English

Fujitsu Software BS2000

SESAM/SQL-Server

Core Manual

User Guide

Valid for:
SESAM/SQL-Server V9.1

Edition November 2022

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Table of Contents

1 Core Manual

Preface

The functions and architectural features of the SESAM/SQL-Server database system meet all the demands placed on a powerful database server in today's world. These characteristics are reflected in its name: SESAM/SQL-Server.

SESAM/SQL-Server is available in a standard edition for single-task operation and in an enterprise edition for multitask operation.

For the sake of simplicity, we shall use the name SESAM/SQL throughout this manual to refer to SESAM/SQL-Server.

Brief product description

SESAM/SQL is the relational database server for BS2000 systems.

SESAM/SQL combines the advantages of the relational data model with all the characteristics expected of a system which is subject to high loads in productive operation. On the one hand, this offers simple operation and data which is independent of the physical storage method used and, on the other, it means that the system is suitable for high transaction rates and large volumes of data and possesses outstanding security and availability characteristics.

The SQL interface implemented in SESAM/SQL has been based across the board on the current SQL standard. This standardized SQL interface means that SESAM/SQL allows you to create portable, future-proof database applications which can be transferred to different database systems and operating systems.

SESAM/SQL fulfils all the demands placed on a modern database system in today's world:

- SESAM/SQL uses SQL, a uniform language and a consistent set of terms for defining, structuring and maintaining a relational database and for creating application programs.
- SESAM/SQL runs on all BS2000 systems and can be used as a powerful SQL server for clients of BS2000, UNIX systems, Solaris, Linux and Windows systems.
- SESAM/SQL enables Unicode characters to be used in tables and takes into account coded character sets.
- SESAM/SQL excels in terms of high availability, security and data integrity.
- SESAM/SQL supports modern parallel processing techniques for multi-user operation and multi-database processing.
- The Universal Transaction Monitor openUTM and the SESAM/SQL database system together form a powerful DB/DC system including fully coordinated transaction processing and restart facilities for online applications.
- The product SESAM/SQL-DCN allows transparent, efficient and trusted access to distributed databases in BS2000 networks.
- With the help of the SESAM-DBAccess release unit supplied with SESAM/SQL, also known as SESDBA for short, it is possible to access SESAM/SQL databases in the client/server environment over the following interfaces.
  - from Java programs or Java Server Pages, Java Servlets, and Java Applets on any platforms
  - via the ADO technology defined by Microsoft (ActiveX Data Objects) and the ADO.NET interface in client applications on Windows systems
  - Via the PDO driver of SESAM/SQL for the PHP interface (**P**HP: **H**ypertext **P**rocessor) on an Apache Webserver

- A large range of add-on products increases the range of application of SESAM/SQL. These products range from database design tools, programming languages and third and fourth generation software development environments through to easy-to-use products for end users and the use of SESAM/SQL in World Wide Web applications.

Structure of the SESAM/SQL server documentation

The documentation for the SESAM/SQL server database system can be found in the following manuals:

- Core manual
  This manual provides an overview of the database system and describes basic principles, concepts, interrelationships, and technical terms. It provides the basis for understanding all the other SESAM/SQL manuals.
- SQL Reference Manual Part 1: SQL Statements
  The manual deals with the embedding of programs and describes the syntax and semantics of the SQL language constructs in alphabetical order.
- SQL Reference Manual Part 2: Utilities
  The utility statements are not included in this alphabetical list and are dealt with separately in the “SQL Reference Manual, Part 2: Utilities”.
- CALL DML Applications
  This manual is aimed at CALL DML programmers and describes the language elements used in the CALL DML interface and explains how to create CALL DML programs.
- Database Operation
  This manual is aimed at the system administrator and covers database operation. It includes details on starting and terminating the DBH and SESDCN and the associated load options and administration statements. The manual also describes the utilities required for database operation.
- Utility Monitor
  This manual describes how to use the utility monitor and the functions it provides. The utility monitor is a component part of SESAM/SQL and provides a menu-driven interface for creating, loading, backing up and reconstructing a database using SQL statements. The utility monitor also provides simple methods of querying the metadata.
- Messages
  The message manual contains information relating to the structure and invocation of messages for the SESAM /SQL server database system and the distribution component SESAM/SQL-DCN. The SQLSTATEs and CALL DML status messages are listed in full here.
- Performance
  This manual is aimed at experienced SESAM/SQL users. It describes how users can identify performance bottlenecks and indicates which parameters can be used to influence system performance.

The following manual describes how to create ESQL-COBOL programs:

- ESQL-COBOL User Guide

The following manuals describe remote access with SESAM-DBAccess:

- SESAM-DBAccess

To search for terms, a PDF index of the SESAM/SQL manuals is integrated into the product SESAM/SQL-Server.

Demonstration database

Included in the SESAM/SQL-Server delivery is the library SIPANY.SESAM-SQL.<ver>.MAN-DB. This library contains all the components that you need to try out the examples in the manuals yourself.

These include:

- Readme files with an overview of the present files
- Start procedures to start the necessary SESAM/SQL programs
- Files relating to the constructed sample database ORDERCUST
- Instruction files and ESQL-COBOL programs with runtime examples of important database statements

If an example in the manual is accompanied by the symbol on the left, this means that it is present as a component in an instruction file or an ESQL-COBOL program
in the library.

The demonstration database is described in chapter “Demonstration database”.

Objectives and target groups of this manual

This manual primarily describes basic principles, concepts and interrelationships and is thus aimed at all SESAM /SQL users.

In addition, the manual is aimed at readers who are interested in gaining an overview of the SESAM/SQL database system, its field of application and the various add-on products available.

It will be easier to understand the manual if you have a broad knowledge of databases and of SQL and if you understand the basics of the BS2000 operating system and the Universal Transaction Monitor openUTM.

Summary of contents

The 'Core Manual' is intended to allow you to familiarize yourself with SESAM/SQL and to provide information on basic principles, concepts and interrelationships.

All the other SESAM/SQL manuals are based on the contents of this manual. For this reason, you should read this manual first. The manual covers the following items:

- the features of the SESAM/SQL database system and information on the fields of application and the interaction with the other products in the environment
- an overview of the range of SQL functions in SESAM/SQL
- use of the utility statements
- the options available for protecting SESAM/SQL databases against unauthorized access
- the backup concept and media recovery options in SESAM/SQL
- fundamental principles of database operation, including distributed processing with SESAM/SQL-DCN
- the stages involved in creating, loading and maintaining SESAM/SQL databases.
- interoperation between SESAM/SQL and openUTM
- the glossary of SESAM/SQL

The other SESAM/SQL manuals are based on the contents of the “Core Manual” and primarily contain descriptions of statements in alphabetical sequence and specialist topics.

Readme file

For information on any functional changes to the current product version or extensions of the manuals for SESAM /SQL-Server, please refer to the product-specific Readme file.

In addition to the product manuals, Readme files for each product are available to you online at http://manuals.ts.fujitsu.com. You will also find the Readme files on the Softbook DVD.

Informations under BS2000

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

`SYSRME.<product>.<version>.<lang>`

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `SHOW-FILE` command or an editor. The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product’s files are stored.

Supplementary product information

Current information, version and hardware dependencies and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at http://manuals.ts.fujitsu.com.

Changes since the last edition of the manual

A list of the most important changes is contained in the table below. The table also shows the manual and chapter /section in which you will find a description of each change. If a topic is described in more than one manual, the one which contains a complete description is listed first. The entries in the “Manual” column have the following meanings:

Core: Core manual

RM P1: Reference Manual, Part 1

RM P2: Reference Manual, Part 2

DBO: Database Operation

Utilmon: Utility Monitor

Perform: Performance

Topic	Manual	Chapter
SQL Reference Manual		
Utility Monitor		
DBH administration		
RECOVER-OPTIONS SYSTEM- and USER-DATA-BUFFER according to DBH options: neuer Default-Wert *STD Neue Spalte im View SYS_VIEW_DALOGS	 DBO RM P1	 Ch.3.2 Ch.10.2
Dump in service task: new administration command diagnosis documentation dump in service task	 DBO	 Ch. 5.2 Ch. 9.5
Remote access to SESAM/SQL databases		
Performance		
Demonstration database		

Table 1: Changes in SESAM/SQL-Server V9.1 made since V9.0

Notational conventions

Because of the frequency with which the server names are used, the following abbreviations are employed to make things simpler and more straightforward:

- **BS2000 servers** for the servers with /390 architecture and the servers with x86 architecture. These servers are operated with the corresponding BS2000 operating system.
- **/390 servers** for the Server Unit /390 of the FUJITSU Servers of the BS2000 SE Series and the Business Servers of the S Series
- **x86 servers** for the Server Unit x86 of the FUJITSU Servers of the BS2000 SE Series and the Business Servers of the SQ Series (x86-64 architecture)

The string `<ver>` specifies the current version if the information is otherwise versionindependent.

The following notational conventions are used in this manual:

—————— ——————	Syntax definitions
UPPERCASE	SQL keywords
underscored	Default values
bold	Used for emphasis in running text
italics	Variables in syntax definitions and running text
`Fixed-space font`	Program text in syntax definitions and examples
`::=`	Definition character The specification to the right of ::= defines the syntax of the element on the left.
[ ]	May be omitted. The brackets are metacharacters and must not be entered in an SQL statement.
{ | | }	Alternative specifications in syntax definitions (on a single line). The braces are metacharacters and must not be entered in an SQL statement.
{ | | }	Alternative specifications in syntax definitions (over several lines). Each line contains one alternative. The braces are metacharacters and must not be entered in an SQL statement.

,...	In syntax definitions, a comma followed by three dots means that you can repeat the preceding specification any number of times, separating each specification with a comma. If you do not repeat a specification, you must omit the comma.
...	In syntax definitions, an ellipsis means that you can repeat the preceding specification any number of times. In examples, the ellipsis means that the rest of the statement is of no significance to the example. The ellipsis is a metacharacter and must not be entered in an SQL statement.
	Indicates notes that are of particular importance.
	Indicates warnings.

SESAM/SQL relational database system

SESAM/SQL is a relational database system which has proved its worth in a wide range of applications. It is employed by several hundreds of thousands of users for a variety of tasks in commerce, administration and science.

The particular strength of SESAM/SQL lies in its use in online transaction processing (OLTP) applications. A large number of SESAM/SQL applications exist which allow several thousand users to access common data resources simultaneously. This involves the usage of tables containing gigabytes of data sometimes with up to 250 million records. There are thus no upper limits on the performance provided by SESAM/SQL. Users can rest assured that SESAM/SQL is able to deal with considerable increases in the number of users and the volume of data handled.

SESAM/SQL fulfills all the demands placed on a modern database system in today's world.

The following sections describe the relational aspects of SESAM/SQL along with the other features and the product environment. This will provide a comprehensive overview of SESAM/SQL with regard to the access options and various fields of application.

Advantages of the relational model

SESAM/SQL is used for administering and processing user data structured in accordance with the relational model. The relational model was originally formulated by E.F.Codd in 1970 on the basis of relational algebra and since then it has been both specified more precisely and expanded a number of times (e.g. in Codd “The Relational Model for Database Management Version 2”, 1990).

The principle characteristics of the relational model are as follows:

- The relational model is characterized by the fact that the data structures are represented in the form of tables comprising rows (records) and columns. Logically, any relational database is made up of tables. The user sets up relationships between tables via the contents of the columns. This representation allows end users to understand the structure of a database intuitively.
- Data manipulation in the relational data model is set-oriented, i.e. it is intended for use with sets of records and not with individual records. Even complex operations can be constructed from simple, set-oriented basic operations.
- Relational database systems guarantee the logical and physical independence of data. Logical independence of data means that a user can create or modify his/her view of the data without changing the overall structure of the data. Existing programs need not be changed if new columns or tables are created. The deletion or modification of columns only means that those programs which access these columns need to be modified.
  Physical independence of data means that the logical and physical structure of a database are independent of each other. Logical relationships between data structures can be formulated in terms of links between columns and require no specifications as to the internal storage structure of the database. Application programs need not be changed if the physical data structure is modified.

SESAM/SQL allows users to exploit fully these advantages of the relational model.

SQL interface in SESAM/SQL

Ever since Codd (1970) first formulated the relational model, developers have been working on a database language based on the relational model. The early eighties saw the first commercial implementations of a language interface of this type under the name SQL (Structured Query Language). SQL was first standardized by the International Organization for Standardization in 1987 (ISO/IEC 9075:1987).
Today, SQL is the most widespread language for processing relational databases. In practice, a database system is referred to as relational if it supports the SQL interface.

In SQL, data is mapped to tables in the same way as in the relational model. Here, data is queried, inserted, updated or deleted. Complex database operations can be formulated by linking tables or the result sets of several different queries.
The basics of the SQL language for relational databases are easily learned and yet it is still suited for complex applications. The English language is used as the basis for the formulation of database operations. SQL includes language resources for simple, setoriented basic operations to which all manipulation of data can be reduced. Unlike the procedural languages used with non-relational database systems, SQL is a descriptive language. This means the user describes the result of a database operation in a set-oriented form (as in the relational model) rather than the various steps which lead to this result. A single SQL statement can thus lead to a number of operations on the database, which would require a large number of statements in a record-oriented language.

SESAM/SQL is based on the **ISO/IEC 9075:<year>** standards, in short: the SQL standard. The current standard is ISO/IEC 9075:2008, in short: SQL08.

SESAM/SQL includes a wide range of standardized SQL language resources for aspects such as

- data definition including the administration of access rights
- data manipulation
- transaction management
- dynamic SQL.

The ISO/IEC 9075:1992 standard distinguishes three levels of conformance: Entry SQL, Intermediate SQL and Full SQL. SESAM/SQL conforms fully with Entry SQL and also includes important functions from Intermediate SQL and Full SQL.

The current standard has not adopted this three-level subdivision. Core SQL has been introduced as the language core. Core SQL is a genuine superset of Entry SQL and is fully supported by SESAM/SQL.

In addition to the SQL functions defined by the standard, SESAM/SQL has its own SQL language resources for handling functions which have not been standardized, such as

- processing of multiple columns
- administration of storage structures
- administration of user entries
- execution of utility statements
  Utility statements are statements which use SQL syntax and which provide utility functions for database administration, for instance for generating, loading, unloading, copying and reorganizing databases.

When users compile SQL programs, they can select the scope of the SQL language supported, thus allowing the development of portable applications.

SESAM/SQL allows SQL statements to be used in a number of different environments:

- within the host language COBOL as embedded SQL statements (embedded SQL=ESQL).
- within DRIVE, a 4th-generation programming language.
- via the utility monitor, which provides a menu-driven interface for database administration.
- via the ADO technology defined by Microsoft (ActiveX Data Objects) and the ADO.NET interface in client applications on Windows systems.
- via the JDBC interface for access to SESAM/SQL from Java programs or Java applets on any platform.
- via the PDO driver of SESM/SQL for the PHP interface (**P**HP: **H**ypertext **P**rocessor) on an Apache Webserver in an Application Unit under Linux on a FUJITSU Server BS2000 of the SE Series.

SESAM/SQL uses the SQL language resources for defining and manipulating data in accordance with the standard and provides all the important database administration functions using SQL syntax. SQL thus represents a standardized, uniform language for the various user groups such as end users, application programmers and database administrators, thus facilitating the task of writing portable database applications. The utility monitor is a component part of SESAM/SQL and provides a simple interface for all database administration tasks.

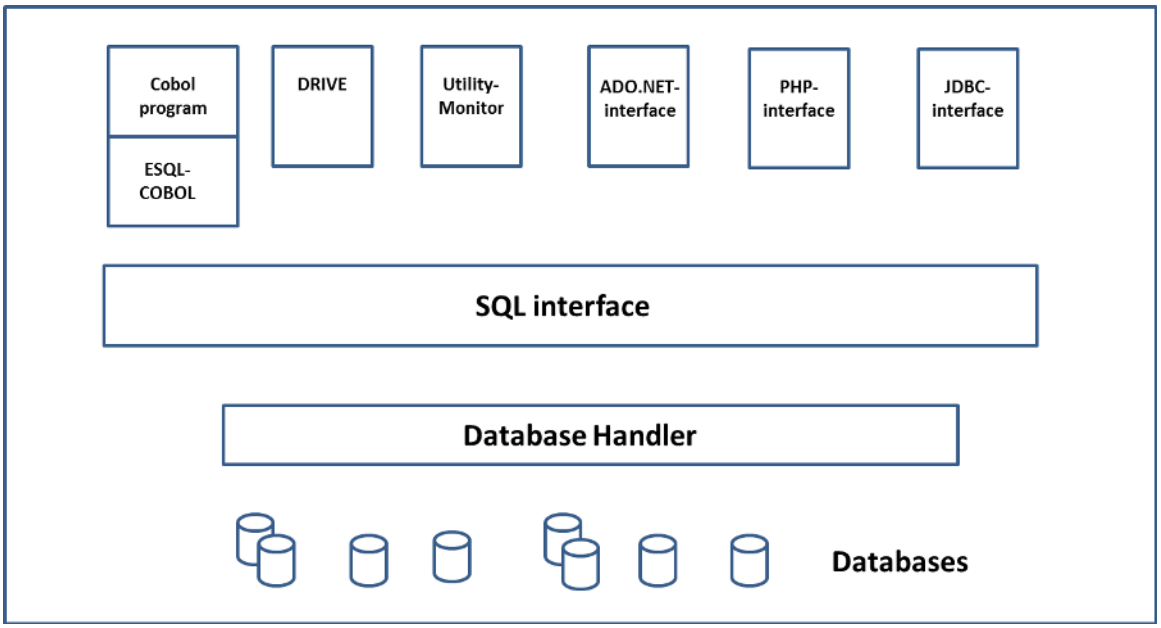

Figure 1: SQL interface

Unicode concept in SESAM/SQL

In the course of increasing internationalization, the Unicode character set is also becoming extremely important in BS2000 and its applications. Detailed information on this subject can be found in the manual “Unicode in BS2000”.

Unicode support in BS2000 is embedded in the existing concept of coded character sets (CCSs), see the manual “**XHCS (BS2000)**”.

A prerequisite for using Unicode is a suitable BS2000 system environment. The products which are currently being relevant for SESAM/SQL are ESQL-COBOL, COBOL, CRTE and XHCS, see the Release Notice for SESAM/SQL-Server.

The concept of Unicode support in SESAM/SQL enables Unicode characters to be used in the columns of tables and takes into account coded character sets in databases, in I/O files and for user programs. This concept influences the SQL language description, the utility functions, and the SESAM/SQL user programs.

EBCDIC character sets

The standard character set in BS2000 is EBCDIC.DF.03IRV (CCS name `EDF03IRV`), a 7-bit character set whose character repertoire is the same as that of the ASCII 7-bit character set, extended by the second control character block of ISO8859-1.

EDF03IRV comprises only 191 characters, 95 of these being printable characters. In their character repertoire the 8-bit character sets EDF04x (x=1, 2, 3, 4, 5, 7, 8, 9, F) are the same as the corresponding character sets ISO8859-x. EBCDIC character sets all contain the same EBCDIC core (79 characters). The 8-bit character sets also contain language-specific characters, e.g. Greek characters in EDF047. No special aspects had to be taken into account for your application in SESAM/SQL V4.0 or earlier. The database’s coded character set (`CODE_TABLE`) had comment characters.

One byte is sufficient for the binary presentation (coding) of an EBCDIC character. The length of an EBCDIC character is consequently one byte.
To ensure compatibility with the Unicode character set, the length in code units (1 code unit = 1 character = 1 byte) is also defined for EBCDIC strings.

Values and columns with alphanumeric (EBCDIC) data types are compared and sorted in binary format:
Latin lower-case letters (a-z) < Latin upper-case letters (A-Z) < numbers (0-9).

In the SESAM/SQL suite of manuals the term “alphanumeric” expresses the affiliation to an EBCDIC character set, e.g. alphanumeric data type, alphanumeric value, alphanumeric literal.

Unicode character set

Unicode incorporates all the known text character in the world in a single character set. Furthermore, Unicode is not dependent on different vendors, systems and countries.

In Unicode each character is assigned a number, which is referred to as the **code point**. A Unicode code point is generally specified in the form `U+n`, where `n` consists of 4 to 6 hexadecimal digits. Example: the Euro character €: `U+0020AC`.
The Unicode character set contains over one million code points.

So-called surrogate pairs are used to represent Unicode values above `FFFF` hexadecimal):

Code point range	UTF-16 encoding (2-byte)	Comments

	D800 ... DBFF	Leading Surrogates
	DC00 ... DFFF	Trailing Surrogates
U+010000 ... U+10FFFF	Surrogate Pair	Leading Surrogates followed by Trailing Surrogates

Table 2: Surrogate representation

“Noncharacters” are code points which are reserved in Unicode for internal purposes. In SESAM/SQL they may not be used in Unicode character strings.

The table below shows the code point ranges of the noncharacters and how these characters are encoded in UTF-16.

Code point range	UTF-16 encoding (2-byte)	Comments
U+00FDDx, U+00FDEx	Not permitted	32 code points
U+0xFFFE, U+0xFFFF	Not permitted	32 code points
U+01FFFE, U+10FFFF	Not permitted	2 code points

Table 3: Unicode noncharacters (x is a hexadecimal digit)

Code points are represented in bytes (encoded) for use in data processing in various ways. For this purpose the Unicode Consortium defines three different encoding forms: UTF-8, **UTF-16** and UTF-32.
The length of Unicode strings in the particular encoding form is specified in “**code units**” here. If only the so-called “Basic Multilingual Plane” (BMP, corresponds to UCS2) is used for encoding form UTF-16, then 1 code unit = 2 bytes.

SESAM/SQL uses the encoding form UTF-16, to represent Unicode characters in the databases.

Consequently precisesly two bytes, i.e. one code unit in UTF-16, are required for binary representation (encoding) of a UTF-16 character in SESAM/SQL. Values and columns with Unicode data types are compared and sorted (in relation to UTF-16) in binary format:Numbers (0-9) < Latin upper-case letters (A-Z) < Latin lower-case letters (a-z)

In the SESAM/SQL suite of manuals the term “national” expresses the affiliation to a Unicode character set, e.g. national data type, national value, national literal.

The technical report of the Unicode standard describes (by analogy to the encoding for UTF-8 for ASCII servers) a UTF-EBCDIC encoding for EBCDIC servers. The characters of UTF-8 are represented in one byte and correspond to the characters of the ASCII 7-bit character set. Analogously, the characters of UTF-EBCDIC are represented in one byte and correspond to the standard character set EDF03IRV of BS2000. The advantage of UTF-EBCDIC (BS2000, CCS name UTFE) is that all BS2000 system programs which are traditionally restricted to the character repertoire EDF03IRV (e.g. the BS2000 command processor) can also process character strings in the encoding form UTF-EBCDIC without any adaptations.

> ¡ The encoding of UTF-EBCDIC in BS2000 is proprietary and is different from the encoding of other vendors.

Support of Unicode in SESAM/SQL

The data types NATIONAL CHARACTER (NCHAR) and NATIONAL CHARACTER VARYING (NVARCHAR) are provided in SESAM/SQL in order to support Unicode. Unicode data can be stored in columns with these data types.

In SESAM/SQL the data is stored in the encoding form UTF-16 (in the scope of the BMP). Since two bytes of memory (one code unit) are required for a UTF-16 character in the internal SESAM/SQL representation, the following maximum lengths apply for the Unicode data types:

- NCHAR: 128 characters (256 bytes)
- NVARCHAR: 16000 characters (32000 bytes)

The Unicode data types can be used in operations such as compare, concatenate and assign.

All functions that were previously provided in SESAM/SQL for CHARACTER (VARYING) are also available for the new data types NATIONAL CHARACTER (VARYING). The relevant constraints must be borne in mind here.

The new data types are also supported in the corresponding DDL and Utility statements, see the CREATE TABLE and ALTER TABLE statements in the „SQL Reference Manual Part 1: SQL Statements“ and the CREATE CATALOG, ALTER CATALOG, LOAD, UNLOAD, EXPORT and IMPORT statements in the „SQL Reference Manual Part 2: Utilities“.

Coded character set of the database

To interpret (alphanumeric) character sets in columns with the CHARACTER and CHARACTER VARYING data types correctly, SESAM/SQL must know the coded (EBCDIC) character set in which the data has been encoded.

When a new database is created (using CREATE CATALOG) or an existing database is modified (using ALTER CATALOG), you can therefore specify or change the database's CCS name using the `CODE_TABLE` parameter. In SESAM/SQL the database's CCS name is used when data of the type (VAR)CHAR is converted to N(VAR)CHAR and vice versa. This is, for example, necessary when the data type of a column is changed and in some variants of the utility statements LOAD and UNLOAD.

Coded character set of the application

To interpret character sets correctly, SESAM/SQL must know the coded (EBCDIC) character set with which the application interprets the character strings.

With the new connection module parameter `CCSN` (CCS name) you notify the independent DBH of the character set which the application uses to interpret character strings. You notify the linked-in DBH of this using the DBH option LINKED-IN-ATTRIBUTES.

SQL statements can be processed in the DBH only if the application's CCS name is the same as the database's CCS name or if no coded character set is used for the database (`CODE_TABLE` has the value `_NONE_`).

Activating Unicode in an existing database

In SESAM/SQL databases up to and including V4.0 the CODE_TABLE parameter in CREATE CATALOG had comment characters and was stored as specified in the metadata. As this parameter now has a meaning, the first time an existing database is accessed by SESAM/SQL as of V5.0 it is assigned the value _NONE_, i.e. the database does not use a coded character set.

In order to use Unicode the database administrator must enter the coded (EBCDIC) character set currently used for the database using the utility statement ALTER CATALOG, e.g. EDF03IRV or EDF041 for a German BS2000 environment. The coded character set should also be defined for the applications (see above).

If the value _NONE_ is retained for the database, no implicit conversions from (VAR)CHAR to N(VAR)CHAR and vice versa are possible. All requests to the database which require such a conversion are rejected.

Example 1

A STAFF table contains the column LASTNAME with the data type CHARACTER(40). The CCS name of the database is EDF041; the data is therefore stored with the CCS name EDF041. You can use Unicode in two different ways:

1. The following DDL statement changes the data type of the LASTNAME column to NATIONAL CHARACTER, and the data stored in this column is converted implicitly from EDF041 to UTF16:

   ```
   ALTER TABLE STAFF ALTER COLUMN LASTNAME SET NCHAR(40)
   ```

2. The following statement defines an additional column:

   ```
   ALTER TABLE STAFF ADD COLUMN ALIAS_LASTNAME NCHAR(40)
   ```

   After this, the data is converted explicitly from their original column of data type CHARACTER to the target column of data type UTF16. The database's coded character set EDF041 is used here:

   ```
   UPDATE STAFF SET ALIAS_LASTNAME=TRANSLATE(LASTNAME USING CATALOG_DEFAULT)
   ```

Example 2

A STAFF table contains the column LASTNAME with the data type CHARACTER(40). The CCS name of the database has the value _NONE_. An additional column is defined using the following statement:

```
ALTER TABLE STAFF ADD COLUMN ALIAS_LASTNAME NCHAR(40)
```

After this, the data is converted explicitly from their original column of data type CHARACTER to the target column of data type UTF16. The coded character set EDF041 is specified explicitly here:

```
UPDATE STAFF SET ALIAS_LASTNAME = TRANSLATE(LASTNAME USING EDF041)
```

The following statement is rejected becasue the database's CCS name is _NONE_ and the conversion can therefore not be performed.

```
UPDATE STAFF SET ALIAS_LASTNAME=TRANSLATE(LASTNAME USING CATALOG_DEFAULT)
```

Transliteration, transcoding

The SQL function TRANSLATE() is provided in SESAM/SQL to convert alphanumeric strings (data type (VAR) CHAR) to Unicode strings (data type N(VAR)CHAR, character set UTF-16) and vice versa (transliteration).

TRANSLATE() also converts alphanumeric strings in the Unicode character set UTF-EBCDIC (BS2000, CCS name UTFE) to the Unicode character set UTF-16 and vice versa (transcoding). See the “SQL Reference Manual Part 1: SQL Statements”.

Normalization

The encoding of a character in Unicode is not unique, i.e. there could be more than one encoding for a character.

A typical example of this is provided by the German umlauts. For example, the character Ä has both the code point U+00C4 (composed form) and the code point combination U+0041 and U+0308 (decomposed form). In normalized presentation forms these differences do not occur. If two normalized strings differ, it is in their different code point presentations.

Non-normalized strings can result in consequent errors after a collation.

In SESAM/SQL the SQL function NORMALIZE() converts a string with national characters (data type N(VAR) CHAR) to a normalized form. Only those characters being taken into account which have code points in the range U+0000 through U+2FFF. Other characters, e.g. surrogates, remain unchanged. See the “SQL Reference Manual Part 1: SQL Statements”.

The normalization procedure requires CPU performance. The data of a SESAM/SQL database should therefore be available in normalized and compressed form.

If it is not certain whether (input) data in normalized form is available, normalization to normalization form C should be performed using the SQL function NORMALIZE().

Example

The code point U+1ED6 corresponds to the Latin upper-case letter „O“ with circumflex and tilde. This character can be generated with the help of three code points: U+00D4 for “Ô” and U+0303 for tilde or U+004F for “O” and U+0302 for circumflex and U+0303 for tilde. The code-point sequence U+00D5 for the „Õ“ with tilde and U+0302 for circumflex also produces this character. The only rule is that the base character must come before the diacritical marks linked to it.

In other words the result of normalization form C (compose) for NORMALIZE() for the uppercase Latin letter “O” with circumflex and tilde is U+1ED6, and the result of the normalization form D (decompose) is the code point combination U+004F, U+0302, U+0303.

> ¡ In BS2000 normalization functions are offered by the XHCS component, see the “ **XHCS (BS2000)** manual”.

Sort sequence

In most programs, character strings are compared in binary form. The binary sort sequence of the characters depends on their encoding:

- For all ISO8859 and Unicode encodings, the following applies:
  Numbers (1-9) < Latin upper-case letters (A-Z) < Latin lower-case letters (a-z)
- For EBCDIC and UTFE, the following applies:
  Latin lower-case letters (a-z) < Latin upper-case letters (A-Z) < numbers (1-9).

Umlauts and diacritical characters are not arranged unambiguously in these collating sequences. This can result in poor collating sequences, e.g. the name “Zuse” can appear ahead of the name “Öhler” in the collation of a list of names.

The Unicode standard defines a linguistic sort algorithm. Each Unicode character is assigned a collation element. The sequence in which the Unicode characters are sorted is defined with the aid of these collation elements. The collation elements are defined by means of a table supplied by XHCS (Default Unicode Collation Table, DUCET). This table contains a priority for the character at various levels. SESAM/SQL recognizes three levels, and these are displayed in the table below. The individual characters are always compared from left to right. The first difference determines the result of the comparison.

Comparison level	**Description**	**Example**
Level 1	Base character	a < b role < roles < rule
Level 2	Diacritics	A < Å role < rôle < roles
Level 3	Uppercase/lowercase	a < A role < Role < rôle

Level 1: Each base character (a,b,c, etc.) is assigned a permanent priority in the Default
Unicode Collation Table. The following characters or diacritics have no influence on the sort sequence of the characters.

Level 2: The base character has a diacritic. A diacritic is an additional character (e.g. accent, slash, dot, cedilla, tilde) which is paced in, above or below a character to define how it is pronounced or stressed. At level 1, a base character with a diacritic has the same priority as the associated base character without a diacritic. At level 2, a character with a diacritic has a higher priority than the same character without a diacritic. If the sort key is otherwise identical, the sort sequence is defined using this diacritic (see the example for level 2: role < rôle).

Level 3: The sort sequence is defined by the distinction between upper- and lowercase letters. Uppercase letters have a higher priority than lowercase letters. Level 3 is taken into account only if level 1 and level 2 are identical for the entire sort key (see the example for level 3: role < Role).

> ¡ The values for the collation elements, which are published on the Unicode Consortium website, may change. For this table, visit: *http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt*.

In BS2000 you can obtain the collation element via XHCS, see also the „**XHCS (BS2000)**“ manual.

The SQL function COLLATE() supplies the corresponding collation element of the Default Unicode Collation Table for national strings.
See the “SQL Reference Manual Part 1: SQL Statements”.

CALL DML interface

In addition to the SQL interface, the user also has access to the powerful, tried-and-tested CALL DML interface for querying and updating data.
Programs written in C, C++, COBOL, PASCAL, Assembler, PL/1, RPG, ALGOL and FORTRAN can process SESAM/SQL databases using this interface.

In addition to statements for updating, deleting and inserting records, CALL DML provides a cursor concept for set-oriented processing, transaction management functions and a wide range of data selection options.

CALL DML and SQL statements can also be used simultaneously within an application program and a CALL DML transaction.

All the functions of CALL DML are described in detail in the “CALL-DM Applications” manual, as are the tools available to CALL DML programmers, for instance for testing purposes.

