** function of the **ADASAV** utility.
- **REPAIR** repairs one or more blocks in Data Storage that, for any reason, have become unusable. The most recent save tape of the database and any protection log tapes created thereafter are used as input to this function.

To minimize the time required to recover after a system failure, the BACKOUT, BACKOUT DPLOG or MPLOG, and REGENERATE functions of ADARES can be executed in multiple threads that simulate the original update environment with multiple commands active at one time.

ADASAV : Save / Restore Database or Files

The ADASAV utility saves and restores the contents of the database, or one or more files, to or from a sequential dataset. ADASAV should be run as often as required for the number and size of the files contained in the database, and the amount and type of updating. For large databases, ADASAV functions may be run in parallel for the various disk packs on which the database is contained.

Special ADASAV functions are available for use with the Adabas Delta Save Facility. For more information, see the *Adabas Delta Save Facility Manual*.

RESTONL functions restore from one or more SAVE datasets created while the Adabas nucleus was **active** (that is, online); RESTORE functions restore from one or more SAVE datasets created while the Adabas nucleus was **inactive** (that is, offline).

RESTONL and RESTORE have the subfunctions GCB, FILES, and FMOVE:

- Without a subfunction, RESTONL and RESTORE restore entire databases.
- With the GCB subfunction, they restore the general control block, Associator RABNs 2–30 of the database, and specified files.
- With the FILES subfunction, they restore one or more files into an existing database to their original RABNs.
- With the FMOVE subfunction, they restore one or more files into an existing database to any free space, allowing changes to extent sizes.

If changes occurred during an online SAVE, the RESTONL function is followed automatically by the RESTPLOG function. RESTPLOG applies the updates that occurred during, and therefore were not included in, the online SAVE.

RESTPLOG is also executed following a RESTONL or RESTONL FILES function that ended before the protection log (PLOG) updates were completely restored. RESTPLOG applies the database updates not applied by the unsuccessful RESTONL function.

The SAVE function to save a database or one or more files may be executed while the Adabas nucleus is active (online) or inactive (offline). If the Recovery Aid option is active, a SAVE database operation begins a new RLOG generation.

ADASEL : Select Protection Data

The ADASEL utility selects information in the Adabas sequential (SIBA), dual, or multiple (PLOG) protection log. ADASEL decompresses the information and writes it to a print dataset (DD/DRUCK) or to a user-specified output dataset.

The protection log contains information on all updates applied to the database during a given Adabas session. Information selected by ADASEL can be used for auditing or as input to a Natural or non-Adabas program.

You can select before-images, after-images, or both for new, updated, and deleted records. You can also select data written to the protection log by an Adabas C5 command.

Database Modification Routines

ADACDC : Changed-Data Capture

ADACDC is an interval-driven, asynchronous mass update feature to generate a sequential file containing all database modifications. This feature is important for open systems and data warehousing solutions.

ADACDC then processes the raw data in the sequential file to isolate the latest status of the data. The ADACDC utility

- takes as input one or more sequential protection logs; and
- produces as output a “delta” of all changes made to the database over the period covered by the input protection logs.

“Delta” of changes means that the last change to each ISN in a file that was altered during this period appears on the primary output file.

This output may be used on a regular basis as input for data warehousing population procedures so that what is applied to the data warehouse database is the delta of changes to a database rather than a copy of the entire database. This affords more frequent and less time consuming updates to the data warehouse, ensuring greater accuracy of the information stored there.

ADACNV : Database Conversion

The utility ADACNV **must** be used to perform all necessary conversions of both operating system-dependent and -independent database system structures when moving in either direction between Adabas versions including 5.2, 5.3, 6.1, 6.2, 7.1, 7.2, and 7.4.

The ADACNV utility converts (CONVERT) an Adabas database from version 5.2 or above to a higher version, and the reverse (REVERT).

To ensure database integrity, ADACNV writes changed blocks first to intermediate storage; that is, to the sequential dataset DD/FILEA. After all changed blocks have been written out to DD/FILEA, a “point of no return” is reached and the changed blocks are written to the database. If ADACNV terminates abnormally after the “point of no return”, the RESTART parameter can be used to begin the ADACNV run by reading the contents of DD/FILEA and writing them out to the database.

The TEST parameter is provided to check the feasibility of a conversion or reversion without writing any changes to the database.

ADADBS : Database Services

All ADADBS functions can also be performed using Adabas Online System (AOS). When the Adabas Recovery Aid is active, using AOS is preferable for file change operations because it writes checkpoints that are necessary for recovery operation.

ADADBS offers a variety of functions, any number of which may be performed during a single execution of the utility.

Database Functions

The ADD function adds a new dataset to the Associator or Data Storage to a maximum of five datasets for each.

The DECREASE function reduces the size of the last dataset currently being used for Associator or Data Storage. The space to be released must be available in the free space table (FST).

The DECREASE function does **not** deallocate any of the specified physical extent space. To deallocate space, you must decrease the database with the DECREASE function; save it with ADASAV SAVE; reformat the datasets with ADAFRM; and restore the database with ADASAV.

The INCREASE function increases the size of the last dataset currently being used for the Associator or Data Storage. This function may be executed any number of times for the Associator. The maximum of five Data Storage space tables (DSSTs) limits Data Storage increases to four before all five Data Storage extents must be combined into a single extent with either the REORASSO or REORDB function of the ADAORD utility.

The RENAME function changes the name assigned to a (file or) database. If a file is not specified or is specified with file number zero, the database is renamed.

The TRANSACTIONS function suspends and resumes update transaction processing; that is, it creates a quiesced state for the database that could be a recoverable starting point.

File Functions

The ALLOCATE/DEALLOCATE functions are used to allocate/deallocate, respectively, a logical extent (an address converter, Data Storage, normal or upper index) of a specific size. Only one extent may be allocated or deallocated per ADADBS execution.

The CHANGE function changes the standard length of an Adabas field but does not modify records in Data Storage. The user is, therefore, responsible for preventing references to the field that would cause invalid results because of an inconsistency between the new standard length as defined to Adabas and the actual number of bytes contained in the record.

The DELETE function deletes an Adabas file from the database. The file may not be coupled. If an Adabas expanded file is specified, the complete expanded file (the anchor and all component files) is deleted. The deletion process deallocates all logical extents assigned to the file, releasing space that may be used for a new file or for a new extent of an existing file.

The DSREUSE function determines, for a specified file, whether Data Storage blocks that become free as a result of record deletion are reused. Block reuse is originally determined when a file is loaded into the database with the ADALOD FILE function, or when the system file is defined with the ADADEF DEFINE function. In both cases, block reuse defaults to “YES”.

To support universal encoding (UES), the ENCODEF function can be used to define encodings for fields in a file that is already loaded:

- an EBCDIC file encoding for alphanumeric fields; or
- a user encoding for the wide-character fields. The file encoding of wide-character fields cannot be changed using this function.

The ISNREUSE function determines, for a specified file, whether Adabas reuses the ISN of a deleted record for a new record. If not, each new record is assigned the next higher unused ISN.

For a specified Adabas file that is not a system file, the MODFCB function modifies parameters such as file padding factors for the Associator or Data Storage; maximum size of secondary logical extent allocations for Data Storage, normal index, and upper index; maximum compressed record length permitted; and whether a user program is allowed to perform a file refresh operation by issuing a special E1 command.

The NEWFIELD function adds one or more fields to a specified Adabas file that is not a system file. The new field definition is added to the end of the field definition table (FDT). NEWFIELD cannot be used to specify actual Data Storage data for the new field; the data can be specified later using Adabas add or update commands, or Natural commands.

The ONLINVERT function allows you to invert files when online applications are active, ensuring continuous access to the files. You can add one descriptor per file per run.

The ONLREORFASSO (reorder Associator), ONLREORFDATA (reorder Data Storage), and ONLREORFILE (reorder both Associator and Data Storage) functions allow you to reorder a list of files when online applications are active, ensuring continuous access to the files. Files are reordered within their existing extents, thus increasing I/O performance as free space is recovered and the sort sequence of data records is changed according to processing needs.

The REFRESH function sets the file to “0” records loaded; sets the first extent for the address converter, Data Storage, normal index, and upper index to “empty” status; and deallocates other extents.

The RELEASE function releases a descriptor from descriptor status. All space currently occupied in the Associator inverted list for this descriptor is released. The space can then be reused for this file by reordering or by ADALOD UPDATE. No changes are made to Data Storage.

The RENAME function changes the name assigned to a file or database. If a file is not specified or is specified with file number zero, the database is renamed.

The RENUMBER function changes the number of an Adabas file that is not a system file. If the new number specified is already assigned to another file, the RENUMBER function will not execute.

The UNCOUPLE function eliminates the coupling relationship between two files.

Other Functions

The CVOLSER function prints the Adabas file extents that are contained on a disk volume specified by its volume serial number.

The DELCP function deletes checkpoint information recorded up to and including a specified date; checkpoint information recorded after the date specified is not deleted. After running ADADBS DELCP, the remaining records are reassigned ISNs to include those ISNs made available when the checkpoint records were deleted. The lower ISNs are assigned but the chronological order of checkpoints is maintained.

The OPERCOM function issues operator commands to the Adabas nucleus. Adabas issues a message to the operator, confirming command execution. In cluster environments, OPERCOM commands can often be directed to another nucleus in the cluster or to all nuclei in the cluster for execution.

The PRIORITY function sets or changes the Adabas priority of a user. A user's priority can range from 0 (the lowest) to 255 (the highest, and the default). The priority value is added to the operating system priority by the interregion communications mechanism. The user for which a priority is to be set or changed is identified by the same user ID provided in the Adabas control block (OP command, additions 1 field).

The RECOVER function recovers space allocated by rebuilding the free space table (FST). RECOVER subtracts file and DSST extents from the total available space.

The REFRESHSTATS function resets statistical values maintained by the Adabas nucleus for its current session. Parameters may be used to restrict the function to particular groups of statistical values:

- ALL (the default) resets values for the combination of CMDUSAGE, COUNTERS, FILEUSAGE, POOLUSAGE, and THREADUSAGE.
- CMDUSAGE resets the counters for Adabas direct call commands such as Lx, Sx, or A1.
- COUNTERS resets the counter fields for local or remote, physical or logical calls, format translations, format overwrites, autorestarts, protection log switches, buffer flushes, and command throw-backs.
- FILEUSAGE resets the count of commands for each file.
- POOLUSAGE resets the high-water marks for the nucleus pools such as the Work pool, the command queue, or the user queue.
- THREADUSAGE resets the count of commands for each Adabas thread.

Adabas maintains a list of the files used by each Adabas utility in the data integrity block (DIB). The DDIB operator command (or Adabas Online System) displays this block to determine which jobs are using which files. A utility removes its entry from the DIB when it terminates normally. If a utility terminates abnormally (for example, the job is cancelled by the operator), the files used by that utility remain “in use”. The RESETDIB function releases any such files and resets the DIB entries for a specified job and/or a particular utility execution.

ADADEF : Define a Database

The ADADEF utility is used to

- define a new database (DEFINE functions), including the checkpoint file,
- set database encoding defaults for a new database or modify them (MODIFY function) for an existing database
- define a new Work file (NEWWORK function) for an existing database.

Databases are defined with name, ID, components (Associator, Data Storage, and Work) with device type and size, and default encodings.

Adabas uses certain files to store system information. The checkpoint file is used to store checkpoint data as well as user data provided with the Adabas CL and ET commands. It is required and must be specified in the ADADEF DEFINE (database) function.

Before database components (Associator, Data Storage, and Work) can be defined with ADADEF, each must be formatted by the ADAFRM utility.

ADAFRM : Format Datasets

The ADAFRM utility formats the Adabas direct access (DASD) datasets; that is, the Associator, Data Storage, and Work datasets as well as the intermediate storage (temp, sort, recovery log, and dual or multiple command/protection log) datasets.

Formatting with ADAFRM comprises two basic operations: creating blocks (that is, RABNs) on the specified tracks/cylinders; and filling the created blocks with binary zeros (nulls).

Any new dataset must be formatted before it can be used by the Adabas nucleus or an Adabas utility. After increasing a dataset with the ADADBS INCREASE or ADD function, new RABNs must also be formatted.

ADAFRM also provides functions to “reset” existing Associator, Data Storage, or Work blocks to binary zeros (nulls).

More than one ADAFRM function (ASSOFRM, DATAFRM, RLOGFRM, and so on) can be performed in the same job. However, each function must be specified on separate statements.

ADAINV : Invert

The ADAINV utility is used to

- create a descriptor (INVERT function); or
- couple two files (COUPLE function).

The INVERT function

- modifies the field definition table (FDT) to indicate that the specified field is a descriptor; and
- adds all values and corresponding ISN lists for the field to the inverted list.

The newly defined descriptor may then be used in the same manner as any other descriptor. This function may also be used to create a subdescriptor, superdescriptor, phonetic descriptor, hyperdescriptor, or collation descriptor.

The COUPLE function adds a common descriptor to two files (updates their inverted lists). Any two files may be coupled provided that a common descriptor with identical format and length definitions is present in both files. A single file may be coupled with up to 18 other files, but only one coupling relationship may exist between any two files at any one time. A file may not be coupled to itself.

Note:

Only files with numbers 255 or lower can be coupled.

Changes affecting a coupled file’s inverted lists are automatically made to the other file. The DBA should consider the additional overhead required to update the coupling lists when the descriptor used as the basis for coupling is updated, or when records are added to or deleted from either file. For example, if a field used as the basis for coupling contains a large number of null values and is not defined with the NU (null suppression) option, the result may be a significant increase in execution time and required disk space to store the coupling lists.

An interrupted ADAINV operation can be restarted without first having to restore the file.

ADAORD : Reorder

Three types of functions are available within the ADAORD utility; only one function may be executed during a given execution of ADAORD.

Reorder Functions

The REORASSO function physically reorders all Associator blocks for all files; REORFASSO reorders the Associator for a single file. This eliminates Associator space fragmentation, and combines multiple address converter, normal and upper index, and Data Storage space table (DSST) component extents into a single logical extent for each component.

The REORDATA function reorders Data Storage for all files in the database; REORFDATA reorders Data Storage for a single file. This condenses extents containing only empty blocks, and also eliminates any Data Storage fragmentation caused by file deletion.

The REORDB function performs both the REORASSO and REORDATA functions in a single ADAORD execution; the REORFILE function performs both the REORFASSO and REORFDATA functions in a single ADAORD execution. The records may be reordered in the logical sequence by a descriptor, by ISN, or in the current sequence.

Restructure Functions

The RESTURCTURE functions are used to relocate a database or specified files to a different physical device.

The RESTRUCTUREDB function unloads an entire database to a sequential dataset; RESTRUCTUREF unloads one or more files to a sequential dataset. This dataset can be used as input to the STORE function.

Store Function

The STORE function loads one or more files into an existing database using the output produced by the RESTRUCTURE functions or the REORDB function.

