Overview

One or more instances of Adabas can be installed and operated on a computer. Each Adabas instance consists of processes, main memory structures (caches), and a disk-based serverdb.

```
                              ______________
                              |            |
                              | Caches     |
                              |____________|
                              |            |
Adabas D Instance             | Processes  |
                              |____________|
                              |            |
                              | Serverdb   |
                              |____________|
```

This chapter covers the following topics:

- Serverdb Structure
- Logging
- Backup
- Restart
- Restore
- Consistency Check and Optimizer Support
- Process Structure
- Caches
- Multiprocessor Configurations
- Client Server
- Availability

Serverdb Structure

A serverdb has the following structure:

System	Transaction	Archive	Data
Devspace	Log Devspace	Log Devspace(s)	Devspace(s)
		... (0..7)	... (1..64)

The term "devspace" denotes a physical disk or part of a physical disk, for example, a Unix raw device or a file.

Adabas assumes that each devspace is located on a different disk. If this is not true, decreased performance is to be expected. The used disks should present uniform performance data (especially access speed) because only then an equal usage of the devspaces can be obtained.

The Adabas devspaces have the following meanings:

System Devspace

The configuration data and the mappings of the logical page numbers to physical page addresses are administered on the system devspace. The size of the system devspace therefore depends directly on the database size and is determined by the database kernel.

Transaction Log Devspace

Modifications to the data are recorded in the transaction log and written to disk at the end of the transaction. The transaction log can be used to ROLLBACK transactions, it is written cyclically. Its size must be sufficient to receive the modifications of all open transactions.

Archive Log Devspaces

All modifications made to the database contents are recorded in the archive log devspace to ensure the recovery of the database contents after a media failure. The log backup functions (Save / Log, Save / Log Segment) can be used to save the archive log devspace contents to tape (DLT, DAT, Video8) and to release the used space afterwards. The size of the archive log devspace must therefore be sufficient to receive all modifications occurring during two backups. The archive log can comprise several devspaces.

Data Devspaces

The user data (tables, indexes) and the SQL catalog (schema information) are stored in the data devspaces. As a rule, an Adabas-internal striping algorithm evenly distributes the data belonging to a table across all data devspaces. The storage space defined by all data devspaces is the total size of the database.

The data devspaces are not directly related to the storage of database objects. An assignment of tables to data devspaces is not possible and not necessary. A table or an index can use one page (4 KB) as a minimum; or a table can use all data devspaces (i.e., the whole database) as a maximum. A table increases or decreases in size automatically without administrative intervention.

The system devspace and all data devspaces of a serverdb can be mirrored to obtain a higher degree of availability. Write operations are performed on each of the two mirrored devspaces, while read operations alternate between one mirrored data devspace and the other to distribute the I/O load.

If the data devspaces become full, database operation stops and Adabas performs an "Emergency Shutdown". The devspace usage level of a serverdb is therefore a critical parameter of database operation and must be monitored. A serverdb can be expanded by additional data devspaces, if necessary, while the database is operational.

Logging

Adabas provides a gradual logging concept to satisfy different data protection and computer configuration requirements. A configuration parameter, LOG MODE, can be used to select one of several variants.

Backup

Adabas supports complete and incremental backups providing the required restore and restart functions in order to make databases operational again after power and media failures. Periodic backups are indispensable for production database environments.

To ensure round-the-clock operation, data backups (complete and incremental) can be performed in warm mode and log segments can be automatically saved as soon as they have been completed.

Data backups are done with checkpoint; i.e., they are consistent. Data backups can be performed in warm database mode. This can impair database operation.

As the low speed of the tape devices involved is a limiting factor for save and restore operations, Adabas provides the option to save to or restore from several tape devices in parallel. Up to 32 tape devices can be used to reduce save and restore times considerably. This is not possible for save log segment.

Restart

If a database failure other than a devspace failure (e.g., a power failure) occurs, the restart of Adabas ensures that the last consistent database state is reestablished using the transaction log; this means, the effects of committed transactions are reapplied on the data devspaces, and the effects of open transactions are rolled back.

Even if a devspace failure (of physical disks) occurs, a restart of the serverdb can suffice as recovery measure to restore the last consistent database state providing the archive log contains all the data required.

Restore

If a media failure occurs on the system devspace or a data devspace, database operation ends (unless the data devspaces and the system devspace are mirrored). After repairing the media failure, the database must be restored using the last complete backup version (Restore / Data).