Convenient administration

Large-scale database applications require a convenient means of administering the database system so that the system and the database administrator are able to handle even a large and ever-increasing number of administration tasks with ease. Therefore, SESAM/SQL provides numerous functions which provide support to the administrator.

SESADM administration program

SESADM allows database operation to be controlled from a form-driven interface compliant with the SDF conventions. SESADM allows system administrators to parameterize and administer SESAM/SQL DBH and the SESAM/SQL distribution component SESDCN using a uniform user interface. SESADM operation is described in the “Database Operation” manual.

Utility monitor SESUTI

The utility monitor is a component of SESAM/SQL and provides a convenient interface for all database administration tasks including constructing, loading, backing up and recovering a SESAM/SQL database.

The utility monitor can be used in interactive or batch mode. The various actions can be logged in a command file, which can then be modified by the user and used for recurrent activities.

The menu-driven interface of the utility monitor allows the database administrator simply to use the SQL language constructs provided for all tasks in interactive mode, i.e. to execute functions without the need to program SQL statements.

The utility monitor executes the following groups of SQL statements:

- data definition statements including those for the administration of access rights
- statements for the administration of the storage structure and user entries
- Utility statements
- statements for manipulating data

The utility monitor also provides functions giving information on the metadata of a database such as descriptions of base tables, views, integrity constraints, access rights etc. and which allow the user to read and modify the information on the SESAM backups available. It is also possible to call SESADM from the utility monitor in order to issue statements for controlling database operation.

The “Utility Monitor” manual provides a detailed description of how to use the utility monitor.

Performance monitor SESMON

The performance monitor SESMON provides the administrator or (via SNMP) a management platform with detailed and clear information on the utilization of resources (e.g. utilization of buffers, disk access). This allows the database system to be tailored to the application involved. Refer to the overview of SESMON and to the “Database Operation” manual for detailed information.

Access from the World Wide Web

You can also access the administration program SESADM, the performance monitor SESMON and the utility monitor SESUTI all from a unified access on the World Wide Web (WWW or Web for short) with the aid of the software product WebTransactions (WebTA).

To access the SESAM programs via the Web, you only need a standard browser in addition to the software product WebTransactions.

Web access is described in the document “WebTA access for SESAM/SQL” shipped together with SESAM/SQL-Server. This document is also available from our manual server under the software product SESAM/SQL.

Utilization of HSMS

HSMS (Hierarchical Storage Management System) can be used to back up tapes in SESAM/SQL. HSMS is used to back up data and support data management on external storage media in BS2000. In particular, HSMS supports the co-ownership system used in the SECOS software product which you can use to administer access rights to external user IDs.

Utilization of ARCHIVE

SESAM/SQL uses the high-performance tape backup system ARCHIVE. Special SESAM/SQL language resources are used to control the ARCHIVE functions. ARCHIVE is a standard backup system under BS2000 and is utilized across a range of other applications.

Sending important information of the DBH session by email

The SESAM/SQL system administrator can also have important information of the DBH session transferred automatically by email via the mail transmitter of the software product interNet Services.
An overview of the procedure is provided on chapter ("Sending important information of the DBH session by email") and it is described in detail in the “Database Operation” manual.

Logging security-relevant events with SAT

SESAM/SQL logs security-relevant events using the component SAT (Security Audit Trail) of the software product SECOS (Security Control System). The SESAM system administrator can enable and disable SAT logging as required.

High performance

SESAM/SQL is a powerful database server for all areas of application and is able to execute different types of tasks without the tasks interfering with each other. A distinction is made between two basic task types:

- OLTP applications
  In OLTP applications, a large number of users access the same databases and application programs. Examples of such applications are order processing systems, reservation systems and inventory management systems. This is often referred to as “productive operation”.
  OLTP transactions normally comprise a relatively small number of read and write statements. This involves repeating a few different transactions a great number of times.
  Refer to section "Online transaction processing (OLTP)" for a more detailed treatment of OLTP.
- End-user computing (EUC)
  End-user computing - e.g. statistical evaluations - usually involves extremely complex database queries. This often involves reading entire databases and performing complex calculations.

SESAM/SQL makes use of state-of-the-art technologies in conjunction with the transaction monitor openUTM to implement highly efficient OLTP operation with fast response times. At the same time, parallelism functions and a powerful optimizer ensure that end-user computing does not impede OLTP operation. This is an important prerequisite for the creation of client/server architectures in which, for example, PC applications (spreadsheets, word processing, etc.) initiate complex queries in order to edit or further process data. The most important technologies and functions are described below.

Multithreading

Multithreading architecture enables the SESAM/SQL Database Handler (DBH) to process jobs in parallel, thus making use of the time in which jobs wait for the completion of I/O operations. The processing of a different executable job is activated for the duration of the I/O operation. This results in a considerable increase in throughput. It also allows longrunning and complex database queries to be processed piecewise without impeding OLTP operation. The load determines the number of threads the DBH uses, i.e. not all the threads are always in use.

Multitasking

SESAM/SQL-Server is available in a standard edition for single-task operation and in an enterprise edition for multitask operation.

In a multitasking architecture, the SESAM/SQL Database Handler (DBH) can be loaded with several tasks when performance requirements are high. This allows the DBH load to be distributed to several processors in multi-processor systems.

the relocation of CPU-intensive actions.

SESAM/SQL relocates CPU-intensive activities to so-called service tasks allowing them to run in parallel to the actual DBH operation. Service tasks are available for instance for CPUintensive database administration functions and for sorting intermediate results.

Cost-based optimizer

When an application issues an SQL statement, SESAM/SQL creates an access plan. This plan describes the type and order of the individual processing steps of the SQL statement. The cost-based optimizer ensures that a particularly efficient access plan is created, in which as few system resources as possible are used (CPU time, I/O accesses, etc.).

Shared SQL

The optimized access plan is maintained in main memory and can be used by several different users. Shared SQL provides significant performance improvements, in particular for OLTP applications where specific processing steps are repeated many times.

Shared record lock

If a read access is executed on a record, this record is normally locked for other transactions. A “shared record lock”, however, makes it possible for other transactions to read this record. This reduces the number of locks and more parallel transactions can be executed. The transaction performance increases. Transaction security is not impeded in any way by this extended locking concept, since shared record locks are only possible when data is not being updated.

Data compression

SESAM/SQL automatically compresses data when it is stored. Thus reduces storage space requirements.

- Only significant values are stored, non-significant values do not occupy any storage space.
- The NUMERIC and DECIMAL data types are stored with their significant length only, without leading zeros.
- The CHARACTER data type is only stored in its significant length with no trailing spaces.

Compressing the data to significant values can configure the database to cope with maximum requirements. Thus making it simple:

- to define columns for which only a few records contain values
- to define columns when setting up the database, even though they are not needed until subsequent applications
- to retain column definitions even if there are now no values for them.

Because the compressed records are shorter and therefore occupy fewer storage blocks, data can be accessed more rapidly. It is also possible to keep more records in main memory, thus reducing the number of disk accesses.

SESAM/SQL-LINK - the linked-in variant of SESAM/SQL

The linked-in variant is designed to allow particularly efficient processing of a single application program. This involves linking the DBH permanently into the application program. The connected databases are assigned exclusively to the application program. The advantages over the independent DBH lie in the fact that the linked-in variant spends no time on communication between the application program and the DBH and expends no resources on administering locks.

The application area of SESAM/SQL-LINK lies in the processing of batch programs which access the database exclusively.

Any differences in functionality or aspects which need to be observed when using the linked-in DBH are described in the SESAM/SQL manuals at the point where the user requires this information.

SESAM/SQL-LINK is not available for SX servers.

Global storage

The performance of a database system is not influenced by the power of the processor alone. In database systems in particular, a large number of read and write accesses to disk are performed which, compared to main memory, are relatively slow. For this reason especially fast storage media were developed for the BS2000 systems. For example, global storage - a battery-buffered semiconductor storage device - provides an access time 2000 times faster than hard disks. SESAM/SQL can use global storage as a data cache thus considerably accelerating read and write access to disk. This in turn leads to a significant improvement in overall performance.

Block mode

In block mode, you can define a cursor. Block mode causes several records to be placed in a buffer the first time the FETCH NEXT statement is executed. When the FETCH statement is executed for the first time, the user is only given the first record. Each time the FETCH statement is executed again, a further record is provided from the buffer (without further task-to-task communication) until the buffer is empty. The next FETCH statement again places several records into the buffer. This block mode speeds up cursor processing considerably.

64-bit load variant of the SESAM/SQL DBH

The 64-bit load variant of the SESAM/SQL DBH is loaded automatically on all current BS2000 servers with SESAM /SQL.
This can be recognized from the DBH start message on the insert “(64-Bit VERSION)” for /390 servers and “(X86-64-VERSION)” for x86 servers.

Enhanced buffer options

The 64-bit load variant permits more powerful handling of the input/output load through a higher maximum value for the buffer for system-access data (DBH option SYSTEM-DATA-BUFFER) and for the buffer for user-access data (DBH option USER-DATA-BUFFER) , see the “Database Operation” manual.

At the same time in this case the buffers do not utilize the normal address space of the task (2 GB), which means that greater values are possible for the other options.

Constant availability and excellent reliability

Depending on organization structures within a company, ever-increasing demands are being made on OLTP applications with regard to availability:

- backup systems must be started automatically in the event of an error
- the bulk of maintenance operations must be performed during normal operation.

In this regard, SESAM/SQL provides a number of functions which allow it to satisfy these requirements.

Dynamic reconfiguration of the DBH session

The DBH administration commands RECONFIGURE-DBH-SESSION and RELOAD-DBH-SESSION are also available to the SESAM/SQL system administrator for increasing the availability of the independent DBH, for importing a correction version of the DBH while DBH operation is in progress, and for dynamic reconfiguration of the DBH session (see the “Database Operation” manual).

Both administration statements are executed without interrupting the DBH session. From the user viewpoint no DBH failure occurs.

Automatic extension of database limits

If, as a result of extensive additions and updates, the defined limits of the database prove to be too narrow, they are extended automatically during the session. Because the extension is performed automatically, availability is increased considerably.

Online data definition and utilities

Database administration tasks such as loading, backing up and applying the modifications logged in the logging files to SESAM backups can be performed during normal operation, i.e. the database does not have to be deactivated.
Databases can also be built online. The logical database description (schema) of an existing database can also be modified online.
This means that situations in which a database has to be deactivated only occur extremely rarely.

Using replications

A replication is a copy of a database which can be used to restore a database, or it can be used as a shadow database for a recovery.

Using replications as shadow databases can speed up recovery considerably.

Utilization of foreign copies

With SESAM/SQL-Server it is possible to create foreign copies without having to shut down database operation. In addition, foreign copies can be used directly for recovery purposes, i.e. without having to use a replication.

Utilization of the warm start time

SESAM/SQL-Server makes it possible to start OLTP operation even during the warm start period and also allows you to influence its duration.

Space concept

SESAM/SQL derives the physical structure of the database automatically from the logical database description (schema).

The Storage Structure Language (SSL) allows users to optimize the organization of storage space in accordance with their own particular needs, thus enabling them to accelerate certain accesses, for instance. One of the measures which affects the storage structure is the physical division of the database into a maximum of 1000 spaces (files). This means that it is often only possible to deactivate the relevant spaces rather than the entire database. This means, for instance, that backup operations or recovery of corrupt portions of the database can be restricted to individual spaces.

On pubsets with “large files” a space can be up to 4 TB in size. Otherwise it can be up to 64 GB in size.

Online transaction processing (OLTP)

OLTP applications are characterized by the fact that a large number of users involved in different tasks work simultaneously and interactively with the same data and programs. This is generally controlled by a transaction monitor. Typical fields of application for OLTP applications are reservation systems, information systems and inventory management systems.

OLTP applications in BS2000 are implemented using the universal transaction monitor openUTM. openUTM is fully integrated into the operating system and, together with SESAM/SQL, forms an extremely high-performance OLTP system. Refer to the openUTM manuals and the openUTM “Concepts and Functions” manual in particular for details on openUTM.

openUTM can be seen as a sort of job processing center, where it coordinates and distributes the jobs received from the various terminals. SESAM/SQL ensures that there are no complications when a number of end users access the databases simultaneously from different processing steps. Processing is carried out in logically discrete units or transactions, which are synchronized between openUTM and SESAM/SQL. Transactions are only committed or rolled back jointly by openUTM and SESAM/SQL. A user working at a terminal knows at any time his /her precise position within a sequence of processing steps. The databases can be returned to a consistent state at any time. Refer to section "Transaction concept" for further details on the synchronization of openUTM and SESAM /SQL transactions.

The power of openUTM and SESAM/SQL is illustrated particularly vividly in large-scale installations where several thousand users are connected and several hundred transactions per second are handled. openUTM is responsible for the following important functions in OLTP applications:

- openUTM ensures that response times remain short, even if large numbers of users are involved. This means that openUTM controls the processing of a large number of user requests with a small number of system tasks. This minimizes the administration overhead for the operating system.
- openUTM is responsible for monitoring, controlling and handling user jobs. openUTM assigns the jobs to the corresponding programs and controls communication with the database system.
- Processing is subject to transaction management, i.e. a sequence of related processing steps are executed either in their entirety or not at all.
- In the event of a system crash, openUTM synchronizes with the database system and activates an automatic restart of the application and of the screen form.

SESAM/SQL and openUTM thus form an ideal basis for secure, high-performance OLTP applications.

Development tools

A wide range of tools are available for developing SQL applications for SESAM/SQL.

Developers who work with 3rd-generation programming languages can create SQL applications under COBOL using ESQL products. For C applications in BS2000, the SQL calls must be bundled in an ESQL-COBOL module called from C.

DRIVE is a 4th generation programming language. It is easy to use and provides powerful language resources, allowing considerable improvements in productivity during the development process.

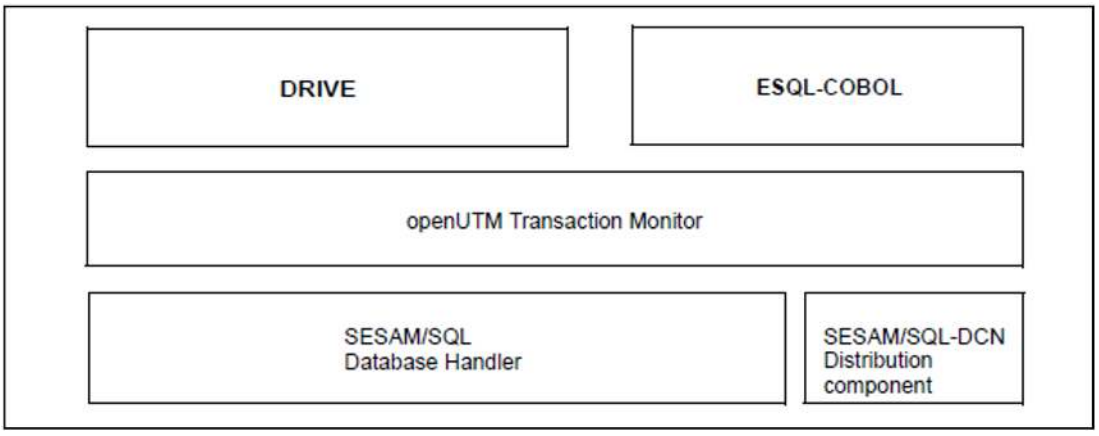

Figure 2: Development tools

DRIVE

DRIVE (BS2000), an easy-to-use 4th-generation programming language, is available for developing SESAM/SQL applications. DRIVE is ideal for developing OLTP applications under openUTM and client/server applications on BS2000 systems and for developing programs for end-user computing (EUC).
The DRIVE compiler is used in the event of particularly high requirements on throughput.

In client/server architectures, DRIVE supports distributed processing with openUTM-D, which is subject to transaction management.

Refer to the DRIVE resp. DRIVE/WINDOWS manuals for a detailed description of the DRIVE functions.

ESQL-COBOL

The ESQL-COBOL (BS2000) precompiler allows the execution of COBOL programs into which SQL statements have been embedded. This involves marking the start and end of an SQL statement, allowing the precompiler to identify the SQL statements. The precompiler then generates an SQL LLM (link and load module) in which all the SQL statements are grouped together. The COBOL program then contains no more SQL statements and is compiled normally using the COBOL compiler. The following diagram illustrates this process.

> ¡ Programs with national data types can only be (pre)compiled with the corresponding BS2000 system environment.

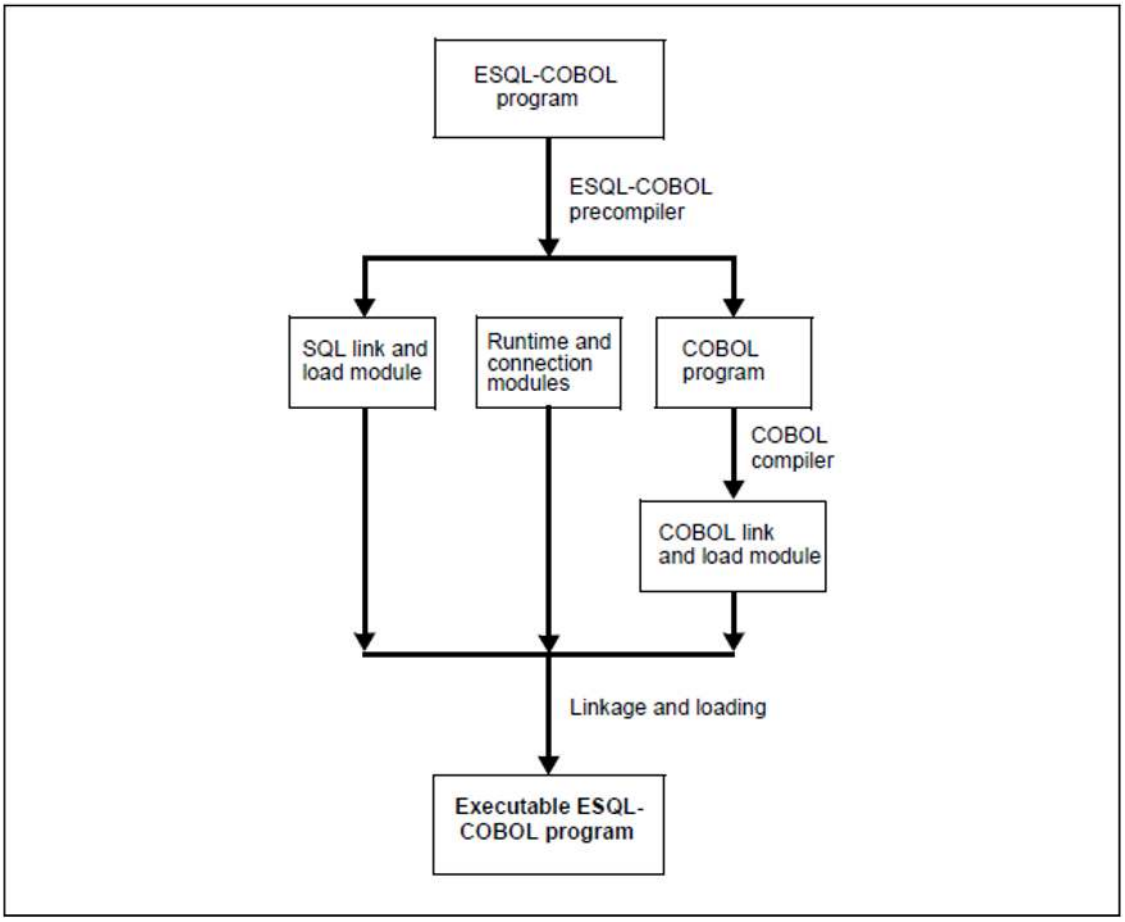

Figure 3: Use of the ESQL precompiler

Precompiler options allow the user to define the precompilation attributes. For instance, it is possible to specify that only SQL statements which conform to the standard are permitted, this allowing the generation of portable programs.

The use of dynamic SQL makes it possible to specify SQL statements during execution of an ESQL program. This means that it is possible to implement dialog applications where queries can be structured flexibly and can modify the database.

An ESQL program can also contain CALL DML statements. SQL statements and CALL-DML statements can also be contained together in a transaction. For this purpose, SQL statements are embedded in a CALL DML transaction.

Refer to the “ESQL-COBOL for SESAM/SQL-Server” manual for a detailed description of the ESQL precompiler.

> ¡ No ESQL-C precompiler is available for SESAM/SQL-Server. We recommend that you extract the SQL statements from a C program and include them in an ESQL COBOL program. The COBOL procedures can then be called from the C program using the SQL statements. The values for SQL host variables are passed to the COBOL procedure in the form of parameters.

SESAM/SQL in a client/server environment

Client/server architectures allow the power of the Data Center to be combined with the convenience of graphical user interfaces and the numerous standard applications available on PCs and workstations to provide an ideal overall system.

Different basic forms of client/server solutions are possible, depending on which of the services provided by the data center the client takes advantage of and which tasks the client handles itself.

Basic types of client/server architecture

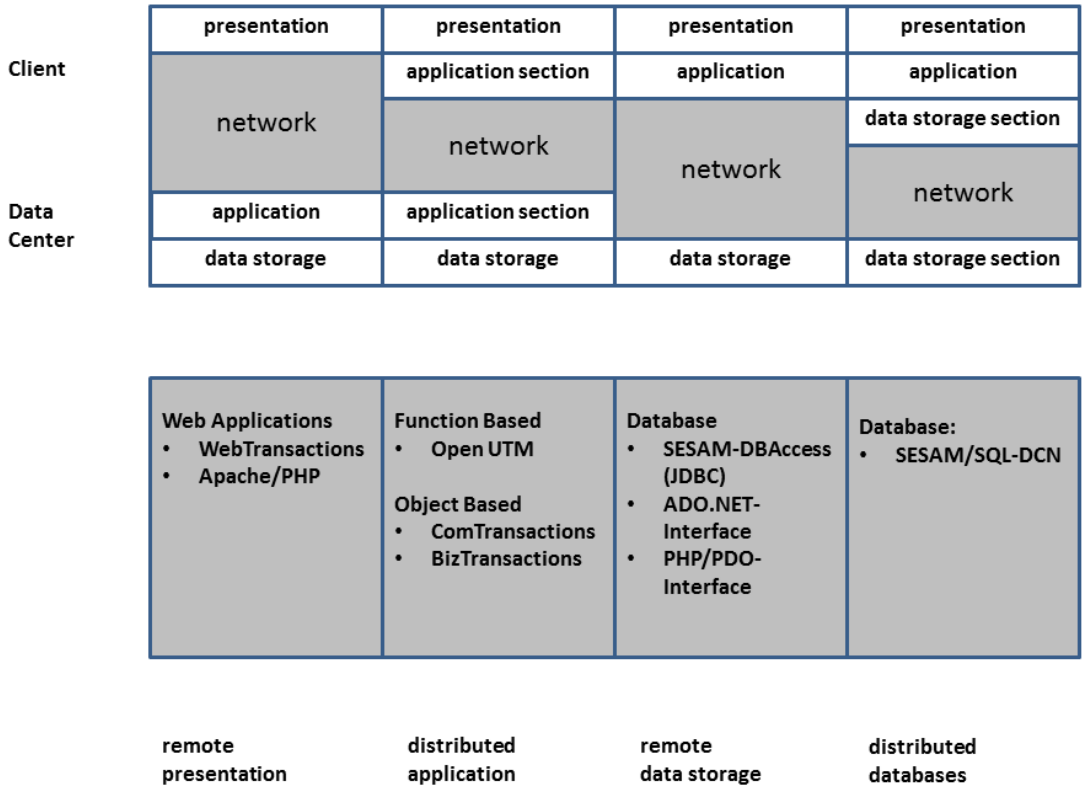

Figure 4: Incorporation in client/server architecture

Remote presentation

Graphical user interfaces use a considerable part of the available computing power for the processing of the display. If this function is transferred to a client system, the data center is relieved of part of its workload and is available for performing other tasks. This combines the qualities of an intelligent graphical workstation and a Data Center. The user can then access local and remote applications which all have the same “look and feel”. Utilization of the client/server option improves productivity and at the same time reduces the load on the Data Center.

Distributed application

With a distributed application, the individual tasks performed by an application are distributed to those systems which are best suited for them. This distribution of tasks means that it is possible to achieve improved overall computer performance.

Remote data storage

In the case of remote data storage, the client application works with a database located on a remote Data Center. This allows data to be stored centrally and thus in a consistent state. At the same time, the Data Center can use all of the available resources for data access, since it has no need to control either the application or the presentation.

Distributed databases are a special form of remote data storage.

Distributed databases

Distributed databases allow the structure of the organization to be reflected by the way in which the data is stored, e. g. in terms of headquarters and branches. The data is stored on the servers on which it is most frequently needed. This has the effect of alleviating the bottleneck caused by the network. This is of particular importance if public networks, which generally only support lower transfer rates, are used. This also means that data transfer costs are kept low.

SESAM/SQL provides additional components in client/server architectures with “remote data storage” and “distributed databases”.

Remote data storage with access via standard interfaces

Add-on products such as SESAM-DBAccess (JDBC interface) or the ADO.NET interface can access Windows systems on SESAM/SQL databases, thus allowing the databases to be processed and updated. The data can, for example, be read into a spreadsheet, or addresses located in the database can be used in a word processing program for the creation of form letters.

The server and client components of SESAM-DBAccess are supplied with SESAM/SQL and do not have to be ordered separately.

Remote access with ADO.NET

SESAM/SQL-Server supports the ADO.NET interface defined by Microsoft (ActiveX Data Objects) for Windows systems to communicate with database systems in the client/server environment.

Via the ADO.NET driver, which is supplied together with SESAM/SQL-Server, you can implement database-independent accesses to SESAM/SQL from the .NET environment in Windows.

Webserver with connection to SESAM/SQL

For FUJITSU Servers of the BS2000 SE Series, a PDO driver is offered on an Application Unit under LINUX for the PHP interface to an Apache Webserver.

Accesses to SESAM/SQL databases can be performed in PHP websites and scripts via this PDO driver.

> The PHP code is executed solely on the Server Unit. It remains hidden from the web user. Only the HTML code is transferred to the client. This satisfies the more stringent security requirements.

In addition to this new PDO driver, the PHP access functionality is, as previously, offered via an Apache Webserver under POSIX in BS2000.

Taking existing SESAM applications onto the Internet/Intranet: WebTransactions

Thanks to WebTransactions, it is an easy matter to make existing SESAM/UTM applications web-compatible. It is particularly easy in the case of applications that were created using DRIVE.

WebTransactions has already been successfully launched on the market and has a wide variety of uses in the integration of business applications and the associated data on the World Wide Web.

WebTransactions enables the following:

- host applications and data can be integrated unchanged into the Internet, or an Intranet or Extranet
- the host application's screen forms can be improved and rendered more attractive
- the host application can be adapted to meet specific needs by merging or extending dialog steps (dialog reengineering)
- it is even possible to merge multiple applications under one web interface and run these in parallel

Currently, all mainframe applications on BS2000 and MVS systems and OLTP applications with openUTM on BS2000, UNIX systems and Windows are supported.

Java: the SESAM/SQL-Server JDBC interface

The Standard Call Level Interface for Java applications, Java Server Pages, Java Servlets, and Java Applets when accessing SQL databases is JDBC (Java Database Connectivity). Java accesses that are independent of the database can run with SESAM/SQL-Server via the JDBC interface. Consequently, external Java applications can also be used with SESAM/SQL-Server.

SESAM/SQL-Server offers the JDBC functionality according to the JDBC standard V6.0. The corresponding drivers are present in SESAM/SQL-Server and are supplied on a volume together with SESAM/SQL-Server. They are type 4 drivers, i.e. they are “nativeprotocol fully Java technology-enabled drivers”. Compared to other types such as “ODBC/JDBC bridges” and “partly Java technology-enabled drivers”, this has the advantage that no binary code has to be installed on the client machine.

Distributed databases using SESAM/SQL-DCN

The supplementary product SESAM/SQL-DCN, the distribution component for SESAM/SQL makes it possible to process SESAM/SQL databases located on different BS2000 computers with a single application program. The application program does not distinguish whether the data is located on a local or remote computer when processing these remote databases.
The application program contains no information on the location of the data, i.e. distribution is transparent from the point of view of the application. All communication operations are invisible to the user. This means that application programs can be used with no modifications on any of the computers within a network.

In the case of update transactions involving databases on different computers, SESAM/SQL-DCN uses a two-phase commit protocol to ensure that data consistency is maintained across the network. SESAM/SQL thus guarantees a consistent data set in the network even in the event of data modifications on a number of computers. Deadlocks and longlocks are recognized and resolved by the system beyond the boundaries of a single computer.

Depending on the field of application, the use of SESAM/SQL-DCN has the following advantages:

- Higher performance:
  The processing of user requests on different computers results in an increase in throughput.
- Higher level of availability:
  If a computer fails, it does not mean that the whole system fails.
- Flexible organization:
  Work processes do not have to be oriented to a central computer center.

Refer to section “Distributed processing with SESAM/SQL DCN” for further details on using SESAM/SQL-DCN.

Other database-related products and applications

SESAM-KLDS - the compatible interface for linear data structures

Above all in public administration, it is normal for users to employ products from different vendors. This means that it is necessary for certain programs to run on all the different systems.

SESAM-KLDS allows you to create system-independent programs for processing linear data structures. The database calls are made via the database interface KLDS, defined under the auspices of the German Ministry of the Interior.
SESAM-KLDS converts database calls in KLDS to CALL DML statements. SESAM-KLDS application programs are also possible within a openUTM environment.

All the functions of the KLDS interface are described in a separate manual, “**SESAM-KLDS (BS2000)**”.

Remote output of information with SNMP

SNMP stands for **S**imple **N**etwork **M**anagement **P**rotocol and was developed as a protocol for network management services in the TCP/IP internet. SNMP’s range of application has since been extended to include system management, application management and even management of middleware products such as databases and transaction monitors. Similarly to TCP/IP, the name SNMP does not just stand for the protocol but for the entire management system which is based on SNMP. SNMP employs a client/server architecture, where the management platform is the client and the management agents are the servers.

There is an agent for the SNMP management of SESAM/SQL databases in BS2000 named SESAM/SQL agent. This subagent is contained in SNMP AGENTS (software product sesAgent) and supplies information on SESAM databases and SESAM-DBHs. SESAM/SQL agent is described in detail in the manual "SNMP Management (NET-SNMP) for BS2000" and the manual "SESAM database operation" (chapter "Outputting operational data with SESMON").

Demonstration database

Included in the SESAM/SQL-Server delivery is the demonstration database ORDERCUST. The structure of this SESAM/SQL database is described in section “Structure of the demonstration database”.

The examples in the SESAM/SQL-SERVER manuals refer to the ORDERCUST database.

You will find the fully structured database in the library SIPANY.SESAM-SQL.<ver>.MAN-DB.

The library SIPANY.SESAM-SQL.<ver>.MAN-DB contains all the components that you need to try out the examples in the manuals yourself as well as to develop your own applications within a clearly structured environment.

The following components are present in the library SIPANY.SESAM-SQL.<ver>.MAN-DB:

- Readme files with an overview of all the present files and a detailed introduction to using the database
- Start procedures to start the necessary SESAM/SQL programs
- The files of the constructed database ORDERCUST
- Instruction files and ESQL-COBOL programs with runtime examples of important database statements

Starting the demonstration database

Do the following to start the demonstration database:

1. Prepare the library SIPANY.SESAM-SQL.<ver>.MAN-DB and adapt it to the runtime environment:

   While some components of the demonstration database can be addressed as library members, the spaces in the ORDERCUST database, configuration files and load files containing user data must be copied to an operating ID.

   In addition, a number of the parameters in the supplied procedures, for example the system file IDs, are dependent on the chosen runtime environment.

   The Readme file in the library SIPANY.SESAM-SQL.<ver>.MAN-DB contains a detailed description of all the steps that are necessary before you first start the database.

2. Starting the DBH:

   The Data Base Handler (DBH) is the SESAM/SQL component that analyzes, executes and coordinates all the database accesses in a DBH session.

   To start the DBH, use the procedure supplied in the library SIPANY.SESAM-SQL.<ver>.MAN-DB.

   For more information on using the DBH, see the “ Database Operation” manual.

3. Starting SESADM (optional):

   You may want to start the SESADM administration program to administer the DBH.

   To do this, use the procedure supplied in the library SIPANY.SESAM-SQL.<ver>.MAN-DB.

   For more information on using the SESADM administration program, see the “ Database Operation” manual.

4. Starting the Utility Monitor:

   You use the Utility Monitor's menu-based interface to administer a database.

   To start the Utility Monitor, use the procedure supplied in the library SIPANY.SESAM-SQL.<ver>.MAN-DB.

   For more information on using the Utility Monitor, see the “ Utility Monitor” manual.

You can now access the database via the Utility Monitor menus.

Using the demonstration database

The ORDERCUST demonstration database enables you to try out some of the given examples in the manuals concerning SQL statements. You can build on these examples to develop your own SQL statements.

Selected SQL statements from the manuals are combined to form executable units for the demonstration database. The instruction files and ESQL-COBOL programs illustrate possible applications for SQL statements and are intended to give you ideas for the development of your own programs.
They are present in the library SIPANY.SESAM-SQL.<ver>.MAN-DB.