ADAZAP : Modify Physical Database Blocks

The ADAZAP utility is used to modify physical database blocks. It can be used to

- write a checkpoint for each VER and REP it processes providing an audit trail of database modifications. SYN3F checkpoints are printed by both Adabas Online System and ADAREP and are ignored by ADARES.
- handle errors according to standard Adabas utility conventions.

Because caution is necessary when running ADAZAP:

- Software AG recommends that you have a current save tape available before running ADAZAP. If an error is encountered while running ADAZAP, it may be necessary to restore the affected file or database.
- a mastercode available only to authorized personnel controls its use. The mastercode is distributed by Software AG on written request.

Audit / Control / Tuning Procedures

ADAACK : Check Address Converter

ADAACK should only be used for diagnostic purposes. It checks

- the address converter for a specified file(s) and ISN range. It is used in conjunction with ADAICK.
- each address converter element to determine whether the Data Storage RABN is within the used portion of the Data Storage extents specified in the file control block.
- the ISN for each record in each Data Storage block within the specified ISN range to ensure that the address converter element for that ISN contains the correct Data Storage RABN.

ADADCK : Check Data Storage

ADADCK should only be used for diagnostic purposes. It checks the Data Storage and the Data Storage space table (DSST) of a specific file (or files) in the database.

ADADCK reads each used Data Storage block (according to the Data Storage extents in the file control block) and checks whether:

- the block length is within the permitted range ($4 \leq \text{block length} \leq \text{physical block size}$).
- the sum of the lengths of all records in the Data Storage block plus 4 equals the block length.
- any record exists with a record length greater than the maximum compressed record length for the file or with a length ≤ 0 .
- any duplicate ISNs exist within one block.
- the associated DSST element contains the correct value. If not, the DSST must be repaired (see REPAIR parameter).

ADAICK : Check Index and Address Converter

ADAICK should only be used for diagnostic purposes. It checks the physical structure of the Associator. This includes validating the index based upon the descriptor value structures and the Associator extents defined by the general control block (GCB) and file control block (FCB).

ADAICK can

- check index and address converter for specific files;
- print/dump the contents of any Associator or Data Storage block in the database; or
- produce a formatted print/dump of the contents of the GCB, FCBs, and FDTs.

ADAMER : ADAM Estimation

The ADAMER utility produces statistics that indicate the number of Data Storage accesses required to find and read a record when using an ADAM descriptor. This information is used to determine

- whether the number of accesses required to retrieve a record using an ADAM descriptor would be less than the standard Adabas accessing method;
- the amount of Data Storage space required to produce an optimum distribution of records based on the randomization of the ADAM descriptor.

The input data for ADAMER is a dataset containing the compressed records of a file produced by the ADACMP or ADAULD utility.

The field to be used as the ADAM descriptor is specified with the ADAMDE parameter. A multiple value field or a field contained within a periodic group may not be used. The ISN assigned to the record may be used instead of a descriptor as the basis for randomization (ADAMDE=ISN).

The ADAM descriptor must contain a different value in each record, since the file cannot be successfully loaded with the ADAM option of the ADALOD utility if duplicate values are present for the ADAM descriptor. The ADAMER utility requires a descriptor field defined as unique (UQ), but does not check for unique values; checking for unique descriptor values is done by the ADALOD utility when loading the file as an ADAM file.

The BITRANGE parameter may be used to specify that a given number of bits are to be truncated from each ADAM descriptor value before the value is used as input to the randomization algorithm. This permits records containing ADAM descriptor values beginning with the same value (for example, 40643210, 40643220, 40643344) to be loaded into the same physical block in Data Storage. This technique can be used to optimize sequential reading of the file when using the ADAM descriptor to control the read sequence, or to remove insignificant information such as a check digit.

ADAREP : Report

The ADAREP utility produces a status report that provides information concerning the current physical layout and logical contents of the database or a qualified save tape.

The information provided in this report includes

- a database overview: the database name, number, creation date/time, file status, and current log number.
- current space resources for Associator, Data Storage, and Work: amount and locations of currently used space, and allocated but unused space.
- summary and detailed file information: summary by file of ISN, extent, padding factor, used/unused Associator and Data Storage space, and file options; and detailed, optionally by file, that includes all summary information plus MINISN/MAXISN settings, detailed space information, creation and last use date/time, field definition table (FDT) contents, and general or extended checkpoint file information.
- checkpoint information: general and extended checkpoint file information.
- physical structure: Associator/Data Storage RABN information including device type, VOLSER number, file number (if appropriate), and usage (AC, NI/UI, Data Storage, DSST, or unused).

The purpose of the save tape report is to determine what the save tape contains.

ADAVAL : Validate the Database

The ADAVAL utility validates any or all files within an Adabas database except the checkpoint and security files.

ADAVAL compares the actual descriptor values contained in the records in Data Storage with the corresponding values stored in the Associator to ensure that the Associator and Data Storage are synchronized, and that there are no values missing from the Associator.

Before running ADAVAL, the consistency of the inverted lists should be checked with the ADAICK utility.

ADAPRI : Print Selected Adabas Blocks

The ADAPRI utility prints the contents of a block (or range of blocks) contained in the Associator, Data Storage, Work, temp, sort, dual or multiple command log (CLOG), dual or multiple data protection log (PLOG), the recovery log (RLOG), or the Delta Save images (DSIM) dataset.

ADABAS SECURITY

Adabas provides the following facilities to prevent unauthorized access to and/or updating of Adabas database files:

- Adabas data encryption (ciphering) which provides data security;
- Adabas multicient files to control access to records in a file;
- Adabas Security and the related security utility ADASCR, a selectable unit, which provides selective user access/update protection at a file, field, and field value level; and
- Adabas SAF Security (ADASAF), a selectable unit, which provides control of Adabas resources at a database/utility, command, or file level through standard security packages based on the System Authorization Facility (SAF) such as RACF, CA-ACF2, and CA-Top Secret. ADASAF is initially available for OS/390, z/OS, and OS IV/F4 (FACOM) operating systems only.

Note:

It is planned that Adabas SAF Security will extend support to all supported operating system in a subsequent release of Adabas.

Security is accomplished by comparing passwords and authorization levels.

Data Encryption

Data encryption is an integral feature of Adabas and requires no options or extra modules. Data may be enciphered before being placed in the database.

The user must provide the cipher key at the time records are stored. This key is not stored and must be available to request or decipher the data. This minimizes the chances of data being compromised by unauthorized access to the system.

To retain maximum control over cipher codes, an Adabas user exit program can be created to insert the currently valid cipher code into user applications; this removes the need to make the codes known to users, and protects the file from corruption that can occur by adding data that is encrypted with the wrong cipher code.

Multiclient Files

Also available as an integral feature of Adabas that requires no options or special modules is the multiclient file.

A single Adabas physical file defined as “multiclient” can store records for multiple users or groups of users. The multiclient feature divides the physical file into multiple logical files by attaching an internal owner ID to each record.

The owner ID is assigned to a user ID. A user ID can have only one owner ID, but an owner ID can belong to more than one user. Each user can access only the subset of records that is associated with the user’s owner ID.

Note:

For any installed external security package such as RACF or CA-Top Secret, a user is still identified by either Natural ETID or LOGON ID.

All database requests to multiclient files are handled by the Adabas nucleus.

Adabas Security and ADASCR

Access/update control is available only with Adabas Security and the related security utility ADASCR that defines and controls Adabas Security functions.

Adabas Security provides two levels of protection: access/update and value.

Access/Update Level Protection

“Access-/update-level” protection applies a basic level of security on a file-by-file basis. Access/update protection can be defined for some files and not for others. It restricts use of a file or field within the file to those having an appropriate access/update profile definition and a password specified by the user of the file.

Access/update permission values ranging from 0 to 14 are defined for each user and attached to that user’s password, and each protected file (and selected field or fields, if desired) has equivalent access/update “threshold” protection values of the same range. Only a user whose permission value equals or is greater than the protection level of the specified file (and, when applicable, field) is permitted to perform that operation type (access or update) on the file or field. An access/update permission level of 0 only allows access/update of unprotected files or fields with protection level 0 or no defined protection password.

Value Level Protection

“Value-level” protection applies restrictions on the type and range of values that can be accessed or updated in specific fields. The restrictions are applied according to user password (files with fields using value-level protection must be password-protected), can be for specific values or for value ranges, and can be either “accept” or “reject” criteria.

Adabas Interface to SAF-based Packages

The System Authorization Facility (SAF) is used by OS/390 and compatible sites to provide rigorous control of the resources available to a user or group of users. Compatible security packages such as IBM’s RACF, Fujitsu’s RACF executing under MSP, and Computer Associates’ ACF2 or Top Secret allow the system administrator

- to maintain user identification credentials such as user ID and password; and
- to establish profiles determining the datasets, storage volumes, transactions, and reports available to a user.

Generally, a security package allows the system administrator to authorize a user’s access to system resources. The security package then monitors all users and their resource usage to ensure that no unauthorized access or change occurs. Attempts by unauthorized users to use either the system or specific system resources are recorded and reported.

A user profile, which can be for a single user or a group of users, defines which system hardware and software resources a user is allowed to use. A resource profile defines access/update privileges for one or more devices, volumes, and/or programs (resources that must be used together to perform certain functions can be defined together in the same profile).

When a user logs on to the system, the security package uses the user’s logon ID to identify that user’s profile. Each time the user attempts to perform a task or access information, the security package uses information in its resource profiles to allow or deny access. Using the profile concept, the security package expands the single point of authorization—the logon ID—to provide extensive control over all system resources.

The resulting security repository and the infrastructure to administer it represent a significant investment. At the same time, the volume of critical information held by a business is constantly growing, as is the number of users referencing the data. The challenge of controlling these ever-increasing accesses requires a solution that is flexible, easy to implement and, above all, one that safeguards the company’s investment.

Adabas SAF Security (ADASAF)

Adabas SAF Security (ADASAF) enhances the scope of SAF-based security packages by integrating Adabas resources into the central security repository. ADASAF enables

- a single control and audit system for all resources;
- industry-standard protection of Adabas data; and
- maximized return on investment in the security repository.

ADASAF operation can be tailored on a nucleus-by-nucleus basis, allowing great flexibility in its implementation. It comprises

- a server operating in each secured Adabas address space;
- router extensions linked with the Adabas SVC;
- an online administration and monitoring system, an application written in Natural and accessed from either the demo or full version of Adabas Online System (AOS); and
- a plug-in routine PINSAF that interfaces with the Adabas error handling facility. It is activated automatically at initialization to aid problem diagnosis.

ADASAF allows you to protect the following Adabas resources:

Resource	Protection
Database Nucleus	Controls the users allowed to start an Adabas nucleus.
Adabas Utilities	Controls the users allowed to execute utilities by utility or database ID; for example, a user or group might be allowed to run ADAREP but not ADASAV against a particular database.
Database Files	Controls the users allowed to access database files.
Database Commands	Controls the users allowed to use access (READ/FIND) and update (STORE/UPDATE/DELETE) commands. To optimize performance, ADASAF disregards commands such as RC that are not file-specific.
Production Environment Data	Controls the users allowed to operate in a production or test environment. Such “cross-level” checking could be used, for example, to prevent damage by an application program inadvertently cataloged against the wrong database ID.
Transaction Data	Controls the users allowed to store or retrieve ET data.
Adabas Operator Commands	Controls the Adabas operator commands that can be issued from the system console.
File Passwords and Cipher Codes	Dynamically applies passwords and codes held in the security repository or supplied by a user exit. This eliminates the need for the application to manage security data and removes the requirement to transmit sensitive information from the client to the database.
Adabas Basic Services	Protects Adabas Basic Services at a selected level (main functions only or main functions and subfunctions) with defined resource profiles and controls user access to those profiles.

In Figure 5-1, all traffic between database users and Adabas is controlled by the Adabas router. With ADASAF installed, the ADASAF router replaces the Adabas router and controls all access to Adabas:

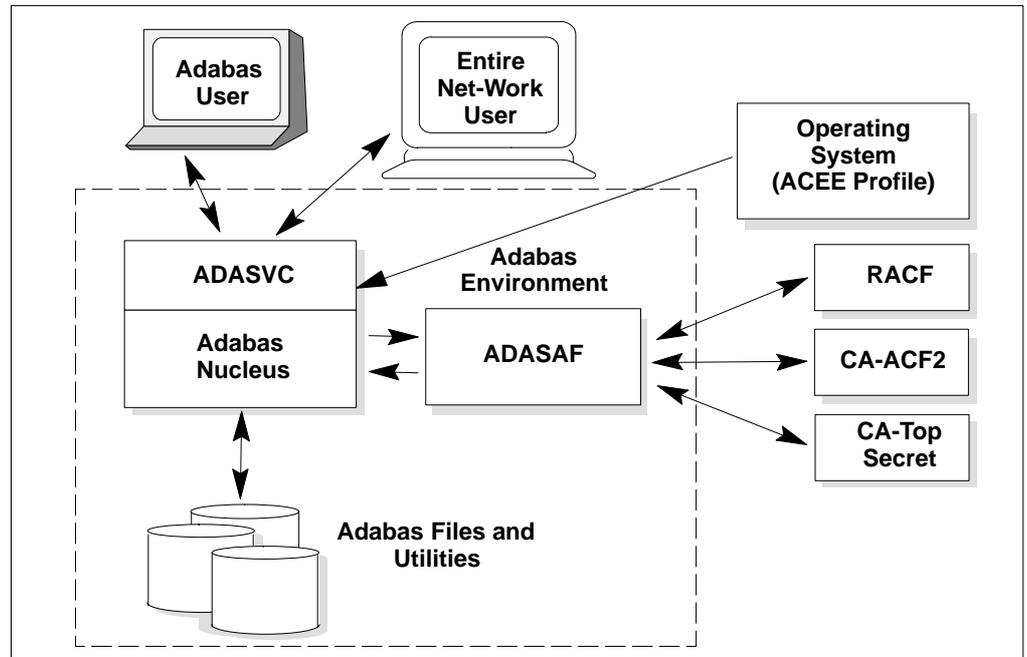


Figure 5-1: Adabas with ADASAF

The central security logon ID is used to log on to the system. Through the operating system or TP monitor, the installed external security package checks the authorization of the logon ID. For calls from a remote workstation or non-IBM platform, a remote logon procedure is used to give the logon ID to ADASAF. The router contains a security exit that extracts the user's logon ID from the ACEE for the user.

Full, flexible control is maintained with a **one user : one definition** approach while previous investments in host-based security systems and infrastructures are enhanced, not discarded.

Related Security Options

Adabas Online System Security

The demo version of Adabas Online System (AOS) distributed with Adabas includes a security facility for restricting access to the Adabas online facilities. AOS Security requires Natural Security as a prerequisite. See the *Adabas Security Manual* for more information.

Natural Security

The Natural Security system provides extensive security for Adabas/Natural users. It is required for AOS Security and recommended for other features of Adabas. See the *Natural Security Manual* for more information.