If the archive log had not been saved in the meantime, the restart has the effect that the database modifications recorded in the archive log are reapplied, thus reestablishing the last consistent state of the database (see section Restart).

If the archive log had been saved in the meantime, Restore / Data must be performed and the backup of the archive log must be restored with Restore / Log) (see also section Troubleshooting When Problems Occur).

If the database needs to be reset to a previous state for organizational reasons, the most recent complete backup (Restore / Data) that was made previous to the desired date and time must be restored, the current archive log must be saved, and the log backups subsequent to the complete backup (Restore / Log) must be restored. The desired database state can be determined by specifying a date and a time (Restore / Log UNTIL).

For a recovery using modified database pages instead of log backups, a similar procedure is used. A sequence of Save / Updated Pages tapes written after the last complete backup will also be availabe in this case. The tapes must be restored one after the other with Restore / Updated Pages. The current contents of the archive log do not need to be saved; they are used to reestablish the last consistent database state during the following restart.

If a media failure occurs on the transaction log devspace or one of the archive log devspaces, database operation ends unless the log mode DUAL defines a mirrored archive log. After repairing the media failure and performing RESTORE LOG FROM DEVSPACE and a subsequent restart, the database is in a consistent state again.

Consistency Check and Optimizer Support

To ensure a safe database operation and good performance, two other activities must be done from time to time: Verify and Update Statistics.

Verify can be executed while the database is operational. It checks the consistency of internal chains within the B* trees used. If inconsistencies are discovered, the database must be restored. A Verify is recommended before each complete backup of the database.

Verify in COLD mode (i.e., before a Restart of the serverdb) has an additional property: pages wrongly recorded as used since an irregular end of database operation are released to the free space management.

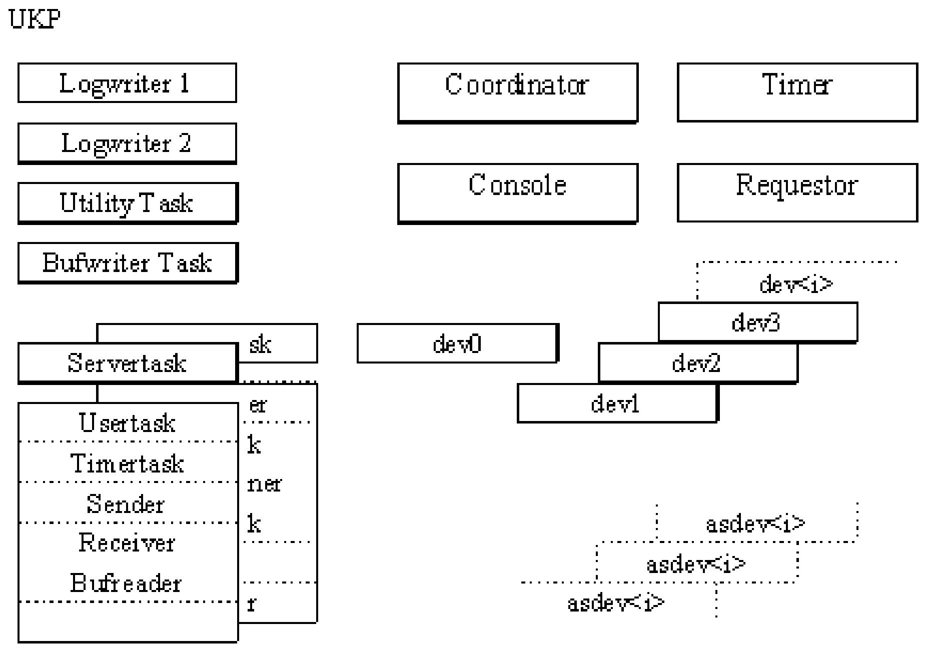
Update Statistics determines the number of rows in tables and the selectivity of individual columns. The Adabas optimizer needs these specifications to determine the best strategy for the processing of complex SQL statements. If the sizes or the value assignments in the database have changed considerably, a new Update Statistics must be performed. Update Statistics should be executed once a week.

If Adabas determines differences between the optimizer assumptions from the last Update Statistics and the current state of a table, it attempts to perform an implicit Update Statistics. If there are conflicting locks, this attempt might be aborted, so that the implicit Update Statistics is not a complete equivalent of the explicit Update Statistics.