Individual SQL statements

Most of the examples in chapter “Range of SQL functions in SESAM/SQL” as well as in the manuals “ SQL Reference Manual Part 1: SQL Statements” and “ SQL Reference Manual Part 2: Utilities” are suitable for testing the ORDERCUST demonstration database.

In the Utility Monitor, call the SQL - SQL-STATEMENTS screen and enter the required SQL statement.

In the SQL screen, you can enter any preparable SQL statements and log these in a file. Consequently, this is a suitable environment for developing new SQL statements and then taking these over from the log file in a syntactically correct form into more complex situations.

For more information on using the Utility Monitor, see the “ Utility Monitor” manual.

Instruction files and ESQL-COBOL programs

Some of the most important components of the library SIPANY.SESAM-SQL.<ver>.MAN-DB consist of statement files and ESQL-COBOL programs. In these, the examples from the manuals are combined in a topic-specific way to produce executable demonstrations.

If an example is accompanied by the symbol on the left, this means that it is present
as a component in an instruction file or an ESQL-COBOL program in the library SIPANY.SESAM-SQL.<ver>.MAN-DB.

The instruction files and ESQL-COBOL programs cover the following topics:

Topic	Runtime environment
Reconstructing the ORDERCUST database	Instruction file
Loading user data in the ORDERCUST database	Instruction file
Unloading user data in different data formats	Instruction file
Focus on DDL statements	Instruction file
Focus on DML statements	Instruction file
Focus on utility statements	Instruction file
Focus on innovations in SESAM/SQL-Server V3.2, V4.0, V5.0, V6.0, V7.0, V8.0, and V9.0	Instruction files
Focus on partitioned tables	Instruction file
Focus on Unicode tables	Instruction file
Focus on procedures and User Defined Functions (UDFs)	Instruction files
Selecting records from a table	ESQL-COBOL program
Inserting records in a table	ESQL-COBOL program
Changing records of a table	ESQL-COBOL program
Deleting records from a table	ESQL-COBOL program
Using dynamic SQL	ESQL-COBOL program
Working with BLOB objects	ESQL-COBOL program
Use of procedures and User Defined Functions (UDFs)	ESQL-COBOL program

Do the following to start the instruction files and ESQL programs:

- Instruction files:
  In the Utility Monitor, open the IFP - INSTRUCTION FILE PROCESSING screen and enter the name of the instruction file.
- ESQL-COBOL program files
  Use the supplied procedure to start the ESQL-COBOL programs. Alongside the executable programs, the library SIPANY.SESAM-SQL.<ver>.MAN-DB also contains the source programs and the procedures required to compile and link a source program.

The Readme file in the library SIPANY.SESAM-SQL.<ver>.MAN-DB contains more detailed descriptions of all the necessary steps.

You can develop new instruction files and ESQL-COBOL programs in the EDT editor and save these as library members in the same way as the existing files.

Structure of the demonstration database

There follows an explanation of the physical and logical structure of the ORDERCUST demonstration database.
You will find references to additional information concerning the used terms and settings at the end of each section.

Storage groups

A storage group is used to group together volumes of a particular BS2000 catalog ID under a single name.

The spaces in the ORDERCUST database are distributed over three storage groups

- Storage group STOGROUP1
  for the catalog Space and the user spaces as well as the DA-LOG files.
- Storage group STOGROUP2
  for the CAT-REC and die CAT-LOG files
- Storage group STOGROUP3
  for the backup copies of the catalog space and the user spaces.

In the supplied version of the ORDERCUST demonstration database, all three storage groups are created in the default pubset of the BS2000 ID. It is therefore possible to access the database irrespectively of the current runtime environment.

However, in productive use the CAT-REC file should be created on a storage medium other than that used for the catalog space and the user spaces so that media recovery remains possible even after the loss of a storage medium.

For more information on the structure of a SESAM/SQL database and on media recovery, refer to section “SQL objects of a SESAM/SQL database” and to section “Media recovery”.

User spaces

The user data in the ORDERCUST database are distributed over the following three user spaces:

- The TABLESPACE space to store tables
- The INDEXSPACE space to store indexes
- The BLOBSPACE space to store BLOB objects

If tables and the associated indexes are created on separate spaces then the media recovery capabilities can be used in an efficient, differentiated way.

In the case of larger databases, you should create each table on a separate space in order to be able to control the access possibilities to the individual tables and to keep the down times due to recovery measures at a low level.

For more information on media recovery, see section “Media recovery”.

Schemas

The ORDERCUST demonstration database is subdivided into the following schemas

- ORDERPROC schema for administering orders,
- PARTS schema for administering components,
- ADDONS schema for administering BLOB objects.

In order to provide a complete description of the tables in the database, the table definition is shown in addition to the data contained in the table. Definition of a table using CREATE TABLE assumes a knowledge of terms such as name, data type and integrity constraint. These are described in section “SQL objects of a SESAM/SQL database”.

NULL values in tables are represented as empty fields.

Schema ORDERPROC

The schema ORDERPROC for the database ORDERCUST implements an information system for processing orders in a small data center. Information is to be recorded on customers, on contacts, on jobs done for these customers and on the services provided for these orders. The schema ORDERPROC contains the following tables: CUSTOMERS, CONTACTS, ORDERS, SERVICE and ORDSTAT.

The figure below shows an overview of the base tables of the ORDERPROC schema and the dependencies between these tables.

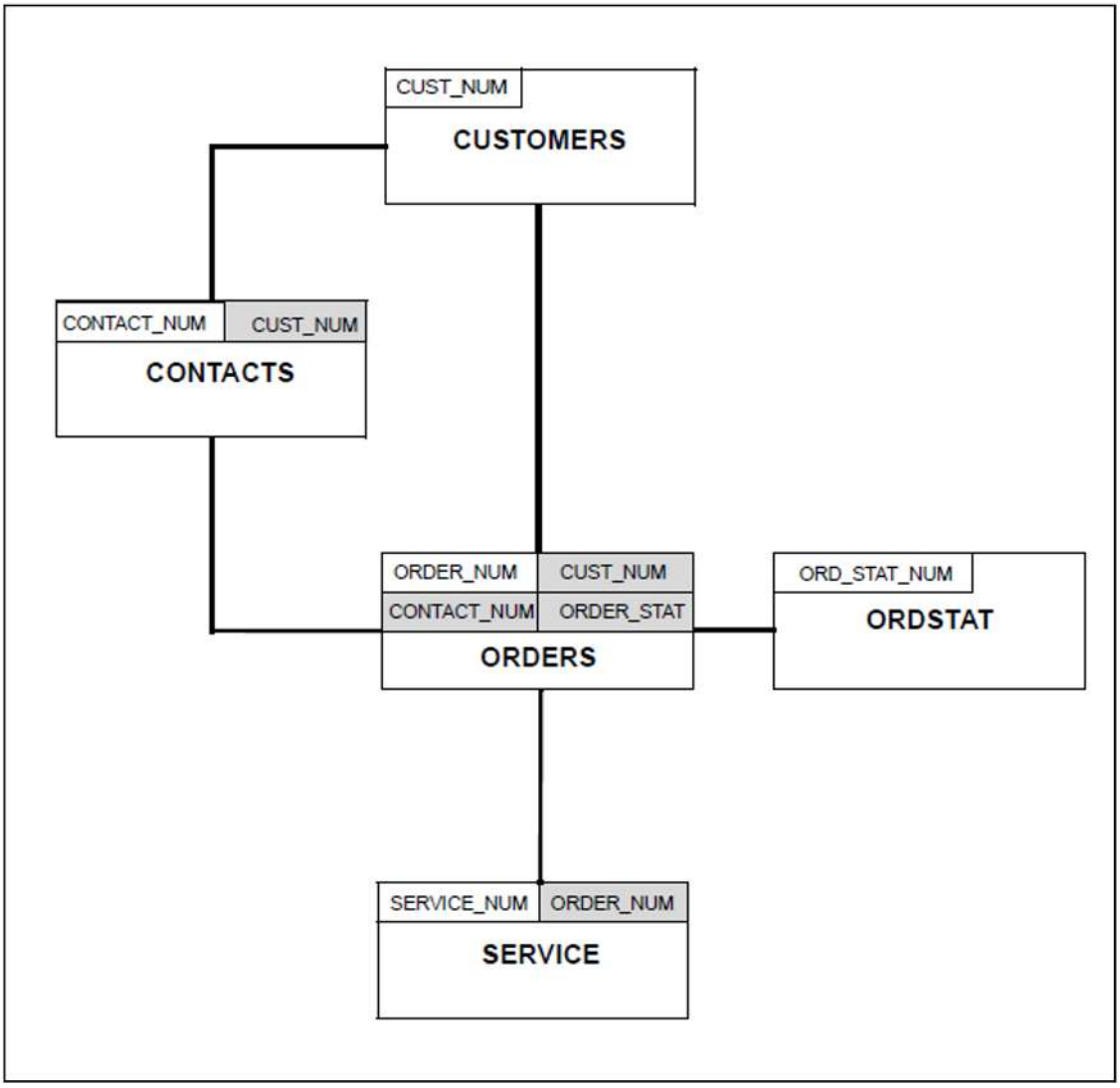

Figure 5: Base table for the ORDERPROC schema; referencing foreign keys have a gray background

CUSTOMERS table

The CUSTOMERS table contains information on the customers. In addition to a unique customer number, the table contains the name, address, telephone number and information on the customer's field of activity. The following table definition is used to create the CUSTOMERS table:

`CREATE TABLE`	`customers`

(cust_num	INTEGER CONSTRAINT cust_num_primary PRIMARY KEY,
company	CHAR(40) CONSTRAINT company_notnull NOT NULL,
street	CHAR(40),
zip	NUMERIC(5),
city	CHAR(40),
country	CHAR(3),
cust_tel	CHAR(25),
cust_info	CHAR(50)
CONSTRAINT PlausZip	CHECK(country IS NULL OR zip IS NULL OR
	(country = 'D' AND zip >= 00000)
	OR (country <> 'D'))
)	

CUST_NUM is the primary key of the CUSTOMERS table. A CHECK constraint has been defined to check the plausibility of the zip codes. The CUSTOMERS table contains the following data:

cust_num	**company**	**street**	**zip**	**city**	**country**	**cust_tel**	**cust_info**
100	Siemens AG	Otto-Hahn-Ring 6	81739	Munich	D	089/636-8	Electrical
101	Login GmbH	Rosenheimer Str. 34	81667	Munich	D	089/4488870	PC networks
102	JIKO GmbH	Posener Str. 12	30659	Hannover	D	0551/123874	Import/Export
103	Plenzer Trading	Paul-Heyse-Str. 12	80336	Munich	D	089/923764	Fruit trade
104	Freddy's Fishery	Hirschgartenstr. 12	12587	Berlin	D	016/5739921	Unit retail
105	The Poodle Parlor	Am Muehlentor 26	41179	Moenchen-gladbach	D	040/873562	Service
106	Foreign Ltd.	26 West York St.		New York, NY	USA	001703/2386532	Commercial agency
107	Externa & Co KG	Berner Weg 78	3000	Berne 33	CH		Law firm

Table 4: Data in the CUSTOMERS table

CONTACTS table

The CONTACTS table contains information on contacts for a specific customer. It comprises a unique contact number, the number of the customer from the CUSTOMERS table, the first name, last name, title and telephone number of the contact and information on the person's position and department along with the reason for the contact. The following table definition is used to create the CONTACTS table:

`CREATE TABLE`	`contacts`
`(contact_num`	`INTEGER CONSTRAINT contact_num_primary PRIMARY KEY,`
`cust_num`	`INTEGER CONSTRAINT contact_cust_num_notnull NOT NULL,`
`fname`	`CHAR(25),`
`lname`	`CHAR(25) CONSTRAINT name_notnull NOT NULL,`
`title`	`CHAR(20),`
`contact_tel`	`CHAR(25),`
`position`	`CHAR(50),`
`department`	`CHAR(30),`
`contact_info`	`CHAR(50),`
`CONSTRAINT`	`contact_cust_num_ref_customers FOREIGN KEY (cust_num)` `REFERENCES customers`
`)`	

CONTACT_NUM is the primary key for the CONTACTS table. A referential constraint has been defined for the table. The foreign key CUST_NUM refers to the primary key CUSTOMERS.CUST_NUM of the referenced table CUSTOMERS.

The CONTACTS table contains the following data:

contact_num	**cust_num**	**fname**	**lname**	**title**	**contact_tel**	**position**	**department**	**contact_info**
10	100	Walter	Kuehne	Dr.	089/6361896	CEO	Personnel	
11	100	Stefan	Walkers	Mr.	089/63640182	Secretary	Sales	
20	101	Roland	Loetzerich	Mr.	089/4488870	Managing director		Networks

25	102	Ewald	Schmidt	Mr.	0551/123873	Training		
26	103	Beate	Kredler	Ms.	089/923764	Organization		SQL course
30	104	Xaver	Bauer	Mr.	016/6739921	Sales exec.		
35	105	Anke	Buschmann	Ms.	02161/584097	Managing director		
40	106	Mary	Davis	Ms.	001703/2386531	Management	Purchasing	
41	106	Robert	Heinlein	Mr.	001703/2386532	Trainer	Purchasing	

Table 5: Data in the CONTACTS table

ORDSTAT table

The ORDSTAT table allows the order status numbers from the ORDERS table (ORDER_STAT column) to be assigned to the corresponding texts. It contains a unique order status number and the relevant text.

The following table definition is used to create the ORDSTAT table:

```
CREATE TABLE    ordstat
(ord_stat_num   INTEGER CONSTRAINT ord_stat_num_primary PRIMARY
                KEY,
ord_stat_text   CHAR(15) CONSTRAINT ord_stat_text_not_null NOT NULL
)
```

The ORDSTAT table contains the following data:

order_status	ord_stat_text
1	planned
2	contract
3	completed
4	paid
5	archived

Table 6: Data in the ORDSTAT table

ORDERS table

The ORDERS table contains the basic data for an order. It contains a unique order number, references to the customer and the contact person, the order date, the name of the order, the planned and actual completion dates and an order status number. The following table definition is used to create the ORDERS table:

CREATE TABLE	orders
(order_num	INTEGER CONSTRAINT order_num_primary PRIMARY KEY,
cust_num	INTEGER CONSTRAINT o_cust_num_notnull NOT NULL,
contact_num	INTEGER,
order_date	DATE DEFAULT CURRENT_DATE,
order_text	CHAR(30),
actual	DATE,
target	DATE,
order_status	INTEGER DEFAULT 1 CONSTRAINT order_stat_notnull NOT NULL,
CONSTRAINT	o_cust_num_ref_customers FOREIGN KEY (cust_num) REFERENCES customers,
CONSTRAINT	contact_num_ref_contacts FOREIGN KEY (contact_num) REFERENCES contacts,
CONSTRAINT	order_stat_ref_ordstat FOREIGN KEY (order_stat) REFERENCES ordstat(ord_stat_num)
)	

ORDER_NUM is the primary key of the ORDERS table. The foreign key CUST_NUM references the primary key CUSTOMERS.CUST_NUM in the CUSTOMERS table, the foreign key CONTACT_NUM references the primary key CONTACTS.CONTACT_NUM in the CONTACTS table, and the foreign key ORDER_STAT references the primary key ORD_STAT_NUM in the ORDSTAT table.
The DEFAULT clause in the column definition for ORDER_DATE sets the current date as the default value using CURRENT_DATE. The DEFAULT clause for ORDER_STAT sets the default value 1. The ORDERS table contains the following data:

order_num	**cust_num**	**contact_num**	**order_date**	**order_text**	**actual**	**target**	**order_status**
200	102	25	4/15/2009	Staff training	5/2/2009	5/2/2009	5
210	106	40	12/15/2009	Customer management	4/12/2010	4/1/2010	3
211	106	41	12/29/2009	Database draft customers	4/9/2010	4/1/2010	4

250	105	35	1/19/2010	Instruction concerning mail merge		3/3/2010	2
251	105	35	1/19/2010	Customer management		5/2/2010	2
300	101	20	2/16/2010	Network test/comparison			1
305	105	35	4/28/2010	Staff training		5/2/2010	2

Table 7: SQL table ORDERS

SERVICE table

The SERVICE table contains the individual services required by the order. It contains a unique service number, the relevant order number, the date the service was provided, the name of the service, the units in which the service is measured, the number of units, the price per unit, the relevant rate of VAT and the invoice number.

The following table definition is used to create the SERVICE table:

`CREATE TABLE`	`service`
`(service_num`	`INTEGER CONSTRAINT service_num_primary PRIMARY KEY,`
`order_num`	`INTEGER CONSTRAINT s_order_num_notnull NOT NULL,`
`service_date`	`DATE,`
`service_text`	`CHAR(25),`
`service_unit`	`CHAR(10),`
`service_total`	`INTEGER CONSTRAINT service_total_pos CHECK (service_total > 0),`
`service_price`	`NUMERIC (5,0),`
`vat`	`NUMERIC(2,2),`
`inv_num`	`NUMERIC(4,0),`
`CONSTRAINT`	`order_num_ref_orders FOREIGN KEY(order_num) REFERENCES orders`
`)`	

SERVICE_NUM is the primary key of the SERVICE table. The foreign key ORDER_NUM refers to the primary key ORDERS.ORDER_NUM in the ORDERS table.

The SERVICE table contains the following data:

service_num	order_num	service_date	service_text	service_unit	service_total	service_price	vat	inv_num
1	200	4/20/2009	Training documentation	Pages	45	75	0.19	3
2	200	4/22/2009	Training	Day	1	1500	0.19	3
3	200	4/23/2009	Training	Day	1	1500	0.19	3
4	211	1/21/2010	Systems analysis	Day	8	1200	0.00	10
5	211	1/28/2010	Database design	Day	10	1200	0.00	10
6	211	2/16/2010	Copies/ transparencies	Pages	30	50	0.19	10
7	211	3/24/2010	Manual	Fixed price	1	200	0.07	10
10	250	2/23/2010	Travel expenses	Fixed price	2	125	0.00	
11	250	2/23/2010	Training	Day	1	1200	0.19	

Table 8: Data in the SERVICE table

Schema PARTS

The PARTS schema is used for managing parts.
It comprises the tables ITEMS, ITEM_CAT, PURPOSE, WAREHOUSE, COLOR_TAB and TABTAB.

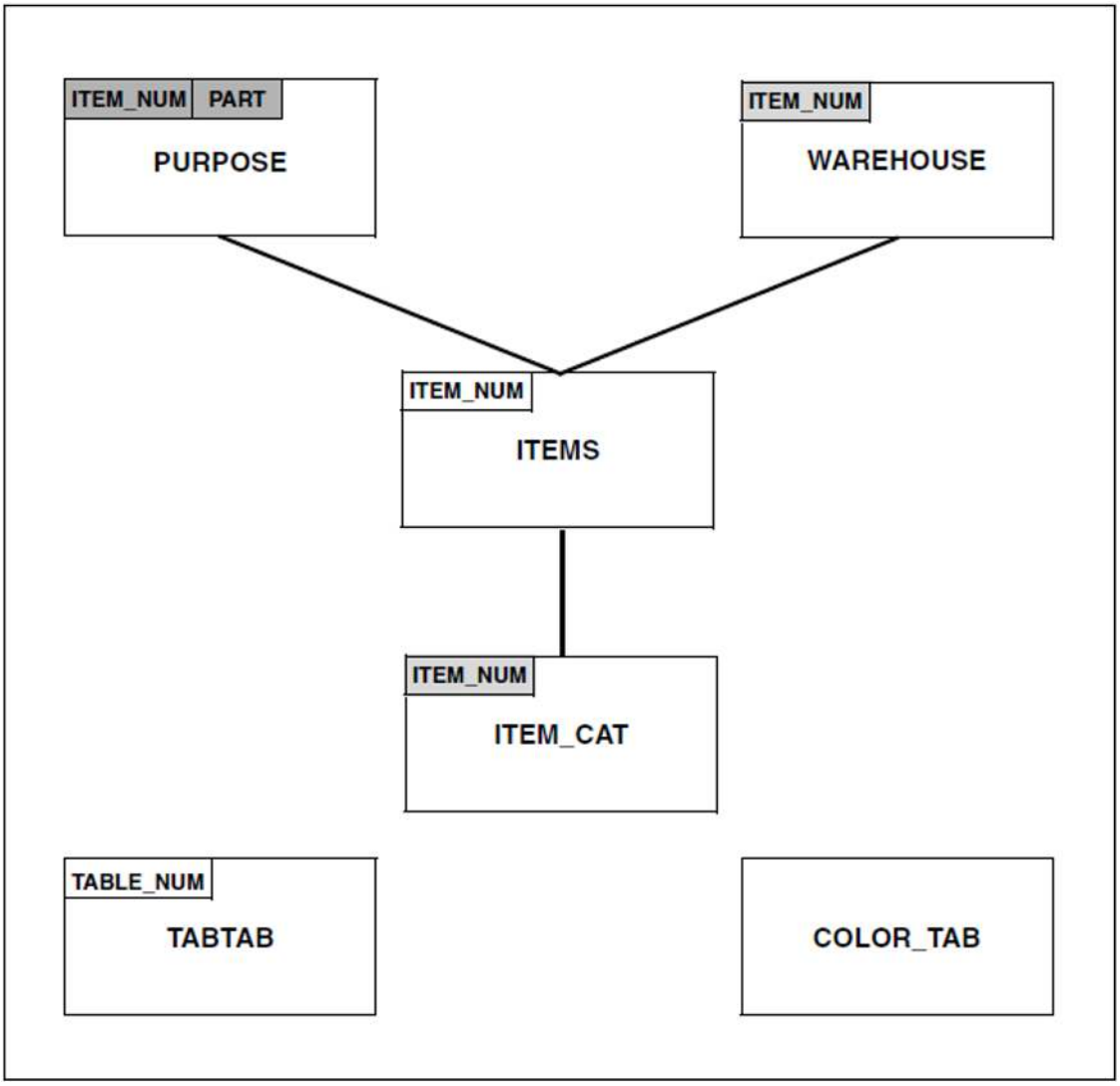

Figure 6: Base tables of the PARTS schema; referencing foreign keys are displayed on a gray background

ITEMS table

The ITEMS table contains information about the stocked articles. It consists of a unique item number, the item name, its color and price, the current and minimum permitted stock levels for the item.

The ITEMS table is defined as follows:

CREATE TABLE	items
(item_num	INTEGER CONSTRAINT item_num_primkey PRIMARY KEY,

item_name	CHARACTER(20) CONSTRAINT item_name_notnull NOT NULL,
color	CHARACTER(15),
price	NUMERIC(8,2) CONSTRAINT price_notnull NOT NULL,
stock	INTEGER CONSTRAINT i_stock_notnull NOT NULL,
min_stock	INTEGER
)	

ITEM_NUM is the primary key of the ITEMS table.

The ITEMS table contains the following data:

item_num	item_name	color	price	stock	min_stock
1	Bicycle	black	700.50	2	1
2	Bicycle	flame	230.00	1	1
10	Frame	black	150.00	10	5
11	Frame	edelweiss	150.00	10	5
120	Front wheel	metallic	40.00	3	5
130	Back wheel	metallic	40.00	12	5
200	Handlebars	metallic	60.00	1	5
210	Front hub	metallic	5.00	15	1
220	Rear hub	metallic	5.00	14	10
230	Rim	black	10.00	9	10
240	Spoke	black	1.00	211	240
500	Screw M5	black	1.10	300	240
501	Nut M5	black	0.75	295	240

Table 9: Data in the ITEMS table

ITEM_CAT table

The ITEM_CAT table contains two REF columns. The REF values in these columns reference BLOB objects in the BLOB tables IMAGES and DESCRIPTIONS in the ADDONS schema. The ITEM_CAT table is defined as follows:

```
CREATE TABLE    item_cat
(item_num       INTEGER CONSTRAINT c_item_num_notnull NOT NULL,
image           FOR REF(images),
desc            FOR REF(descriptions),
)
```

The ITEM_CAT table contains the following data:

item_num	image	desc
2	ADDONS/IMAGES? UID=942acb5a471511db8700a9de6f05052d &OID=1	ADDONS/DESCRIPTIONS? UID=9c3d3d64471511db8600bb19c953d9f8 &OID=1
120	ADDONS/IMAGES? UID=a9f7656a471511db8f019d75c8d27c82 &OID=2	ADDONS/DESCRIPTIONS? UID=b0e1d5cc471511db8300ae9379299332 &OID=2
500	ADDONS/IMAGES? UID=bdfe5e10471511db8600e510dc40a634 &OID=3	ADDONS/DESCRIPTIONS? UID=c279d1ea471511db8a01e1958d1b9863 &OID=3
501	ADDONS/IMAGES? UID=cf7c2fbe471511db8e018591fbfe609b &OID=4	ADDONS/DESCRIPTIONS? UID=d49b3a08471511db8c0096edddbf555c &OID=4

Table 10: Data in the ITEM_CAT table

PURPOSE table

The PURPOSE table specifies the individual components involved in the construction of an item and the number of each of these that is required. Some items are used separately and as a component of another item. The PURPOSE table contains the item number of an article in the ITEMS table, the item number of the part in the ITEMS table and the number of times that this part is used in the item (NUMBER).

The PURPOSE table is defined as follows:

```
CREATE TABLE    purpose
```

(item_num	INTEGER CONSTRAINT p_item_num_notnull NOT NULL,
part	INTEGER CONSTRAINT part_notnull NOT NULL,
number	INTEGER CONSTRAINT number_notnull NOT NULL,
CONSTRAINT	p_item_num_ref_items FOREIGN KEY (item_num) REFERENCES items,
CONSTRAINT	part_ref_items FOREIGN KEY (part) REFERENCES items
)	

The foreign keys ITEM_NUM and PART reference the ITEMS.ITEM_NUM primary key of the ITEMS table.

The PURPOSE table contains the following data:

item_num	part	number
1	10	1
1	120	1
1	130	1
1	200	1
120	210	1
120	230	1
120	240	15
120	500	5
120	501	5
200	500	10
200	501	10

Table 11: Data in the PURPOSE table

WAREHOUSE table

The WAREHOUSE table contains information about the item stocks in the individual warehouses. It consists of an item number in the ITEMS table and the stock of the item at a given warehouse location.

The WAREHOUSE table is defined as follows:

CREATE TABLE	warehouse
(item_num	INTEGER CONSTRAINT w_item_num_notnull NOT NULL,
stock	INTEGER CONSTRAINT w_stock_notnull NOT NULL,
city	CHAR(25),
CONSTRAINT	w_item_num_ref_items FOREIGN KEY (item_num) REFERENCES items
)	

The foreign key ITEM_NUM refers to the primary key ITEMS.ITEM_NUM in the ITEMS table. The WAREHOUSE table contains the following data:

item_num	stock	city
1	2	Main warehouse
2	1	Main warehouse
10	10	Main warehouse
11	10	Main warehouse
120	3	Main warehouse
130	3	Main warehouse
130	9	Parts warehouse
200	1	Main warehouse
210	15	Main warehouse
220	8	Main warehouse
220	6	Parts warehouse
230	6	Main warehouse
230	3	Parts warehouse
240	11	Main warehouse
240	200	Parts warehouse
500	120	Main warehouse

500	180	Parts warehouse
501	248	Main warehouse
501	47	Parts warehouse

Table 12: Data in the WAREHOUSE table

COLOR_TAB

The COLOR_TAB table contains information about how individual colors can be constructed from differing proportions of red, green and blue. It consists of the name of the color and the different proportions of red, green and blue

The COLOR_TAB table is defined as follows:

```
CREATE TABLE    color_tab
(color_name     CHARACTER(15),
rgb             (3)NUMERIC(2,2)
)
```

The COLOR_TAB table contains the following data:

color_name	**rgb**		
flame	0.98	0	0
orange	0.9	0.3	0
skyblue	0	0	0.99
aquamarine	0	0.1	0.99
edelweiss	0.99	0.99	0.99
black	0	0	0
metallic	0	0.2	0.3

Table 13: Data in the COLOR_TAB table

TABTAB table

The TABTAB table contains information about the tables present in the PARTS schema. It consists of a unique table number, the table name and the function of the table.

The TABTAB table is defined as follows:

```
CREATE TABLE    tabtab
(table_num      INTEGER CONSTRAINT table_num_primkey PRIMARY KEY,
```

`table_name`	`CHARACTER(20) CONSTRAINT table_name_notnull NOT NULL,`
`comment`	`CHARACTER(50)`
`)`	

TABLE_NUM is the primary key of the TABTAB table. The TABTAB table contains the following data:

table_num	**table_name**	**comment**
1	items	Parts data
2	purpose	Related bicycle data
3	warehouse	Warehouse locations
4	color_tab	Permissible colors
5	tabtab	Tables used
6	item_cat	Basic data for catalog

Table 14: Data in the TABTAB table

ADDONS schema

The ADDONS schema contains the two BLOB tables IMAGES and DESCRIPTIONS. These tables store two different classes of BLOB object. The table contents cannot be depicted due to the structure of the BLOB tables.

BLOB table IMAGES

The BLOB table IMAGES contains BLOB objects of image class. The BLOB table contains the images referenced by the REF column image in the ITEM_CAT table. The BLOB table is defined as follows:

```
CREATE TABLE    images OF BLOB
(MIME           ('image/gif'),
USAGE           ('images for parts.item_cat.image'),
                'Photographer: Hans Sesamer'
)
```

In the supplied demonstration database, the BLOB table IMAGES contains images in GIF format that can be edited using commercially available graphics programs. For example, it includes the following picture of a bicycle:

BLOB table DESCRIPTIONS

The DESCRIPTIONS table contains BLOB objects of Word document class. The BLOB table contains the documents referenced by the REF column DESC in the ITEM_CAT table. The BLOB table is defined as follows:

```
CREATE TABLE    description OF BLOB
```

(MIME	('application/msword'),
USAGE	('word documents for parts.item_cat.desc'),
	'<AUTHOR>Herta Sesamer</AUTHOR>'
)	

In the supplied demonstration database, the BLOB table DESCRIPTIONS contains Word documents that can be edited with Microsoft Word. For example, the following document is included as a description of the bicycle:

Range of SQL functions in SESAM/SQL

This chapter gives an overview of the most important SQL terms used in the SESAM/SQL manuals. These terms refer to SQL objects created and addressed using the SQL statements or are additional important terms used in the context of SQL. It is assumed that the reader has a general knowledge of relational database systems and of SQL.

The SQL functions are only described in as much detail as is required to understand the remaining chapters of this manual. The SQL statements and language elements are not described in detail. Please refer to the manuals " SQL Reference Manual Part 1: SQL Statements" and " SQL Reference Manual Part 2: Utilities" for a complete and systematic description of the SQL statements in SESAM/SQL.

- In section "SQL objects of a SESAM/SQL database", you will find a description of the SQL objects which can be addressed using SQL statements as well as a number of other important SQL concepts. For each of the concepts, a brief description is provided of those SQL statements which are of importance in that context.
- In section "SQL statements", you will find overviews of the individual classes of SQL statements.
- In section "Fundamental SQL language resources" you will find descriptions of the language constructs SESAM /SQL uses in SQL statements. These are illustrated briefly, primarily on the basis of examples. This section deals with
  - the names of SQL objects
  - the data types recognized by SESAM/SQL
  - how values, expressions and functions are specified in SQL
  - how to select rows from a table using a search condition
  - how to select rows and columns from tables using a query expression
  - the select expression as an elementary query expression
  - the join expression, which allows you to link tables together.
  - the statements SQL provides for manipulating data.
- In section "SQL transaction" you will find a description of the SQL statements and concepts which are important in the context of transactions.
- In section "Embedding of SQL in programs" you will find a description of how SESAM/SQL users can
  - access a SESAM/SQL database from a COBOL program
  - access the rows in a table efficiently using a cursor
  - implement flexible applications using dynamic SQL statements (i.e. statements which are only interpreted at runtime.

Important concepts regarding access protection in SQL are dealt with in section "Access protection based on privileges in SQL".

SQL objects of a SESAM/SQL database

This section describes the SQL objects which can be addressed using SQL statements.

“SQL objects” is a collective term for all the elements of a database that can be created using SQL statements and, with the exception of privileges (see "Access protection based on privileges in SQL"), can also be named using SQL statements.

This section deals with the following SQL objects:

- SESAM/SQL database/(catalog)
- Space
- Storage group
- Schema
- Table
- Column
- Integrity constraints
- Index
- Routines (procedures and User Defined Functions (UDFs))
- BLOB constructs

The figure 7 illustrates the relationships between a number of fundamental SQL objects for the demonstration database ORDERCUST (see "Structure of the SESAM/SQL server documentation"):

The metadata of the ORDECUST **database** contains the ORDERPROC **schema**. The user can access parts of the metadata via the **schema** INFORMATION_SCHEMA. The ORDERPROC schema contains the definition of the CUSTOMERS **table**. The user can specify where the tables containing the user data are to be created (in which **user space** or BS2000 file). The metadata is located in the **catalog space**. The CUSTOMERS table is created in the TABLESPACE and the **index** IND_CUST_INFO is created in the INDEXSPACE. User spaces and the catalog space can in turn be assigned to **storage groups**, which group together the physical storage media on which the space files are created. TABLESPACE and INDEXSPACE are assigned to the storage group STOGROUP1 and the catalog space is assigned to the storage group STOGROUP2.