Using the SAF Repository to Secure Software AG Products

Adabas SAF Security or ADASAF (see page 78) is one of several Software AG security products that enhance the effectiveness of the SAF central security repository:

Product	Protects
Adabas SAF Security	Adabas
Adabas SQL Server SAF Security	Adabas SQL Server
Entire Net-Work SAF Security	Entire Net-Work version 5.6 and above
EntireX Security	EntireX, Entire Broker, Broker Services
Natural SAF Security	Natural

Entire Security SAF Gateway

Entire Security SAF Gateway can be installed under OS/390, MVS/ESA, MSP F4 EX and AE. Version 4.1.1 can be used to secure the following when using a SAF-compatible security system:

- Adabas (version 6.2 and earlier) for mainframes
- Adabas SQL Server operating in MVS environments
- Natural RPC using Entire Broker

- Natural for mainframes
- Entire Broker (pre-EntireX) operating in MVS, UNIX, and Windows environments (the security built into EntireX now protects Entire Broker and Broker Services as well).
- Entire Net-Work version 5.5 and lower (the Entire Net-Work SAF Security Interface is used for Entire Net-Work version 5.6 and above; see page 83).
- API Facility for Windows and UNIX applications

SAF Gateway protects client/server, peer-to-peer, and standard application systems. The software is implemented at specific points where communication between clients, servers, and peers is secured using definitions made in the SAF-based security system.

SAF Gateway comprises at least two separate components in any implementation:

- The main component, referred to as the SAF Gateway started task, operates in its own MVS address space as a gateway to SAF-based security systems. As a node in the Software AG network, it focuses SAF-based processing for the products being protected. The SAF Gateway started task can operate in combination with an existing Adabas database.
- The second component depends on what is being protected and represents the various different distributed and mainframe scenarios listed above. It can be located in the application software itself; for example, mainframe Natural. Distributed applications are protected by authenticating clients/servers and securing the communication between different components.

The API facility for Windows and UNIX applications is already being used to secure

- Web Servers operating under Windows NT and UNIX
- Visual Basic applications under Windows
- PowerBuilder applications under Windows
- Delphi applications under Windows
- C and C++ applications in Windows and UNIX

Entire Net-Work SAF Security (NETSAF)

The Entire Net-Work SAF Security (NETSAF) is a separate, optional product for OS/390 and z/OS environments running Entire Net-Work version 5.6 or above. It allows Entire Net-Work clients to access SAF-secured data sources (targets); for example, Adabas, Adabas SQL Server, Entire Broker, and Entire System Server.

NETSAF can be activated on a link-by-link basis. If only one node of several communicates externally, security can be activated for that node alone and only for external links.

To secure Entire Net-Work, it is necessary to define resource profiles in the SAF repository. Resource profiles are defined for each host target. Adabas resource profiles can be defined at the file level. The command type determines the access level required for successful authorization: valid access levels are READ, UPDATE, and CONTROL. CONTROL applies to AOS commands, for example.

Point-of-access verification of incoming requests is made against the SAF-based central security repository: all access from mainframe clients can be verified against the same security profile.

Security checks are based on a trusted user ID, which must exist in the central security repository. In some cases, the user ID is authenticated in the caller's home environment or is fixed by, for example, the Entire Net-Work configuration. A user ID can be lost if calls are routed through an intermediate gateway node.

OPTIONAL EXTENSIONS

The selectable units discussed in this chapter are available to Adabas customers who have exercised a separate purchase agreement for the feature or product.

Adabas Online System

Note:

Adabas includes a demo version of the Adabas Online System to illustrate its capabilities and to provide access to selected other services.

Adabas Online System (AOS) provides an online database administration tool for Adabas. The same functionality is available using a batch-style set of utilities.

AOS is an interactive menu-driven system providing a series of services used for online Adabas database analysis and control. These services allow a database administrator (DBA) to

- display Adabas user statistics, monitor and control access and operation of one or all users;
- display and modify Adabas fields and files: add fields, allocate and remove file space, change file and database layout, view and remove field descriptors;
- restrict file use to utility users only, or lock/unlock file access completely.

AOS is written in Natural, Software AG's fourth generation application development facility. AOS security functions are available only if Software AG's Natural Security is installed and operating.

AOS includes functions that are comparable to the Adabas operator commands and utilities.

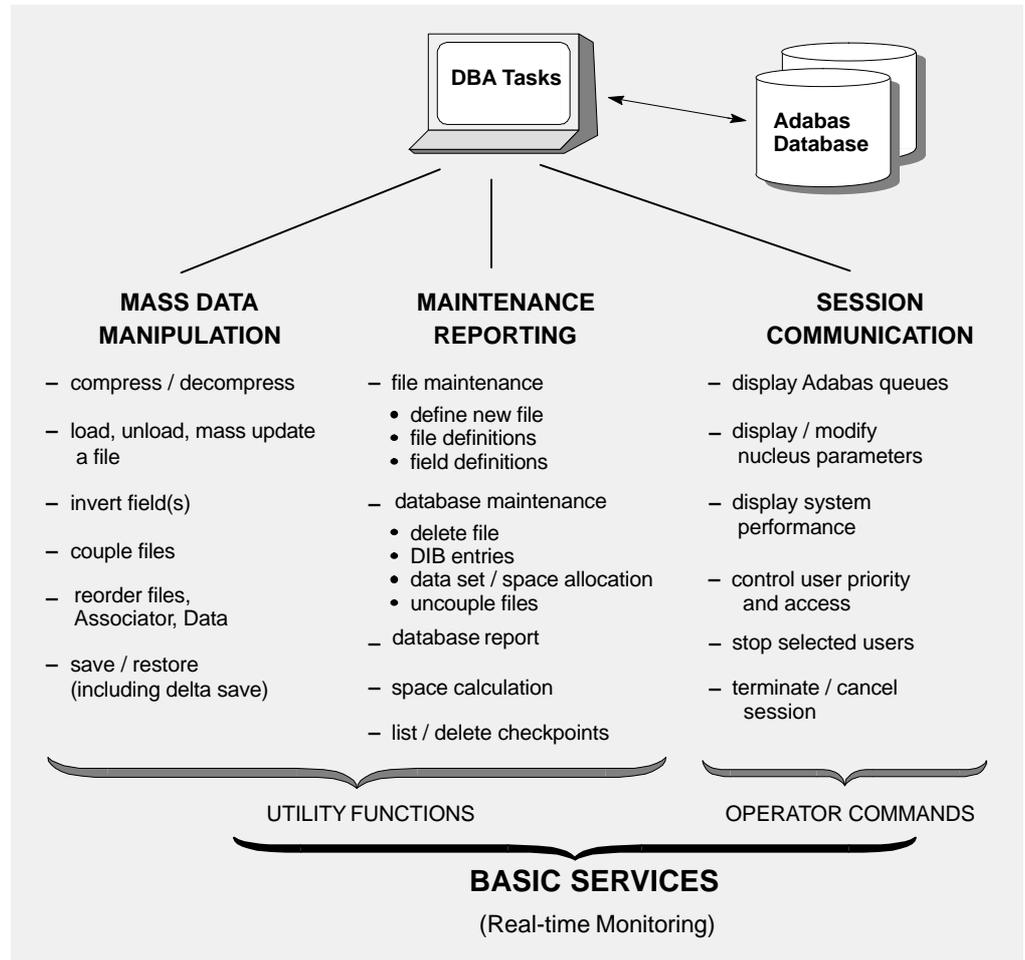


Figure 6-1 Overview of Adabas Online System/Basic Services

Basic Services makes it possible for the DBA to interactively monitor and change aspects of an Adabas database while an Adabas session is active. Using menu options or direct commands, the DBA can view resource status and user queues, display and revise space allocation, change file and database parameters, define a new file online, and stop a selected user or current Adabas session.

Adabas Caching Facility

Adabas Caching Facility helps improve system performance and make full use of ESA functions by augmenting the Adabas buffer pool in extended memory, data space, hiperspace and, in z/OS version 1.2 and above environments, 64-bit virtual storage.

Adabas Caching Facility augments the Adabas buffer manager by reducing the number of read Execute Channel Programs (EXCPs; UPAM SVCs for BS2000) to the database. This allows you to use the available operating system facilities without monopolizing valuable virtual memory resources.

Note:

Write EXCPs are always issued to maintain the integrity of the database.

Adabas Caching Facility is functionally similar to the Adabas buffer manager, but offers the following additional capabilities:

- User-specified RABNs (blocks) can be cached or “fenced” to make them readily accessible when demand arises, even though the activity against them is not sufficient to keep them in the active buffer pool. RABN-fencing reduces the required I/O response time if the Adabas nucleus needs to reread those RABNs.
- The Adabas Work dataset parts 2 and 3 can be cached to improve performance in environments that service large numbers of complex queries. Adabas Work parts 2 and 3 serve as temporary work areas used to resolve and maintain the ISN lists of complex queries. Reducing the number of read and write EXCPs to Work parts 2 and 3 for these complex queries can decrease processing time dramatically and improve performance substantially.
- A file or range of files can be specified to cache all associated RABNs. It is also possible to cache only Associator or Data Storage blocks if required. Files can be prioritized by assigning a “class of service” which determines the percentage of the maximum available cache space that a given file can use and when the file’s RABN blocks will be purged from the cache.
- Operator commands are available to dynamically respond to a changing database environment by modifying
 - the RABNs to be cached by RABN range, file, or file range;
 - the RABNs to be enabled or disabled by RABN range, file, or file range;
 - when to acquire and release the system resources used by the Adabas Caching Facility.

- When processing serial Adabas commands (e.g., read logical, read physical, histogram, and searches using non-descriptors) with the read-ahead caching option, a single EXCP is issued to read all the consecutive ASSO and/or DATA blocks that reside on a single track of the disk device. The blocks are kept in cache and are immediately available when the nucleus requests the next block in a sequence. This feature may enhance performance by reducing the number of physical read I/Os for a 3380 ASSO by as much as 18:1.

The integrity of the database is preserved because Adabas RABNs are not kept redundantly in both the Adabas buffer pool and the cache. An Adabas RABN may reside either in the Adabas buffer pool or in the cache area, but never in both. All database updates and consequently all buffer flushes occur only from the Adabas buffer pool. Unlike other caching systems, this mechanism of non-redundant caching conserves valuable system resources.

The demo or full version of Adabas Online System is required to use the online cache-maintenance application Cache Services. For more information, see the *Adabas Caching Facility Manual*.

Adabas Delta Save Facility

The Adabas Delta Save Facility (DSF) offers significant enhancements to ADASAV utility processing. It reduces the volume of save output produced and shortens the duration of save operations; this increases database availability. By allowing more frequent save operations to be performed, it also reduces database recovery time.

DSF achieves these objectives by saving only those Associator and Data Storage blocks that have changed (delta portion) since the last save operation. The result of this operation is called a **delta save tape**. Because a much smaller volume of output is written to delta save tapes, contention for secondary (tape, cassette etc.) storage is reduced.

DSF can

- maintain a log of changed database blocks (RABNs);
- create and merge interim “delta” save tapes while the database remains online, if required;
- consolidate delta save tapes with the most recent database save tape to create an up-to-date full save tape;
- restore the database from the most recent full save tape and all subsequent delta save tapes.

DSF is intended for Adabas sites with one or more large, heavily updated databases that need to be available most of the time. It is particularly beneficial when the volume of data changed on a day-to-day basis is considerably smaller than the total database volume.

The demo or full version of Adabas Online System is required to use DSF. For more information, see the *Adabas Delta Save Facility Manual*.

Adabas Fastpath

Adabas Fastpath optimizes response time and resources by bringing reference data closer to the user, reducing overhead, and reducing response times by servicing requests locally (that is, within the same region or partition).

Fastpath satisfies an Adabas query from within the application process, thus avoiding the operating system overheads needed to send a query to and from the database. Database activity such as command queue processing, format pool processing, buffer pool scanning, and decompression are also avoided.

Fastpath uses a query sampler to efficiently identify

- the most commonly issued direct access queries; that is, queries where the client identifies the data being sought (e.g., ISN, search value).
- sequential access queries; that is, queries where the client identifies a series of data items related by sequence or search criteria.

The query sampler reports interactively and at shutdown the exact types of queries that can be optimized and their relative popularity.

For each type of query, Fastpath uses algorithms to recognize and retain the most popular data and discard or overwrite the least popular data. Given a particular amount of memory to use that is available to all clients within an operating system, Fastpath retains the results of popular data queries so that they can be resolved in the client process when repeated. The retained results comprise a common knowledge base that reflects the experience gained from past queries. The knowledge base is dynamic in that it is continually updated; the least popular data held there is discarded or overwritten.

A Fastpath component attached to the DBMS ensures that any changes to popular data are reflected in the results returned to the knowledge base. Fastpath data is always consistent with the DBMS data.

Before a query is passed to the DBMS, the Fastpath optimizer attempts to resolve the query from the knowledge base. If successful, the query is satisfied faster, without interprocess communication or DBMS activity. Fastpath optimizes sequences by dynamically applying Adabas prefetch read-ahead logic to reduce DBMS activity. As many as 256 data items can be retrieved in a single visit to the DBMS.

Fastpath optimization occurs in the client process, but requires no change to application systems. Different optimization profiles can be applied automatically at different times of the day. Once started, the Fastpath buffer can be left active without intervention; Fastpath reacts automatically to DBMS startup and shutdown.

Adabas Vista

Adabas Vista allows you to “partition” data into separately managed files without reconstructing your business applications, which continue to refer to one (simple) Adabas file entity even though the physical data model is partitioned and possibly distributed across a wide-ranging computer complex.

Data can be partitioned across multiple Adabas database services. When a large file is partitioned across two or more databases, the processing load is actually being spread across the computer service. With more than one CPU engine in your computer, greater use is made of the parallel availability of the CPU engines.

Adabas Vista partitions are truly independent Adabas files:

- partitions need not be identical. Provided all the partitions support all of the views to be used by Adabas Vista, the files can operate with different physical layouts (FDTs). Of course, the Adabas source fields that are common to all partitions must be defined identically in each FDT.
- partitions can be maintained individually. You can size, order, and restore according to the needs of the individual partition. You do not have to make all partitions operate from the same physical constraints. You may choose some to be large, others medium, etc. You can also tune the ASSO space according to the partition.

You can select the applications for which Adabas Vista provides a single file image for all the partitions. You can also set an application for “mixed access mode” so that a program can access a partition directly by its real file number, even while using the single file image.

A file is usually partitioned based upon the overall dominance of a key field such as location or date: the “partition criteria”. However, it is possible to partition a file without a partition criteria.

Applications generally access the file with search data based to some extent on the key field. Adabas Vista minimizes processing overhead by detecting access explicitly or implicitly based upon the partition criteria, interrogating the search argument, and directing the access to the specific partition(s) needed. This is referred to as “focused access”.

The partition outage facility of Adabas Vista allows you to control what happens when a partition becomes unavailable. You can set sensitivity to partition outage unilaterally and allow business application to override it on a user basis. For example, if your partition criteria is location and only data in a particular location is critical to users in that location, you can set partition outage so that users are interrupted by outages in their own location but unaffected by outages in other locations. This can greatly increase the overall availability of your data, which can significantly enhance the effectiveness of your business.