Process Structure

An Adabas instance consists of a set of Unix processes or Windows threads. Under Unix, a process acts as UKP (user kernel process) or as special process with special tasks. Under Windows, the term "thread" is used instead of process; consequently, there are UKTs (user kernel threads). Strictly speaking, the Adabas instance is realized in Windows by a process subdivided into threads. The required number of UKPs/UKTs and of special processes/threads depends on the number of used devspaces, the hardware configuration, and the database parameters.

In the following example, each box represents one UKP/UKT. Some UKPs/UKTs bundle up several tasks, others realize just one special task. The example is valid for a process structure in a Unix system with one CPU.



A UKP/UKT bundles up a subset of all tasks (internal tasking). There are the following tasks:

Usertasks

Each user or each application program is assigned a usertask when connecting to the database. The usertask ensures the processing of SQL statements for the session. The number of available usertasks is defined by the database parameter MAXUSERTASKS.

Servertasks

The main purpose of servertasks is to perform data backups. If the installed database is a distributed system, servertasks also realize access to remote data. When configuring the database, the number of servertasks is automatically computed from the number of data devspaces and the number of provided backup devices.

Servertasks ensure the writing to secondary storage. They become active when a savepoint is being performed. A savepoint means that the modifications done to the data cache are also performed to the data on the disk.

Logwriter 1

The logwriter 1 ensures the writing of modification information (before and after images) to the transaction log.

Logwriter 2

The logwriter 2 ensures the writing of before and after images to the archive log(s).

Bufreader

For a large data cache, savepoint writing takes a long time. The bufreaders become also active between two savepoints to write data asynchronously from the data cache to disk. The number of bufreaders to be activated, if needed, must be defined in *xparam*. It depends primarily on the data cache size and the number of data devspaces.

Utility Task

The utility task is reserved for the database operating. It is only used to handle administrative tasks. As there is only one utility task for each serverdb instance, no parallel operating actions can be done.

Bufwriter Task

Adabas allows a special trace, the so-called vtrace, to be activated for diagnose purposes. The bufwriter task is provided for this purpose.

Sender and Receiver

In a distributed database installation, these tasks perform the communicative operations between the serverdbs involved.

Timertask

The timertask handles all kinds of timeout situations.

Special processes/threads are activated in addition to UKPs/UKTs. There are the following special processes/threads:

Requestor

The requestor receives the local communication requests (connect) as well as requests from the network and assigns them to a UKP/UKT. For example, the requestor informs the corresponding UKP/UKT when a user disconnects abnormally.

Timer

The timer monitors the time for timeout control.

Dev Processes/Threads

Dev processes/threads ensure that write and read operations to be done for the corresponding tasks are actually performed. Their number primarily depends on the number of devspaces in the installed database. Usually, two dev processes/threads are activated for each data devspace and the system devspace, one dev process/thread is activated for the log devspaces and for vtrace writing, if this has been enabled.

The process/thread dev0 plays a special part. Dev0 coordinates and monitors the dev processes/threads. For example, if a mirrored devspace fails in warm mode (bad devspace), dev0 ensures that the corresponding dev processes/threads are terminated. Database operation is not impaired in this case.

If the database is enlarged in warm mode by adding another data devspace, dev0 ensures that new dev processes/threads are generated.

All the other dev<i-> processes/threads write data to or read it from the devspaces.

Temporary Dev Process

Processes/threads are temporarily activated to read and write data for data backups. These processes are called asdev<i>. Their number depends on the number of data devspaces and of the number of backup devices.

In Unix systems, dev0 coordinates these processes.

On a Windows system, the special thread async0 coordinates these processes.

Coordinator

The coordinator process/thread has a special meaning. It monitors all kernel processes/threads of the instance. When starting the database instance, the coordinator is the first process/thread that becomes active coordinating the start of the other processes/threads. For example, if a process/thread fails in warm mode, the coordinator stops all the other processes/threads in the worst case.

There are some more special processes/threads in addition, according to the operating system:

Clock Thread

The clock thread is only used On a Windows system. It computes internal times; for example, to determine the time needed to execute an SQL statement.

Console Process

In Unix systems, the console process gathers information produced by other processes that could be useful to the administrator and writes it to the knldiag operating message file.

On a Windows system, there is also a knldiag operating message file into which the information is entered by each thread.

Console Thread

On a Windows system, this special thread satisfies requests made by the x_cons console. x_cons communicates with the console thread for this purpose.