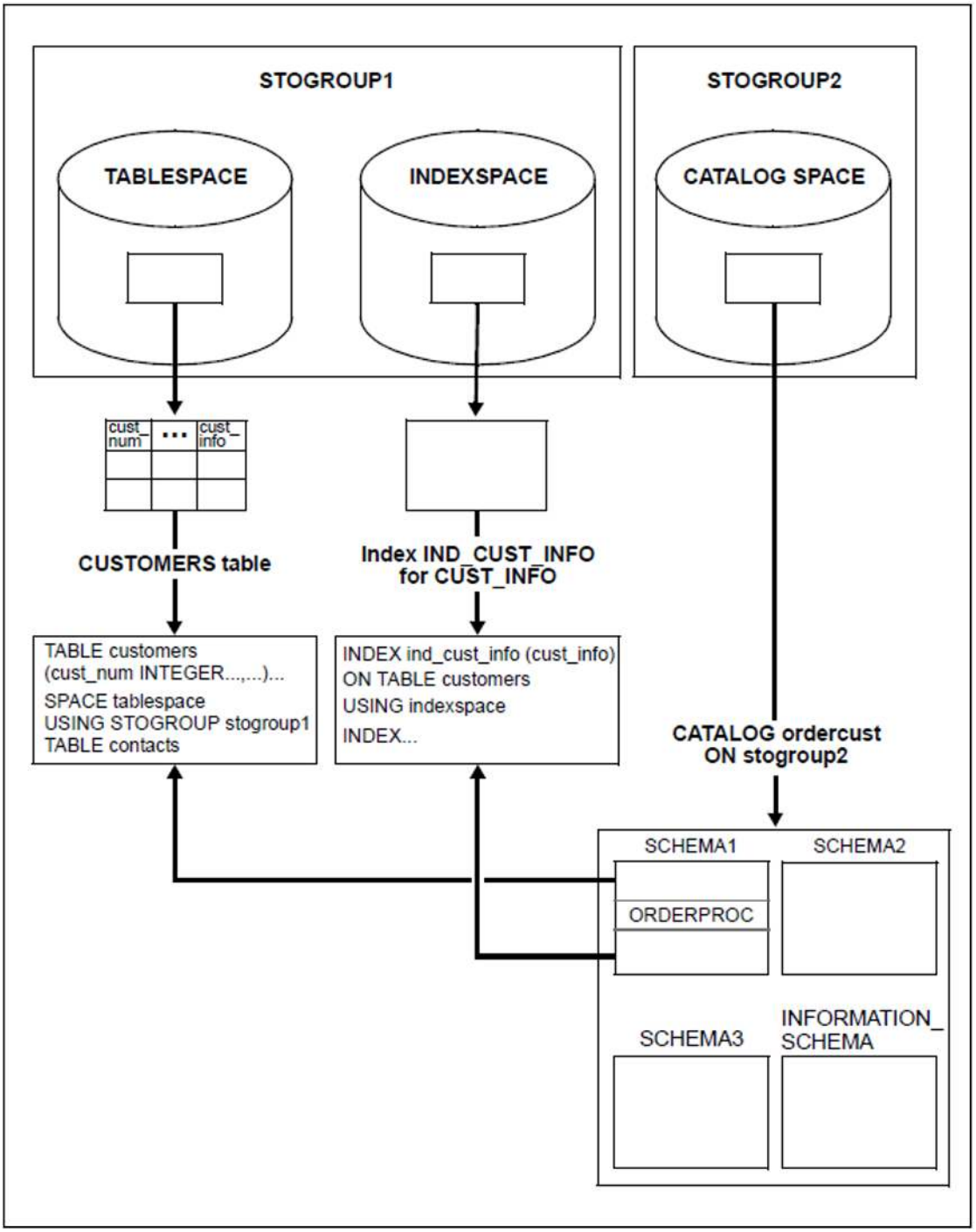

Figure 7: Example of a SESAM/SQL database

SESAM/SQL database

A database (synonym: a catalog) can be regarded as a set of related data that is administered with the aid of a database system.

In SESAM/SQL, a database consists of the *metadata* in the *catalog space* and the *user data* in the associated *user spaces*. A database is identified by the database name.

User data

From the user's point of view, a relational database can be considered to be a set of tables. SQL users are responsible for the structure and content of the tables that they define using SQL statements. As far as the SQL user is concerned, these tables represent user data.

Metadata

In addition to the user data, a database system requires internal data which describes the structure of the user data and which is required for managing the user data. This internal data is also referred to as metadata. The metadata for a database is stored in the catalog space for that database. The user can access parts of the metadata via the schema INFORMATION_SCHEMA.

Space

User data and metadata are located on so-called spaces. Spaces are BS2000 files. They play a key role in backing up and recovering databases (see "Backup concept").

There is a distinction between user spaces, which are used to store the user data, i.e. tables and indexes, and the catalog space, which contains the metadata. In physical terms, a SESAM/SQL database comprises the user data in the corresponding user spaces and the relevant metadata in the catalog space.

On pubsets with “large files” a space can be up to 4 TB in size. Otherwise it can be up to 64 GB in size.

Database (catalog)

You create a database by issuing the utility statement CREATE CATALOG (see "Creating the database's catalog space") to create the catalog space for the database. This defines the database name. The names of all the objects which belong to the database with this catalog space are qualified using the database name.

In order to improve portability of application programs (e.g. when switching from a test environment to a productive environment), SESAM/SQL distinguishes between the logical and physical database names. The physical database name is the actual name of an existing SESAM/SQL database.

The logical database name is the name by which an application program addresses a SESAM/SQL database. If no SESAM/SQL database corresponding to this logical database name exists, it must be assigned to an existing database via the physical database name in the SQL database catalog. The logical database name is always used to qualify SQL objects with the database name.

User space

The SQL statement CREATE SPACE is used to create a user space, see "Creating, modifying and deleting user spaces".

The name of the user space can be specified in the CREATE TABLE and CREATE INDEX statements in order to create the table or index on a specific space. Any given base table or index must be located in its entirety in a single space.

ALTER SPACE allows you to change the properties of the catalog space or a user space and the name of the relevant storage group.

You use DROP SPACE to delete a user space.

Storage group

When building a SESAM/SQL database, users do not need to concern themselves about the volumes on which the user data and metadata for a database are created. There are, however, times when it makes sense for users to influence how the spaces are distributed across the various volumes, for instance in order to place next to each other sets of data which are often accessed together or to place the most frequently accessed data on those disks with the shortest access times. SESAM/SQL users have this flexibility as a result of being able to specify which disks are to be used for storing the spaces of a database.

To achieve this, users can create a “storage group” with the required name using the SQL statement CREATE STOGROUP.

A storage group is used to group together volumes of a particular BS2000 catalog ID under a single name. All volumes in a storage group should be of the same device type. The individual spaces of a single database can be distributed over a number of storage groups.

Once a storage group has been created with the CREATE STOGROUP statement, subsequent CREATE SPACE statements can be used to assign user spaces to this storage group. The user then uses the CREATE TABLE and CREATE INDEX statements to specify the space on which a given table or index is to be created.

The storage group for the catalog space of a database is specified in the utility statement CREATE CATALOG. The name of the catalog space is generated from the database name:

:*catid*:*system-user-id*.*catalog*.CATALOG

If no catalog ID is specified for CREATE STOGROUP, the catalog ID of the default pubset for the DBH session is used.

The ALTER STOGROUP statement allows you to change the definition for a storage group. This allows you to add new volumes or remove individual volumes from the storage group.

DROP STOGROUP deletes a storage group if it is no longer required by a space.

Schema

A database is split into so-called schemas. A distinction is made between user-defined schemas and information schemas.

User-defined schema

Every user-defined schema in a database is assigned to one user, the owner of the schema. A user-defined schema contains metadata which defines the formal structure of the base tables, views, indexes, integrity constraints, privileges and routines, all of which are defined by the owner of the schema.

Every user-defined schema has a name and an owner, identified by a so-called authorization identifier for this schema (see "Specifying an SQL user's authorization identifier"). The schema is created with the SQL statement CREATE SCHEMA and can be modified by other SQL statements for schema definition and administration.

The statements CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, GRANT and CREATE INDEX can be specified as parts of the CREATE SCHEMA statement or as separate statements. One requirement for all SQL statements for schema definition and administration is that the DBH start statement ADD-SQL-DATABASE-CATALOG-LIST (see the “ Database Operation” manual) has been issued with the parameter ACCESS=*PARAMETERS (CAT-ADMINISTRATION=*YES).

A schema can be deleted with the SQL statement DROP SCHEMA.

Information schemas

In addition to user-defined schemas, every database possesses two so-called information schemas with the names INFORMATION_SCHEMA and SYS_INFO_SCHEMA.

The INFORMATION_SCHEMA comprises tables containing part of the metadata for a database. Any user can query this information using an ESQL program or the utility monitor.

The SYS_INFO_SCHEMA contains system-specific data and can only be accessed by the universal user (see "SQL users with universal authorization"). The tables for the information schemas are described in the “ SQL Reference Manual Part 1: SQL Statements”.

Table

The data in a SESAM/SQL database is organized in tables. A table is a two-dimensional arrangement of data comprising rows and columns.

The rows in a table correspond to records. There is no particular sequence to the order of rows in a table. The number of rows in a table is not defined.

Every row in a table has the same number of columns. The name and data type of each column and the sequence of the columns is defined in the table definition. All the values in a given column are of the same data type. Although, according to the relational model, a table is not allowed to include duplicate rows, SQL allows them in principle. The SQL language does, however, does have the means to suppress duplicate rows.

There are the following types of tables:

- base tables
- partitioned tables (base tables which are distributed on a number of user spaces
- views
- derived tables
- abstract tables
- “read-only” tables

Base table

Base tables are tables which contain the user data for a database. A base table is defined with a CREATE TABLE statement and stored permanently in the database until it is deleted with a DROP TABLE statement.

The ALTER TABLE statement allows you to add or delete table constraints to or from an existing base table (see section “Integrity constraint”), to add columns or indexes, or to modify or delete columns.

SESAM/SQL distinguishes the following types of base tables:

- tables which can only be processed with SQL (SQL tables)
- tables which only contain BLOB objects and can only be processed using SQL (BLOB tables)
- tables which can only be processed with CALL DML (CALL DML only tables)
- tables which can be processed with CALL DML and, to a limited extent, with SQL (CALL DML/SQL tables)

SQL tables, BLOB tables and CALL-DML/SQL tables can also be created as partitioned tables, see the section “Partitioned table”.

CALL DML only tables and CALL DML/SQL tables are referred to by the term CALL DML tables.

The table style is of significance in the context of the utility statement MIGRATE. The MIGRATE statement is used to convert databases produced with SESAM/SQL V1.1 or an earlier version to base tables of a SESAM/SQL database for the current version.

Certain points must be observed with CALL DML tables: Only the data types CHARACTER, NUMERIC, DECIMAL, INTEGER and SMALLINT are permitted for CALL DML tables. No default value can be defined for the column with DEFAULT.

The only default value which can be specified is the non-significant value in the CALL DML clause. CALL DML tables must have a primary key constraint (see "Integrity constraint") for a simple or compound primary key. The name of the primary key constraint is used as the name of the compound primary key. Other than this primary key constraint, no integrity constraints are permitted.

Storage structure of base tables

When a table is created using CREATE TABLE, storage space which is known as the contiguous area of the table is reserved. The records which are to be inserted are stored in this area. If a record is to be inserted or extended and there is no longer enough space in this area, relocation takes place, i.e. a free block is created. Logically this block belongs to the table, but it is no longer physically contained in the contiguous area of the table. The next time the user space is reorganized using the REORG SPACE statement all the existing tables and indexes are recovered in the user space. The relocations then also disappear.

Converting a base table to a partitioned table

A non-partitioned base table with primary key can be converted to a partitioned table using the utility statement ALTER PARTITIONING FOR TABLE, see section “Changing the partitioning of a base table” and the “ SQL Reference Manual Part 2: Utilities”.

Partitioned table

A partitioned table is a base table whose data is stored in a number of user spaces. The table data contained on a single space is referred to as a partition. All partitions in a table must be located on different spaces. 2 to 16 partitions are possible per table. In SESAM/SQL the data is distributed to the partitions row by row; the allocation criterion is the primary key of a row.

Partitioned tables have the following advantages over non-partitioned tables:

- Particular criteria, e.g. by month, can be used to structure the user data in a straightforward manner and to store it in different partitions or user spaces. A partition then contains the user data which is currently accessed.
- When suitable structuring/partitioning is used, the size of the user spaces which are currently accessed can be reduced.

  Access to the user spaces which are currently required is possible even when other partitions or user spaces of the table are not available, see the section “Partial availability”.
- The backup and repair times can be reduced since the individual user spaces are smaller.
- The table can be larger than 64 GB.

The table below shows the most important properties of non-partitioned and partitioned tables:

Property	**Non-partitioned table**	**Partitioned table**
Number of user spaces	1	2 to 16 (all spaces must be disjunctive and have been created beforehand)
Primary key	Optional	Mandatory (possible in single or multiple columns)
Modifying the primary key values of an existing row with UPDATE or MERGE	Permitted	Not permitted (row must be deleted and readded with a modified value)
Maximum number of rows	Approx. 4.3 billion	Approx. 268 million per partition, with 16 partitions at most 4.3 billion rows
Smallest backup and repair unit	Complete table	One partition (or the user space belonging to it)
Metadata	INFORMATION_SCHEMA: • BASE_TABLES	INFORMATION_SCHEMA: • BASE_TABLES [1] • PARTITIONS

	SYS_INFO_SCHEMA: • SYS_TABLES	SYS_INFO_SCHEMA: • SYS_TABLES[1] • SYS_PARTITIONS
CALL DML only tables	Possible	Not possible
Password protection with SEPA	Possible	Not possible

Table 15: Properties of non-partitioned and partitioned tables

[1]"_PARTITIONS_" is entered for the space name

Further properties of partitioned tables

- The partitioning of a partitioned table can be altered or canceled with the utility statement ALTER PARTITIONING FOR TABLE, see section "Changing the partitioning of a base table" and the " SQL Reference Manual Part 2: Utilities".

  Partition boundaries can also be altered by exporting the table into an export file, deleting the existing table with DROP TABLE, and then importing the export file as a partitioned table with the same or a different name and different partition boundaries.
- The definition of columns, integrity constraints, indexes, and default values always relates to all partitions.
- Indexes created implicitly or explicitly without the USING SPACE clause are always stored on the user space of the first partition.
- In the case of CHECK FORMAL, error files which are created by SESAM/SQL are created for each space affected. In the case of LOAD and UNLOAD, one error file is created for all partitions.
- Other base tables and indexes or a partition of another table may be stored on a user space on which a partition is stored. However, this procedure is not recommended as the benefit of "small backup and repair units" is reduced.

Partial availability

For many types of access to a suitably partitioned table only the user spaces which are currently required need be available. Other partitions or user spaces in the table do not have to be available. This is referred to as partial availability of partitions.

In addition, a distinction is made between physical and logical availability of a partition.

Physical and logical availability

Physically available means that the space belonging to the partition can be accessed physically and that the space is in the "space o.k." status.

Logically available means that the space belonging to the partition is flagged as physically available within SESAM /SQL. This flag is used to expedite the accesses as only the flag is checked and not the space itself. The flag is created or updated when a partitioned table is accessed for the first time in a DBH session using a DML statement. The same applies for the first access after a database has been entered in the SQL table catalog or after a DBH start or DBH restart.

The physical and logical availability plays a role particularly in DML, DDL, and utility statements:

- DML statements require that the partitions affected should be logically available. Decisive here is the data that is searched, not the number of hits. In other words all partitions which are involved in a search operation must be logically available. When you add a row it is, for example, possible that an adjacent partition must also be searched.
- DDL statements require that the partitions affected should be physically available. With the CREATE INDEX, DROP INDEX, ALTER TABLE and DROP TABLE statements all partitions must be physically available. With the CREATE TABLE statement all spaces on which the partitioned table is to be created must be physically available.
- In most cases utility statements require that the partitions affected should be physically available. The LOAD ONLINE, UNLOAD ONLINE ... WHERE, EXPORT ... WHERE and CHECK CONSTRAINTS statements are an exception. Details on the availability of partitions for utility statements are provided in the “ SQL Reference Manual Part 2: Utilities”.

Changing logical availability

The logical availability of a partitioned table remains unchanged during a DBH session even if the physical availability of partitions changes. Only after RECOVER SPACE is the logical and physical availability of all specified user spaces redefined.

The following administration statements enable you to obtain information on and change the logical availability:

- SHOW-PARTITIONS enables you to obtain information on the logical availability of the individual partitions of a partitioned table.
- CLOSE-SPACE enables you to set all the partitions located on this space to “logically not available”.
- REUSE-PARTITIONS enables you to update the logical availability of the partitions of a partitioned table. Partitions which were until now physically but not logically available are then set to “logically available”.

View

A view is a table which provides a user with a defined view of other tables in the database. In contrast to a base table, a view is not stored permanently in the database. The contents of a view are derived when needed. The data in a view only exists in the underlying base tables.

Thus a view always contains the values which are contained in the database at the point in time when the view is evaluated.

Views provide a number of advantages: flexibility when querying the database, reduced storage space requirements and the option of a graded data protection mechanism:

- Correctly defined views allow data to be grouped in such a way that it exactly meets the information requirements of each user.
- It is not necessary to create new storage space each time different combinations of data are required.
- It is possible to grant nonprivileged users access to the data in the database only via appropriate views on selected data (see "Access protection in connection with views").

The CREATE VIEW statement creates a view; the DROP VIEW statement deletes a view. If the view is queried using this name, it represents a base table as far as the user is concerned.

Updatable views can be used to insert, modify and delete rows in the underlying base tables. A view is updatable if a query expression is specified in the CREATE VIEW statement and the underlying query expression is updatable (see "Query expression").

Derived table

A derived table is a table created as a result of the evaluation of a query expression. Unlike base tables and views, a derived table has no name. The values in the derived table are taken from the values in the underlying tables at the time the query expression is evaluated.

Abstract table

An abstract table is a base table whose individual rows are not permanently stored. SESAM/SQL calculates the values of such rows on the basis of values in other tables when the tables are actually used by SESAM/SQL. Abstract tables always output the current vallues. In SESAM/SQL, abstract tables are used as the basis for views in the Information Schema.

“Read-only” tables

Table functions return “read-only” tables whose content does not depend on SQL data which is stored persistently.

- The table function DEE() returns a table with a row without columns
- In the case of the table function CSV() the values of a table are read from a BS2000 file

Column

When a base table is created or modified (CREATE TABLE, ALTER TABLE), the column definition defines the name and the attributes of a column.

Every column has a name and a data type. Refer to section “Data types” for a list of the data types which can be specified in SESAM/SQL.

SESAM/SQL distinguishes between atomic and multiple columns. In an atomic column, exactly one value can be stored in each row. In a multiple column, several values of the same type can be stored in each row. A multiple column is made up of a number of column elements. In the case of a single column, a single value is stored for each row.

The value of a multiple column in a row is referred to as an aggregate and the value of a single column element in a row is referred to as an occurrence. An aggregate is made up of the occurrences of the individual column elements. It is possible to reference either a column element or a contiguous range of column elements in a multiple column. A single element is specified in the form *column*[*n*] (or *column*(*n*)) and a range is specified in the form *column*[*n..m*] (or *column*(*n..m*)), where $m>n>0$.

In order to integrate BLOB objects in “normal” SQL base tables, the FOR REF is used to define a REF column. This can then contain references that point to the BLOB values of a BLOB table. BLOB objects, tables and REF values are described in detail in section “BLOB constructs”.

Within a column definition, a default value for a single column can be defined. If a single column does not contain a value when a row is inserted, the column is assigned the default value. The default value can be specified as a literal, a date function, one of the special literals CURRENT_USER or SYSTEM_USER or the value NULL. The default value must be compatible with the data type for the column and must comply with any integrity constraint. SESAM/SQL enters the REF value of the class by default in the REF column. This designates the entire class of BLOB values for the BLOB table specified in the FOR REF clause.

The CONSTRAINT clause allows you to define an integrity constraint (see section “Integrity constraint”) for the column.

In the case of CALL DML tables, the CALL DML clause specifies the non-significant value and the column name.

Integrity constraint

An integrity constraint is a rule which restricts the possible range of values for a column or for several columns. In much the same way as only those values are added to the database which are compatible with the data type defined for the corresponding column, SESAM/SQL only permits values which fulfil the defined integrity constraints. An integrity constraint can be seen as a search condition formulated for one column or for several columns and for which the truth value must never be false. A row can only be added to or deleted from a table and a column value can only be changed if all the relevant integrity constraints continue to be fulfilled after the change is made.

An integrity constraint can be defined during definition of the table with CREATE TABLE. The ALTER TABLE statement allows an integrity constraint to be added to or deleted from an existing base table.

An integrity constraint can be specified as a table constraint or a column constraint. A table constraint is an integrity constraint which is declared for one column or a combination of columns. If the integrity constraint references one column only, it can be specified as a column constraint during definition of the column concerned.

An integrity constraint has a name which can be assigned explicitly when the integrity constraint is defined or which is assigned implicitly by SESAM/SQL. If an integrity constraint is not fulfilled, a message which references this name is issued.

When a user defines a new integrity constraint, SESAM/SQL checks whether the constraint is fulfilled by the existing data in the database. If this is not so, the definition of the integrity constraint is rejected. If the table is empty, the integrity constraint is always true.

SESAM/SQL distinguishes the following integrity constraints:

NOT NULL constraint

The NOT NULL constraint requires that a column contain no NULL values. The NOT NULL constraint can only be specified as a column constraint.

UNIQUE constraint

The UNIQUE constraint for a single column requires that any value other than null must occur once only in the specified column. The UNIQUE constraint for a combination of columns requires that any combination of values which does not contain NULL must occur only once in the specified combination of columns.

PRIMARY KEY constraint

The PRIMARY KEY constraint defines a column or set of columns as the primary key of a table. The PRIMARY KEY constraint requires that the column or set of columns satisfy the UNIQUE and NOT NULL constraints.

A table can only have one primary key. The primary key must not be of the data type VARCHAR or NVARCHAR.

It is only possible to define a primary key constraint for CALL DML tables.

Referential constraint

A referential constraint ([FOREIGN KEY]...REFERENCES) defines a column or a combination of columns as a foreign key for a table. The foreign key references one or more columns in another table. A UNIQUE constraint must have been declared for these columns. The table containing the foreign key is known as the **referencing** table and the table for whose columns the UNIQUE constraint must have been declared is known as the **referenced** table. The number and data types of the associated columns must be identical in the referencing and referenced tables. The same base table can be taken for the referencing table and the referenced table.

A row in the referencing table fulfils the referential constraint either if one of the referencing columns contains the null value or if all the values in the referencing columns are non-null and there is a row in the referenced table whose referenced columns contain identical values. If the referential constraint is not fulfilled when a row is inserted in the referencing table or the referencing columns are updated, or when rows in the referenced table are updated or deleted, SESAM/SQL rejects the relevant table operations.

In the case of single-column foreign keys, the referential constraint requires that every value other than NULL for the foreign key of a table occurs as the value of a particular column in a different table where a UNIQUE constraint is fulfilled.

In the case of multiple-column foreign keys, every combination of values which occurs and which does not include a null value must occur in the corresponding columns in the referenced table. This combination of columns must fulfil a UNIQUE constraint. Thus, in SQL, a row fulfils the referential constraint if it contains a null value in at least one column of a multiple-column foreign key.

If no column(s) are specified for the referenced table after REFERENCES, the primary key of the referenced table is used.

Check constraint

The check constraint requires that every column value or every combination of column values fulfils a search condition. The search condition may only reference the table to which the column(s) belong and must not contain a subquery or transliteration between EBCDIC and Unicode. In addition, the search condition may not contain conversion of uppercase letters to lowercase letters or lowercase letters to uppercase letters if the string to be converted is a Unicode string. Since integrity constraints must not be dependent on a particular application, you are not allowed to specify a host variable, a time function or a special literal in the search condition. The search condition may not reference a multiple column.

Deleting integrity constraints

A primary key constraint can only be deleted by deleting the table concerned with the DROP TABLE statement. The other integrity constraints can be deleted with the ALTER TABLE DROP CONSTRAINT statement or are deleted implicitly with DROP TABLE.

A UNIQUE constraint can only be deleted with ALTER TABLE DROP CONSTRAINT RESTRICT if it does not apply to the referenced columns of a referential constraint.

Index

An index for a base table is used to accelerate access to a table. An index is a tree-type access structure assigned to a column or combination of columns in a particular table and which contains cross-references the rows in this table. An index uses a so-called inverted list to assign to every value in a column those rows which contain the value in this column. Combinations of columns are dealt with in the same way.

An index is referred to as simple or compound, depending on whether it refers to one or more columns.

SESAM/SQL uses indexes

- to provide rapid access to rows containing specific values in the index columns
- to return the rows of a table in sorted sequence according to the values in the index columns
- to evaluate integrity constraints for one or more columns of the index without the need to access the base table.

Administration of an index increases the overhead involved in insert and update statements and in recovery operations. The following guidelines should therefore be observed:

- Indexes should not be used for tables which only contain a few rows. The time advantages of an index are lost in the case of small tables as a result of the time taken to open and search the index file.
- No index should not be used for a column which contains only a small number of different values.
- Indexes are more suitable for tables used primarily for retrieving data than for tables which are updated regularly.

An index is created with the SQL statements CREATE INDEX or ALTER TABLE and deleted with DROP INDEX.

Storage structure of indexes

When an index is created using CREATE INDEX, storage space which is known as the contiguous area of the table is reserved. The values which are to be inserted are stored in this area. If a value is to be inserted and there is no longer enough space in this area, relocation takes place, i.e. a free block is created. Logically this block belongs to the index, but it is no longer physically contained in the contiguous area of the index. The next time the user space is reorganized using the REORG SPACE statement all the existing tables and indexes are recovered in the user space. The relocations then also disappear.

Index for a UNIQUE constraint

SESAM/SQL requires an index for every column or combination of columns for which a UNIQUE constraint is defined. If an index has already been created for the relevant column or combination of columns using CREATE INDEX, then this index is also used for the UNIQUE constraint.

If this is not the case, SESAM/SQL generates the index automatically. The name of an index generated in this way begins with UI, followed by a 16-digit number.

REORG STATISTICS statement

REORG STATISTICS rebuilds the global statistics for an index. It makes sense to use REORG STATISTICS after large volumes of changes or insertions have been made. Updated statistics allow more accurate internal estimation of costs and supports the decision regarding the most efficient access plan (see "High performance").

Routine

SESAM/SQL distinguishes between the following routines:

- **Procedures (Stored Procedures)**
- **User Defined Functions (UDFs)**.

> In SESAM/SQL, the generic term **routine** is used for procedures and User Defined Functions (UDFs) if the information applies both for procedures and for UDFs.
>
> The generic term “SQL-invoked routine” from the SQL standard is not used in SESAM/SQL.

A routine is used to store sequences of SQL statements in the database which can be executed later with a single call. A routine is comparable to a subroutine which runs entirely in the DBH, in other words without exchanging data with the application program.

The text of a routine in SESAM/SQL is written entirely in the SQL programming language.

In addition to the usual DML statements, a routine can also contain local definition, control, and diagnostic statements. Definition statements define local data and the cursor and specify special error handling procedures. Control statements control execution of the routine,e.g. by means of loops or conditions. Diagnostic statements provide information on possibly faulty execution of the routine.

The EXECUTE privilege is required to execute a routine.

Information on routines is provided in the information schemas.

Procedures can be defined with input and output parameters. Input parameters are supplied with arguments in the procedure call. Output values are stored by the procedure at predefined locations.
A procedure is generated with CREATE PROCEDURE, executed with the SQL statement CALL, and deleted with DROP PROCEDURE.

UDFs can be defined with input parameters. Input parameters are supplied with arguments in an expression when the function is called. UDFs have precisely one return value (SQL statement RETURN).
A UDF is generated with CREATE FUNCTION, executed by a function call, and deleted with DROP FUNCTION.

BLOB constructs

With SESAM/SQL, you can save multimedia data contents in a database in a permanent and fail-safe way.

Basically, multimedia data contents are subdivided into four main categories:

- Text
- Graphics
- Audio (Sound)
- Video

These large sets of data are known as BLOBs (**B**inary **L**arge **Ob**jects). More precisely,

BLOBs can be variable length sequences of bytes up to a length of 2^{31}-1 bytes. SESAM/SQL stores these BLOBs as BLOB objects. The safety mechanisms established in SESAM/SQL apply to BLOB objects.

When handling BLOB objects it is necessary to distinguish between:

- the administration of BLOB objects which is performed using SESAM/SQL
- the processing of the contents of BLOB objects which is performed outside of SESAM/SQL using programs which are suited to the data content in question (example: editing of images with any image processing program).

The following elements are used in SESAM/SQL to work with BLOB objects. The interaction between these elements is depicted in figure 8:

BLOB objects	In SESAM/SQL, BLOBs are known as BLOB objects because they consist not only of a value but also of the associated attributes.
BLOB tables	These are special base tables that serve as storage locations for BLOB objects. They are structured in a way which permits the storage of BLOB objects. The BLOB objects in a table are known as a class.
BLOB values	The value of a BLOB object is known as a BLOB value.
Attribute	A BLOB object possesses a number of different attributes. These contain information about the creation date and last modification date together with additional specifications that describe the object. The individual attributes are identified by names (tags) to make it possible to distinguish between them.
REF values	Each BLOB object and each class possesses a unique REF value which references the object/class. SESAM/SQL uses this REF value to access the BLOB object or the class.

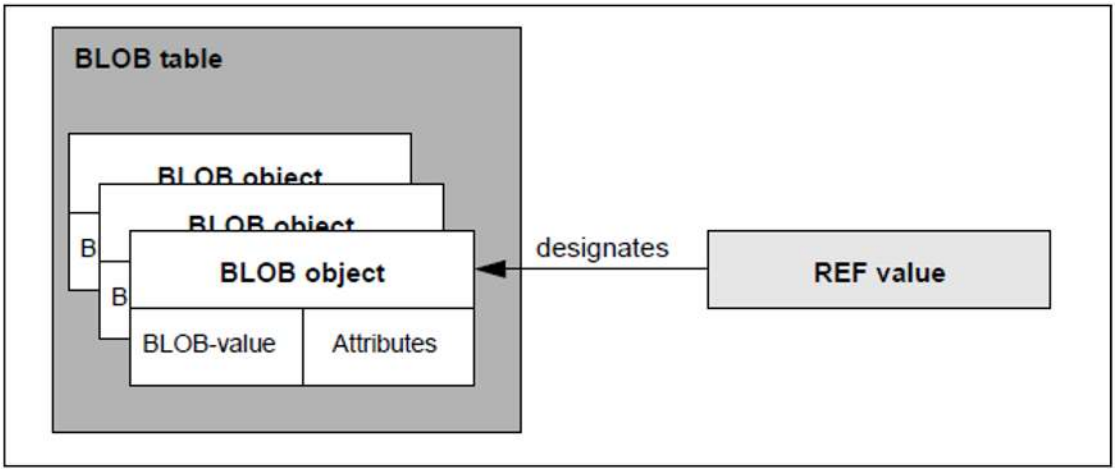

Figure 8: Structure of the BLOB constructs

When you create a BLOB object, SESAM/SQL stores its BLOB value and the assigned attributes in a BLOB table. When this is done, the BLOB value is written slice-by-slice to multiple lines of the BLOB table. This storage method allows for efficient sequential access to BLOB values.

To make it possible to integrate BLOB objects in any required base table, SESAM/SQL generates a unique REF value for each BLOB object. This designates the BLOB object as long as it exists. You can save this REF value in the REF columns of any base tables of your choice (see section “Column”).

The BLOB objects in a BLOB table form the objects of a class. Correspondingly there is also a class REF value which designates all the BLOB objects in a BLOB table. To address BLOB objects, classes and the attributes of BLOB objects as well as BLOB values you must use the SESAM-CLI interface (**C**all **L**evel **I**nterface).

BLOB tables

You create BLOB tables using the statement CREATE TABLE OF BLOB. These tables are used to store BLOB objects. SESAM/SQL basically handles BLOB tables in the same way as other SQL tables. They are displayed as “normal” base tables in the information schemas. There are no differences from other SQL tables in terms of view and index definitions.

The BLOB values of the BLOB object are saved slice-by-slice in multiple table rows when SESAM-CLI is called. To do this, SESAM/SQL saves slices of 31000 bytes at a time. This value was chosen to make optimum use of the bandwidth during the transfer of the BLOB value between the SQL client and SQL server.