The restricted partition facility of Adabas Vista allows you to “hide” partitions even though the data is available. You can use this facility to limit data to particular users based on role, location, or other business definition for security or performance reasons.

The consolidation facility of Adabas Vista allows you to impose a single file image upon multiple, previously unrelated files. The files may well be different but they support the same consolidated view.

Adabas Vista can be used in IBM mainframe environments (OS/390, z/OS, VM/ESA, z/VM, VSE/ESA) with all supported versions of Adabas.

Adabas Vista supports Adabas calls from 3GL programs as well as from Natural. An online services option is available in a Natural environment.

Adabas Vista comprises a stub (client) part and a server part. By design, most processing occurs in the client process rather than the server to

- minimize CPU usage;
- minimize the impact of overhead associated with partitioning on the database service; and
- spread the load among as many CPU engines in parallel or even computers as possible.

Adabas Transaction Manager

Adabas Transaction Manager (ATM) is a selectable unit of Adabas that

- coordinates changes to Adabas databases participating in a “global” transaction (see page 125);
- (function not available in first release) processes two-phase commit directions from transaction managers that take a higher-level, controlling role in coordinating global transactions such as IBM’s RRMS or the CICS Syncpoint Manager allowing transactions to encompass both Adabas and non-Adabas databases operating within a single operating system image; and
- plays a key role in coordinating global transactions that change Adabas databases on more than one system image. In this case, the communication mechanism between the components across the system images in Entire Net-Work.

Each ATM instance (one per operating system image) executes in its own address space as a special kind of Adabas nucleus. Each ATM is aware of and “partners” with the other ATMs in the distributed system and the databases they coordinate. At any time, each ATM can account for the status of the global transactions it is coordinating.

Adabas Review

Adabas Review (formerly Review Database) provides a set of monitoring, accounting, and reporting tools that enable you to monitor the performance of the Adabas environment and the applications executing within them.

Information retrieved about Adabas usage helps you tune application programs to achieve maximum performance with minimal resources.

In addition to the “local” mode with Adabas Review running in the Adabas address space, Adabas Review offers the “hub” mode, a client/server approach to the collection of performance data for Adabas:

- the Adabas Review interface (the client) resides on each Adabas nucleus.
- the Adabas Review hub (the server) resides in its own address space, partition, or region.

The Hub Server

The Adabas Review hub is the data collector and the reporting interface for the user. The hub handles the data consolidation and reporting functions for monitoring an Adabas database, including usage information related to applications, commands, minimum command response time (CMDRESP), I/O activity, and buffer efficiency.

An interactive reporting facility allows you to pinpoint problems quickly, providing detailed and summary data about Adabas activities. Specific information about each database is also available.

Proven Adabas and Review components are combined in the centralized collection server (hub) with the following advantages:

- A single hub can collect information from multiple Adabas nuclei and from Adabas nucleus clusters managed by either Adabas Parallel Services or Adabas Cluster Services. This means that the number of Adabas Review nuclei required to support an enterprise-wide distribution of Adabas nuclei is minimized, resource requirements are minimized, and performance increases.
- Removing the Review subtask from the address space, partition, or region of each Adabas nucleus improves the performance of the Adabas main task. At the same time, the isolation minimizes the impact of future Adabas releases on the functioning of Adabas Review.

The hub comprises

- ADAREV, a logic module that manages and supervises the incoming Review data calls and requests;
- REVHUB, a module to establish and maintain the environmentals for Adabas Review; and
- the Review nucleus and subsystems including RAOSAUTO, the autostarted report parameter generation routine, and RAOSHIST, the historical data population routine.

The Interface Client

The Adabas Review interface constructs and then transmits the Review data from the Adabas nucleus to the Adabas Review hub. An Adabas Review interface is integrated with each Adabas nucleus that is monitored.

The interface utilizes the existing Adabas interregion communication process; that is, ADALNK, ADASVC, and ADAMPM. This communication process is consistent across supported platforms.

When all supported platforms and systems are networked correctly, Adabas Review supports a multiple platform, multiple operating system, Adabas database environment.

The interface comprises the following:

- ADALOG, the Adabas command logging module;
- RAOSDAEX, the Adabas Review command log extension module that is responsible for acquiring additional information not present in the Adabas command log record; and
- ADARVU, which handles the environment conditions for RAOSDAEX and the Adabas API requirements for transmitting the Review data to the Adabas Review hub.

Interface Calls

To maximize performance, the ADARVU module issues an “optimistic” call from an Adabas nucleus to the Adabas Review hub without waiting for a completion or “post” from the hub; ADARVU assumes that the Review data was successfully passed to the hub.

However, ADARVU does perform an initialization step to ensure that the hub is active prior to any command processing by the Adabas nucleus. If the hub is not active, ADARVU informs you using WTOs and/or a user exit. If a user exit is used, you are given the option to wait for the hub to be activated, or continue initialization and call the hub only when it is active.

On the hub side of the call, the elimination of the cross-memory “post” call enhances performance by reducing the overhead of active communication with the Adabas clients. This allows the hub to remain a passive data collector.

Example Client/Server Environment

Figure 6-2 shows the major components of the Adabas Review interface (Adabas nucleus address space) and the Adabas Review hub (Adabas Review hub address space) in a client/server architecture.

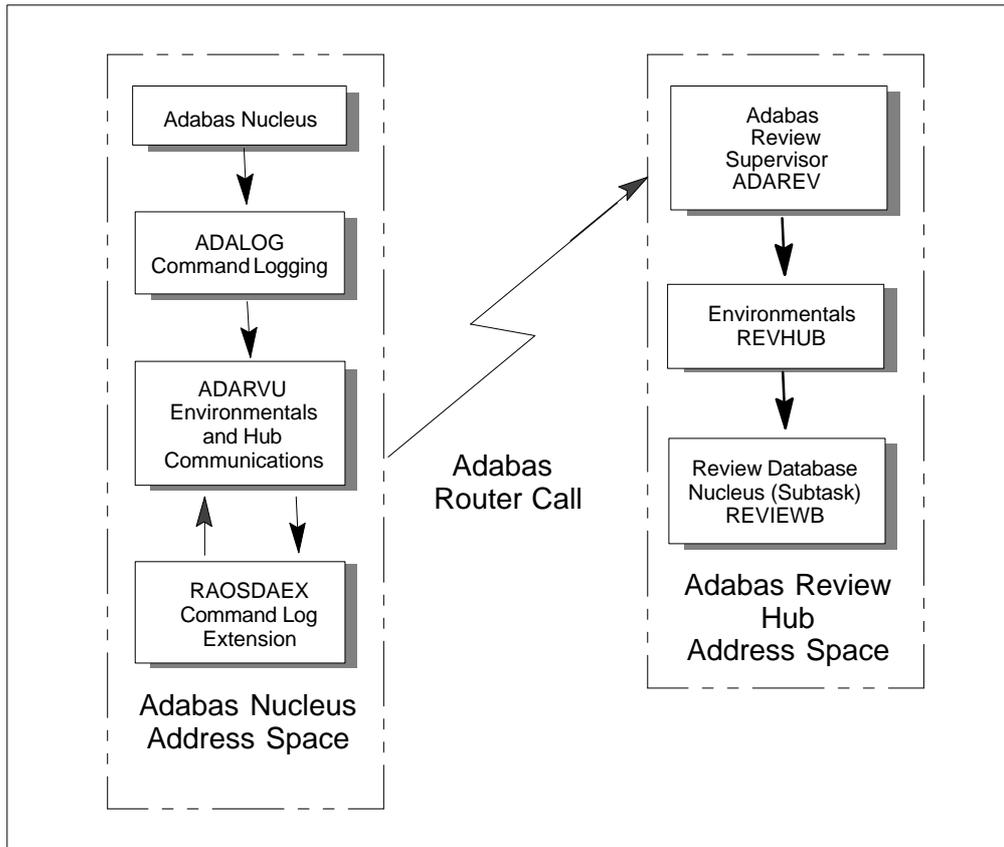


Figure 6-2 : Example Adabas Review Environment (OS/390 or z/OS)

Adabas Statistics Facility

A database administrator (DBA) regularly checks the status of the database (such as disk and memory utilization) and plans for the long term, such as ensuring that the future disk space requirements can be met, based on current trends.

For Adabas, the DBA can check the status of individual databases and files using the ADAREP (Database Report) utility, the nucleus end session protocol, and ad hoc inquiries made using the Adabas Online System. This is often a time consuming process.

The Adabas Statistics Facility (ASF) provides an automated environment comprising

- a program to collect data concerning groups of databases and files specified by the DBA; and
- a set of programs to evaluate the data collected.

Data Collection Program

ASF uses a data collection program called the “store program” to collect database status information at the start of, at the end of, or during an active nucleus session. This program is normally scheduled to run as a batch program at regular intervals (perhaps once per day) over a period of weeks or months to collect data that can be statistically evaluated. The store program can also be started by the DBA on an ad hoc basis, using commands in the ASF online menu system.

The DBA defines “store profiles”, each specifying a different set of databases and files to be monitored, that are specified as input to the store program when it runs. Several store programs, each with a different store profile, can run concurrently.

Approximately 170 criteria called “data fields” are used for monitoring Adabas databases and files. The data fields represent aspects of an Adabas database such as disk and buffer usage, thread usage, database load, ADARUN parameters, pool usage, and frequency of use of particular Adabas commands. All data fields are stored for each database and file specified in the store profile.

Start and end nucleus data, when accumulated over periods of weeks or months, give an indication of the long term database growth, and permit projections of future database requirements. Nucleus performance data, such as main memory usage and pool usage, provide information for tuning Adabas nucleus parameters.

Data Evaluation Programs

ASF uses a set of programs called “evaluation programs” to evaluate the statistics gathered by the Store Program and produce summary reports called “evaluation reports” that can be viewed online, printed, or downloaded to a PC.

For each evaluation report, the DBA uses the online menu system to define an evaluation profile specifying

- the databases and files for which data is to be evaluated (data for these must have been collected by a store program);
- for each database and file specified, one or more data fields to be analyzed in the report;
- the units of measurement of the data fields specified;
- upper and lower values representing “critical” levels for the data fields specified; and
- one of ten report types (01–10) which determines the format of the report heading.

The main types of reports are

- **General evaluation:** an analysis of the past and present database status. The statistical tables generated provide an overview of the status of various databases and files. The maximum and minimum values in rows of the output tables can be displayed, as well as statistical quantities such as the sum and average of the values.
- **Trend evaluation:** tables of projected statistics, in time steps of days, weeks, or months, until a specified end date.
- **Critical Report:** a report of databases and files for which the specified data fields have reached or exceeded the specified “critical limits”.
- **Critical Trend Report:** a report of databases and files for which the specified data fields will reach or exceed their critical limits within a time frame, based on an extrapolation of current trends.

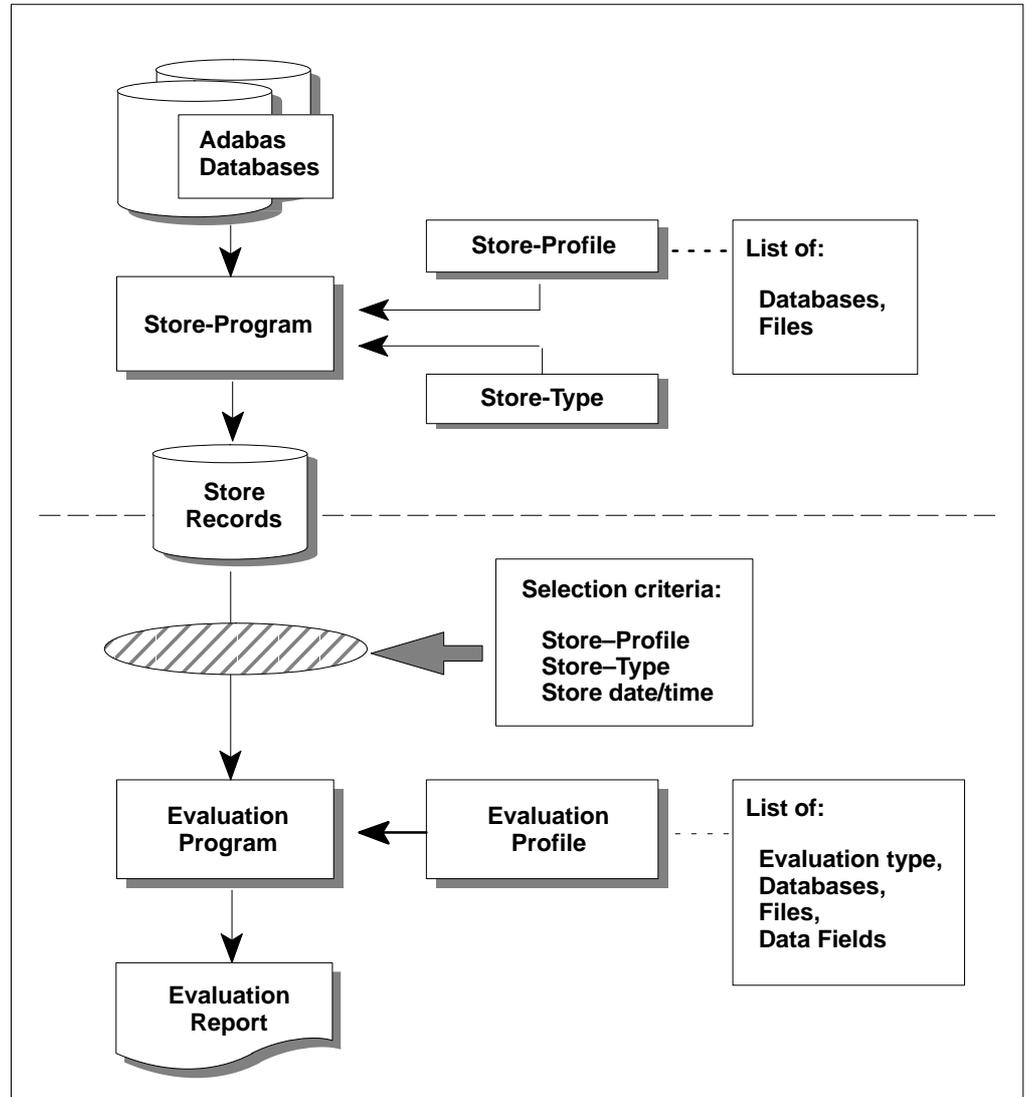


Figure 6-3 Adabas Statistics Facility Components

Adabas Parallel Services

Adabas Parallel Services (formerly ADASMP) implements multinucleus, multithread parallel processing and optimizes Adabas in a multiple-engine processor environment on a single operating system image.

Up to 31 Adabas nuclei in an Adabas Parallel Services “cluster” are distributed over the multiple engines that are synchronized by the operating system.

All nuclei in the cluster access a single physical database simultaneously. A “single physical database” is one set of Associator and Data Storage datasets identified by a single database ID number (DBID).

The nuclei communicate and cooperate with each other to process a user’s work. Compression, decompression, format buffer translation, sorting, retrieving, searching, and updating operations can all occur in parallel.