In Unix systems, x_cons receives the required information from the processes’ shared memory.

Death Process (Unix only)

The death process monitors the coordinator process. If the coordinator process fails in an operative database, the death process stops all the other processes.

Network Process (Unix only)

If REMOTE-ACCESS is set to YES in *xparam*, this process - instead of the vserver - is used for communication between a remote application (remote SQL) and the kernel. A network process can serve several connections.

Caches

Read and write operations to the devspaces are buffered in order to save disk accesses. The pertinent main memory structures are called caches. They can be dimensioned appropriately. Adabas defines the following caches:

Data Cache

This cache contains the last read- or write-accessed pages of the data devspaces. The data cache is shared by all simultaneously active users. The hit rate, i.e. the relation between successful and unsuccessful accesses to the data cache, is decisive for the performance. Successful access means that the required data was already available in the data cache.

Converter Cache

The converter cache and its hit rate are also decisive for performance. The converter cache contains the last read- or write-accessed pages of the system devspace. The converter cache is shared by all simultaneously active users. For the converter cache, you should strive for hit rates as close to 100% as possible.

Proc Code Cache

This structure contains the code of the last executed DB procedures, triggers, or DB functions. The proc code cache is shared by all simultaneously active users.

Proc Data Cache

This cache exists for each active user (or for each database session). It contains the parameters or variables belonging to the last executed DB procedures, triggers, and DB functions.

Catalog Cache

This cache exists for each active user (or for each database session). It contains the last catalog objects used by a database session and the internal representation (application plans) of the last exec commands. Displacements from the catalog cache first move the data into the data cache.

Temp Cache

This cache exists for each active user (or for each database session). It contains the last database objects (SELECT results, temporary tables) generated or temporarily used by a database session.

Applications that generate large join results or frequently work with temporary tables can improve their performance by configuring a temp cache with an appropriate size. Displacements from the temp cache first move the temporary data into the data cache.

Multiprocessor Configurations

For an optimal usage of multiprocessor configurations, Adabas supports an external/internal tasking that can be configured. The aim hereby is to support as many database sessions as possible with a minimum number of operating system processes. One operating system process is required for each CPU that resides in the computer and is to be used by an Adabas serverdb.

The degree of the external/internal tasking is controlled by the two configuration parameters MAXUSERTASKS and MAXCPU.

The parameter MAXUSERTASKS indicates the maximum number of simultaneously active users (database sessions). Overconfiguration exceeding the actual requirements results in increased address space (especially shared memory) requirements.

The parameter MAXCPU indicates the number of CPUs to be made available to the serverdb.

For example, if you want to use up to 800 simultaneously active database sessions on a 4-processor computer, MAXUSERTASKS must be set to 800 and MAXCPU to 4. The serverdb can then utilize the four processors by establishing four operating system processes each of which performs an internal tasking for up to 200 users.

If the number of configured database sessions is exhausted, no other user can connect to the serverdb. The number of active sessions is therefore a critical parameter of database operation and must be monitored.

Client Server

To open a serverdb for remote SQL client operation, only the remote SQL server must be started. It acts as an agent for the remote clients.

To be able to use this connectivity built into Adabas via TCP/IP sockets, the corresponding TCP/IP entries must have been previously configured. Information required for this purpose is contained in the "User Manual Unix" or "User Manual Windows".

To connect to a serverdb, the name of the serverdb and the network name of the corresponding computer or network node (servernode) must be specified in addition to a valid user name/password combination. When connecting to a local serverdb, the servernode specification can be omitted.

Note:
Control can only be used on the local serverdb.

Availability

The availability of a serverdb can be increased by using the corresponding hardware, operating system, or database features.

For mission-critical applications, we recommend RAID-5 configurations as disk peripherals for data devspaces. Then a failure and the exchange of a disk does not impair database operation. For performance reasons, the log devspaces must not be created on RAID-5 systems but on special disks.

The same applies to operating system mirror disks. These, however, require double disk capacity.

Regardless of the hardware and operating system properties, Adabas provides a mirroring of the system devspace and all data devspaces. (Independent of these mirrored devspaces, log mode DUAL can be used to define mirrored archive log devspaces.) Mirroring the system devspace and the data devspaces is controlled by the configuration parameter MIRRORED and requires the definition of a corresponding number of mirrored devspaces. In a mirrored configuration, read operations alternate between the original and the mirrored devspace; write operations concern both devspaces.