SESAM/SQL defines the structure of the BLOB table at creation time. Users cannot therefore perform any column definitions at this point.

A BLOB table consists of the following columns:

- The OBJ_NR column is of data type INTEGER and contains the serial number of the BLOB within the table.
- The column SLICE_NUM is of data type INTEGER and contains the sequential number of the BLOB value slice.
- The SLICE_VAL column is of type VARCHAR(31000). It contains the individual components of the BLOB value. The entries as of slice number 1 contain the value in slices of 31000 bytes in size. Obviously, the last segment may be shorter than this. The row containing slice number 0 is used to store administrative information on the BLOB. The default values for this column are the user-defined attributes in the OF BLOB clause such as MIME and USAGE. In addition, the defaults contain the attributes CREATED and UPDATED. This attributes specify the date on which the BLOB was created and last updated.
- The OBJ_REF column is of type CHAR(237). In the row containing slice number 0, it specifies the REF value of the BLOB. For all other slices, the column value is NULL. The default value for the column is the REF value for the table class. The column is defined with the UNIQUE constraint.

The OBJ_NR and SLICE_NR columns together form the primary key of a BLOB table. As usual, for this primary key constraint, SESAM/SQL assigns internally generated numbers which cannot be re-used within the same schema.

It is possible to append columns using ALTER TABLE. ALTER TABLE also allows you to modify BLOB tables. However, such changes may result in it being impossible for the BLOB table to be accessed by means of CLI calls since the CLI calls are adapted to the structure that is predefined by SESAM/SQL.

BLOB tables can be deleted with DROP TABLE. This call also deletes all the contained BLOB objects.

BLOB objects

A BLOB object possesses a BLOB value and a number of attributes.

BLOB values are stored in “normal tables” via the REF values. Direct storage is not possible. This has the advantage that you can execute all the habitual SQL operations with REF values in base tables. This would not be possible with BLOB values.

The attributes of a BLOB object are initially assigned the default values for the class when a BLOB object is created. The class defaults are defined with *mime_clause*, *usage_clause* and *alphanumeric_literal* when the BLOB table is created.

These attributes are identified by the names (tags) MIME and USAGE. Alongside these optional attributes, the time of creation and last modification are recorded by means of the CREATED and UPDATED tags. You can replace the MIME and USAGE attributes by means of the CLI call SQL_BLOB_TAG_PUT.

You use SESAM-CLI calls to create, read and set BLOB objects and values. However, you do not edit the content of BLOB values in SESAM/SQL but in object-specific programs. To transfer BLOB values from SESAM/SQL to another system, for example to BS2000, you can use one of two methods:

- The BLOB value can be read in a single step with SQL_BLOB_VAL_GET.
- You can use the command sequence SQL_BLOB_VAL_OPEN, SQL_BLOB_VAL_FETCH and SQL_BLOB_VAL_CLOSE to read the BLOB value sliceby-slice (see section “SESAM-CLI”).

Because of their size, you should delete any BLOB objects you no longer require.

REF values

When a new BLOB object is created, SESAM/SQL generates an REF value. You can use this value in the columns of any required table to reference the BLOB object. For this reason, the REF columns are defined in base tables. The syntax used for column definition is described in section “Column”.

Alongside the data type CHAR(237), the REF column also contains the REF value of the class as the default value. Certain SESAM-CLI calls, for example for the creation of a new BLOB object, may require the REF value of the class.

A REF value has the basic structure: *ss*/*tt*?UID=*uuuu*&OID=*nn*

ss is the simple schema name of the BLOB table without a database name.

tt is the simple table name of the BLOB table without a schema or database name.

uuuu is a unique ID number for the BLOB object which consists of 32 hexadecimal digits. In the case of the class REF value, all the digits are 0.

nn is the number of the BLOB object in the BLOB table. In the case of the class REF value, this number is 0.

Two REF values are then identical when they designate the same BLOB object. If two REF values are different then they also reference different BLOB objects. In contrast, their BLOB values may be different. If a BLOB object is deleted then its associated REF value can never again designate a BLOB object.

SESAM-CLI

BLOB objects, their values and attributes as well as the BLOB object classes are addressed by calling the SESAM-CLI interface (**C**all **L**evel **I**nterface). CLI calls can be made from C or COBOL programs.

Below is an overview of the SESAM-CLI calls and their associated functions. The individual calls are described in detail in the “ SQL Reference Manual Part 1: SQL Statements”.

Operations involving BLOB classes

CLI call	Short form	Function
SQL_BLOB_CLS_REF	SQLBCRE	Output REF value of the class
SQL_BLOB_CLS_ISBTAB	SQLBCIS	Check whether BLOB table exists

Table 16: CLI calls for operations involving BLOB classes

Creating and deleting BLOBs

CLI call	Short form	Function
SQL_BLOB_OBJ_CREATE	SQLBOCR	Create object (object number sequential)
SQL_BLOB_OBJ_CREAT2	SQLBOC2	Create a BLOB (object number area-specific)
SQL_BLOB_OBJ_DROP	SQLBODR	Delete a BLOB

Table 17: CLI calls for BLOB objects

Reading and setting BLOB attributes

CLI call	Short form	Function
SQL_BLOB_TAG_GET	SQLBTGE	Read an attribute value
SQL_BLOB_TAG_PUT	SQLBTPU	Set an attribute value

Table 18: CLI calls for BLOB attributes

Reading and setting BLOB values

CLI call	Short form	Function
SQL_BLOB_VAL_GET	SQLBVGE	Output BLOB value
SQL_BLOB_VAL_PUT	SQLBVPU	Set BLOB value
SQL_BLOB_VAL_LEN	SQLBVLE	Output the length of a BLOB value

Table 19: CLI calls for BLOB values

Sequential processing of BLOB values

CLI call	Short form	Function
SQL_BLOB_VAL_OPEN	SQLbvop	Open an access handle
SQL_BLOB_VAL_CLOSE	SQLbvcl	Close an access handle
SQL_BLOB_VAL_FETCH	SQLbvfe	Read a BLOB value sequentially
SQL_BLOB_VAL_STOW	SQLbvst	Set a BLOB value sequentially

Table 20: CLI call for individual sequences of BLOB values

SQL statements

SQL statements can be classified according to various aspects, e.g.:

- whether they start a transaction (see section “SQL transaction”)
- whether they are executable SQL statements (see "ESQL program")
- whether they can be compiled dynamically (see "Dynamic SQL")

SQL statements can also be classified by function. The SQL standard distinguishes the following classes of statements:

- SQL statements for schema definition and administration
- SQL statements for querying and updating data
- SQL statements for designing and managing routines
- SQL statements for transaction management
- SQL statements for session control
- SQL statements for dynamic SQL
- WHENEVER statement for ESQL error handling

SESAM/SQL includes the following additional classes of SQL statements:

- SQL statements for managing the storage structure
- SQL statements for managing user entries
- Utility statements

The SQL statements for schema definition and administration are also known as DDL statements (“Data Definition Language”) and the SQL statements for querying and updating data are known as DML statements (“Data Manipulation Language”).

The following sections list the SQL statements for each of the classes and provide a brief description of each statement. Extensions specific to SESAM/SQL are shown on a gray background. The functions provided by the SQL statements are described in section “SQL objects of a SESAM/SQL database”.

SQL statements for schema definition and administration

SQL statement	Function
`CREATE SCHEMA`	Creates a schema
`DROP SCHEMA`	Deletes an empty schema
`CREATE TABLE`	Creates a base table
`ALTER TABLE`	Alters columns, indexes and integrity constraints in a table
`DROP TABLE`	Deletes a base table and its indexes
`CREATE VIEW`	Creates a view
`DROP VIEW`	Deletes a view definition
`GRANT`	Grants table privileges or special privileges to a user
`REVOKE`	Revokes table privileges or special privileges from a user

Table 21: SQL statements for schema definition and administration

SQL statements for querying and updating data

SQL statement	Function
SELECT[1]	Transfers the values of a row to a host variable
UPDATE	Updates column values in rows determined by a search condition
DELETE	Deletes from a table rows determined by a search condition
INSERT	Inserts rows in a table
MERGE	Inserts new records in a table and modifies existing records
CALL	Executes a procedure
DECLARE CURSOR[2]	Declares a cursor
OPEN[2]	Opens a cursor
FETCH[2]	Positions on a row in the cursor table and reads this row
UPDATE...CURRENT[2]	Updates the current row in the cursor table
DELETE... CURRENT[2]	Deletes the current row in the table
CLOSE[2]	Closes a cursor
STORE[2]	Stores the current cursor position
RESTORE[2]	Restores a cursor stored with STORE

Table 22: SQL statements for querying and updating data

[1]A dynamically compilable form exists for the SELECT statement
[2]These statements are also available for a dynamic cursor

SQL statements for designing and managing routines

SQL statement	Function
`CREATE PROCEDURE`	Creates a procedure
`CREATE FUNCTION`	Creates a User Defined Function (UDF)
`COMPOUND statement, CASE, FOR, IF, ITERATE, LEAVE, LOOP, REPEAT, RETURN, SET, WHILE`	Control statements for a routine (in the CREATE PROCEDURE or CREATE FUNCTION statement)
`GET DIAGNOSTICS; SIGNAL; RESIGNAL`	Diagnostic statements for a routine (in the CREATE PROCEDURE or CREATE FUNCTION statement)
`DROP PROCEDURE`	Deletes a procedure
`DROP Function`	Deletes a User Defined Function (UDF)

Table 23: SQL statements for designing and managing routines

SQL statements for transaction management

SQL statement	Function
`SET TRANSACTION`	Defines the isolation level and transaction mode
`COMMIT WORK`	Commits the changes made during a transaction
`ROLLBACK WORK`	Rolls back the changes made during a transaction

Table 24: SQL statements for transaction management

SQL statements for session control

SQL statement	Function
`SET CATALOG`	Defines the default database name for dynamically compilable SQL statements and cursor descriptions
`SET SCHEMA`	Defines the default schema name for dynamically compilable SQL statements and cursor descriptions
`SET SESSION AUTHORIZATION`	Defines an authorization identifier for the current SQL session
`PERMIT`	Has no effect; provided only for compatibility with SESAM/SQL V1

Table 25: SQL statements for session control

SQL statements for dynamic SQL

SQL statement	Function
`PREPARE`	Checks a dynamically compilable SQL statement or cursor description and prepares it for execution
`EXECUTE`	Executes a statement already prepared with PREPARE
`EXECUTE IMMEDIATE`	Checks a dynamically compilable statement and executes it
`ALLOCATE DESCRIPTOR`	Creates an SQL descriptor area
`GET DESCRIPTOR`	Reads entries from an SQL descriptor area
`SET DESCRIPTOR`	Updates an SQL descriptor area
`DEALLOCATE DESCRIPTOR`	Releases an SQL descriptor area
`DESCRIBE`	Provides information on derived columns or input values in a dynamically compiled statement or cursor description

Table 26: SQL statements for dynamic SQL

WHENEVER statement for ESQL error handling

SQL statement	Function
`WHENEVER`	Defines the reaction to a failed SQL statement

Table 27: WHENEVER statement for ESQL error handling

SQL statements for managing the storage structure

SQL statement	Function
`CREATE STOGROUP`	Creates a new storage group
`ALTER STOGROUP`	Alters a storage group definition
`DROP STOGROUP`	Deletes a storage group
`CREATE SPACE`	Creates a space and enters the space name in the metadata
`ALTER SPACE`	Changes the properties of a space
`DROP SPACE`	Deletes a user space
`CREATE INDEX`	Creates an index for a base table
`DROP INDEX`	Deletes an index
`REORG STATISTICS`	Rebuilds global statistics for an index

Table 28: SQL statements for managing the storage structure

SQL statements for managing user entries

SQL statement	Function
`CREATE USER`	Creates a new authorization identifier
`DROP USER`	Deletes an authorization identifier and the corresponding system entries
`CREATE SYSTEM_USER`	Assigns to users authorization identifiers for a database
`DROP SYSTEM_USER`	Cancels the assignment to system users of authorization identifiers for a database

Table 29: SQL statements for managing user entries

Utility statements

Utility statements are statements which employ SQL syntax and which provide utility functions for database administration (see chapter “Utility concept”). These statements are SESAM/SQL-specific extensions. Unlike SQL statements, utility statements can only be issued outside transactions, i.e. no further transactions may be open when a utility statement is issued. A utility statement is processed internally as a sequence of internal transactions and can therefore not be rolled back.

Fundamental SQL language resources

This section describes basic SESAM/SQL language constructs used in SQL statements.

The following topics are covered:

- Names of SQL objects
- Data types
- Values
- Expressions
- Functions
- CASE expression
- CAST expression
- Search condition
- Predicates
- Query expression
- SELECT expression
- Subquery
- Join expression

Names

Names are strings used to identify SQL objects.

The name of an object is usually defined when the object itself is defined using the appropriate SQL statement. The name has then been introduced and the object can be referenced using this name in any subsequent statements.

SESAM/SQL distinguishes between unqualified names and qualified names.

Unqualified names

Unqualified names are either regular names that are not enclosed in double quotes, or special names, which must be enclosed in double quotes.

Regular names

Regular names are made up of alphabetic characters, numerics and the underscore character (_). The first character of a regular name must be an alphabetic character. Lowercase characters are converted to uppercase during interpretation. Reserved SQL keywords must not be used as regular names.

Special names

Special names, on the other hand, can contain reserved SQL keywords and characters which are not permitted for regular names. A distinction is made between uppercase and lowercase letters. Special names are enclosed in double quotes. The character following the first quote must not be an underscore (_). Quotes within a special name must be entered twice. The two double quotes then count as a single character.

Blanks at the end of special names are ignored. I.e. the following representations are all equivalent:

CUSTOMERS Customers “CUSTOMERS” “CUSTOMERS ”

Qualified names

To allow the unique identification of different objects with the same name within an SQL application, it is possible to qualify the names. Names of objects can be qualified explicitly in SQL by prefixing the name with a database name, schema name, table name or correlation name, followed by a period.

The following qualifications are possible

- Schema, space, storage group, table, index. Integrity constraint and routine with the database name.
- Table, index, integrity constraint and routine with the schema name.
- Column with the table or correlation name.

With the exception of column names, names of SQL objects must generally be qualified, either explicitly or implicitly. If a name is not qualified explicitly using the database or schema name, it is implicitly qualified by the default database or schema name. The default database and schema names are specified in the precompiler option SOURCE- PROPERTIES (see the “ ESQL-COBOL for SESAM/SQL-Server” manual) or in the configuration data for the utility monitor (see the “ Utility Monitor” manual). In the case of dynamically compilable statements, the user can specify the default database or schema name with the SET CATALOG and SET SCHEMA statements respectively (see "Defining default values in dynamic statements").

Example

CUSTOMERS table in the ORDERPROC schema of the ORDERCUST database

```
ordercust.orderproc.customers
```

Data types

The data type specifies the range of permitted values for a column. In SQL, the data type for a column is defined with the column definition (see "Column") in the statements CREATE TABLE or ALTER TABLE.

BLOBs (Binary Large Objects) are based on existing data types in SESAM/SQL and are therefore not a new data type in themselves. The structure and method for processing such objects are described in section “BLOB constructs”.

Data type	SQL syntax	Range of values
Alphanumeric data type: fixed-length alphanumeric string	CHAR[ACTER] [(*length*)] *length:* fixed column length in characters; unsigned integer, 1 through 256	Alphanumeric string of length *length* The CHAR data type is stored with its significant length only, without trailing spaces.
Alphanumeric data type: alphanumeric string of length 0 through *max*	{ CHAR[ACTER] VARYING(*max*) | VARCHAR (*max*) } *max:* maximum length of column in characters; unsigned integer, 1 through 32000	Alphanumeric string shorter than or equal to the length *max*
National data type: fixed-length national string	{ NATIONAL CHAR[ACTER] | NCHAR } [(*cu_length* [CODE_UNITS])] *cu_length:* fixed column length in code units (1 code unit = 2 byte); unsigned integer, 1 through 128	National sring of length *cu_length* The NCHAR data type is stored with its significant length only, without trailing spaces
National data type: national string of length 0 through *cu_max*	{ NATIONAL CHAR[ACTER] VARYING | NCHAR VARYING | NVARCHAR } (*cu_max* [CODE_UNITS]) *cu_max*: maximum length of column in code units (1 code unit = 2 byte); unsigned integer, 1 through 16000	National stringshorter than or equal to the length *cu_max*
Numeric: small integer	SMALLINT	-2^{15} through 2¹⁵through $2^{15}-1$
Numeric: large integer	INT[EGER]	-2^{31} through $2^{31}-1$

Numeric: fixed-point number	NUMERIC [(*precision*[,*scale*])] *precision:* number of decimal digits; unsigned integer, 1 through 31 *scale:* number of digits after the decimal point; unsigned integer, 0 through *precision*	Fixed-point numbers with a value of 0 or in the range 10^{-scale} through $10^{precision-scale}-10^{-scale}$ The NUMERIC and DECIMAL data types are stored with their significant length only, without leading zeros.
Numeric: fixed-point number	DEC[IMAL] [(*precision*[,*scale*])] *precision:* number of decimal digits; unsigned integer, 1 through 31 *scale:* number of digits after the decimal point; unsigned integer, 0 through *precision*	
Numeric: single-precision floating-point number	REAL	Floating-point numbers with a value of 0 or in the range $5.4E^{-79}$ through $7.2E^{+75}$
Numeric: double-precision floating-point number	DOUBLE PRECISION	
Numeric: floating- point number	FLOAT[(*precision*)] *precision:* minimum number of binary digits for the mantissa; unsigned integer, 1 through 53	
Time data type: date	DATE	Date specification from the range 0001-01-01 through 9999-12-31
Time data type: time	TIME(3)	Time specification from the range 00:00:00.000 through 23:59:61.999
Time data type: time stamp with date and time	TIMESTAMP(3)	Date specification as for DATE, time as for TIME(3)

Tabelle 30: Data types in SESAM/SQL

Values

Values can be specified in SQL statements to enter or update column values. They can also be linked by operators in expressions and used in comparisons. In routines they can also be used as parameters and local variables. Values can be entered directly as literals or passed in host variables (see "Host variables"). In dynamically compilable statements (see "Dynamic SQL"), the character “?” can be used as a placeholder for a value. Every value has a data type.

SESAM/SQL makes a distinction between NULL values and non-NULL values. Depending on the data type, non-null values are classified as strings (alphanumeric and national values), numeric values and date and time values.

Each of these classes has corresponding formats for specifying literals.

REF values that occur in connection with BLOBs (Binary Large Objects) are not values in the conventional sense. They are used to reference so-called BLOB objects in base tables. Information on defining REF values in base tables can be found in the section “Column”. The structure and method of processing BLOB objects are described in section “BLOB constructs”.

NULL values

NULL values are provided to distinguish missing values from values. The term “null value” does not refer to any specified value, it conveys the meaning “value unknown” or “irrelevant”. In particular, it must not be confused with a blank or a zero.

The keyword NULL can be seen as a literal for the null value. It can be specified for instance in INSERT, MERGE or UPDATE statements in order to enter a null value in a column. A null value can also be defined as a default value within a column definition using the DEFAULT NULL clause.

If NOT NULL or PRIMARY KEY is specified for a column in the column definition, the column can contain non-null values only. If no NOT NULL or PRIMARY KEY constraint and no default value (see "Column") other than NULL has been defined, SESAM/SQL enters a null value automatically if no value is specified for a column when a row is inserted.

Depending on the type of link (see "Search condition"), a null value in a condition will generally return the truth value unknown. One exception in this context is the predicate *column* IS [NOT] NULL, which returns only the truth values true or false.

Values for multiple columns

A value for a range of column elements in a multiple column is specified by a *value*, which can be a host variable for a vector, a placeholder in the form “?” or an aggregate. An aggregate is specified in the form

<*value1*, *value2*, ...>

value1, *value2*, ... are here the values of the occurrences of a multiple column.

Strings

Strings are sequences of any characters in EBCDIC or Unicode. EBCDIC strings are termed “alphanumeric values”, Unicode strings are termed “national values”.

In SESAM/SQL, alphanumeric literals, national literals and special literals are used to represent strings.

Alphanumeric values

Alphanumeric literals are specified in the form '[*character*...]' or X'[*hex hex*...]' respectively and can contain any EBCDIC character. The two single quotes then count as a single character. The data type is CHAR (*n*), where *n* is the number of characters.

In addition to alphanumeric literals, it is possible to specify alphanumeric values with special literals. For example, SYSTEM_USER returns the name of the current system user.

The operator || is available for alphanumeric values. This operator concatenates two strings to form a single string. When strings are concatenated, either both operands must be alphanumeric (CHAR or VARCHAR) or both must be of the national type (NCHAR or NVARCHAR).

An alphanumeric literal can comprise substrings, each contained in a separate line:

'*substring_1*'
'*substring_2*'
...
'*substring_n*'

This corresponds to the alphanumeric literal

'*substring_1*'||'*substring_2*'|| ... ||'*substring_n*'

Comments and blanks are permitted between the substrings.

National values

National literals are specified in the form N'*[*character**...**]' *or* NX'[*4hex*...]*'* or *U&*'[character ...esc+4hex... character...].

N literals can contain Unicode characters which are also included in the coded character set EDF03IRV. A single quote in a national literal must be duplicated; the duplicated single quote is regarded as one character. The data type is NCHAR (*cu_length*), where *cu_length* is the number of code units (1 code unit in UTF-16 = 2 bytes).

NX literals only contain Unicode characters in hexadecimal representation.

U& literals contain both Unicode characters from the coded character set EDF03IRV and Unicode characters in hexadecimal representation which are identified by a preceding escape character, U&'[*esc 4hex*...]' or U&'[*esc+6hex* ...]'. However, each character must be invalidated.

For national values there is the operator ||, which concatenates two strings to form one string (concatenation). When strings are concatenated, either both operands must be alphanumeric (CHAR or VARCHAR) or both must be of the national type (NCHAR or NVARCHAR).

A national literal can comprise substrings, each of which is contained in a row:

N'*substring_1*'
'*substring_2*'
...
'*substring_n*'

This corresponds to the national literal

N'*substring_1*' | |N'*substring_2*'|| ... ||N'*substring_n*'

Special literals

Special literals are converted to the corresponding values of their runtime environment at runtime:

Special literal	Meaning
CURRENT_CATALOG	Name of the preset database
CURRENT_ISOLATION_LEVEL	Isolation level of the current transaction
CURRENT_REFERENCED_CATALOG	Name of the database which the current statement references
CURRENT_SCHEMA	Name of the preset schema
[CURRENT_]USER	Name of the current authorization identifier
SYSTEM_USER	Name of the current system user

Table 31: Special literals

Numeric values

Numeric values are integers, fixed-point numbers and floating-point numbers.

Time values

SESAM/SQL distinguishes three date and time values:

- date (`DATE'`*year-month-day*`'`)
- time (`TIME'`*hour*`:`*minute*`:`*second*`.`*fraction_of_second*`'`)
- time stamp (`TIMESTAMP'`*year*`-`*month*`-`*day hour*`:`*minute*`:`*second*`.`*fraction_of_second*`'`).

The time functions (see "Time functions") CURRENT_DATE, CURRENT_TIME(3) and CURRENT_TIMESTAMP(3) and LOCALTIMESTAMP(3) can also be specified as date and time values. These return the current date, and/or the current time.

You can specify a number between 00 and 61 for *second*. The numbers after the decimal point specify the fractions of a second.

Date and time values can be entered in columns which have the appropriate data type. They are used

- in the time functions CURRENT_DATE, CURRENT_TIME(3), LOCALTIME(3), CURRENT_TIMESTAMP(3) and LOCALTIMESTAMP(3)
- in the aggregate functions COUNT, MAX and MIN
- in the numeric functions EXTRACT and JULIAN_DAY_OF_DATE
- in comparisons with another date and time value of the same data type.

Expressions

The evaluation of an expression returns a valuei.e. a numeric value, an alphanumeric value, a national value or a date and time value.

The data type of this value determines whether the expression is an alphanumeric expression, a national expression, a numeric expression or a date and time expression.

Expressions can occur in:

- Column selection (SELECT expression, SELECT expression)
- predicates in search conditions (e.g. WHERE clause, HAVING clause, IF clause)
- assignments (INSERT, MERGE, UPDATE or SET statement)

An expression consists of the following operands and can include operators. The operators are used on the results of the operands.

- values (see "Values")
  - as strings (alphanumeric and national literals)
  - as special literals
  - as host variables
  - as parameters
  - as local variables
  - as placeholders in SQL statements for dynamic compilation
- columns (see "Column")
- functions (see "Functions")
  - Time functions
  - Numeric functions
  - String functions
  - Aggregate functions
  - Table functions
  - Cryptographic functions
  - User Defined Functions (UDFs)
- CASE expression (see "CASE expression")
- CAST expression (see "CAST expression")
- subqueries (see "Subquery")

Functions

A function consists of its function name and parameters. It is used to calculate a value or returns a table (table function). Functions can be called from within expressions. When an expression is evaluated, any function which occurs in the expression is evaluated and replaced by the calculated value or the table which is returned.

SESAM/SQL functions fall into two groups:

- Time functions
- String functions
- Numeric functions
- Aggregate functions
- Table functions
- Cryptographic functions
- User Defined Functions (UDFs)

Time functions

Time functions determine

- current date (CURRENT_DATE)
- current time (CURRENT_TIME(3) or LOCALTIME(3))
- time stamp with the current date and current time (CURRENT_TIMESTAMP(3) or LOCALTIMESTAMP(3))
- the Julian Day number corresponding to an integer value (DATE_OF_JULIAN_DAY) (see also the inverse function JULIAN_DAY_OF_DATE on "Numeric functions").

LOCALTIMESTAMP(3) and CURRENT_TIMESTAMP(3) are equivalent in SESAM/SQL, as are LOCALTIME(3) and CURRENT_TIME(3). In the SQL standard, CURRENT_TIMESTAMP and CURRENT_TIME are provided for a language extension making it possible to calculate on the basis of different time zones.

String functions

String functions

- extract substrings (SUBSTRING)
- transliterate alphanumeric strings to national strings or vice versa (TRANSLATE)
- transcode national strings from UTFE to UTF-16 or vice versa (TRANSLATE)
- remove leading or trailing characters of strings (TRIM)
- convert uppercase letters to lowercase letters or lowercase letters to uppercase letters (LOWER, UPPER)
- convert a value of any data type to the internal presentation (as an alphanumeric string or in hexadecimal format) and vice versa (HEX_OF_VALUE, VALUE_OF_HEX, REP_OF_VALUE, VALUE_OF_REP)
- for national strings, supply the collation element in accordance with the Default Unicode Translation Table (COLLATE)
- convert national strings to normal form (NORMALIZE)

Numeric functions

Numeric functions achieve various purposes:

- ABS(), CEILING(), FLOOR(), MOD(), SIGN() and TRUNC() execute the corresponding mathematical functions on the specified numeric expressions.
- CHARACTER_LENGTH( ), OCTET_LENGTH( ) and POSITION( ) calculate the number of bytes or code units in a string or the position of a string in another string.
- JULIAN_DAY_OF_DATE( ) converts a date into an integer value.
- EXTRACT() extracts specific components of a time value.

When a numeric function is evaluated, a numeric value is returned.

Aggregate functions

Aggregate functions return the average (AVG), count (COUNT), maximum value (MAX), minimum value (MIN) or sum (SUM) of a set of values or the number of rows in a derived table.

> The aggregate functions MIN() and MAX() reference the set of all values in a column in a table. They differ in this way from a CASE expression with MIN / MAX (see "CASE expression"), which references different expressions.

Table functions

Table functions generate tables whose content depends on the call parameters or is derived from external data sources, e.g. files.

The following table functions are provided in SESAM/SQL:

- CSV()
  returns a table whose values are read from a BS2000 file (the so-called CSV file)
- DEE()
  returns a table with a row without columns

CSV format (CSV: Comma Separated Values) is used to display SQL tables in files here. This is a standardized format for the platform-independent exchange of table data. The file contains the sequence of table rows, each row containing its column values sequentially as a string. Such files can be generated with a large number of software products (e.g. with Microsoft EXCEL). Further details on CSV format and on interpreting CSV files is provided in the “ SQL Reference Manual Part 1: SQL Statements”.

Cryptographic functions

The ENCRYPT() and DECRYPT() functions are used to encrypt and decrypt individual values. Sensitive data is protected against unauthorized access by encryption. Only the users who know the “key” can decrypt the data.

Introductory information on data access control by means of data encryption in SESAM/SQL is provided in section “Data access control by means of data enryption”.

User Defined Functions (UDFs)

A UDF is used to store sequences of SQL statements in the database which can be executed later with a single function call. UDFs have an almost identical function scope to procedures, see section “Routine”.

CASE expression

A CASE expression is a conditional expression, i.e. an expression containing conditions. An expression or NULL is assigned to each condition. When the CASE expression is evaluated, the assigned expression value or NULL value is returned to whichever condition is true.

There are different types of CASE expression:

- CASE expression with search condition
- Simple CASE expression
- CASE expression with NULLIF
- CASE expression with COALESCE
- CASE expression with MIN or MAX

> **i** A CASE expression with MIN / MAX references different expressions. In this way it differs from the aggregate functions MIN() and MAX() (see "Aggregate functions") which reference the set of all values in a column in a table.

CASE expressions allow you, for instance, to recode column values or replace specific column values by NULL.

CAST expression

The CAST expression allows you to change the value of a specific data type to a value of a different data type.

Search condition

A search condition is a condition used to select rows. Only those rows are selected which return the truth value true for the specified search condition. Search conditions can occur in a number of SQL statements which allow selection of rows in WHERE, HAVING or ON clauses. They can also occur in a check constraint (see "Integrity constraint") or in a case expression (see "CASE expression").

A search condition comprises predicates (logical expressions) which can be linked by the logical operators AND, OR or NOT.

A search condition is evaluated by first evaluating the predicates and then applying the logical operators to the results. The usual precedence rules apply for evaluation: NOT takes precedence over AND, which takes precedence over OR. Parentheses can be used to influence the evaluation sequence.

A search condition returns the truth value true, false or unknown.

AND Operand2 Operand1	**true**	**false**	**unknown**
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

Table 32: Truth values for AND operations

OR Operand2 Operand1	**true**	**false**	**unknown**
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

Table 33: Truth values for OR operations

NOT Operand1	
true	false
false	true
unknown	unknown

Table 34: Truth values with NOT

Predicates

Predicates are components of search conditions. A predicate consists of operands and operators.

Predicates can be grouped together as follows according to the operator involved:

- Comparison of two rows
  Two rows are compared lexicographically according to a comparison operator. If both rows only have one column, you will obtain the normal comparison of two values. The following comparison operators are allowed:

=	equal to
<	smaller than
<=	smaller than or equal to
>	greater than
>=	greater than or equal to
<>	not

- Quantified comparison (comparison with the rows of a table)
  The value of a row is compared with the rows of a table. It is determined whether the comparison holds true either for all the rows of the table, or else for at least one row (SOME/ANY).
- BETWEEN predicate (range query)
  It is determined whether the row lies within a range (BETWEEN) or outside a range (NOT BETWEEN), specified its lower and upper limits.
- CASTABLE predicate (convertibility check)
  This checks whether an expression can be converted to a particular data type. The CASTABLE predicate enables you to check whether a corresponding CAST expression (see section “CAST expression”) can be executed before it is executed and to react appropriately.
- IN predicate (elementary query)
  This determines whether a row occurs in a table (IN) or does not occur in a table (NOT IN).
- LIKE predicate (simple pattern comparison)
  A LIKE predicate determines whether an alphanumeric or a national value matches a specified pattern.
- LIKE_REGEX predicate (pattern comparison with regular expressions)
  A check is made to see whether an alphanumeric value matches a specified regular expression. Regular expressions are precisely defined search patterns which go far beyond the options of the search patterns in the LIKE predicate. Regular expressions are a powerful means of searching large data sets for complex search conditions. They have long been used, for example, in the Perl programming language.
- NULL predicate (comparison with the NULL value)
  A check is made to see whether a column, a parameter or a local variable contains the NULL value.
- EXISTS predicate (existence query)
  This checks whether the specified derived table contains values.

The data types of the operands must be comparable.
All numeric values can be compared with other numeric values, all alphanumeric strings can be compared with other alphanumeric strings and all national strings can be compared with other national strings. Date, time and datetime can only be compared with date, time and datetime respectively.

Query expression

Query expressions are the SQL language constructs for selecting rows and columns from base tables and views. A query expression results in a derived table containing the selected rows.

Query expressions are used, for example, to define a view or a cursor or to select a set of rows to be inserted into a table with the INSERT statement.

The UNION operator links two query expressions in such a way that the derived table contains all those rows which occur in at least one of the derived tables for the query expressions linked by the operator.

The EXCEPT operation in the query expression is comparable to the difference between two sets in set theory.