In addition to the increased throughput that results from parallel processing, Adabas Parallel Services increases database availability during planned or unplanned outages: the database can remain available when a particular cluster nucleus requires maintenance or goes down unexpectedly.

An unlimited number of Adabas Parallel Services clusters can operate in the same operating system image under the same or different SVCs; that is, an unlimited number of separate databases can be processed, each with its own Adabas Parallel Services cluster of up to 31 nuclei.

Applications see only one database target; no interface changes are required. Applications still communicate with their intended databases and communicate with an Adabas Parallel Services cluster of nuclei without modification.

Adabas Cluster Services

Adabas Cluster Services implements multinucleus, multithread parallel processing and optimizes Adabas in an IBM Parallel Sysplex (systems complex) environment. The Adabas nuclei in a sysplex cluster can be distributed to multiple OS/390 or z/OS images that are synchronized by a Sysplex Timer[®] (IBM). One or more Adabas nuclei may be run under an OS/390 or z/OS image.

Adabas Cluster Services comprises software components that ensure intercommunicability and data integrity among the OS/390 or z/OS images and the associated Adabas nuclei in each sysplex nucleus cluster. An unlimited number of sysplex clusters each comprising up to 32 clustered nuclei can reside on multiple OS/390 or z/OS images in the sysplex.

Software AG's Entire Net-work (see also page 111) is used to send Adabas and Adabas Cluster Services commands back and forth between OS/390 or z/OS images. It provides the communication mechanism among the nuclei in the sysplex cluster. No changes have been made to Entire Net-work to accommodate Adabas Cluster Services. To support a sysplex environment that includes more than one OS/390 or z/OS image, a limited Entire Net-work library is included as part of Adabas Cluster Services.

The ADACOM module is used to monitor and control the clustered nuclei. For each cluster, the ADACOM module must be executed in each OS/390 or z/OS image that either has a nucleus that participates in the cluster or has users who access the cluster database.

The Adabas Cluster Services SVC component SVCCLU is prelinked to the Adabas SVC and is used to route commands to local and remote nuclei. CSA space is used to maintain information about local and remote active nuclei, and currently active users.

The sysplex cache structure is used to hold ASSO/DATA blocks that have been updated during the session. It synchronizes the nuclei, users, and the OS/390 or z/OS images; ensures data integrity, and handles restart and recovery among the nuclei.

Cluster Services With Other Adabas Products

Adabas Online System communicates with all nuclei within the sysplex cluster.

Adabas Caching Facility supports clustered nuclei and can provide a performance boost to the cluster.

Advantages of Using Entire Net-Work

In an Adabas Cluster Services environment, Entire Net-Work allows users on various network nodes to query a logical database across multiple OS/390 or z/OS images. Users access a cluster database as they would a conventional, single-node database.

A request to Adabas can be made from within an existing application, without change. The request is processed automatically by the system; the logistics of the process are transparent to the application.

Entire Net-Work ensures compatibility by using Adabas-dependent service routines for the operating system interface as well as for interregion communication. Job control statements for running Entire Net-Work are much like those needed to run Adabas. For example, the EXEC statement invokes the ADARUN program for Entire Net-Work just as it does for Adabas, and the ADARUN parameters for Entire Net-Work are a subset of Adabas parameters.

Because status information is broadcast to all nodes whenever a target or service establishes or terminates communication with the network, there is no need to maintain or refer to database or target parameter files at a central location.

Allowing only one Entire Net-Work task on each node enforces control over the network topology by maintaining all required information in one place. This avoids confusion in network operation and maintenance. If, however, more than one Entire Net-Work task is required, this can be accomplished by installing additional routers.

Each Entire Net-Work node maintains only one request queue and one attached buffer pool for economical use of buffer storage. All buffers that are not required for a particular command are eliminated from transmission. In addition, only those portions of the record buffer and ISN Buffer that have actually been filled are returned to the user on a database reply.

Buffer size support in Entire Net-Work is comparable to that in Adabas, ensuring that all buffer sizes that are valid for Adabas can also be transmitted to remote nodes.

XCF Line Driver

Actual network data traffic is controlled by the Entire Net-Work XCF line driver, an interface to IBM's "cross-system coupling facility" (XCF) which allows authorized applications on one system to communicate with applications on the same system or on other systems. XCF transfers data and status information between members of a group that reside on one or more OS/390 or z/OS images in a sysplex.

The XCF line driver, which is installed on each Entire Net-Work node, provides high performance, transparent communications between OS/390 or z/OS images that reside on different central processors in the sysplex. Multiple connections to other nodes are supported, and the line driver's modular design permits easy addition of new access method support to the system.

A "member" is a specific function (one or more modules/routines) of a multisystem application that is defined to XCF and assigned to a group by the multisystem application. A member resides on one OS/390 or z/OS image in the sysplex and can use XCF services to communicate (send and receive data) with other members of the same group. Each Entire Net-Work node running the XCF line driver is identified as a different member in a group specifically set up for Entire Net-Work connectivity.

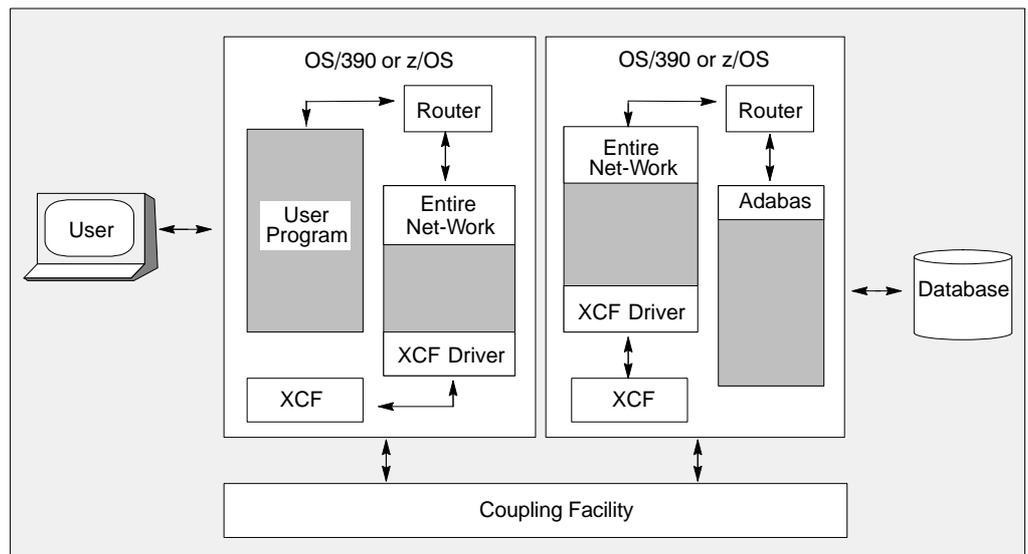


Figure 6-4 : Cluster Services with Entire Net-Work

Adabas Text Retrieval

Adabas Text Retrieval is an extension of Adabas that allows you to develop applications that access both formatted and unformatted (that is, text) data simultaneously. Adabas Text Retrieval manages the index information and not the content of the data, making it possible to store the document contents at any location: Adabas, sequential files, CD-ROM, PC, etc. The call interface for Adabas Text Retrieval can be embedded in Natural, Software AG's fourth generation application development environment, or in any third generation language such as COBOL or PL/I.

Documents comprise "chapters" designated as either free text to be managed by Adabas Text Retrieval or formatted fields to be managed by Adabas itself. Free-text chapters comprise paragraphs and sentences, which can be individually searched.

Text substrings of an entered text are identified or "tokenized" by identifying a character defined in the Adabas Text Retrieval character table, by using algorithms to identify characters based on their contexts, or by using translation tables to sort or limit previously identified characters.

Document index entries are created when unformatted data is inverted. The full text can be inverted, or the inversion can be limited by a controlled thesaurus or by ignoring words in a stopword list.

Searches can be based on words, parts of words (right/left/middle truncations are possible), phonetics, synonyms, integrated thesaurus relations (broader/narrower terms), proximity operators (adjacent, near, in sentence, in paragraph), relational operators, Boolean operators, references to previous queries (refinement), or sorts (ascending or descending). Searches can be independent of structure, meaning that they may encompass any combination of free text and formatted fields. Search returns can be highlighted.

Natural users can expand the functions of Adabas Text Retrieval using Natural Document Management, which provides complete document management services.

Adabas Bridges

Adabas Bridge technology allows you to access the DL/I (and IMS/DB) and VSAM application development environments efficiently. Emulation requires no modifications to application programs and the delay and expense of traditional conversion are avoided.

Note:

Solutions are also available for TOTAL and SESAM.

Adabas Bridge technology provides

- user application transparency (it is not necessary to change any existing application programs or third party application software using native VSAM or DL/I calls);
- support for batch and online processing environments and the RPG, COBOL, PL/I, FORTRAN, and Assembler programming languages;
- data and application integrity.

Adabas Bridge for VSAM

Adabas Bridge for VSAM (AVB) allows application software written to access data in the VSAM environment to access data in an Adabas environment. It operates either in batch mode or online (under CICS) and is available for OS/390, z/OS, and VSE operating environments.

AVB may be executed in an environment with Adabas files only; VSAM files only; or both Adabas and VSAM files (mixed environments). The ability to operate in a mixed environment means that your migration schedule can be tailored to your needs and resources. You can migrate files from VSAM to Adabas as needed, one application or even one file at a time.

AVB uses a “transparency table” to map the names and structures of VSAM files to the numbers and structures of corresponding Adabas files. Once a VSAM file has been migrated to Adabas and defined in the AVB transparency table, it can be “bridged” to Adabas. When a VSAM file is bridged, AVB converts each request for the VSAM file to an Adabas call; the Adabas file is accessed instead of the VSAM file.

When AVB is active, it intercepts each file OPEN and CLOSE request and performs a series of checks to determine whether to process the request against an Adabas file. If not, AVB passes the request to the operating system to open the referenced VSAM file.

When AVB detects an OPEN or CLOSE for a bridged file, it converts it to an Adabas command and calls Adabas to open or close the corresponding Adabas file. After the OPEN, all requests to read or update the VSAM file are passed directly to AVB.

AVB allocates VSAM control blocks and inserts information the application needs to process results as if they were returned from a standard VSAM file request. After the Adabas call, AVB returns the results to the application using standard VSAM control blocks and work areas.

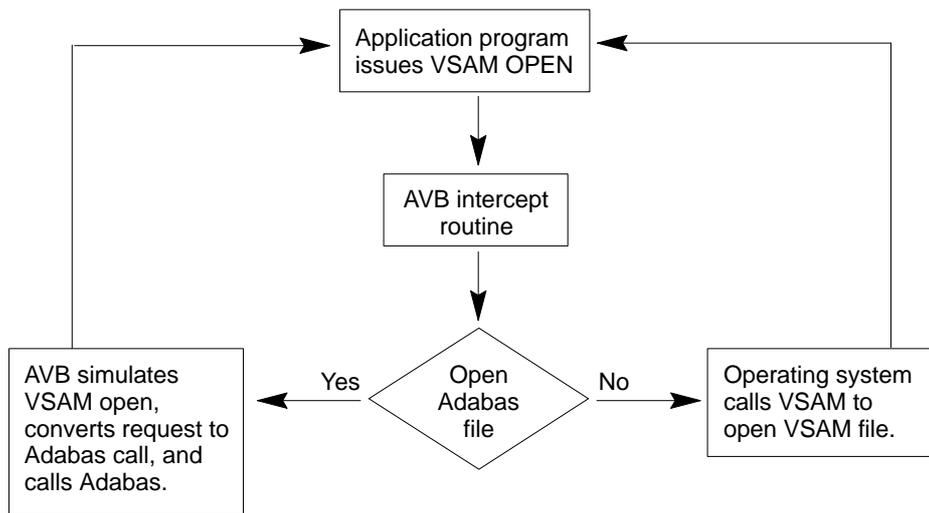


Figure 6-5 Adabas Bridge for VSAM Operation

Advantages of Adabas over VSAM

The availability of Adabas in an environment in which only VSAM file structures were previously used results in the following benefits:

- Applications can be extended with the powerful indexing facilities of Adabas. You can query, retrieve, and manipulate data using efficient views and paths.
- Applications can be extended with programming languages such as Natural and SQL.
- Application programs are independent of the data structure, which reduces maintenance costs and increases programmer productivity.
- Automatic restart/recovery ensures the physical integrity of the database in the event of a hardware or software failure.
- Data compression significantly reduces the amount of online storage required and allows you to transmit more information per physical I/O.
- Security is improved by password protection at both the file and field levels and, on the basis of data values, at the record level as well.
- Adabas provides encryption options, including a user-provided key that drives the encryption process.

After migration, your application programs have the same view of data as before, but you can structure the new Adabas files to optimize the benefits outlined above.

The tables that identify files to Adabas are external to the applications and may be changed without relinking the application programs. This feature is especially useful when you want to change file or security information, or move applications from test to production status.

Adabas Bridge for DL/I (and IMS/DB)

Adabas Bridge for DL/I (ADL) is a tool for migrating DL/I or IMS/DB databases into Adabas. The term “DL/I” is used as a generic term for IMS/VS and DL/I DOS/VS. ADL operates either in batch mode or online (under CICS or IMS/DC) and is available for OS/390, z/OS, and VSE operating environments.

DL/I applications can continue to run without change: the migrated data can be accessed by Natural and by SQL applications if the Adabas SQL Server is available. ADL can also be used to run standard DL/I applications against Adabas databases.

Functional Units

ADL comprises six major functional units:

- A collection of conversion utilities automatically converts DL/I databases into Adabas files called “ADL files” to emphasize the special properties of these files as opposed to native Adabas files.
- As a result of the conversion process, the DL/I database definitions (DBDs) and the related Adabas file layouts are stored on an Adabas file called the “ADL directory”, which also contains the ADL error messages and other information.
- A menu-driven application written in Natural provides a number of online services, including reports on the contents of the ADL directory.
- A special set of integration programs uses data from the ADL directory to generate Predict definitions for ADL files, which can then be used to generate Natural views.
- A call interface allows DL/I applications to access ADL files in the same way as original DL/I databases (and both concurrently in “mixed mode”). It supports Assembler, COBOL, PL/I, RPG, FORTRAN, and Natural for DL/I. A special precompiler is provided for programs using the “EXEC DLI” interface.
- A “consistency” interface provides access to ADL files for Natural applications or programs using Adabas direct calls. This interface preserves the hierarchical structure of the data, which is important for ongoing DL/I applications.