Updatable query expression

The concept of an updatable query expression is of significance for defining an updatable view or an updatable cursor. A query expression is updatable if it fulfills the following criteria:

- The query expression does not contain a join expression.
  Join expressions are described in the section “Join expression”.
- The query expression must not contain any UNION operation or EXCEPT operation.
- Only column names can be specified in the SELECT list. Other elements of an expression, e.g. subqueries, function calls or literals, are not permitted. Atomic columns cannot be specified more than once. Subareas from multiple columns cannot overlap.
- Only a table or updatable subquery can be specified in the FROM clause. If a table is specified, it must be a base table or an updatable view.
- No subquery can occur in the WHERE clause.
- The keyword DISTINCT cannot be specified.
- The SELECT expression cannot include a GROUP BY or HAVING clause.

SELECT expression

The select expression is a query expression introduced by the keyword SELECT. A select expression produces a derived table.

The select expression is made up of a number of individual clauses, each of which relates to a specific operation. These clauses jointly produce the final derived table.

- select list: this specifies the columns of the derived table.
- FROM clause: this specifies the tables; the derived table is the Cartesian product of these tables.

  If two or more tables are specified in the FROM clause, the result is the Cartesian product of these tables. The Cartesian product of two tables is produced by concatenating all the rows of the first table with all the rows of the second table. The number of rows of the Cartesian product thus corresponds to the product of the number of rows in the underlying tables.
- WHERE clause: this specifies a search condition. The derived table comprises those rows of the derived table produced by the FROM clause which match the search condition.

  The **join condition** is a special form of a search condition in the WHERE clause. This allows data from two or more tables to be linked. A join condition across two tables is formulated by linking one column in one of the tables to a corresponding column in the other table with a comparison operator. Only those rows are selected from the Cartesian product of the two tables which fulfil this join condition. The link between the two tables (and sometimes the resultant derived table) is referred to as a join).

  In addition to the option of linking tables by a join condition in the WHERE clause, there is the more detailed option of formulating joins on the basis of join expressions. Join expressions are described on "Join expression".
- GROUP BY clause: this specifies grouping attributes. The derived table is the same as the derived table from the WHERE clause, except that the rows are grouped according to the values in the grouping attributes.
  If the GROUP BY clause is not specified, the table is not arranged in groups.
- HAVING clause: this allows you to specify a search condition. The derived table comprises those groups which match the search condition.

The correct sequence of clauses must be observed in the select expression. The derived table is built step by step in the same sequence specified in the select expression, with the exception of the select list. This is only evaluated after the FROM, WHERE, GROUP BY and HAVING clauses have been evaluated.

Subquery

A subquery is a query expression that can be used in

- expressions:
  The subquery must return a single-column derived table with a maximum of one row. The value of the subquery is then the value in the derived table or the NULL value if the derived table is empty.
- predicates:
  In the predicates ANY, SOME, ALL, IN and EXISTS the subquery returns a derived table.
- In the FROM clause of SELECT expressions:
  The subquery returns a derived table.
- In join expressions:
  The subquery returns a derived table.

A subquery is always enclosed in parentheses.

Correlated subquery

In a nested query expression, the inner subquery is referred to as a correlated subquery if it references columns in a table which is used in an outer query expression.

Correlated subqueries are evaluated by evaluating the inner subquery for all the rows in the table of the outer query. Non-correlated subqueries are only evaluated once as they do not depend on the outer query.

Join expression

The description of the WHERE clause on "Query expression" introduced the simplest form of the join, which allows tables to be linked using a join condition. The join expression provides greater flexibility in using joins.

A join expression consists of the tables to be joined, the desired join operation and possibly a join condition.

A join expression can be specified

- as a query expression in an SQL statement
- in the FROM clause of a SELECT expression or SELECT statement
- in a subquery in the SELECT list and HAVING clause

The derived table of a join expression cannot be updated.

If two tables are linked with a join, this is referred to as simple join. If more than two tables are linked, it is referred to as a compound join. To form a compound join, join expressions are nested within other join expressions. The use of parentheses can influence the grouping of the operators in nested joins of this type.

The keywords CROSS, INNER, OUTER and UNION define the type of join.

CROSS JOIN

A CROSS JOIN between two tables results in the Cartesian product (cross product) of the tables. In the derived table of a CROSS JOIN, every row of the first table is concatenated with every row of the second table.

INNER JOIN

INNER defines a so-called inner join. In the case of an inner join, only those rows of the Cartesian product whose join columns fulfil the join condition are included in the derived table.

OUTER JOIN

OUTER defines a so-called outer join. The source table for an outer join is the same as for an inner join, namely the Cartesian product. In a one-way outer join, a distinction is made between a dominant table and a dependent table. Unlike the inner join, the outer join also contains a row if there is no row in the dependent table which fulfils the join condition for a given row in the dominant table. A row containing null values is used for the missing row in the dependent table.
The keyword LEFT OUTER defines the table to the left of the LEFT OUTER operator as the dominant table and the keyword RIGHT OUTER defines the table to the right of the RIGHT OUTER operator as the dominant table. In the case of a FULL OUTER join the tables to the right and left of the FULL operator are each defined once as the dominant and as the dependent table. The derived table of the FULL OUTER join contains all the rows of the LEFT OUTER and of the RIGHT OUTER join. Any row which occurs in the derived table of the LEFT OUTER and RIGHT OUTER joins is only included once in the derived table of the FULL OUTER. Duplicates in the LEFT OUTER table and in the RIGHT OUTER table are included as duplicates in the derived table of the FULL OUTER join.
Inner and outer joins can be combined in a join expression.

UNION JOIN

Unlike the inner and outer joins, the Cartesian product is not the source table for UNION JOIN. Instead, the tables to the left and right of the UNION operator are used as source tables. The UNION JOIN is formed as follows:

- The first table of the UNION JOIN is expanded to the right to include the columns of the second table and NULL is entered for the column values of the second table.
- The second table is expanded to the left to include the columns from the first table and NULL is entered for the column values of the first table.
- UNION is employed to join the first table (expanded to include NULL values) to the second table (expanded to include NULL values).

Manipulating data under SQL without a cursor

SQL statements for querying and updating data can be classified as followed:

- SQL statements which access rows via a cursor The function of a cursor and the SQL statements which address a cursor are described in section “Data manipulation under SQL using cursors”.
- SQL statements which access rows without using a cursor The following SQL statements do not use a cursor:

SELECT	SELECT is used to select values from a row and pass them to host variables (see "Host variables")
INSERT	Insert rows in a table
MERGE	Combine the functions INSERT and UPDATE in one operation. Depending on the result of the search condition, MERGE changes column values of records which already exist or adds new records to an existing table.
UPDATE	Update column values in selected rows of a table
DELETE	Delete rows from a table

The following conditions must be fulfilled in order to make changes to a table with the statements mentioned above:

- the user must have the appropriate privileges (see "Access protection based on privileges in SQL"); the SELECT privilege is also required when changes are made using a query expression or a search condition.
- the transaction mode (see "SQL transaction") for the current transaction must be READ WRITE
- the DBH start statement ADD-SQL-DATABASE-CATALOG-LIST (see the “ Database Operation” manual) must have set ACCESS=*PARAMETERS (WRITE=*YES) or ACCESS=*PARAMETERS (CAT-ADMINISTRATION=*YES).

SQL transaction

An SQL transaction is a sequence of related SQL statements which move a database from one consistent status to an new consistent status. Changes to tables are made either entirely or not at all at the end of an SQL transaction.

An SQL transaction starts when no other transaction is open and an SQL statement is issued which opens a transaction. SQL statements which open transactions include all SQL statements other than

- ALTER TABLE with the pragma UTILITY MODE ON
- DECLARE CURSOR (not executable)
- PERMIT
- SET CATALOG
- SET SCHEMA
- SET SESSION AUTHORIZATION
- SET TRANSACTION
- WHENEVER (not executable)
- Utility statements

The statements EXECUTE and EXECUTE IMMEDIATE open a transaction if the relevant dynamically executable statement opens a transaction.

In SESAM/SQL, SQL statements for querying and updating data (see "SQL statements for querying and updating data") may not be executed in a transaction in which an SQL statement for schema definition and administration, for storage structure management or for the administration of user entries is executed. The statements SET SESSION AUTHORIZATION and SET TRANSACTION and utility statements can only be executed outside a transaction. A utility statement is processed internally as a sequence of internal transactions and can therefore not be rolled back.

An SQL transaction is terminated either when the SQL statement COMMIT [WORK] or ROLLBACK [WORK] is executed or if it is rolled back internally by the DBH.

Changes made to the database since the beginning of a particular transaction are only committed after the COMMIT statement has been executed successfully (*exception*: autonomous transactions, see "SQL transaction").

If a transaction is terminated with ROLLBACK, all the changes made to the database since the start of the transaction are canceled.

If an irrecoverable error, a longlock or deadlock occurs during a transaction, SESAM/SQL executes an implicit ROLLBACK WORK.

If openUTM is used, the termination of a transaction is carried out using UTM language resources only. The SQL statements COMMIT and ROLLBACK must not be used in UTM applications. A UTM transaction ends when the next synchronization point is set.

An SQL transaction is characterized by a specific isolation level (see "SQL transaction") and a specific transaction mode (see "SQL transaction"). The isolation level and transaction mode can be set with the SQL statement SET TRANSACTION. The settings made with SET TRANSACTION are only valid for the SQL statements in the immediately following transaction. It is irrelevant whether it is an SQL or CALL DML transaction or a mixed transaction. Once the transaction has been terminated, the default values are restored.

An autonomous transaction (see "SQL transaction") enables you to write to a database irrespective of the result of the surrounding transaction.

Isolation level

The isolation level specifies the extent to which consistent reading of rows within a transaction can be affected by concurrent accesses by other transactions. Concurrent transactions are simultaneous attempts by two or more application programs to access a single row. Depending on the selected isolation level, the following phenomena can occur with concurrent accesses:

- dirty read
  A transaction updates a row or inserts a new row. A second transaction reads this updated or new row before the first transaction has committed the changes. This means that the second transaction has read a row which may still be modified or deleted by the first transaction, i.e. which has not been finalized.
- non-repeatable read
  A transaction reads a row. While this transaction is still open, a second transaction updates or deletes the row concerned and commits the changes. If the first transaction again attempts to access this row, either it receives different values or the attempt to access the row is not successful.
- phantoms
  A transaction selects a number of rows from a table on the basis of a specific condition in a query. While this transaction is still open, a second transaction adds rows to the table which would also fulfil the condition. If the first transaction repeats the same query, the derived table additionally contains the new rows.

For reasons of compatibility, it is possible to use CONSISTENCY LEVEL to specify the consistency level instead of the isolation level.

The following table shows the relationship between the consistency level, the isolation level and the various phenomena which can occur.

Isolation level	Consistency levels	dirty read	non-repeatable read	Phantoms
READ UNCOMMITTED	0	x	x	x
-	1	x	x[1]	x
READ COMMITTED	2	-	x	x
REPEATABLE READ	3	-	-	x
SERIALIZABLE	4	-	-	-

Table 35: Isolation level, consistency level and associated phenomena

[1]The phenomenon “non-repeatable read” may occur if a row was read earlier with a dirty read.

The default value for the isolation level is SERIALIZABLE, which provides complete protection against concurrent transactions. If the isolation level or the consistency level are included in the configuration file for the application program, this value is taken as default. This value must not be higher than the value permitted by the DBH option MAX-ISOLATION-LEVEL. Otherwise SESAM/SQL reports SQLSTATE 91SCL.

Transaction mode

The transaction mode specifies whether rows can only be read or read and updated within a transaction. READ ONLY is default for READ UNCOMMITTED or for consistency levels 0 and 1. In all other cases, READ WRITE is the default. READ UNCOMMITTED is not permitted if the transaction mode READ WRITE is specified at the same time.

Autonomous transaction

The pragma AUTONOMOUS TRANSACTION enables data to be written to a database irrespective of the surrounding transaction. In particular, the data is written persistently to the database before the SQL statement ROLLBACK WORK has possibly executed the transaction.

The pragma is effective only in the case of SQL statement which are to be updated, in other words with INSERT, UPDATE, DELETE (with a search condition), MERGE, and CALL.

The SQL statement after the pragma AUTONOMOUS TRANSACTION is executed in the user's current transaction, but in a separate runtime environment (own thread, own transaction context).

SQL statements of the user's for transaction management have no effect. In other words, the user statements COMMIT WORK and ROLLBACK WORK do not influence persistent writing of data by updating SQL statements of autonomous transactions. The user's SET TRANSACTION statement has no effect on autonomous transactions. The transaction mode of autonomous transactions is READ/WRITE, and the isolation level is the maximum value which the DBH option permits.

Information on user identification, lock conflicts, and aborting the application is provided in the description of the pragma AUTONOMOUS TRANSACTION in the manual " SQL Reference Manual Part 1: SQL Statements".

Switching CALL DML applications

In order to make switching CALL DML applications to the SQL interface easier, you can use certain SQL statements within a CALL DML transaction. This function can only be used with independent DBH. As in the past, application with linked-in DBH may only contain CALL DML or SQL transactions. For further information, see section “Transaction in application programs” and the “ SQL Reference Manual Part 1: SQL Statements”.

Embedding of SQL in programs

Programming language-specific interfaces that allow you to incorporate SQL statements in a program are available, thus allowing you to access a database from a program. SESAM/SQL provides an interface for the programming language COBOL.

ESQL program

An ESQL program is a program in a specific programming language in which additional SQL statements have been embedded. The underlying programming language of an ESQL program is also referred to as the “host language”. SESAM/SQL recognizes ESQL programs for the host language COBOL (see the “ ESQL-COBOL for SESAM/SQL-Server” manual).

In order to allow the compilers to distinguish between statements in the host language and SQL statements, the beginning and end of an SQL statement are marked in the ESQL program. An ESQL precompiler then separates the host language statements from the SQL statements and generates an SQL object module in which the SQL statements are grouped together. The rest of the COBOL program, which now contains no more SQL statements, is compiled using the COBOL compiler. The resulting module is linked with the SQL module and the runtime system of the COBOL compiler to form a load module.

In ESQL-COBOL programs, embedded SQL statements are introduced by the keyword “EXEC SQL” and terminated with “END EXEC”. Executable SQL statements can occur anywhere in a program where an executable statement is possible in the host language. All SQL statements can be executed with the exception of DECLARE CURSOR and WHENEVER. National data types can only be used with an appropriate BS2000 system environment including the corresponding ESQL precompiler and COBOL compiler, see "Unicode concept in SESAM/SQL".

Host variables

SQL statements can contain variables from the host language, known as host variables. Host variables allow you to pass data from the database to the host language program, to process this data in the program and then to pass it back to the database. Within SQL statements, all host variables must be prefixed with a colon. Host variables can bear the same names as columns.

Host variables must be declared in a DECLARE SECTION before they are used. A DECLARE SECTION in ESQL-COBOL starts with “EXEC SQL BEGIN DECLARE SECTION END-EXEC” and ends with “EXEC SQL END DECLARE SECTION END-EXEC”. An ESQL program may include any number of DECLARE SECTIONs.

DECLARE SECTIONs may only occur in locations where variable declarations are permitted in the host language. If a host variable is used in an executable SQL statement, a reference to the declaration must be permitted at this location in accordance with the syntax of the host language.

When defining host variables, you must ensure that the data types of the host language are compatible with the SQL data types. This compatibility must be ensured not only when passing values to or from the database, but also when using host variables in expressions.

Indicator variables

Indicator variables are host variables which can be used to check whether information is lost when transferring an alphanumeric or a national value to a host variable or whether a column contains a null value. Indicator variables are also used to pass null values to the database.

If an alphanumeric or a national value was passed to a host variable without truncation, the indicator variable contains the value “0”. If the length of the value passed was greater then the length of the host variable, the indicator variable contains the actual length of the value passed.

If NULL is to be passed from the database to a host variable, the indicator variable is assigned the value “-1” and the host variable is unchanged. If NULL is to be passed from a host variable to the database, the relevant indicator variable must be assigned a negative value.

An indicator variable is assigned to a host variable in an SQL statement by specifying the indicator variable immediately following the host variable, with the optional keyword INDICATOR.

Indicator variables can be specified in the following situations:

- in the RETURN INTO or VALUES clause of the INSERT statement
- in the VALUES clause of the MERGE statement
- in the INTO clause of the SELECT or FETCH statement
- in the SET clause of UPDATE or MERGE statements
- in the USING and INTO clause of the EXECUTE statement
- in the USING clause of the OPEN statement
- in expressions, e.g. in DELETE or DECLARE CURSOR statements.

Monitoring success and error handling

After an SQL statement has been executed, the ESQL program should check whether execution was successful or whether an error occurred. The ESQL interface provides the host variables SQLSTATE and SQLCODE for this purpose. These are used to store information on the execution of an SQL statement. SQLSTATE contains an SQL status code and SQLCODE contains an SQL return code which has the same function as the SQL status code and which is supported for reasons of compatibility with earlier versions of SESAM/SQL.

The 5-character, alphanumeric SQL status codes are made up of a 2-character part for the class of the status code and a 3-character part for the subclass. If an SQL statement is executed without errors, this is indicated by values for SQLSTATE which have the classes “successful execution”, “warning” or “no data”. If execution was errored, SQLSTATE contains information on the type of error.

The numeric values for SQLCODE are as follows: 0 for successful execution, 100 if a table is empty or the end of the table has been reached and 10 or 50 if a warning is issued. In the event of an error, SQLCODE contains a negative value indicating the error type.

There are two ways of checking whether an SQL statement was executed properly and initiating appropriate action in the host program depending on the SQL status code or SQL return code. On the one hand, the ESQL programmer can use host language statements to control program execution on the basis of the value in SQLSTATE (or SQLCODE). Another option involves the use of the SQL statement WHENEVER.

Data manipulation under SQL using cursors

A cursor is used in SQL in order to process rows of a derived table within an ESQL program. A cursor is a pointer within a special derived table, known as the “cursor table”, which allows users to access the rows of the table.

The cursor table is defined by a query expression which can be specified when defining a cursor with DECLARE CURSOR. This query expression is only evaluated when the cursor is opened with an OPEN statement. If the query expression contains host variables, the current values for the host variables are used. It is now possible to query the rows or, in the case of an updatable cursor, to update or delete them until the cursor is closed using a CLOSE statement.

FETCH statement positions an open cursor on a row in the derived table and makes this row the current row.

The UPDATE... CURRENT statement updates the current row (the row on which the cursor is positioned).

DELETE... CURRENT deletes the row on which the cursor is positioned.

The STORE and RESTORE statements are used to prevent the position of a cursor being lost at the end of a transaction. The current cursor position is only retained if it is stored with the STORE statement before it is closed. RESTORE is used to restore a cursor which was stored with the STORE statement.

Block mode cursors (see also "Dynamic cursors"**)**

You can define a cursor with “block mode” by specifying the PREFETCH pragma in the DECLARE-CURSOR statement. Block mode causes several records to be placed in a buffer the first time the FETCH NEXT statement is executed. When the FETCH statement is executed for the first time, the user is only given the first record. Each time the FETCH statement is executed again, a further record is provided from the buffer (without further task-to-task communication) until the buffer is empty. The next FETCH statement again places several records into the buffer. This block mode speeds up cursor processing considerably. A block mode cursor must always be addressed with the same FETCH statement, i.e. the same FETCH statement in a loop or a subroutine.

The size of the buffer can be defined with the PREFETCH-BUFFER statement (see "Connection module parameters") specified in the configuration file of the application program. The PREFETCH-BUFFERS screen in the SESAM/SQL performance monitor SESMON provides information on the memory requirements for the buffer and on actual usage (see the “ Database Operation” manual).

Dynamic SQL

The SQL statements described up to this point can be used to create an ESQL program which is compiled once and can then be executed as often as required. The form of every SQL statement is known at the time the program is compiled.

If, for instance, you wanted to perform a wide range of queries on a database, you could try to predict all possible queries in an ESQL program. Any given query could then be analyzed by the ESQL program and the appropriate SQL statement called. It is clear that an approach of this type would soon reach its limitations. In order to implement applications of this type, you need language constructs which allow you to specify SQL statements at runtime.

Applications of this type can be implemented using dynamically compiled SQL statements, i.e. statements which are only compiled at runtime. Special statements are available for compiling and executing dynamically compilable SQL statements. These are known as “statements for dynamic SQL” or simply “dynamic SQL”. In addition to the SQL statements for dynamic SQL on "SQL statements for dynamic SQL", they also include SQL statements for dynamic cursors, i.e. cursors defined only at runtime. In contrast to dynamic SQL, those statements which must be defined before compilation are referred to as static SQL.

The following overview illustrates the increasing levels of flexibility provided by the various dynamic SQL constructs:

SQL language resources	Number of parameters known at compilation time	Data types of the parameters known at compilation time	Value of the parameter
Static SQL	yes	yes	Can be passed in host variables
Dynamic SQL EXECUTE IMMEDIATE	no parameters	no parameters	no parameters
Dynamic SQL PREPARE EXECUTE with host variables	yes	yes	Placeholders as dummy input parameters
Dynamic SQL PREPARE EXECUTE with descriptor areas	no	no	Placeholders as dummy input parameters

Table 36: Increasing flexibility when using dynamic SQL

This gain in flexibility is offset by an increased effort when programming applications using dynamic SQL.

The most comprehensive mechanism for setting and reading input parameters for dynamically compiled SQL statements is the SQL descriptor area. An SQL descriptor area acts as an interface for input and output parameters between the application program and the database. SQL statements which are only entered at runtime can contain variable input parameters which are stored in the SQL descriptor area and which can be queried in this area. It is also possible to store the results of querying the database in the SQL descriptor area and to read information on the number, name, data type, values etc. of the derived columns and to take them into account in the ESQL program. It is thus possible, for example, to use descriptor areas to print edit the results of any queries to the database in ESQL programs.

Dynamic preparation and execution of SQL statements

To start with, a brief example will be used to introduce dynamic SQL. This example will not use an SQL descriptor area. In the example, the SQL statements are marked with EXEC SQL and END-EXEC as in ESQL-COBOL. The actions carried out by the host language of the ESQL program are shown in italics.

Example

In the ORDERS table, the order status is to be changed to 3 in the row with job number 251 and customer number 105. This can be done with the following statement in static SQL:

```
UPDATE orders SET order_status=3 WHERE order_num=251 AND cust_num=105
```

This statement corresponds to the following sequence of statements in dynamic SQL:

a. *Declaration of the host variables Sourcestmt and Condition Sourcestmt='UPDATE orders SET order_status=3 WHERE order_num=251'*
b. *Read the text 'AND cust_num=105' into Condition Sourcestmt=Sourcestmt||Condition*
c. `EXEC SQL EXECUTE IMMEDIATE :Sourcestmt END_EXEC`

Explanation

a. *Sourcestmt* is a host variable declared in the ESQL program and which is assigned an alphanumeric string using the UPDATE statement.
b. The UPDATE statement is expanded to include a condition by concatenating the text contained in *Sourcestmt* with the text in *Condition* read in at runtime. Whereas static SQL requires the statement text for the UPDATE statement to be defined before compilation, dynamic SQL only requires the statement to be defined at runtime.
c. The EXECUTE IMMEDIATE statement checks, compiles and executes the statement.

Example

A more generalized form of the UPDATE statement shown above can be achieved with the following statement in static SQL:

```
UPDATE orders SET order_status=:ORDER_STATUS WHERE order_num=:ORDER_NUM

AND cust_num=105
```

Here, the values for ORDER_STATUS and ORDER_NUM are replaced by two host variables whose values are only defined at runtime. This statement corresponds to the following sequence of statements in dynamic SQL:

a. *Declaration of the host variables SOURCEMT, CONDITION, HOSTVAR1 and HOSTVAR2 Sourcestmt='UPDATE orders SET order_status= ? WHERE order_num= ?'*
b. *Read the text 'AND cust_num=105' into CONDITION sourcestmt=sourcestmt||condition*
c. `EXEC SQL PREPARE dynstmt FROM :SOURCESTMT END-EXEC`
d. *Read the value 3 into HOSTVAR1 and 251 into HOSTVAR2*

   ```
   EXEC SQL EXECUTE dynstmt USING :HOSTVAR1,:HOSTVAR2 END-EXEC
   ```

Explanation

a. The UPDATE statement contains two placeholders, represented by question marks in place of the values for ORDER_STATUS and ORDER_NUM in the WHERE condition.
b. The UPDATE statement is expanded to include a condition by concatenating the text contained in SOURCESTMT with the text in CONDITION read in at runtime.
c. This dynamically compilable UPDATE statement is compiled with the following PREPARE statement. DYNSTMT stands for the dynamically compiled SQL statement. This name can then be used to address the statement in an EXECUTE statement.
d. Before the EXECUTE statement is executed, values for the host variables HOSTVAR1 and HOSTVAR2, which were declared in the ESQL program, are read in. These values are to replace the placeholders “?” in the UPDATE statement. The EXECUTE statement then executes the statement DYNSTMT as prepared by PREPARE.

Example The cursor CUR_SERVICE is defined for the SERVICES table. The cursor description is to be generated at program runtime. The user can enter the order number and the condition that are to apply to the services that are to be output.

1. *Declaration of the host variables ORD_NUM_ENTRY, CONDITION and DESCRIPTION. In addition, the SQL variable ORDER_NUM is defined. This takes the value of the order number from ORD_NUM_ENTRY.*
2. Define the cursor CUR_SEVICE:

```
EXEC SQL
     DECLARE cur_service CURSOR FOR CUR_DESCRIPTION
END-EXEC
```

3. *Entry of values for ORD_NUM_ENTRY, and CONDITION by the user at program runtime.*
4. *Generation of the description DESCRIPTION:*

```
STRING "SELECT * FROM services WHERE order_num = ? AND "
condition
```

5. Generate cursor description:

```
EXEC SQL
     PREPARE CUR_DESCRIPTION FROM :DESCRIPTION
END-EXEC
```

6. Open the cursor CUR_SERVICE. The placeholder “?” in the cursor description is now replaced by the order number selected by the user:

```
EXEC SQL
     OPEN cur_service USING :ORDER_NUM
END-EXEC
```

The statements EXECUTE IMMEDIATE, PREPARE and EXECUTE used in the examples are described in detail below. There first follows an overview of the statements which can be dynamically compiled.

Dynamically compilable statements

The following statements can be dynamically compiled (refer to "SQL statements for schema definition and administration" for details on the classification of the statements):

- SQL statements for schema definition and administration
- SQL statements for designing and managing routines
- SQL statements for querying and updating data:
  - SELECT (without INTO clause)
  - UPDATE
  - DELETE
  - INSERT (without RETURN INTO clause)
  - MERGE
  - CALL
- SQL statements for transaction management
- SQL statements for session control
- SQL statements for managing the storage structure
- SQL statements for managing user entries
- Utility statements

In addition, cursor descriptions can be dynamically compiled.

The following statements cannot be dynamically compiled

- the SQL statements for dynamic SQL
- the following SQL statements for working with a cursor:
  - DECLARE CURSOR
  - OPEN
  - FETCH
  - CLOSE
  - STORE
  - RESTORE
- the SQL statements INCLUDE and WHENEVER.

EXECUTE IMMEDIATE statement

You use the EXECUTE IMMEDIATE statement to prepare and execute a dynamic statement in one step. In other words, EXECUTE IMMEDIATE corresponds to a PREPARE statement immediately followed by an EXECUTE statement. The statement does not, however, remain prepared and cannot be executed again with EXECUTE.

The statement text cannot include any host variables or question marks as placeholders for unknown values.

PREPARE statement

You use PREPARE to prepare a dynamic statement or the cursor description of a dynamic cursor for execution at a later time.

You execute a statement prepared with PREPARE with the EXECUTE statement.

The statement text cannot include any host variables. Question marks are specified as placeholders for unknown values .

Placeholder

The dynamically compilable statement or dynamically compilable cursor description in statement variable may not contain any host variables. Instead, they may contain placeholders for input values which are represented by question marks. These placeholders are provided with values with the USING clause of a subsequent EXECUTE or OPEN statement.

An SQL data type is determined for every placeholder. There are a number of rules which restrict the use of placeholders in *statement_variable*. A placeholder is only permitted in a context in which the data type of the placeholder can be determined uniquely.

For this reason, placeholders are not permitted

- as elements of a select list in the form “?” (“SELECT ?+1” would, however be permitted)
- as an operand of a monadic operator
- as an argument of an aggregate function
- in a query for a null value
- if both operands of a dyadic operator or a comparison operator are placeholders
- if, in a range query with BETWEEN where the first operand is a placeholder, one of the other two operands is also a placeholder
- as the first operand in pattern matching with LIKE
- if, when comparing an expression with a set of values, both the expression to be compared and the first value in the list following IN are placeholders.
- if all operands are placeholders in a CASE expression. If the CASE expression contains one or more THEN or ELSE clauses, not all the operands in these clauses can be placeholders.
- as operands of the string functions LOWER and UPPER
- as the first operand (*character*) and/or second operand (*expression*) of the string function TRIM (e.g. TRIM (TRAILING FROM ?))
- as the first operand of the string function SUBSTRING (e.g. SUBSTRING ? FROM 1 FOR 5)).
- if both operands of the numeric function POSITION are placeholders.

EXECUTE statement

You use EXECUTE to execute a statement prepared with PREPARE. Placeholders for input values in the dynamic statement are replaced by specific values.

If the statement is a SELECT statement, the column values of the derived rows are stored in host variables or in an SQL descriptor area.

You can use EXECUTE to execute a previously prepared statement any number of times.

A statement can only be executed with EXECUTE in the compilation unit in which it was previously prepared with PREPARE.

Host variables or SQL descriptor areas can be used for input and output values in the EXECUTE statement. The use of SQL descriptor areas is described in the following section.

SQL descriptor area

In the dynamic SQL statements described up to this point, the number and data types of the placeholders and derived columns for a dynamically compilable SQL statement or a dynamic cursor description were known at the time the ESQL program was compiled. So-called descriptor areas are available in SQL to allow you to use any number and type of input parameters and to process any results from queries in ESQL programs.

A descriptor area is an area in memory in which values and information on the number and data types of inputs (placeholders) and outputs (derived columns) in dynamically compiled SQL statements or cursor descriptions are stored.

Every descriptor area comprises a number of items. One item is created for every simple column or value or, in the case of multiple columns or aggregates, for every column element or every occurrence. Every item is subdivided into fields. The values in these fields can be set with DESCRIBE or SET DESCRIPTOR or read with GET DESCRIPTOR. Depending on the type of the item, only certain fields are assigned values. Every descriptor area has a field with the name COUNT, which contains the number of entries in the descriptor area.

The descriptor area contains the following fields for a given item:

Keyword	Meaning	Can be set in SQL program
COUNT	Number of items in the descriptor area	yes
TYPE	Data type of item: -42 NVARCHAR -31 NCHAR 1 CHAR 2 NUMERIC 3 DECIMAL 4 INTEGER 5 SMALLINT 6 FLOAT 7 REAL 8 DOUBLE PRECISION 9 Time and date data type 12 VARCHAR	yes
LENGTH	Length or maximum length in characters in the case of alphanumeric data types and time data types or length or maximum length in code units in the case of national data types	yes
OCTET_LENGTH	Maximum memory requirement in bytes in the case of alphanumeric data types, national data types, numeric data types and time data types	No, can be read only

PRECISION	Only for numeric data types and TIME and TIMESTAMP: Total number of decimal digits (for NUMERIC, DECIMAL, TIME, TIMESTAMP) or binary digits (in all other cases)	yes
SCALE	Only for integers and fixed-point numbers: number of digits after the decimal point	yes
DATETIME_ INTERVAL_CODE	Only for time and date data types (TYPE =9): 1 DATE2 TIME3 TIMESTAMP	yes
REPETITIONS	For multiple columns and aggregates: contains the number of components in the first item and 1 in the following items. For single columns: contains the value 1 only.	yes
NULLABLE	○ 1 Output value can be NULL ○ 0 in all other cases	no
INDICATOR	○ In a descriptor area for derived columns: 0 DATA contains the value read -1 NULL was read >0 Original length of an alphanumeric or a national value if information was lost during transfer ○ In a descriptor area for placeholders: <0 Value is NULL 0 in all other cases	yes
DATA	Value of an item. The data type is defined in by the entries in TYPE, LENGTH, PRECISION, SCALE and DATETIME_INTERVAL_CODE	yes
NAME	Column name	no
UNNAMED	○ 1 NAME contains column name ○ 0 in all other cases	no

Tabelle 37: Fields of a descriptor area

The COUNT field occurs once in every descriptor area. The data type of NAME is CHAR(*n*) or VARCHAR(*n*) with n >= 128 and the data type of DATA is determined by the entries in TYPE, LENGTH, PRECISION and SCAL. The remaining fields have the data type SMALLINT.

Users do not need to concern themselves with the internal structure of the SQL descriptor area, but can address the fields of the descriptor area using SQL statements. Depending on the statement concerned, only certain fields in the descriptor area are addressed.

There are a number of SQL statements and variants of the EXECUTE, OPEN and FETCH statements available for working with an SQL descriptor area:

- Creating a descriptor area
  A descriptor area must first be created with ALLOCATE DESCRIPTOR. This statement specifies the maximum number of items in the descriptor area. The items in the descriptor area are undefined after the ALLOCATE DESCRIPTOR statement has been executed.
- Describe the number and data types of the placeholders
  DESCRIBE INPUT allows you to describe the number and data types in a dynamically compiled SQL statement or cursor description and to store these in a descriptor area.
- Pass values for placeholders
  The values for the placeholders in a dynamically compiled statement are taken from a descriptor area in the EXECUTE... USING SQL DESCRIPTOR statement. The values for the placeholders in a dynamically compiled cursor description are taken from a descriptor area in the OPEN... USING SQL DESCRIPTOR statement. In both cases, the required values must first be set using SET DESCRIPTOR.
- Describe the number and data types of the derived columns
  DESCRIBE OUTPUT allows you to describe the number and data types of the derived columns in a dynamically compiled SELECT statement or cursor description and to store these in a descriptor area.
- Store column values from a derived row
  The column values from a derived row in a dynamically compiled SELECT statement are stored in a descriptor area with EXECUTE... INTO SQL DESCRIPTOR. The column values of a row of the derived table in a dynamic cursor are stored in a descriptor area with FETCH... INTO SQL DESCRIPTOR.
- Modifying the descriptor area
  The number of items or the contents of an item of the descriptor area can be modified with SET DESCRIPTOR.
- Querying the descriptor area
  The number of items or the contents of fields in an item of a descriptor area can be queried with GET DESCRIPTOR.
- Releasing the descriptor area
  The memory occupied by a descriptor area can be released with DEALLOCATE DESCRIPTOR.

Dynamic cursors

Cursors declared without a cursor description are referred to as dynamic cursors.

A dynamic cursor is declared with DECLARE CURSOR.
A dynamic cursor is associated with a dynamically compilable cursor description using PREPARE. As with a static cursor, the user can access the rows of a table with the OPEN, FETCH and CLOSE statements and update or delete rows in a table with the UPDATE CURRENT AND DELETE CURRENT statements.

As with dynamic SQL statements, dynamic cursors provide an increasing degree of flexibility, which is offset by the increased effort involved in programming:

SQL language resources	Number of parameters known at compilation time	Data types of the parameters known at compilation time	Values of parameters
Static cursor	yes	yes	Can be passed in host variables
Dynamic cursor DECLARE CURSOR PREPARE OPEN... USING FETCH ... INTO CLOSE with host variables	yes	yes	Placeholders as dummy input parameters
Dynamic cursor DECLARE CURSOR PREPARE OPEN... USING FETCH ... INTO CLOSE with descriptor areas	no	no	Placeholders as dummy input parameters

Table 38: Increased flexibility when using dynamic cursors

If a dynamically compilable cursor description contains placeholders, the corresponding values must be made available in the USING clause of the OPEN statement via host variables or a descriptor area previously supplied with values.

An SQL descriptor area allows the following operations to be performed for a dynamically compiled cursor description:

- define the data types of the placeholders with DESCRIBE INPUT
- define the number and data types of the derived columns and store them in a descriptor area with DESCRIBE OUTPUT
- derive the values for the placeholders from a host variable or a descriptor area using OPEN USING
- store the values of a row in the derived table in a descriptor area with FETCH INTO.

Dynamic block mode cursor

In the same way as for a static cursor (see "Data manipulation under SQL using cursors"), block mode can be defined for a dynamic cursor. This is done by specifying the PREFETCH pragma in the cursor description for the dynamic cursor.

If a FETCH statement is used with a descriptor area, this descriptor area must not be addressed in SET DESCRIPTOR, DESCRIBE or DEALLOCATE DESCRIPTOR statements as long as block mode is activated. In addition, the same restrictions apply for a static cursor as for a static block mode cursor.

Example

```
DECLARE dyn_cursor CURSOR FOR dynstmt
```

Read SOURCESTMT:

```
'--%PRAGMA PREFETCH 10 SELECT cust_num, company FROM customers WHERE
country = 'D ''
PREPARE dynstmt FROM :SOURCESTMT
OPEN dyn_cursor
```

Loop until SQLSTATE = 00200:

```
FETCH NEXT FROM dyn_cursor INTO :CUST_NUM, :COMPANY
```

In the cursor description for the dynamic cursor DYN_CURSOR, the PREFETCH pragma is specified with a blocking factor of 10. On the first FETCH NEXT statement, the user receives the first row of the cursor table and the next 9 rows are read into the buffer at the same time. A block mode cursor must always be addressed with the same FETCH statement, i.e. the same FETCH statement in a loop or a subroutine. The next time this FETCH statement is executed, the second row of the cursor is read from the buffer far more quickly.

Defining default values in dynamic statements

The statement SET CATALOG allows the user to specify a default database name used to qualify simple schema names in dynamically compilable SQL statements.

SET SCHEMA defines a default schema name with used to qualify simple names for tables, integrity constraints and indexes in dynamically compilable SQL statements.

The default database or schema name is derived from the value of an alphanumeric literal or from the value of a host variable with an alphanumeric data type. The database or schema name is valid until the next SET CATALOG or SET SCHEMA statement. It is also possible to set a default database name and schema name together by specifying a qualified schema name in the SET SCHEMA statement.

Dynamic SQL with descriptor areas

Example

Any SQL statement is read in at runtime. It is assumed that the number and data types of the input values are not known when the program is written. The DESCRIBE statement is used to determine both the input values and the number and data types of the column values for the derived rows.

In the example, the SQL statements are marked with EXEC SQL and END-EXEC as in ESQL-COBOL. The actions carried out by the host language of the ESQL program are shown in italics.

Error handling is only demonstrated once in this example. A complete example would, of course, require SQLSTATE to be queried after every executable SQL statement and appropriate error handling to be initiated. It is also assumed that there are no multiple columns.

Declaration of the host variables H_COUNT, H_TYPE, H_LENGTH, H_NAME etc. to receive the values for COUNT, TYPE, LENGTH, NAME etc. in the fields of the descriptor area.

Declaration of host variables for each of the SQL data types: H_CHARACTER, H_NUMERIC1, H_NUMERIC2 etc. for different value ranges etc.

Declaration of the host variable SOURCESTMT to receive the dynamically compilable statement

1. `EXEC SQL DECLARE dyn_cursor SCROLL CURSOR FOR dynstmt END-EXEC`
2. `EXEC SQL ALLOCATE DESCRIPTOR GLOBAL 'Placehold' END-EXEC`
   `EXEC SQL ALLOCATE DESCRIPTOR GLOBAL 'Results' END-EXEC`
3. *Read in SOURCESTMT*

   `EXEC SQL PREPARE dynstmt FROM :SOURCESTMT END-EXEC`
4. a. `EXEC SQL DESCRIBE INPUT dynstmt`
      `USING SQL DESCRIPTOR GLOBAL 'Placehold' END-EXEC`

      Error handling if SQLSTATE(1:2) is not equal to “00”

   b. `EXEC SQL DESCRIBE OUTPUT dynstmt USING SQL DESCRIPTOR GLOBAL 'Results' END-EXEC`

      Error handling if SQLSTATE(1:2) is not equal to “00”
5. `EXEC SQL GET DESCRIPTOR GLOBAL 'Placehold' :H_COUNT = COUNT END-EXEC`
6. *Loop from I=1 to I=H_COUNT*

   ```
   EXEC SQL GET DESCRIPTOR GLOBAL 'Placehold' VALUE :I
   :H_TYPE =TYPE,
   :H_LENGTH = LENGTH, ..., END-EXEC
   ```
7. *Selection of a suitable user variable in accordance with the placeholder's data type. Adjusting the data type and reading the value for the placeholder into the descriptor area, e.g.:*

   if :H_TYPE = 1 or 12 Read the value for the placeholder into H_CHARACTER

   ```
   EXEC SQL SET DESCRIPTOR GLOBAL 'Placehold' VALUE :I TYPE = 1,
   LENGTH=256, INDICATOR=0, DATA=:H_CHARACTER END-EXEC
   ```

   if :H_TYPE =2 or 3 and 10 to 18 digits before the decimal place Read the value for the placeholder into H_NUMERIC1

   ```
   EXEC SQL SET DESCRIPTOR GLOBAL 'Placehold' VALUE :I TYPE=2,
   ```

```
PRECISION=18, SCALE=0, INDICATOR=0, DATA=:H_NUMERIC1 END-EXEC
```

if :H_TYPE =2 or 3 and 1 to 9 digits before the decimal place Read the value for the placeholder into H_NUMERIC2

```
EXEC SQL SET DESCRIPTOR GLOBAL 'Placehold' VALUE :I TYPE=2,
PRECISION=18, SCALE=9, INDICATOR=0, DATA=:H_NUMERIC2 END-EXEC
```

if :H_TYPE =2 or 3 and 0 digits before the decimal place Read the value for the placeholder into H_NUMERIC3

```
EXEC SQL SET DESCRIPTOR GLOBAL 'Placehold' VALUE :I TYPE=2,
PRECISION=18, SCALE=18, INDICATOR=0, DATA=:H_NUMERIC3 END-EXEC
```

etc.

End of loop

8. `EXEC SQL GET DESCRIPTOR GLOBAL 'Results' :H_COUNT=COUNT END-EXEC`

   if :H_COUNT=0

   ```
   EXEC SQL EXECUTE dynstmt USING SQL DESCRIPTOR GLOBAL 'Placehold'
   END-EXEC
   ```

   Go to (12)

9. *else*

   Loop from I=1 to I=H_COUNT

   ```
   EXEC SQL GET DESCRIPTOR GLOBAL 'Results' VALUE :I
   :H_TYPE=TYPE
   :H_LENGTH=LENGTH ... END-EXEC
   ```

   Selection of a suitable user variable in accordance with the data type of the column values in the derived table. Adjusting the data type in the descriptor area, like (6) for the placeholders, e.g.:

   if :H_TYPE = 1 or 12

   ```
   EXEC SQL SET DESCRIPTOR GLOBAL 'Results' VALUE :I TYPE = 1, LENGTH=256
   END-EXEC
   ```

   etc.

   End of loop

10. `EXEC SQL OPEN dyn_Cursor USING SQL DESCRIPTOR GLOBAL 'Placehold' END-EXEC`

11. *Execute the following FETCH statement in a loop until the “end of table” condition occurs*

    ```
    EXEC SQL FETCH NEXT FROM dyn_cursor INTO SQL DESCRIPTOR GLOBAL
    'Results' END-EXEC
    ```

    Error handling if SQLSTATE(1:2) is not equal to “00”

    Loop through all columns of the derived row: if :H_TYPE =1 or 12

    ```
    EXEC SQL GET DESCRIPTOR GLOBAL 'Results' VALUE :I
    :H_NAME=NAME, :H_CHARACTER=DATA END-EXEC
    ```

    if :H_TYPE =2 or 3 and 10 to 18 digits before the decimal place

    ```
    EXEC SQL GET DESCRIPTOR GLOBAL 'Results' VALUE :I
    :H_NAME=NAME, :H_NUMERIC1=DATA END-EXEC
    ```

    etc.

    Output H_NAME and the corresponding host variable

End of loop

End of loop

12. `EXEC SQL CLOSE dyn_cursor END-EXEC`
13. `EXEC SQL DEALLOCATE DESCRIPTOR GLOBAL 'Placehold' END-EXEC`

    `EXEC SQL DEALLOCATE DESCRIPTOR GLOBAL 'Results' END-EXEC`

Meaning

1. Declare the dynamic cursor DYN_CURSOR. DYN_CURSOR has a dynamically compilable cursor description DYNSTMT.
2. Create the descriptor areas: Placehold for the placeholders and Results for the column values of a derived row.
3. Read the statement text into SOURCESTMT and prepare it for subsequent execution. The statement text can be a dynamically compilable statement or a cursor description.
4.
   a. After successful execution of the PREPARE statement, the number and data types of the placeholders in the dynamically compiled SQL statement DYNSTMT are queried with DESCRIBE INPUT. If the DESCRIBE INPUT statement was successful, DYNSTMT is dynamically compiled and the SQL descriptor area Placehold is filled with the type descriptions for the dynamic parameters.
   b. In the same way, DESCRIBE OUTPUT is used to write the number and data types of the columns of a derived row to the descriptor area Results. If the DESCRIBE OUTPUT statement was successful, DYNSTMT is dynamically compiled and the SQL descriptor area Results is filled with the type descriptions for the columns of a derived row.
5. GET DESCRIPTOR is used to determine the number of placeholders H_COUNT.
6. GET DESCRIPTOR is used to read the data type, length etc. of every placeholder and to transfer these to the appropriate host variables H_TYPE, H_LENGTH etc.
7. A host variable is selected to suit the data type of the placeholder. The value of the placeholder is read into this host variable. In order to transfer the value of a placeholder to the descriptor area, the SET DESCRIPTOR statement must be used to set the data type, length, number of digits before and after the decimal point etc. in the descriptor area to suit the data types of the host variables. The demonstration program, for example, allows for three data types, corresponding to the SQL data types NUMERIC(18,0), NUMERIC(18,9) and NUMERIC (18,18). Depending on the data type of the placeholder, one of these data types is set in the descriptor area. After the SET DESCRIPTOR statements have been executed, the descriptor area Placehold contains the values for the placeholders.
8. GET DESCRIPTOR queries the number of columns in the derived table. If this number is 0, the dynamically compiled statement is executed with EXECUTE. The descriptor area Placehold is specified in the USING clause. This area contains the values for the placeholders in the dynamically compiled statement. The program is then continued at (13).
9. Otherwise, the rows of the derived table are read using the cursor declared in (1).In the same way as for steps (6) and (7), the data types and lengths of the derived columns are first read for the descriptor area Results using GET DESCRIPTOR. SET DESCRIPTOR is then used to adjust the data types, lengths etc. in the descriptor area.
10. The cursor DYN_CURSOR is opened. The descriptor area Placehold is specified in the USING clause. It contains the values for the placeholders in the cursor description.
11. The derived table is written to the descriptor area row by row using the FETCH statement. GET DESCRIPTOR is used to transfer the column values from the descriptor area to suitable host variables and to output them.

12. The cursor DYN_CURSOR is closed.
13. Finally, the descriptor areas are released with DEALLOCATE.

Utility concept

This chapter describes

- SESAM/SQL's utility functions
- the embedding of utility statements in the system
- the embedding of utility statements in programs.

The section “SESAM/SQL utility functions” describes the utility functions provided by SESAM/SQL and the utility statements that implement these functions.

The section “Integration of utility statements in the system” describes peculiarities associated with the execution of utility statements by SESAM/SQL. In addition, it covers the different states of user spaces occurring as a result of utility statements.

The section “Embedding of utility statements in programs” deals in brief with the incorporation of utility statements in an ESQL program, as well as success monitoring and error handling.

SESAM/SQL utility functions

SESAM/SQL provides utility functions through utility statements.

The table below shows the utility functions provided by SESAM/SQL and indicates which utility statements implement the individual utility functions.

Utility function	**Utility statement**
Modify properties of the database	ALTER CATALOG
Create a database (catalog space)	CREATE CATALOG
Create a SESAM backup copy	COPY
Create a replication	CREATE REPLICATION
Update a replication	REFRESH REPLICATION
Extend a replication	REFRESH SPACE
Recover a database, catalog space, user space(s)	RECOVER [USING]
Reset database, catalog space, user space(s)	RECOVER TO
Reset database to any time	RECOVER CATALOG [USING] TO ANY *time_stamp*
Rebuild indexes	RECOVER INDEX
Load user data into a base table	LOAD
Unload user data from base table	UNLOAD
Export table from database to an export file	EXPORT TABLE
Import table from export file into database	IMPORT TABLE
Shuffle column values, anonymize user data	ALTER DATA FOR TABLE
Reorganize the catalog space, user spaces or base tables	REORG
Changing the partitioning of a base table	ALTER PARTITIONING FOR TABLE
Check integrity constraints	CHECK CONSTRAINTS
Check the format of base tables and indexes	CHECK FORMAL
Edit media table	ALTER MEDIA DESCRIPTION CREATE MEDIA DESCRIPTION DROP MEDIA DESCRIPTION
Maintain information (metadata) on the SESAM backup copies	MODIFY

Migrate databases and tables	MIGRATE

Table 39: SESAM/SQL utility functions

In chapter “Building, loading and maintaining databases”, the utility statements are described in connection with the tasks involved in developing and maintaining a SESAM/SQL database.

The individual utility statements are described in full in the “SQL Reference Manual Part 2: Utilities”.

Integration of utility statements in the system

Utility statements are statements in SQL syntax, but they may not be executed within SQL transactions. A utility statement does not initiate an SQL transaction and it cannot be rolled back. A utility statement opens an internal transaction, however, which is closed again when the statement is terminated.
Utility statements can be compiled dynamically (section “Dynamic SQL”).

Execution of utility statements by SESAM/SQL

SESAM/SQL uses locking mechanisms for transactions to guarantee that utility statements are synchronized:

- The utility statement waits till all open transactions have been terminated which already have locks on the relevant spaces).
- While the utility statement is being executed, all subsequent accesses wait until the internal transaction of the utility statement has been terminated if they need to access the spaces concerned.

CALL-DML-OPENs are closed and stored SQL cursors are invalidated when the relevant table has been modified by the utility. Subsequent CALL-DML accesses receive status 9U, accesses to the SQL cursor receive SQLSTATE 24SA5. The utility statements concerned are:

- REORG [CATALOG_]SPACE
- CHECK CONSTRAINTS
- LOAD OFFLINE
- IMPORT TABLE
- RECOVER CATALOG / CATALOG_SPACE
- RECOVER SPACE
- REFRESH REPLICATION
- REFRESH SPACE
- ALTER CATALOG
- ALTER PARTITIONING FOR TABLE
- ALTER DATA FOR TABLE

The following table shows the spaces affected by the execution of the individual utility statements.

Utility statement	**Spaces affected by the utility statement**
ALTER CATALOG	Catalog space
CREATE CATALOG	Catalog space
COPY	Specified backup unit (catalog space and all user spaces, catalog space, space set, one or more user spaces)
CREATE REPLICATION	Specified unit (replication)
REFRESH REPLICATION	Specified unit (replication or partial replication)
REFRESH SPACE	Specified spaces
RECOVER [USING/TO]	Specified unit for recovery (catalog space and all user spaces, catalog space, space set, space list, individual user space)

RECOVER USING/TO REPLICATION	Specified unit for the recovery (catalog space and all user spaces of the replication or of the partial replication, catalog space, space list, single user space)
RECOVER INDEX	All user spaces in which the specified indexes are located
REORG [CATALOG_]SPACE	Catalog space, specified user space
REORG ONLINE TABLE	User space in which the base table or the partition of the base table is located
LOAD	User space in which the base table is located into which user data is loaded with LOAD
UNLOAD	User space in which the base table is located from which user data is unloaded with UNLOAD
EXPORT TABLE	User space in which the base table that is to be exported to an export file is located
IMPORT TABLE	User space into which a base table is to imported by means of an export file; possibly user space in which the indexes are to be rebuilt
ALTER DATA FOR TABLE	User space on which the base table is located
ALTER PARTITIONING FOR TABLE	User spaces on which the relevant partitions of the base table are located
CHECK FORMAL	User space to be formally checked or the user space in which the base table or index to be checked is located
ALTER MEDIA DESCRIPTION CREATE MEDIA DESCRIPTION DROP MEDIA DESCRIPTION	Media table in the catalog space
MODIFY	RECOVERY_UNITS and DA_LOGS catalog tables in the catalog space (resp. CAT-REC file)
MIGRATE	Catalog space and user space in which the base table to be migrated is located

Table 40: Spaces affected by utility statements

Depending on the utility statement in question, other users can perform full or limited updates or queries, or no updates or queries, on the space affected by the utility statement while it is executing.

Concurrent access is not permitted in connection with:

- ALTER CATALOG
- ALTER PARTITIONING FOR TABLE (only for the spaces concerned)
- ALTER DATA FOR TABLE
- CREATE CATALOG
- CREATE REPLICATION
- MIGRATE
- RECOVER
- RECOVER INDEX
- REFRESH REPLICATION
- REFRESH SPACE
- REORG [CATALOG_]SPACE
  (while copying or renaming the work file to/in the user space and while reorganizing the catalog space)
- LOAD OFFLINE (if the GENERATE INDEX clause is specified)
- IMPORT TABLE

Concurrent queries on tables and indexes in the space not affected by the utility statement are permitted in connection with:

- CHECK CONSTRAINTS
- LOAD OFFLINE (if the GENERATE INDEX clause is not specified)

Concurrent queries on the whole of the space affected by the utility statement are permitted in connection with:

- CHECK FORMAL
- COPY
- UNLOAD
- EXPORT TABLE (if user data is exported)
- REORG [CATALOG_]SPACE (for the time taken to set up the work file)

Concurrent queries and updates on the whole of the space affected by the utility statement are permitted in connection with:

- COPY ONLINE
- EXPORT TABLE (if no user data is exported)
- LOAD ONLINE
- MODIFY
- ALTER/CREATE/DROP MEDIA DESCRIPTION

- REORG ONLINE TABLE
- UNLOAD ONLINE

SESAM/SQL denies illegal access attempts, returning a status code, and treats spaces, tables and indexes to which access is not permitted as unavailable spaces, tables and indexes.

Parallel RECOVER statements

You can shorten RECOVERY runs by running multiple RECOVER statements concurrently.

You can run the following RECOVER statements concurrently:

- RECOVER of a single space: `RECOVER SPACE` *space* `USING` *rec_unit*
- RECOVER of a space list: `RECOVER SPACE` *space* `,` *space* `[,...] USING` *rec_unit*
- RECOVER of a space set: `RECOVER SPACESET AT CATALOG` *catalog* `USING` *time stamp*

Prevalent conditions for parallel processing

- The spaces specified in the RECOVER statements must be disjointed, i.e. a space only may be processed by one of the concurrent RECOVER statements.
- The parameters TO, RESTART and ADJUST must not be specified for concurrent RECOVER statements. This means that only those RECOVER statements are permitted where the modifications logged in the logging files are to be applied.
- A sufficient number of service tasks must be started (one service task per concurrent RECOVER statement).

If one of these conditions is not fulfilled, the RECOVER statements concerned will be serialized by the locking mechanism.

> ¡ If a partitioned table is located on the spaces to be recovered, for reasons of consistency these spaces should be recovered using a **single** RECOVER statement.

Space state after the execution of utility statements

The user spaces, tables and indexes affected by the execution of the utility remain locked in the following cases even after execution of the utility statement has been completed:

- After LOAD OFFLINE, when the indexes have not yet been recovered.
  Index state: “defective”
  User space state: OK
- Once user data has been loaded into a base table in the user space with LOAD OFFLINE, the integrity constraints have not yet been checked. The primary key constraint is always checked when LOAD is executed.
  Base table state: “check pending”
  User space state: “check pending”
- After IMPORT TABLE in the following situations:
  - If user data is being imported and logging is activated for the user space.
    User space state: “copy pending”
  - If indexes have not yet been constructed.
    Index state: “defective”
    User space state: OK
- After ALTER DATA FOR TABLE, a SESAM backup of the user space has to be created before processing of the user space continues if logging is enabled for this user space.
  User space state: “copy pending”
- After ALTER PARTITIONING FOR TABLE, a SESAM backup of the user spaces has to be created before processing of the user space continues if logging is enabled for these user spaces.
  User space state: “copy pending”
- During the execution of the CHECK CONSTRAINTS statement, SESAM/SQL detected an integrity constraint violation in connection with at least one base table in the user space.
  Base table state: “check pending”
  User space state: “check pending”
- After RECOVER TO, the user data violates integrity constraints.
  User space state: “check pending”
- After LOAD OFFLINE and MIGRATE, a SESAM backup of the user space has to be created before processing continues if logging is enabled for the user space in question.
  User space state: “copy pending”
- After MIGRATE, LOAD OFFLINE or IMPORT TABLE, if loading is cancelled because of an error.
  User space state: “load running”
- After RECOVER USING, if, for a partitioned table, changes must be applied on this space with CREATE INDEX or DROP INDEX, but not all the user spaces who accomodate partitions of this partitioned table are contained in the RECOVER statement.
  State of the user spaces on which indexes are located which where newly created using RECOVER: “copy pending”

- In the following situations after a RECOVER TO:
  - Logging is enabled for the user space and the data needed to be adapted to the definitions in the catalog space.
    User space state: “copy pending”
  - The indexes had to be recovered.
    State of the user space in which those indexes are located which were rebuilt with RECOVER TO: “copy pending”
- The current state of the user space could not be recovered during repair (e.g., because the DA LOG files are missing or corrupt).
  User space state: “recover pending”
- An error occurred while the user space was being processed (e.g., a utility statement terminated with an error) or SESAM/SQL detected a data error in the user space.
  User space state: “space defect”

In section “Working on locked user spaces” you will find a description of how the database administrator can edit locked user spaces.

For information on which utility statements can be executed in connection with a specific user space state and on the state to which the user space switches as a result of the given statement, see the “SQL Reference Manual Part 2: Utilities”.

Embedding of utility statements in programs

Program structure

Utility statements are specified in ESQL programs in the same way as SQL statements. However, utility statements may not be executed within SQL transactions. Further details on program structure are provided in the “SQL Reference Manual Part 1: SQL Statements”.

Monitoring success and error handling

A utility statement issues a return code in the same way as an SQL statement: SESAM/SQL returns an SQL status code. For further details are provided in the “SQL Reference Manual Part 1: SQL Statements”.

Security policy

Security, i.e. ensuring the confidentiality and integrity of stored information, is an important aspect in today's dp and SQL systems.

A distinction is made between the following security criteria:

- System access control (identification, authentication)
  Users must be identified and authenticated before interaction can begin.
- Data access control (administration of rights, verification of rights)
  The system must manage access rights between users (subjects) and objects. Each time users attempt to access objects which are subject to administration of rights, the operating system must check whether they are authorized to do this. Unauthorized access attempts must be rejected.
- Logging
  The system must contain a logging component which is able to log events which are relevant to security.
- Reprocessing
  Before being reused by other users, sensitive stored objects must be processed in such a way that no conclusions can be drawn about their previous content.

General information on the subject of security in dp systems and information on security in BS2000 is provided in the " SECOS (BS2000)" manual.

The security officer of the dp system and the SESAM/SQL administrator must use the protective mechanisms of the dp system, in other words of BS2000, and of SESAM/SQL to develop a security policy which offers the required security and permits regular checks. The protective mechanisms of BS2000 are described in the " SECOS (BS2000) " manual; the protective mechanisms of SESAM/SQL are described in this chapter.

The SESAM/SQL database system includes a wide range of protective mechanisms for effective data protection.

- The files of a SESAM/SQL database can be protected by BS2000 passwords, see section "BS2000 passwords for files in the database".
- Users must be authorized to access the database (system access control), in other words they must be registered with the database system as authorized users. Details of the requirements for working with a SESAM /SQL database are provided in section "Access permission for a SESAM/SQL database". This section introduces the concepts of a system user ID and an authorization identifier, and describes how to set up a user with comprehensive privileges.

  Here the SESAM/SQL administrator can also use the technical options of BS2000 and openUTM, e.g. access check via chipcard or electronic signature.
- Users must be privileged to perform the relevant operations within the database (data access control). They have to have been assigned the relevant privilege for each operation they would like to carry out. Access protection in SQL through privilege assignment is described in section "Access protection based on privileges in SQL". The section begins by describing features of access protection under SQL, such as special privileges and table privileges. It also covers aspects of assigning and revoking privileges that require special attention.
- In section "Access protection in connection with views" you will find a description of how views can enhance access protection under SQL.
- Data access control can also be implemented for CALL DML tables using the SEPA utility routine, see section "Password protection with SEPA for CALL DML tables".

- The section “Data access control by means of data enryption” describes how sensitive data can be protected by means of encryption. This data is also protected in accordance with the security criterion “reprocessing”. Using different keys also enables data from different security areas (e.g. credit card numbers and medical data) to be encrypted differently.
- The section “Protection of person-related data by means of anonymization” describes how sensitive data is anonymized with SESAM/SQL.
- The section “Logging security-relevant events with SAT” describes the logging component of SESAM/SQL.
- The SESCOSP and SEDI70 utility routines (see the “ Database Operation” manual) log database accesses and create logs for auditing the SESAM/SQL system.

BS2000 passwords for files in the database

The files in the SESAM/SQL database (database files, database-specific files, and DBH-specific files) are ordinary files that can be password-protected from the point of view of BS2000. The database administrator can assign a BS2000 password in the utility statement CREATE CATALOG when creating a database. SESAM/SQL automatically applies this password to all files in the database.

If the database is to be created under a DB user ID, the database administrator must declare the DBH user ID in the DB user ID as a co-owner of the files concerned or create the files and assign the BS2000 password him or herself, see section “Database files and job variables on foreign user IDs”. The password must be specified in the CREATE CATALOG utility statement.

Before they can work with a SESAM/SQL database protected by BS2000 passwords, system administrators must specify the relevant password in the DBH statement ADD-SQL-DATABASE-CATALOG-LIST when starting DBH. The password is then stored in DBH's SQL database catalog.

If the password for a SESAM/SQL database is to be changed, the database administrator must change it specifically for each individual file in the database with the aid of the command MODIFY-FILE-ATTRIBUTES. When doing this, care must be taken to ensure that the same password is assigned to each file.

If the database's password has been changed since a specific SESAM backup copy was created, the password that was valid when the SESAM backup copy was created has to be specified when the SESAM backup copy is recovered with the utility statement RECOVER.

Access permission for a SESAM/SQL database

- System entry
- SQL users with universal authorization
- Creation of further SQL users by the universal user
- Specifying an SQL user's authorization identifier
- Overview of access authorization to a SESAM/SQL database

System entry

Users (TIAM or UTM users) require a BS2000 system user ID in order to be able to work in BS2000.
TIAM users are identified in BS2000 by a symbolic processor name and the BS2000 user ID. UTM users are identified in the openUTM system by a symbolic processor name, the name of the UTM application and the KDCSIGN or LSES name.

In order to use SQL to work with a SESAM/SQL database, BS2000 users require an SQL user authorization identifier. The database must contain a system access for that authorization code. A system entry is a system user ID and authorization identifier combination. An SQL user is represented by all system accesses in the database system which contain his or her authorization code.

When a database is moved to a different computer, you must create a system entry for the database on the new computer **before** the database is moved. If this is not done, it is not possible to access the database after the move.

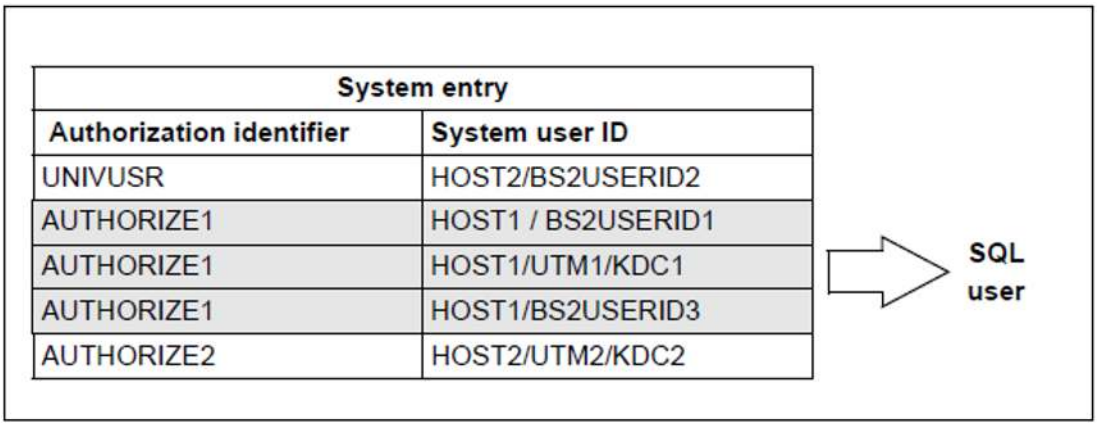

Figure 9: Example of system access by an SQL user

In section “Specifying an SQL user's authorization identifier” there is a description of how an SQL user identifies himself/herself to SESAM/SQL with an authorization code.

SQL users with universal authorization

A prerequisite for the development of a SESAM/SQL database is that the system user ID of an authorized person is specified in the DBH option ADMINISTRATOR. This entry authorizes the relevant BS2000 user to issue the utility statement CREATE CATALOG.

When a database is created with the utility statement CREATE CATALOG, a system entry (among other things) is generated and is stored in the database's catalog space: the database administrator specifies an authorization identifier in the USER clause in CREATE CATALOG and assign the key to a system user ID. Normally, the system user ID and the associated CREATE CATALOG authorization are used (i.e. the system user ID specified in the DBH option ADMINISTRATOR).

The system entry defined with CREATE CATALOG is the database's first system entry and identifies the so-called “universal user”. The universal user is equipped with comprehensive rights that allow him to set up additional users and assign privileges. The universal user can be seen as a central agency that holds all rights, particularly those associated with the SESAM/SQL database, and can assign these rights freely to other users.