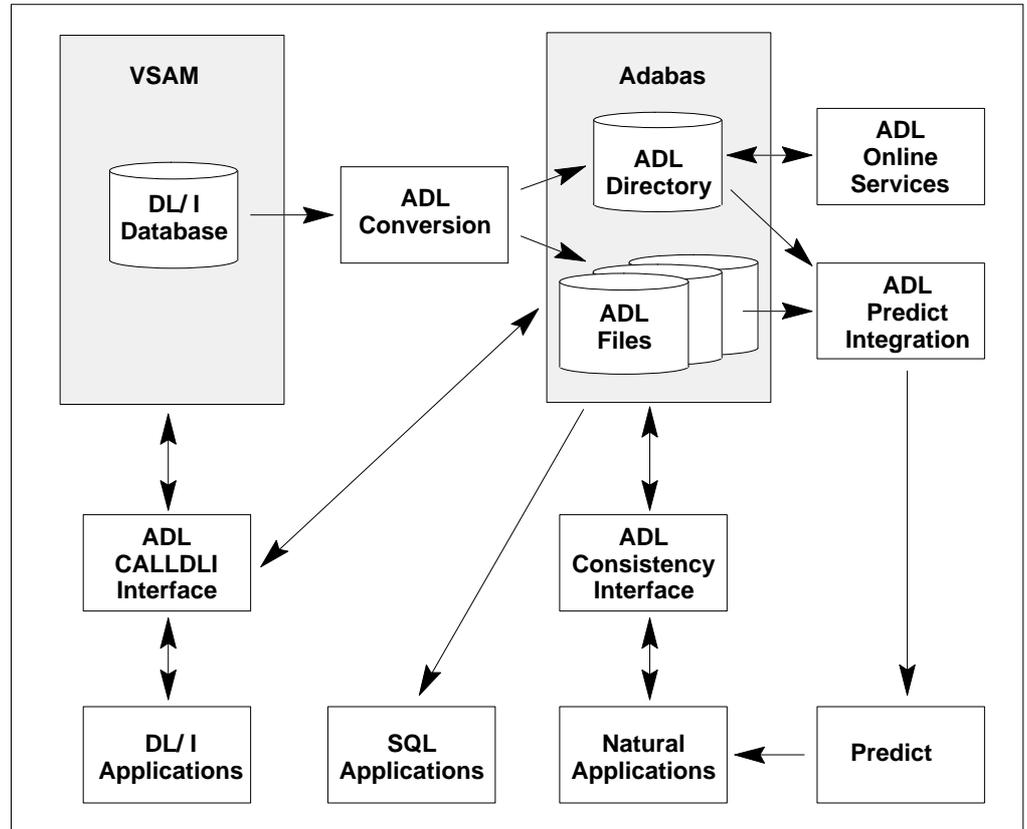


Figure 6-6 Adabas Bridge for DL/I Functional Relationships

Advantages of Adabas over DL/I

The availability of Adabas in an environment in which only DL/I file structures were previously used results in the following benefits:

- Data can be manipulated by Natural, Software AG's fourth generation language;
- Data is compressed at field level automatically by Adabas;
- Deleted data records are released from storage immediately. This is in contrast to DL/I, which simply sets a flag in such records but does not release the storage used by them. With Adabas, the released space can be reused immediately for new records: there is no requirement to maintain records that are marked as "deleted", and less reorganization of the database is required;
- All converted DBDs have full "HIDAM" functionality, regardless of the original access method;
- The length of a field can be increased without unloading and reloading the data;
- Segments can be added to the end of a DBD without unloading and reloading the data;
- Trace facilities are available for online and batch;
- The batch CALLDLI test program is available.
- Since Adabas (unlike DL/I) does not run in the CICS region/partition, online system resources are reduced.
- A symbolic checkpoint facility is available under VSE/ESA.
- HD databases are available under OS/390 and z/OS.

Entire Transaction Propagator

Entire Transaction Propagator (ETP) allows Adabas users to have duplicate, or replicate, database files in a single database or distributed network. The copies can be distributed throughout a network to provide quick, economical access at user locations.

The concept of a distributed database provides operating efficiency and flexibility while at the same time offering almost unlimited data capacity. Such a “networked” database structure means that the portion of the database data needed by a particular department can be located on local systems and still be available corporate-wide as part of the common database resource.

One particularly appealing feature of distributed databases is the possibility of having duplicate copies of data at those locations where the data is needed most. This concept allows duplicate copies of a data file to be located throughout the database network, yet the copies are viewed logically by users as a single file.

Normally, a replicated file requires an intricate control process to ensure data integrity in all file copies after each change. For distributed systems with a high ratio of read transactions compared to write transactions, however, such critical control may be unnecessary. ETP provides an alternative replicated file concept using a less critical control process, but with virtually all the other advantages of replicated files. Using a “master/replicate” system of control, ETP resynchronizes all replicate copies with a master copy at user-specified intervals.

Entire Net-Work Multisystem Processing Tool

Entire Net-Work, Software AG's multisystem processing tool, provides the benefits of distributed processing by allowing you to communicate with Adabas and other service tasks on a network-wide scope. This flexibility allows you to

- run Adabas database applications on networked systems without regard to the database location;
- operate a distributed Adabas database with components located on various network nodes;
- perform specific types of tasks on the network nodes most suitable for performing those services without limiting access to those services from other network systems;
- access Entire System Server (formerly Natural Process) to perform operating system-oriented functions on remote systems;
- access Entire Broker in order to implement your client/server applications.

Mainframe Entire Net-Work supports BS2000, OS/390, MVS/ESA, VSE/ESA, VM/ESA, and Fujitsu MSP. It provides transparent connectivity between client and server programs running on different physical or virtual machines, with potentially different operating systems and hardware architectures.

Entire Net-Work is additionally available on the midrange platforms OpenVMS, UNIX, and AS/400 and on the workstation platforms OS/2, Windows, and Windows NT.

At its lowest level, Entire Net-Work accepts messages destined for targets or servers on remote systems, and delivers them to the appropriate destination. Replies to these requests are then returned to the originating client application, without any change to the application.

The method of operation and the location and operating characteristics of the servers are fully transparent to the user and the client applications. The servers and applications can be located on any node within the system where Entire Net-Work is installed and communicating. The user's view of the network targets and servers is the same as if they were located on the user's local node. Note that due to possible teleprocessing delays, timing of some transactions may vary.

Entire Net-Work insulates applications from platform-specific syntax requirements and shields the user from underlying network properties. It also provides dynamic reconfiguration and rerouting (in the event of a down line) to effect network path optimization and generate network-level statistics.

Entire Net-Work is installed on each participating host or workstation system requiring client/server capability. The configuration for a given system comprises an Entire Net-Work control module, control module service routines, and any required line driver. Each system with Entire Net-Work installed becomes a node in the network. Each node's adjacent links to other nodes are defined by name and driver type.

Each Entire Net-Work node maintains a request queue for incoming requests. This queue is similar to the Command Queue used by Adabas; it allows the node to receive Adabas calls from locally executing user/client programs, which Entire Net-Work then dequeues and transports to the nodes where the requested services reside.

Each local Entire Net-Work node also keeps track of all active network services, and therefore can determine whether the user's request can be satisfied or must be rejected. If the request can be serviced, the message is transmitted; otherwise, Entire Net-Work advises the calling user immediately, just as the Adabas router would do for a local database request.

Actual network data traffic is controlled by Entire Net-Work line drivers, which are interfaces to the supported communications access methods, such as VTAM, IUCV, DCAM, XCF (see page 102), and TCP/IP, or directly to hardware devices, such as channel-to-channel adapters (CTCAs). Each Entire Net-Work node contains only those line drivers required by the access methods active at that node. In addition, each line driver supports multiple connections to other nodes; this modular line driver design permits easy addition of new access method support to the system.

The Entire Net-Work XTI interface allows users to write their own client/server applications, typically in "C", which are independent of the Adabas structures. XTI is an internationally accepted vehicle for creating truly portable applications. In theory, an application created according to XTI specifications can easily be ported to any other platform that supports the XTI implementation.

The Entire Net-Work XTI implementation supports communication between programs running on the same machine and programs running on different machines. Entire Net-Work is viewed as the “transport provider” from the application programmer’s point of view.

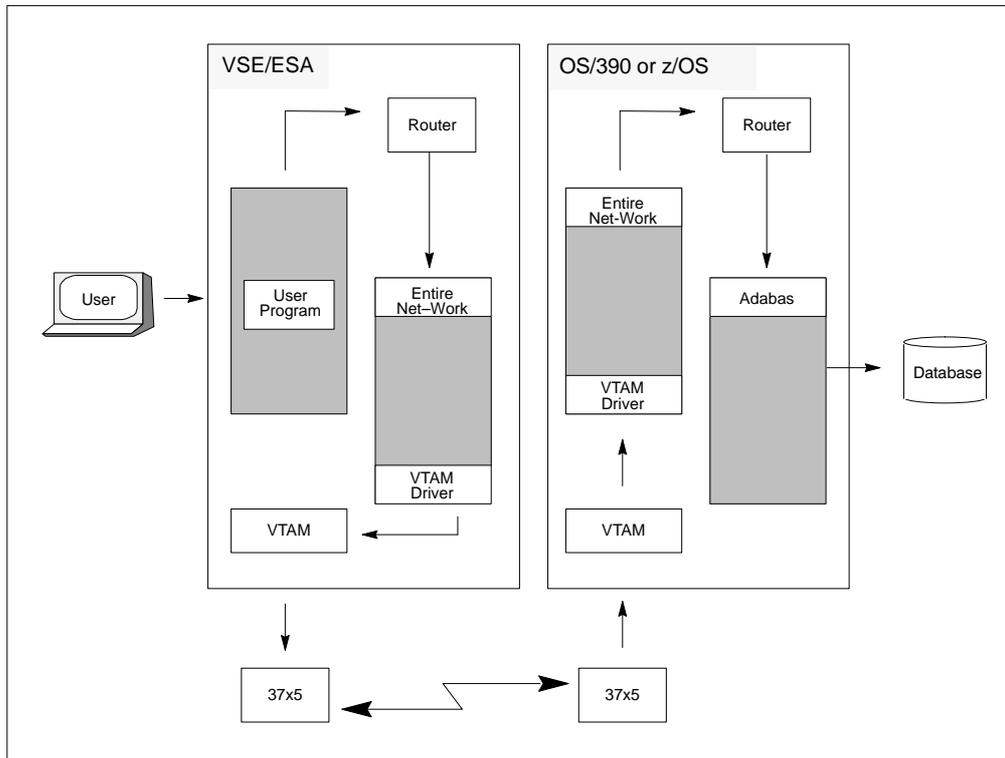


Figure 6-7 Sample Entire Net-Work Data Flow

Adabas Native SQL

Adabas Native SQL is Software AG's high-level, descriptive data manipulation language (DML) for accessing Adabas files from applications written in Ada, COBOL, FORTRAN, and PL/I.

Database access is specified in an SQL-like syntax embedded within the application program. The Adabas Native SQL precompiler then translates the SQL statement into a transparent Adabas native call.

Software AG's Adabas, Natural, Predict, and the Software AG Editor are prerequisites for Adabas Native SQL, which makes full use of the Natural user view concept and the Predict data dictionary system to access all the facilities of Adabas.

Using your Natural field specifications, Adabas Native SQL automatically creates Ada, COBOL, FORTRAN, or PL/I data declarations with the correct set of fields, field names, field sequence, record structure, field format and length.

Adabas Native SQL uses information about file and record layouts contained in Predict to generate the data structures that the generated Ada, COBOL, FORTRAN, or PL/I program needs to access the database. Then, as Adabas Native SQL processes the program, it records in Predict "active cross-reference" (Xref) information including the names of the files and fields that the program accesses.

These features help to eliminate the risk of writing incorrect data declarations in programs that access the database. In addition, they create comprehensive records in the data dictionary that show which programs read from the database and which programs update it, providing the DBA with an effective management tool.

Adabas SQL Server

Adabas SQL Server is Software AG's implementation of the ANSI/ISO Standard for the standard database query language SQL. It provides an SQL interface to Adabas and an interactive facility to execute SQL statements dynamically and retrieve information from the catalog.

The server supports embedded static and dynamic SQL, as well as interactive SQL and SQL2 extensions. It automatically normalizes complex Adabas data structures into a series of two-dimensional data views that can then be processed with standard SQL.

Adabas SQL Server accesses and manipulates Adabas data by submitting statements

- embedded in a Natural application program.
- embedded in the third generation host languages C, COBOL, or PL/I.
- using a direct, interactive interface.

Currently, Adabas SQL Server provides precompilers for SQL statements embedded in C, COBOL, and PL/I. The precompiler scans the program source and replaces the SQL statements with host language statements. Due to the modular design of Adabas SQL Server, the functionality is identical regardless of the host language chosen.

Because certain extensions not provided for by the standard are available to take full advantage of Adabas functions, one of three SQL modes must be selected when compiling an application program: ANSI compatibility mode, DB2 compatibility mode, or Adabas SQL Server mode.

Both local and remote clients can communicate with the server using Entire Net-Work as the protocol for transporting the client/server requests. With the Adabas ODBC Client, an ODBC driver allows access to Adabas SQL Server using ODBC-compliant desktop tools. Entire Access, also ODBC-compliant, provides a common SQL-eligible application programming interface (API) for both local and remote database access representing a client-server solution for Adabas SQL Server.

Software AG is committed to making Adabas SQL Server available in most hardware and operating system environments where Adabas itself is available. The core functionality of the Adabas SQL Server will be identical across platforms.

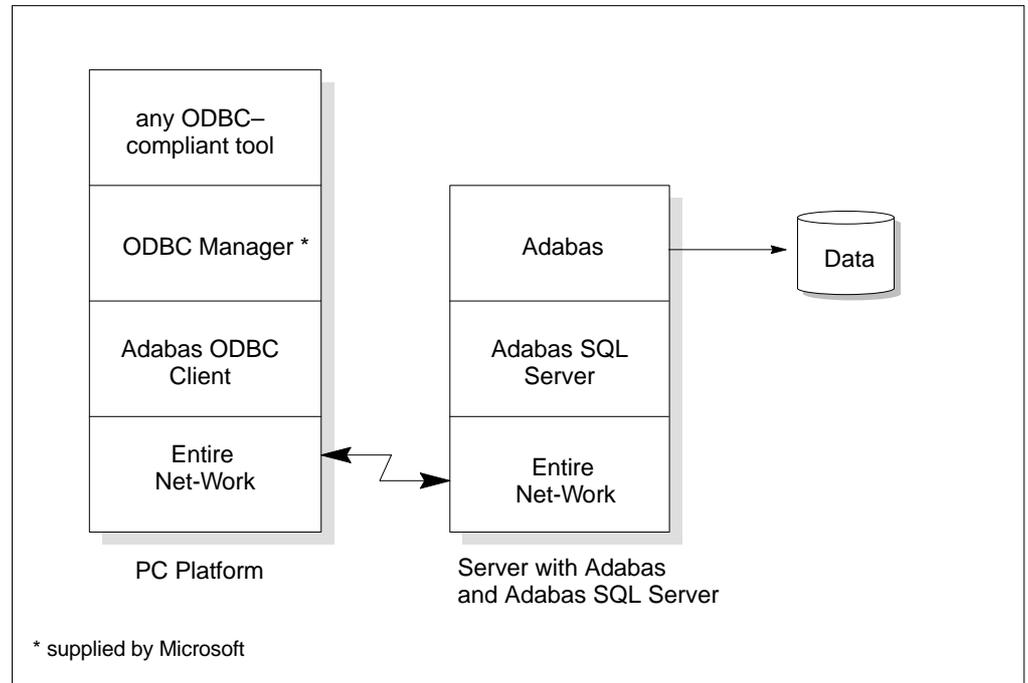


Figure 6-8 Typical Adabas SQL Server environment

Natural Application Development Environment

High-level access to Adabas is provided by Natural, Software AG's advanced fourth generation application development environment and the cornerstone of Software AG's application engineering product family which includes analysis/design, code-generation, and repository facilities.

The access can either be directly from Natural to Adabas or using an Entire Access call via Adabas SQL Server.

Whereas the FDT defines the physical records in an Adabas file, Natural programs define and use logical views of the physical file to access the file. There can be two levels of views: data definition modules and user views.

“Data definition modules” (DDMs) are Natural modules that look much like the Adabas FDT. They consist of a set of fields and their attributes (type, format, length, etc.), and may contain additional specifications for reporting formats, edit masks, and so on.

A DDM can include all the fields defined in the FDT or a subset of them. There must be at least one DDM for each Adabas file. For example, the Adabas file “Employees” could have a DDM called “Employees”. The Natural statement “READ EMPLOYEES BY NAME” actually refers to the DDM rather than the physical file; the DDM links the Natural statement to the Adabas file.

You can define multiple DDMs for an Adabas file. Multiple DDMs are a way of restricting access to fields in a file. For example, a DDM for a program used by managers could include fields that contain restricted information; these fields would not be included in a DDM for a general-use program. In a workstation group, a database administrator may define a standard set of DDMs for the group.

A new Adabas FDT can be created from an existing Natural DDM. Conversely, Adabas can generate or overwrite a DDM automatically when an FDT is created or changed.

Note:

When you delete a field from an Adabas file, you must also eliminate it from Natural programs that reference it.

A Natural “user view” often contains a subset of the fields in a DDM. User views can be defined in the Data Area Editor or within a program or routine. When a user view references a DDM, the format and length do not need to be defined, since they are already defined in the DDM. Note that in a DDM or user view, you can define the sequence of fields differently from the FDT sequence.

Adabas access is field-oriented: Natural programs access and retrieve only the fields they need. Natural statements invoke Adabas search and retrieval operations automatically.

Adabas supports a variety of sequential and random access methods. Different Natural statements use different Adabas access paths and components; the most efficient method depends on the kind of information you want and the number of records you need to retrieve.

Predict Data Dictionary System

Predict, the Adabas data dictionary system, is used to establish and maintain an online data dictionary. Because it is stored in a standard Adabas file, it can be accessed directly from Natural.

A data dictionary contains information about the definition, structure, and use of data. It does not store the actual data itself, but rather data about data or “metadata”. Containing all of the definitions of the data, the dictionary becomes the information repository for the data’s attributes, characteristics, sources, use, and interrelationships with other data. The dictionary collects the information needed to make the data more useful.

A data dictionary enables the DBA to better manage and control the organization’s data resources. Advanced users of data dictionaries find them to be valuable tools in project management and systems design.

Database information may be entered into the dictionary in online or batch mode. The description of the data in the Adabas dictionary includes information about files, the fields defined for each file, and the relationship between files. The description of use includes information about the owners and users of the data in addition to the systems, programs, modules and reports that use the data. Dictionary entries are provided for information about

- network structures
- Adabas databases
- files, fields, and relationships
- owners and users
- systems, programs, modules and reports
- field verification (processing rules)

Standard data dictionary reports may be used to

- display the entire contents of the data dictionary
- print field, file, and relationship information
- print field information by file



GLOSSARY OF TERMS

Adabas

Adabas, the adaptable database, is a high-performance, multithreaded, database management system for mainframe platforms where database performance is a critical factor. It is interoperable, scalable, and portable across multiple, heterogeneous platforms including mainframe, midrange, and PC.

Adalink

A generic term for that part of the Adabas API (application program interface) that is specific to a particular teleprocessing (TP) monitor. The Adabas API is used to link application programs to Adabas. The actual module name depends on the TP monitor being used; for example, the module name for linking to a batch or TSO program is ADALNK, and for CICS, the module name is ADALNC. The term “Adalink” refers to the module appropriate for the given environment. The terms “Adalink(s) and “ADALNK(s)” are synonyms.

ADARUN

The ADARUN control statement defines and starts the Adabas operating environment. The ADARUN control statement also starts Adabas utilities. ADARUN

- loads the ADAIOR module, which performs all database I/O and other operating-system-dependent functions;
- interprets the ADARUN parameter statements; then loads and modifies the appropriate Adabas nucleus or utility modules according to the ADARUN parameter settings; and
- transfers control to Adabas.

The ADARUN statement, normally a series of entries each specifying one or more ADARUN parameter settings, is specified in the DDCARD (OS/390, MVS/ESA, MSP, VM/ESA, or BS2000) or VSE/ESA CARD dataset.



address converter

Adabas stores each database record in a Data Storage block identified by a relative Adabas block number (RABN). Each record's RABN is kept in a table called the address converter. The address converters, one for each database file, are stored in the Associator. Address converter entries are in ISN order (that is, the first entry tells the RABN location of data for ISN 1, the 15th entry holds the RABN location of data for ISN 15, and so on).

address space

The storage area assigned to a program task/work unit. In OS/390, MVS/ESA, or MSP, an address space is a region; in VSE/ESA, a partition; and in BS2000, a task. In this manual, the term "region" is used as a synonym for "partition" and "task".

buffer flush

Associator and Data Storage blocks in the buffer pool that have been updated since the last "buffer flush" have write flags set "on". When the buffer is "flushed", these blocks are written to Associator and Data Storage datasets, respectively. The "flushed" blocks remain in the buffer pool with their write flags set "off".

A buffer flush can be synchronous or asynchronous. Update commands cannot be selected during a synchronous flush, but can be selected during an asynchronous flush.

data compression

Data compression significantly reduces the amount of storage required. It also permits the transmission of more information per physical transfer, resulting in greater I/O efficiency.

Adabas retains data records in compressed form. It defines and executes compression at the field level. Three compression options are supported: default compression, null suppression, and fixed format. The last two options are added as field options.

Default compression deletes trailing blanks in alphanumeric fields and leading zeros in binary fields. An **inclusive length byte (ILB)** at the beginning of the field indicates the total number of stored bytes, including the ILB. Thus, if "Susan" is entered in a "first-name" field defined with a 20-character length and default compression, its stored size will be six bytes: five bytes for the letters of the name, plus one byte for the ILB. In addition, empty fields in a record are not stored; an empty field is replaced by a one-byte **empty field counter (EFC)**. Adabas can store up to 63 contiguous empty fields in a single hexadecimal byte.

Null suppression (NU field option) adds to default compression in that searches on descriptor fields defined with null suppression do **not** return records in which the descriptor field is empty.

Fields defined as **fixed format** (FI field option) do not include a length byte and are **not** compressed. This option actually saves storage space for one-byte fields or fields that are nearly always full (e.g., a field containing the social security number).

database

In Adabas, a “database” is a group of related files.

A **physical database** identified by its database ID number (DBID) is defined with Adabas utilities. A **single physical database** is one set of Associator and Data Storage datasets identified by a single DBID. An Adabas nucleus running in an address space allows access to the physical files in the physical database.

Database Administrator (DBA)

Controls and manages the database resources. Tasks include defining database distribution, assigning a structure and resources, creating and maintaining programming and operation standards, ensuring high performance, resolving user problems, defining and teaching user training, controlling database access and security, and planning for growth and the integration of new database resource applications and system upgrades. Also known as the Database Analyst.

descriptor

A descriptor is a search key. A **unique descriptor** has a different (i.e., unique) value for each record in the file. Entries are made in the Associator’s inverted list for descriptor fields, adding disk space and processing overhead requirements.

Any field can be used within a selection criterion. When a field that is used extensively as a search criterion is defined as a descriptor (key), the selection process is considerably faster since Adabas is able to access the descriptor’s values directly from the inverted list without reading any records from Data Storage.

A descriptor field can be used as a sort key in a search command, as a way of controlling a logical sequential read process (ascending or descending values), or as the basis for file coupling.



A portion of a field may be defined as a **subdescriptor**; combinations of fields or portions thereof may be defined as a **superdescriptor**; a user-supplied algorithm may be the basis of a **hyperdescriptor** or a **collation descriptor**; and a “sounds-like” encoding algorithm may be the basis of a **phonetic descriptor**, which may be customized for specific language requirements. See page 33 for more information.

field

In Adabas, a “field” is the smallest logical unit of information (e.g., current salary) that may be defined and referenced by the user.

Adabas supports four field types:

	Single Value per Record	Multiple Values per Record
Single Field	Elementary	MU
Multiple Fields	Group	PE

The two basic field types are elementary and multiple-value. An **elementary** field has only one value per record. A **multiple-value** (MU) field can have up to 191 values, or occurrences, in a single record. Each multiple-value field has a **binary occurrence counter** (BOC) that stores the number of occurrences.

When two or more consecutive fields in the FDT are frequently accessed together, you can reference them by defining a **group** field. Other than its level and Adabas short name, a group field has no attributes defined. It immediately precedes its member fields in the FDT. A higher field **level** number is used to assign the member fields to the group field. Adabas supports up to seven field levels.

A **periodic** (PE) group field defines consecutive fields in the FDT that repeat together in a record. Like the members of a normal group field, PE members immediately follow the PE group field, have a higher level number than the PE field, and can be accessed both individually and as a group. Each PE has a BOC that stores the number of occurrences.

A portion of a field (**subfield**) or any combination of fields (**superfield**) may be defined as an elementary field. Subfields and superfields may be used for read operations only. They may only be changed by updating the original fields.

field definition table (FDT)

A table that defines each file's record structure and content. There is one FDT for each database file. FDTs, stored in the Associator's fixed area, have three parts: the first is a list of the file's fields in physical record order, the second part is a "quick index" to the records in the first part, and the third part defines the files sub/superfields and sub-/super-/hyper- and phonetic descriptors.

file

In Adabas, a "file" is a group of related records that have the same format (with some exceptions; see page 24).

The disk storage space allocated to a single Adabas database is segmented into **logical** Adabas files. A certain part of the overall space within the database is allocated to each logical file. When this space is filled with records from that file, Adabas automatically allocates more space to the file from the common free space pool. This dynamic space allocation, together with the dynamic recovery of released space, allows Adabas databases to run without intervention for long periods of time.

A **physical** Adabas file contains database records. Each physical file is identified by a file number. The number of physical files (and physical file numbers) per physical database is limited to 5000 or one less than the ASSOR1 block size, whichever is lower.

An **expanded file** is a logical file comprising physical files in one or more locations. The physical files have the same field definition table (FDT), but non-overlapping ISN ranges. The data content of at least one field (the field value criterion) determines the physical file in which a data record is located.

A **multiclient file** is an Adabas file with records accessible through an owner ID. Only records identified by the same individual or group owner ID can be accessed or updated by the related user. This allows the file to be maintained as a single Adabas file, but to be used as multiple logical files (each record group belonging to an owner ID is a "logical file"). "Super" owner IDs allow access to all records in the file.

global transaction

A "global" transaction is a unit of work that involves changes to resources under the control of more than one database operating in one or more operating system images.



internal sequence number (ISN)

Every Adabas record is assigned an internal sequence number (ISN) to identify the record. Each record keeps its original ISN, regardless of where it is located.

Records in a physical database file have four-byte ISNs ranging from MINISN to MAXISN. In replicated files, a record has the same ISN in all file copies. In partitioned files, the ISN ranges are non-overlapping for each physical file.

I/O buffer

The Adabas “I/O buffer” area, which can be resized for each Adabas session, contains the most frequently used data and data relationships; it helps to minimize physical input/output (I/O) activity and thus saves computer time. It is loaded into main memory at startup, along with the Adabas nucleus.

The buffer contains blocks read from the database and blocks to be written to the database:

- For blocks read from the database, a “buffer algorithm” ensures that the most frequently accessed blocks stay in memory. When a block from the database is needed, the buffer content is checked to determine if the block is already in memory, thus avoiding unnecessary reads.
- Multiple updates are accumulated in a block before it is written (“flushed”) to external storage.

nucleus

The “nucleus” is a set of programs that drive Adabas, coordinate all work, and translate user program statements into Adabas commands. All programs access Adabas files through the nucleus. All database activities such as data access and update are managed by the Adabas nucleus. In most cases, a single nucleus is used to manage a single physical database.

Adabas Parallel Services makes it possible to run a cluster of up to 31 Adabas nuclei on a single operating system against a single database. Adabas Cluster Services supports the IBM parallel sysplex environment making it possible to run a cluster of up to 32 Adabas nuclei on multiple operating systems set up as a sysplex. Again the cluster runs against a single database.

Note:

*See the **Optional Extensions** chapter starting on page 85 for information about running multiple nuclei against a single physical database under a single operating system image (ADASMP) or under multiple OS/390 or MVS/ESA images (Adaplex+).*

operator commands

Adabas operator commands are entered during an Adabas session or during utility operation to

- terminate an Adabas or user session;
- display nucleus or utility information;
- log commands into CLOG;
- change Adabas operating parameters or conditions.

procedure

A procedure is a Natural subprogram that is written and tested using standard Natural facilities. The same types of parameters are passed to the subprogram whether it is a **trigger** or a **stored procedure**.

RABN (relative Adabas block number)

The basis of Adabas storage addressing. Adabas divides Data, Associator, and Work disk space into device-dependent logical blocks. The blocks in each of the three areas are numbered consecutively in ascending sequence beginning with RABN 1. The data blocks themselves as well as their addresses are referred to throughout Software AG publications as “RABNs”. In other words, the sentence, “Adabas assigns RABNs 1–10 to the Associator” means ten Adabas storage blocks numbered 1–10 are assigned—not just the block numbers, whereas “Adabas assigns 50 RABNs to the Associator” means 50 blocks of storage with unspecified RABN numbers is assigned.

record

In Adabas, a “record” is a collection of related fields that make up a complete unit of information (e.g., all the payroll data for a single employee).

record buffer

The portion of the calling program’s parameter area, called the user buffer, that contains the data transferred during Adabas read, search, and update operations. When reading data field definitions, Adabas also returns the field definition information in the record buffer.



region

This manual uses “region” to collectively refer to storage space allocated to user jobs by OS/390, MVS/ESA, MSP, VSE/ESA, and BS2000 operating systems.

router

A central routine for communication within the boundaries of one operating system. The routine is called by users with Adalink routines, and by targets with ADAMPM. The router’s main purpose is to transfer information between the Adalink and Adabas. The router also maintains the ID table. VM/ESA and BS2000 environments divide router functions among Adalink or other Adabas functions. The Adabas SVCs in OS/390, MVS/ESA, MSP, and VSE/ESA are examples of routers.

service

A processor of Adabas calls and issuer of replies. An Adabas nucleus is an example of a service. See also **target**.

session

A **user session** is a sequence of Adabas calls optionally starting with an OP command and ending with a CL command. A **user** is either a batch mode program or a person using a terminal. The uniqueness of each user is assured by the user ID, a machine, an address space, and a terminal ID.

An **Adabas session** starts when Adabas is activated and continues until Adabas is terminated. During this time, the Adabas nucleus creates a sequence of protection entries in exact historical sequence reflecting all modifications made in the database. The sequence of protection entries is written to the Work dataset (part 1) and to a protection log in blocks. Each block contains the nucleus session number, a unique block number, and a time stamp.

stored procedure

A “stored procedure” is a procedure executed by Adabas, but invoked directly or manually by an application.

target

A receiver of Adabas calls. A target maintains a command queue, and communicates with routers using ADAMPM. A target is also classified as a **service**. The Adabas nucleus is a target.

thread

Adabas provides multithreaded processing to maximize throughput. If I/O activity suspends command processing in an active thread, Adabas automatically switches to another thread. The user may set the number of 8-kilobyte threads to be used for an Adabas session up to a maximum of 250.

transaction

Adabas data protection, recovery, and user restart is based on the concept of a “logical transaction”: the smallest unit of work (as defined by the user) that must be performed in its entirety to ensure that the information contained in the database is logically consistent.

A logical transaction may comprise one or more Adabas commands that together perform the database read/update required to complete a logical unit of work. A logical transaction begins with the first command that places a record in hold status and ends when an ET (end transaction), BT (back out transaction), CL (close), or OP (open) command is issued for the same user.

The ET command must be issued at the end of each logical transaction. Successful execution of an ET command ensures that all the updates performed during the transaction are physically applied to the database, regardless of subsequent user or Adabas session interruption.

Updates performed during transactions for which ET commands are not successfully executed are backed out, either manually by issuing the BT command or automatically by the Autobackout routine (see page 52).

translator

A process that converts a logical ID of a user’s Adabas call into a corresponding physical ID for a target.

trigger

A procedure that is executed automatically by Adabas when a specified set of criteria is met. The set of criteria is determined for each Adabas command sent to the DBMS and is based on the target file number and optionally the command type and/or field. The command type refers to the commands FIND, READ, STORE, UPDATE, and DELETE. The field must be in the corresponding format buffer of the command.

two-phase commit

“Two-phase commit” processing ensures commercial transaction integrity by securing or rejecting transactions as a whole across separately managed resources.



user

A batch or online application program that generates Adabas calls and uses an Adalink for communication.

work load balancing

Adabas Parallel Services attempts to balance the work load evenly across the cluster nuclei based on the number of users and the number of commands processed by each of the nuclei.

INDEX

A

- Accept, security-by-value criterion, overview, 77
- Access methods
 - random, 43
 - ADAM, 44
 - sequential, 41
- ACF2, Adabas interface to, 77
- ADAACK utility, check address converter, 70
- Adabas
 - definition of, 3, 121
 - implementing SAF security with, 78
 - version 6.2 and below, 81
 - session, definition of, 128
- Adabas Bridge for DL/I, overview, 107
- Adabas Bridge for VSAM
 - overview, 104
 - use of transparency table, 104
- Adabas Caching Facility
 - online services, 88
 - overview, 87
 - read-ahead caching, 88
- Adabas Cluster Services, overview, 100
- Adabas Delta Save Facility
 - online services, 89
 - overview, 88
- Adabas Online System
 - overview, 85
 - requirement for delta save, 89
 - security, 81
 - use with Adabas Cluster Services, 100
- Adabas Parallel Services
 - multiple thread processing, 99
 - overview, 99
 - parallel processing, 99
- Adabas Review
 - hub (server) component, 93
 - interface (client) component, 94
 - interface calls, 94
 - overview, 92
 - sample environment, 95
- Adabas SQL Server, 115
 - implementing security with, 81
- Adabas Statistics Facility
 - data fields, 96
 - online menu system, 96
 - report type
 - critical, 97
 - critical trend, 97
 - general evaluation, 97
 - trend evaluation, 97
 - store profiles, 96
- Adabas Transaction Manager, 92
- ADACMP utility, compress/decompress data, 55
- ADADBS utility, database services, 63
- ADADCK utility
 - block length check within range, 71
 - check data storage, 70
 - correct value in DSST, 71
 - duplicate ISNs in block, 71
 - max compressed record length, 71
 - record length sum, 71
- ADADEF utility, define a database, 67
- ADAFRM utility, format Adabas direct access (DASD) datasets, 67
- ADAICK utility, check index and address converter, 71
- ADAINV utility, create a descriptor or couple two files, 68
- Adalink, definition of, 9, 121
- ADALOD utility, load a file into Adabas, 57
- ADAM
 - bypassing the inverted lists, 19
 - estimation using ADAMER utility, 71
 - random access retrieval, 44

Adabas Concepts and Facilities

ADAMER utility, ADAM estimation, 71
ADAORD utility, reorder databases and files, 69
ADAPLP utility, print data protection records, 59
ADAPRI utility, print selected Adabas blocks, 73
ADARAI utility, database recovery aid, 59
ADAREP utility
 produce database status report, 72
 produce save tape status report, 72
ADARES utility, database recovery and restart, 60
ADARUN, definition of, 121
ADASAF
 description, 78
 router, 80
ADASAV utility, save/restore database/files, 61
ADASEL utility, select and write data protection log, 62
ADAULD utility, unload an Adabas file, 58
ADAUSER, link with Adabas API, 9
ADAVAl utility, validate the database, 73
Address, areas, by operating system, 10
Address converter
 check using ADAACK utility, 70
 check using ADAICK utility, 71
 definition of, 122
 function of, 19
Address space, definition, 122
AITM/DC, operation with Adabas, 4
Alphanumeric fields, no conversion option (NV), 31
API, link applications to Adabas, 9
API security facility, 82
Associator
 component of Adabas, 14, 18
 function of, 7
 read commands (L9, LF), 38
 reorder, using utility, 69
Autobackout, 52
Autorestart, 52
 buffer flush check, 53

B

Backout, 51
 remove changes between checkpoints, 60
Binary occurrence counter, definition of, 31, 124
Blocks, reorder, using utility, 69
Bridges, to VSAM, DL/I, IMS/DB, TOTAL, SESAM, 104
Buffer flush
 database status after, 52
 definition of, 122
 from I/O buffer, 7, 126
 session interruption during, 53
Buffer manager, augmented by Caching Facility, 87

C

CA-ACF2, use with ADASAF, 77
CA-Top Secret, using with ADASAF, 77
Cache, Work parts 2 and 3, 87
Checkpoint file, identify using ADALOD parameter, 22
Checkpoints
 command to write, 39
 reapply changes between (ADARES REGENERATE), 60
CICS, operation with Adabas, 4
Cipherng, of critical data, 75
Cluster of nuclei, 126
Collation descriptor, 35
Command ID, release, using command, 39
Command log, 21
 copy from disk to sequential dataset (CLCOPY), 60
 merge across a cluster, 60
Commands
 operator, 12, 127
 types of direct calls, 37
Com-plete, operation with Adabas, 4
Compression, forward index, 17
CTCA, driver with Entire Net-Work, 4

D

Data compression
 default, 16, 122
 definition of, 122
 options, 16
 fixed-storage (FI), 31, 123
 null-value suppression, 31, 123

Data definition, field options
 DE – descriptor, 29
 FI – fixed storage, 30
 LA – long alphanumeric, 31
 MU – multiple-value, 31
 NC – null not counted, 33
 NN – not null, 33
 NU – null value suppression, 30
 NV – no conversion, 31
 overview, 29
 PE – periodic group, 31
 UQ – unique descriptor, 29

Data dictionary, function and use, 119

Data protection area, command to write information to, 39

Data redundancy
 logical, 25
 physical, 25

Data Storage
 check using ADADCK utility, 70
 component of Adabas, 14
 function of, 7
 read commands (L1–L6), 38
 reorder, using utility, 69
 repair blocks, 60

Database
 accessing from programs, 37
 definition of, 13, 123
 definition of physical, 123
 definition of single physical, 99, 123
 maintaining integrity of, 47

modification commands (A1, E1, N1/N2),
 overview, 38
 query commands (Sx), overview, 38
 repair after nucleus ABEND, 52
 restructure, using utility, 69
 single physical, defined, 99
 supported models, 5

DBA, definition of, 123

DCAM, driver with Entire Net-Work, 4

Deadlock, avoiding resource, 49

Descriptor
 collation, 35
 definition of, 29, 123
 hyperdescriptor, 35
 phonetic, 35
 subdescriptor, 36
 superdescriptor, 36
 value to order inverted list, 18

Direct calls, 8
 types of commands, 37

DL/I, bridging files to Adabas, 107

E

Elementary, field type, 31

Empty field counter, definition of, 16, 122

Entire Broker, implementing security with, 82

Entire Net-Work
 SAF Security Interface, description, 83
 SAF security protection, 82
 using with Adabas Cluster Services, 101
 XCF line driver, 102

Entire Net-work
 operation with Adabas, 4
 supported drivers, 4

Entire Security SAF Gateway, description, 81

ET logic, 47

Execute Channel Program (EXCP), read and write, 87

Expanded files, definition of, 24, 125

Adabas Concepts and Facilities

F

- FDT, definition of, 26, 125
- Field type, 31
 - elementary, 124
 - group, 124
 - multiple value, 124
 - periodic group, 124
- Fields
 - definition of, 13, 124
 - elementary, 31
 - group, 27
 - levels, 27, 124
 - multiple value, 31
 - parent, 33
 - periodic group, 32
 - short names, 28
- File coupling
 - logical, definition of, 23
 - physical, definition of, 22
- Files
 - definition of, 13, 125
 - physical, 125
 - restructure, using utility, 69
 - security
 - access/update level, 76
 - by password, 76
 - system, 22
- Format ID, command to delete global, 39

G

- Group, field type, 27, 124

H

- Hold facility
 - command to release record hold status, 39
 - command to set record hold status, 39
- Hyperdescriptor, 35

I

- I/O Buffer
 - algorithm for, 7, 126
 - purpose of, 7
- I/O buffer, definition of, 126
- IMS
 - bridging IMS/DB files to Adabas, 107
 - operation with Adabas, 4
- Inclusive length byte, definition of, 16, 122
- Index, check using ADAICK utility, 71
- Intercepts, OPEN or CLOSE, 105
- Inverted list
 - Associator element, 18
 - function of, 18
 - normal index (NI), 18
 - upper index (UI), 18
- ISNs
 - definition of, 14, 126
 - reusing, 14
- IUCV, driver with Entire Net-Work, 4

L

- Log, types of, 21
- Long alphanumeric (LA), field option, description, 31

M

- Modes of operation
 - multiuser, 10
 - single-user, 10
- Multiclient files
 - definition of, 25, 125
 - security use, 76
- Multiple-value fields
 - definition of, 124
 - field type, 31

N

- Natural
 - DDM, 117
 - implementing SAF security with, 82
 - use with Adabas, 117
 - user view, 118
- Natural RPC, implementing security with, 81
- Natural Security, 81
- Nucleus
 - cluster, definition of, 99
 - definition of, 6, 126
 - number per operating system image, 100
- Nucleus cluster, 126
- Null value, SQL, meaning of, 33

O

- OpenEdition MVS, support for, 9
- Operating systems, supported, 100
- Operations
 - environments supported, 4
 - highlights, 3
 - overview of Adabas, 6
 - TP monitors supported, 4
- Operator commands, 12, 127

P

- Padding area, function of, 15
- Parent field, of special descriptor, 33
- Password, protection, overview, 76
- Periodic groups
 - definition of, 124
 - field type, 32
 - restrictions on using, 32
- Phonetic descriptor, 35
- PINSAF, 78
- Predict, using with Adabas, 119
- Procedure, definition of, 127

- Profile, resource/user, description of a, 77
- Protection log, 21
 - copy sequential dataset (COPY), 60
 - copy to sequential dataset (PLCOPY), 60

R

- RABNs
 - definition of, 14, 127
 - fencing, 87
 - location, 88
 - RACF
 - Adabas interface to, 77
 - using with ADASAF, 77
 - Record buffer, definition, 127
 - Records
 - definition of, 13, 127
 - hold and release, 49
 - resource deadlock, 49
 - structure of, 27
 - Recovery, 51
 - Recovery log, 21
 - Region, address space as a, 10
 - Reject, security-by-value criterion, 77
 - Remote processing, 101
 - Resources, access/update privileges for, 77
 - Restart, 51
 - processing after system failure, 52
 - Router
 - ADASAF, 80
 - definition, 128
- S**
- SAF Security Interface
 - Adabas, 78
 - Entire Net-Work, 83
 - SAF-based security, gateway to, 81
 - SAF-based security system, securing Software
 - AG products with, 81
 - Searches, complex, 40

Adabas Concepts and Facilities

Security
 options available, 75
 package (non-Software AG)
 general description, 77
 general operation, 77
 SAF-based systems, gateway to, 81
 system file, 22
Service, definition, 128
Session
 Adabas, 10, 51, 128
 command to
 close, 39
 open, 39
 types of, 10
 user, 10, 51, 128
 utility, 10
Shadow, operation with Adabas, 4
Sort, dataset for, 21
Space, management, 14
SQL, interface to Adabas, 115
Stored procedure, definition of, 9, 45, 128
Subdescriptor, 36
Subfield, 36, 124
Superdescriptor, 36
Superfield, 36, 124
SYSACF application, online cache maintenance, 88

T

Target, definition, 128
TCP/IP, driver with Entire Net-Work, 4
Temp, dataset for, 21
Threads
 multithread processing, 7
 size and number, 7
Threshold (protection) levels, overview, 76
TIAM, operation with Adabas, 4
Timeout control

 non-activity limit, 50
 transaction limit, 50
Top Secret, Adabas interface to, 77
TP monitor, overview in Adabas operation, 9
Transaction
 control commands (ET/BT), 39
 definition of, 47, 129
Translator, definition of, 129
Trigger
 definition of, 9, 45, 129
 system file, 22
Triggers and stored procedures, overview, 45
TSO, operation with Adabas, 4

U

Universal encoding support (UES), no conversion field option (NV), 31
UNIX, API security facility for, 82
Updating
 competitive, 48
 exclusive control, 49
 reapply backed-out update, 60
User
 definition of, 130
 exclusive control, updating, 49
 isolating within a file, 25, 76
 multiuser operating mode, 10
 profile, 77
 program, relationship to Adabas operation, 8
 session, 10
 definition of, 51, 128
 single-user operating mode, 10
User data, read, using direct call command, 39
User exits, controlling cipher codes with, 75
Utilities
 overview, 8, 55
 session, definition of, 10
UTM, operation with Adabas, 4

V

Value, security by, overview, 77
Virtual machines, term used to define storage space, 10
VSAM, bridging files to Adabas, 104
VTAM, driver with Entire Net-Work, 4

W

Wide-character fields, no conversion option (NV), 31

Windows, API security facility for, 82

Work

component of Adabas, 14
function of, 7, 21

X

XCF, driver with Entire Net-Work, 4
XCF line driver, 102
XCF member, definition, 102

Adabas Concepts and Facilities

Adabas Concepts and Facilities