The universal user assigns these rights by the following means:

1. Creation of new SQL users (see the section that follows).
2. Assignment of special privileges to new SQL users (see section “Access protection based on privileges in SQL”)

Creation of further SQL users by the universal user

The universal user creates an authorization identifier with the SQL statement CREATE USER. He or she then uses CREATE SYSTEM_USER to create an associated system entry by assigning a BS2000 system user ID to the authorization identifier. The database is aware of users created in this way, but they are not authorized in any way to access objects in the database with SQL. To access database objects, SQL users require the relevant privileges (see "Access protection based on privileges in SQL").

The universal user can pass on the property “universal user” to other users by assigning additional system entries with his or her authorization identifier. From the database system's point of view, all system user IDs with this authorization identifier have equal rights. It goes without saying that this means of assigning universal rights should be used judiciously and with extreme care.

The universal user is also able to assign existing users the right to set up additional users (see section “Special privileges”).

Specifying an SQL user's authorization identifier

SQL users identify themselves to SESAM/SQL by means of the authorization identifier. Users can specify the authorization identifier in the SOURCE-PROPERTIES option of the ESQL precompiler under AUTHORIZATION (see the “ ESQL-COBOL for SESAM/SQL-Server” manual). An authorization identifier can be specified within an ESQL program with the SQL statement SET SESSION AUTHORIZATION. This authorization identifier will only become the current authorization identifier if no authorization identifier is specified in the SOURCE-PROPERTIES option. The configuration file must contain the authorization identifier if the utility monitor is to be used.

Overview of access authorization to a SESAM/SQL database

Authorization	creating with	Parameter	Specification
User authorized for the CREATE CATALOG statement	DBH option	ADMINISTRATOR=...	a new system user ID
the universal user	ESQL program Utility Monitor	CREATE CATALOG USER=... PASSWORD=...	the first system entry
BS2000 password for the database	ESQL program Utility Monitor		BS2000 password
SQL user	ESQL program Utility Monitor	CREATE USER CREATE SYSTEM_USER	a new authorization identifier a new system entry (system user ID, authorization identifier)

Table 41: Creating new SQL users in SESAM/SQL and assigning BS2000 passwords

Authorization	identifying in	Parameter	Specification
Authorization keys for SQL users	Precompiler option	SOURCE-PROPERTIES-AUTHORIZATION=...	Authorization identifier
	Configuration file, or relevant screen in utility monitor	SEE-AUTHID=... parameter for utility monitor	Authorization identifier
	ESQL program utility monitor	SET SESSION AUTHORIZATION	Authorization identifier
BS2000 password for the database	DBH start statement	ADD-SQL-DATABASE-CATALOG-LIST ... PASSWORD=...	BS2000 password

Table 42: Identifying SQL users in SESAM/SQL and specifying the BS2000 password

Access protection based on privileges in SQL

In SQL, access protection is based on the assignment of privileges. An authorized user (known as the “grantor”) authorizes another user (known as the “grantee”) to carry out a certain action on a specific object by granting that user the relevant privileges.

SESAM/SQL distinguishes between special privileges, table privileges and privileges for routines. In the case of special privileges, the object is the database or, if the privilege is USAGE ON STOGROUP, the object is a storage group. In the case of table privileges, the object is a specific table within a specific schema. In the case of privileges for routines, the object is a procedure or a User Defined Function (UDF) within a schema.

A privilege is therefore identified in terms of the grantor, grantee, action and object. Consequently, two otherwise identical privileges that differ only in that they are assigned to the grantee by two different grantors are regarded as different privileges.

Special privileges

The universal user is authorized to grant all special privileges with the exception of USAGE ON STOGROUP. Only the owner of a given storage group is authorized to grant the privilege USAGE ON STOGROUP for that storage group.

the creation and deletion of SQL users (CREATE USER)

The special privilege CREATE USER authorizes the grantee to carry out the following tasks:

- Create a new SQL user
  The first step in creating an SQL user is to assign an authorization identifier with the SQL statement CREATE USER. This authorization identifier then has to be assigned a system user ID with the SQL statement CREATE SYSTEM_USER. In this way, one or more system entries can be created for an SQL user that represent that SQL user.
- Delete an SQL user
  An SQL user is deleted by using DROP USER to delete all system entries that contain the SQL user's authorization identifier.
- Remove a specific system entry for a specific user with the SQL statement DROP SYSTEM_USER.

the creation of a schema (CREATE SCHEMA)

The special privilege CREATE SCHEMA authorizes a user to create a schema. The owner of this schema then possesses all the table privileges for this schema (see section “Table privileges” below). The schema owner is the user who issued the relevant CREATE SCHEMA statement, or the user who is specified in the AUTHORIZATION clause of the CREATE SCHEMA statement.

Execution of the utilities (UTILITY)

The special privilege UTILITY authorizes a user to execute utilities to work on the database.

Specification of storage media for the spaces (CREATE STOGROUP)

The special privilege CREATE STOGROUP authorizes a user to issue the SQL statement CREATE STOGROUP to specify storage media on which user spaces are to be stored. This user is then the owner of the storage group in question and is assigned the special privilege USAGE ON STOGROUP for this storage group.

Use of a storage group (USAGE ON STOGROUP)

The owner of a storage group has the special privilege USAGE ON STOGROUP for that storage group. The special privilege USAGE ON STOGROUP entitles users to create space on a specific storage group with CREATE SPACE. If users wish to alter a user space with ALTER SPACE and the USING STOGROUP clause is specified in ALTER SPACE, they must have the special privilege USAGE ON STOGROUP.

all special privileges (ALL SPECIAL PRIVILEGES)

ALL SPECIAL PRIVILEGES gives the grantee all special privileges that the grantor is entitled to grant for a specific database or storage group with the exception of USAGE-STOGROUP.

Table privileges

Users who create a schema with CREATE SCHEMA (without an AUTHORIZATION clause), or users specified in the AUTHORIZATION clause, own the schema in question. As the owner of a schema, users have all the table privileges for objects in that schema. In addition, they are allowed to grant table privileges for the schema to other users.

Selection of rows (SELECT)

The table privilege SELECT for a specific tables entitles users to select rows in that table and to define a view of the table with CREATE VIEW. The table privilege SELECT remains valid if columns are added after the privilege was granted.

Insertion of rows (INSERT)

The table privilege INSERT for a specific table entitles users to insert rows in that table. Before rows can be inserted, the user must have the table privilege SELECT on the tables named in the query expression of the INSERT or MERGE statement. The table privilege INSERT remains valid if columns are added after the privilege was granted.

Deletion of rows (DELETE)

The table privilege DELETE for a specific table entitles users to delete rows in that table. Before rows can be deleted, the user must have the table privilege SELECT on the tables named in the query expression of the DELETE statement. The table privilege DELETE remains valid if columns are added after the privilege was granted.

Updating of values in rows (UPDATE)

The table privilege UPDATE for a specific table entitles users to update the values in specific columns in the tables rows. Before rows can be updated, the user must have the table privilege SELECT on the tables named in the query expression of the UPDATE or MERGE statement. If no columns are specified in the table privilege UPDATE, the privilege applies to the whole of the table, including any columns added to it after the privilege was granted.

Specification of columns in integrity constraints (REFERENCES)

The table privilege REFERENCES for selected columns in a specific table entitles users to use these columns when defining integrity constraints. If no columns are specified in the table privilege, it applies to the whole of the table, including all the columns added after the privilege was granted.

All privileges (ALL PRIVILEGES)

ALL PRIVILEGES for a specific table gives grantees all the privileges that the grantor is entitled to grant for that table.

Privileges for routines

The authorization identifier which wishes to execute a routine (procedure or a User Defined Function (UDF)) requires the EXECUTE-privilege for this routine.

To execute a routine it is not necessary to have the various table or column privileges which are required to execute the DML statements contained in the routine.

The authorization identifier which generates the routine automatically receives the EXECUTE privilege for this routine.

The authorization identifier must have the EXECUTE privilege for the routines called directly in the routine. It must also, for all tables and columns which are addressed in the routine, have the privileges which are required to execute the DML statements contained in the routine.

If it even has authorization to pass on the relevant privileges, it may also pass on and subsequently revoke the EXECUTE privilege to or from other authorization identifiers.

Granting privileges with the SQL statement GRANT

The following users have specific privileges:

- The universal user has all special privileges for a database, except for USAGE ON STOGROUP.
- The owner of a storage group has the special privilege USAGE ON STOGROUP for that storage group.
- The owner of a schema has all table privileges for the base table in his or her schema.
- A view owner's access rights to his or her view are described on "Granting privileges with the SQL statement GRANT".

The universal user, all schema owners and the owners of storage groups and views who were granted their privileges with “WITH GRANT OPTION”, can grant their respective privileges to other users with the SQL statement GRANT.

The SQL statement GRANT can be represented schematically as follows:

```
GRANT privilege(s) ON object TO grantee(s) [WITH GRANT OPTION]
```

GRANT can be specified as an independent SQL statement or as part of the CREATE SCHEMA statement.

The “grantor” is the user who grants the privilege with GRANT; the “grantee” is the user to whom this privilege is granted. Depending on the privilege, the “object” can be a table, a database, or a storage group.
Users can only pass on a privilege they have been granted with GRANT to other users if the privilege was granted “WITH GRANT OPTION”. In other words, grantees can act in turn as grantors and grant the same privilege with or without “WITH GRANT OPTION”. This feature can be used to construct a hierarchically organized access protection scheme.

A grantor can specify either the authorization identifiers of selected users or the keyword PUBLIC for the grantee in the GRANT statement. PUBLIC grants the privilege to all users entitled to work with the database.

If a grantor assigns a grantee the same privilege twice – once with and once without the “WITH GRANT OPTION” clause – then “WITH GRANT OPTION” always applies.

A grantor can also use the GRANT statement to grant a table privilege to the owner of a different schema. This allows schema owners, for example, to define views of their schemas that are based on tables that have a different owner.

Special privileges and table privileges should be assigned on the basis of a well-planned data-protection scheme. Special privileges in particular should only be granted to a select group of individuals, and the “WITH GRANT OPTION” should be used with extreme caution.

Special aspects of granting privileges for the creation of views

Users only receive the right to create a view with CREATE VIEW via the table privilege SELECT for the table on which the view is based. Here, there are two specific cases to differentiate between:

- The view is updatable.
  In this case the view is based on just **one** base table. If the view's owner has the table privileges INSERT, UPDATE or DELETE for the underlying base table, he or she automatically also receives the relevant table privilege for the view.
- The view is not updatable.
  The owner automatically receives the table privilege SELECT for the view.

One can imagine the automatic granting of privileges for a view as follows: SESAM/SQL grants the relevant table privilege for the view with an implicit GRANT statement.

The owner of the view can only pass on a table privilege for the view with GRANT if he or she himself has the “WITH GRANT OPTION” for the base table on which the view is based.

Example

The example that follows outlines the granting of privileges to the owner of a view and the passing on of privileges to another user.

User A owns the base table T and therefore has the table privileges SELECT, INSERT, DELETE, UPDATE and REFERENCES. A is entitled to pass theses privileges on to other users.

A passes the table privileges DELETE “WITH GRANT OPTION” and SELECT to user B.

```
GRANT DELETE ON T TO B WITH GRANT OPTION

GRANT SELECT ON T TO B
```

Because user B has SELECT authorization for base table T, he or she can create a view of T:

```
CREATE VIEW V AS SELECT * FROM T
```

B thus automatically, i.e. from SESAM/SQL, obtains the table privileges granted for table T (SELECT and DELETE "WITH GRANT OPTION") for view V, too. The following table privileges for T and V therefore exist for B:

Grantor	Grantee	Object	Table privilege	WITH GRANT OPTION
A	B	T	DELETE	yes
A	B	T	SELECT	no
SESAM/SQL	B	V	DELETE	yes
SESAM/SQL	B	V	SELECT	no

Table 43: Table privileges for base table T and view V

A then grants the following table privileges with GRANT:

```
GRANT UPDATE ON T TO B,C

GRANT INSERT ON T TO B WITH GRANT OPTION
```

B grants the following table privilege:

```
GRANT DELETE ON V TO D WITH GRANT OPTION
```

This means that the following new table privileges have been granted:

Grantor	Grantee	Object	Table privilege	WITH GRANT OPTION
A	B	T	UPDATE	no
A	C	T	UPDATE	no
SESAM/SQL	B	V	UPDATE	no
A	B	T	INSERT	yes
SESAM/SQL	B	V	INSERT	yes
B	D	V	DELETE	yes

Table 44: New table privileges for the base table T and view V

Revoking privileges with the SQL statement REVOKE

With the SQL statement REVOKE, users can revoke privileges they have granted. In order to guarantee consistent access protection, SQL requires that users comply with certain rules when revoking privileges.

The SQL statement REVOKE can be represented schematically as follows:

```
REVOKE privilege(s) ON object FROM grantee(s) { CASCADE | RESTRICT }
```

Users use REVOKE to revoke the specified privileges previously granted to “grantees”. Depending on the privilege, the “object” can be a table, a database, or a storage group. A privilege may only be revoked from a grantee by one user, namely the user who originally granted the privilege.

REVOKE ... FROM PUBLIC allows users to revoke a specific privilege from all other users, provided they granted the privilege with GRANT... TO PUBLIC in the first place.

REVOKE ... RESTRICT only allows a user to revoke the specified privileges from a grantee if the grantee has not in turn granted the privileges to other users or if these other users no longer possess the privileges. If privileges have been passed on in this way, the privileges specified with REVOKE are not revoked and an error message is issued.

REVOKE ... CASCADE on the other hand, allows a user to revoke the specified privileges from the grantees in all cases. In this case, all the those privileges are revoked which were passed on to other users by the grantees on the basis of the specified privileges.

The user can use the tables of the INFORMATION_SCHEMA to determine the sequence in which the privileges have to be revoked with REVOKE ... RESTRICT. The INFORMATION_SCHEMA describes which privileges are assigned to which authorization identifiers in the tables TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES and CATALOG_PRIVILEGES.

Since a **single** REVOKE ... CASCADE statement can, under certain circumstances revoke a large number of privileges and delete views (see "Revoking privileges with the SQL statement REVOKE") and referential constraints (see "Revoking privileges with the SQL statement REVOKE"), it is recommended that you obtain information on the existing privileges in the tables of the INFORMATION_SCHEMA before issuing a REVOKE ... CASCADE statement.

As already mentioned, two ostensibly identical privileges granted to the grantee C by two different grantors, A and B, are regarded as different privileges.
A, for example, has passed the same privilege to C and with “WITH GRANT OPTION” to B. B, on the other hand, has in turn passed this privilege to C. If A now revokes this privilege from grantee C with `REVOKE` *Privilege* `ON` *Object* `FROM C RESTRICT`, C still has the privilege granted by B. Only if B revokes the privilege from the grantee C, is the privilege actually revoked from C.

If instead of this A now also revokes the privilege from grantee B with `REVOKE` *Privilege* `ON` *Object* `FROM B CASCADE`, B and C no longer have the privilege.

The revoking of privileges with the aid of REVOKE is demonstrated below in a simple example.

A privilege is generally identified by the following:
(grantor, grantee, action, object, [WITH GRANT OPTION]).

The following initial situation exists:

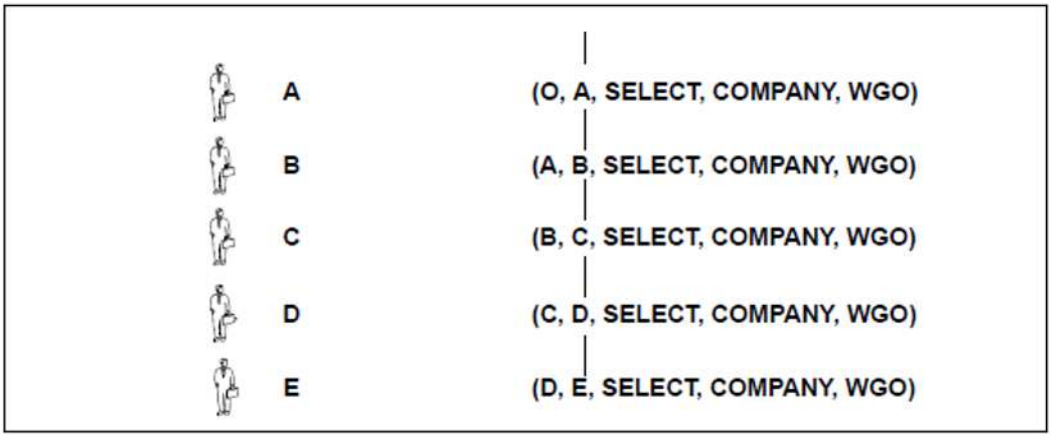

Figure 10: The passing on of a privilege

O, the owner of the schema, has granted user A the table privilege SELECT for the table COMPANY with the clause “WITH GRANT OPTION” (abbreviated to WGO in the example). This privilege is described as (O, A, SELECT, COMPANY, WGO), see figure 10:

User A has passed on this privilege to user B and has included the “WITH GRANT OPTION”. B has passed on the privilege to C, C has passed it on to D, and, finally, D to E. B, C and D have included the privilege “WITH GRANT OPTION”.
The following privileges are currently assigned:
(O, A, SELECT, COMPANY, WGO)
(A, B, SELECT, COMPANY, WGO)
(B, C, SELECT, COMPANY, WGO)
(C, D, SELECT, COMPANY, WGO)
(D, E, SELECT, COMPANY, WGO)

In figure 10, the fact that “User B has been granted the privilege by user A” is apparent from B's position directly under A. A line between related privileges indicates that A has not yet revoked the privilege from B.

D revokes the privilege from E with the statement:

```
REVOKE SELECT ON company FROM E RESTRICT
```

Once the REVOKE has been carried out, E no longer has the privilege (D, E, SELECT, COMPANY, WGO), see figure 11.

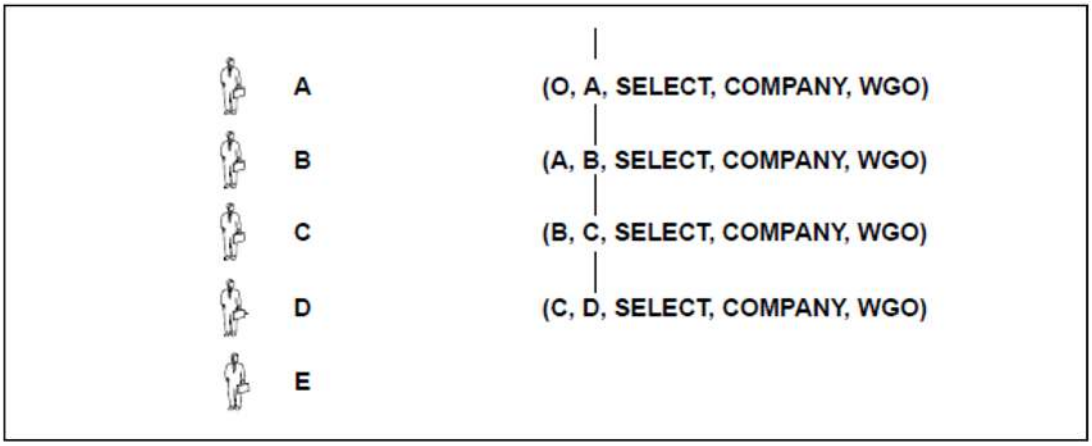

Figure 11: The revoke of the privilege (D,E,SELECT,COMPANY,WGO)

In the same way, C could now revoke the privilege from D, B from C, etc. For D to revoke the privilege from E, other users must not have been granted the same privilege by E and must not still be in possession of it. This is expressed in figure 10 and figure 11 as follows: No lines connect E or D with users below E or D.

Avoidance of abandoned privileges

In the situation shown in figure 11, A wishes to revoke the privilege from B. However, the system rejects the following REVOKE, because it is invalid:

```
REVOKE SELECT ON company FROM B RESTRICT
```

If the REVOKE had been executed, it would have resulted in the following situation:

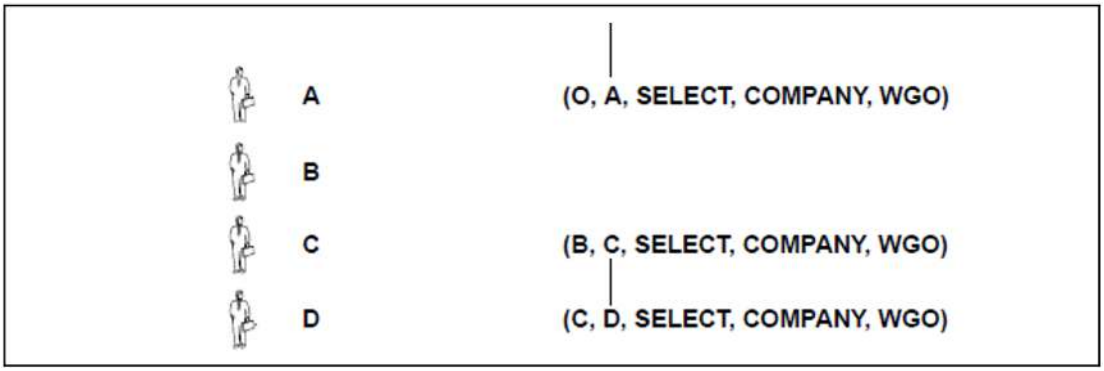

Figure 12: Hypothetical situation: abandoned privileges

C still has the privilege granted by B (B, C, SELECT, COMPANY, WGO). B, however, no longer has the privilege granted by A (A, B, SELECT, COMPANY, WGO); this means that the privilege (B, C, SELECT, COMPANY, WGO) in figure 12 is “abandoned”. As a result, the privilege (C, D, SELECT, COMPANY, WGO) is likewise abandoned because the privilege (B, C, SELECT, COMPANY, WGO) is abandoned.

The abandoned privileges are indicated as follows in figure 12:
No connecting line exists any more from such a privilege upward to a privilege, or no connecting line exists upward to an abandoned privilege.

Generally, a privilege which has been granted and still exists is referred to as “abandoned” if the privilege still exists, but the grantor no longer possesses the privilege. Every privilege derived from an abandoned privilege is also referred to as “abandoned”. The term “abandoned privilege” refers to a purely hypothetical situation, since SESAM/SQL prevents them from occurring in the first place when REVOKE is used.

If REVOKE ... RESTRICT is used, SESAM/SQL prevents abandoned privileges by rejecting a REVOKE ... RESTRICT statement if execution of the statement would lead to an abandoned privilege. Before issuing a REVOKE ... RESTRICT statement, the user should therefore check whether revocation of a given privilege would lead to abandoned privileges and would therefore be rejected. Privileges which would be abandoned must be revoked with REVOKE ... RESTRICT by the user who granted the privilege.

Execution of a REVOKE ... CASCADE, on the other hand, can never lead to a situation in which abandoned privileges can occur. Consequently a REVOKE ... CASCADE is also never rejected for this reason. If, for example, in the situation illustrated in figure 10 grantor

A revokes the SELECT privilege from grantee B with `REVOKE SELECT ON company FROM B CASCADE`, all privileges which were granted by other grantees on the basis of the "WITH GRANT OPTION" privilege granted to B are then automatically revoked in a kind of "chain reaction": Initially grantor D revokes the privilege (D, E, SELECT, COMPANY, WGO) from grantee E, then grantor C revokes the privilege (C, D, SELECT, COMPANY, WGO) from grantee D, etc. This ensures that abandoned privileges can never occur.

The two examples that follow depict situations that would or would not lead to abandoned privileges.

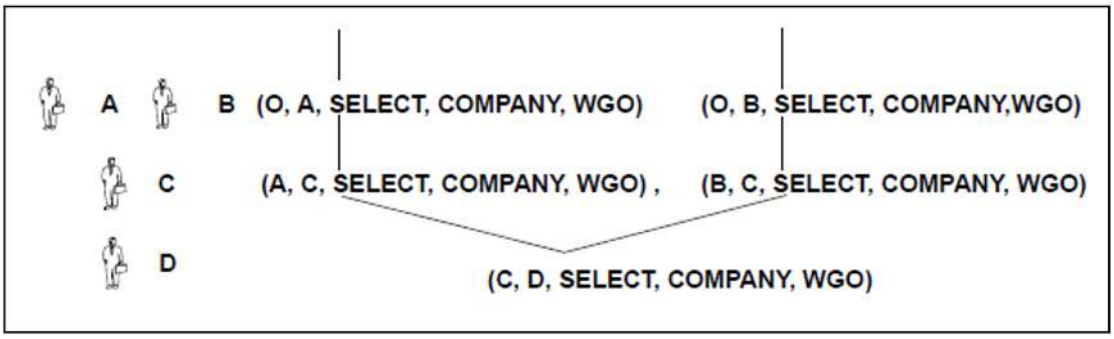

Figure 13: Privilege granted by two users (C, D, SELECT, COMPANY, WGO)

The figure 13 depicts the following initial situation: C has been granted the same privilege by two different users (grantor A and grantor B). C, in turn, has passed on this privilege to D. A would now like to revoke the privilege granted to user C (A, C, SELECT, COMPANY, WGO).

To do this, A issues the statement: `REVOKE SELECT ON company FROM C RESTRICT`

The REVOKE succeeds. The figure 14 shows the situation after the REVOKE

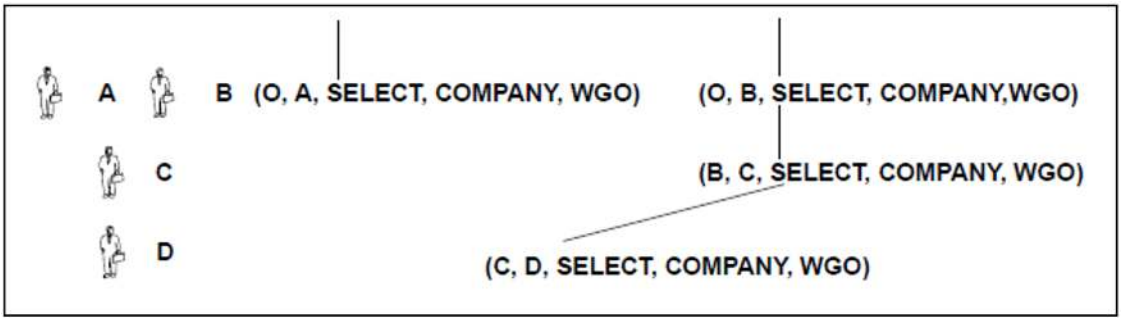

Figure 14: The revoke of the privilege (D,E,SELECT,COMPANY,WGO)

The revoke would not produce an abandoned privilege because user C still has the privilege issued by B (B, C, SELECT, COMPANY, WGO). However, an attempt by B to revoke this privilege with the REVOKE (`REVOKE SELECT ON company FROM C RESTRICT`) would be rejected. If the REVOKE were to succeed, it would lead to an abandoned privilege (C, D, SELECT, COMPANY, WGO).

The next example is based on the following situation (see figure 15): Two users, C1 and C2, have been granted the same privilege by two different users (grantor A in the case of C1 and grantor B in the case of C2). C1 then grants user D the privilege (C1, D, SELECT, COMPANY, WGO), and C2 grants user D the privilege (C2, D, SELECT, COMPANY, WGO).

Figure 15: Privileges (C1, D, SELECT, COMPANY, WGO) and (C2, D, SELECT, COMPANY, WGO)

An attempt by A to revoke the privilege (A, C1, SELECT, COMPANY, WGO) from C1 with `REVOKE SELECT ON company FROM C1 RESTRICT` is rejected because the revoke would cause the privilege (C1, D, SELECT, COMPANY, WGO) granted to D by C1 to be abandoned. The figure 16 shows the hypothetical result if the REVOKE were to be carried out illegally.

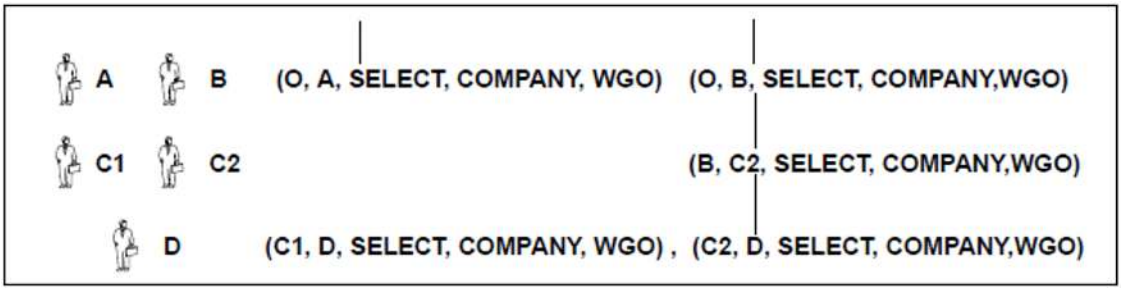

Figure 16: Hypothetical situation: an abandoned privilege (C1,D,SELECT,COMPANY,WGO)

The last two examples (figure 13 and figure 14, and figure 15 and figure 16) underline once again why a REVOKE ... RESTRICT is always rejected if it would lead to an abandoned privilege.

In addition to “abandoned” privileges there are also “abandoned” referential constraints and “abandoned” views.

Avoidance of abandoned referential constraints

The concept of “abandoned” constraints is explained in the following.

A, the owner of base table T, wishes to revoke from user B with REVOKE ... RESTRICT the table privilege REFERENCES originally granted with GRANT, but user B has defined a referential constraint for columns in the table and the constraint has not yet been deleted with ALTER TABLE ... DROP CONSTRAINT. The REVOKE ... RESTRICT statement is rejected.

If the REVOKE ... RESTRICT statement were to be executed, this would leave a referential constraint applied to table T, for which the relevant table privilege REFERENCES no longer exists. As a result, the referential constraint would be abandoned.

On the other hand, an appropriate REVOKE ... CASCADE statement allows owner A to revoke from user B the REFERENCES privilege, even if a referential constraint is defined for the columns of table T. The REVOKE ... CASCADE statement executes an implicit ALTER TABLE...DROP CONSTRAINT statement for this referential constraint.

Avoidance of abandoned views

In addition to abandoned privileges and abandoned referential constraints there is the concept of “abandoned” views. This term will be explained on the basis of the following initial situation:
As grantor, the owner of a schema A assigns the table privilege SELECT for particular tables in a schema to the owner of schema B. The owner of schema B uses CREATE VIEW to create a view V_B on which these tables are based.

The owner of A would now like to revoke the SELECT privilege granted to the owner of B with REVOKE ... RESTRICT. However, the REVOKE ... RESTRICT statement is rejected. If the REVOKE ... RESTRICT were successful, the owner of B, i.e. the owner of V_B, would still be able to access the tables of schema A on which V_B is based via V_B despite the fact that the owner of B no longer has the table privilege SELECT for these tables, consequently may no longer access the table directly, and should therefore not even be permitted to define view V_B.

A view definition would exist, even though the owner of B is no longer entitled to issue the relevant CREATE VIEW statement. The right to the definition of view V_B has been revoked and the view is “abandoned”. Generally, a view is termed abandoned if the view's owner's table privilege SELECT has been revoked for one of the tables on which the view is based. A REVOKE ... RESTRICT statement that would normally produce an abandoned view if executed is always rejected. Users should therefore make sure that all views that required a SELECT privilege in order to be defined have been deleted with the SQL statement DROP VIEW before issuing the REVOKE ... RESTRICT that is to revoke that privilege.

If, in the above example, the owner of A revokes the privilege from the owner of B with a REVOKE ... CASCADE statement, all dependent views are deleted. The REVOKE ... CASCADE statement does not only revoke from the owner of B the SELECT privilege for the tables of his/her schema A, but executes the following statement implicitly:

```
DROP VIEW B.V_BCASCADE
```

This deletes view V_B and all the views which use view V_B in their definition.

Access protection in connection with views

Views provide added convenience to access protection.

The principle on which views are based is that groups of users are only shown the information that they need. In combination with privilege-based access protection as described in section “Access protection based on privileges in SQL”, views make it possible to ensure that individual groups of users only have access to the information that is intended for them. Through views and privileges SESAM/SQL provides flexible and comprehensive means of implementing data protection that meets both user's information requirements and data protection requirements.

For example, it might be desirable to create a view of a table containing data on all employees that only comprises the first name, surname and department columns. All employees can be given the SELECT privilege for this view. By contrast, the SELECT right for a second view that also includes other columns, such as salary, birth date, address, etc., should only be granted to a select circle of persons, such as personnel department staff. Restrictions should be even tighter in connection with the granting of rights to alter individual columns.
A further option would be to use department-specific views to limit to the SELECT privilege granted to all employees so that they can only ever view data pertaining to colleagues within their own departments. In this case one would grant a department-specific privilege for the relevant view.

Password protection with SEPA for CALL DML tables

Users can protect CALL DML tables from unauthorized access by means of passwords. This functionality is provided by the utility SEPA (see the “Database Operation” manual).

When a table is protected by a password, users must enter the relevant password for each CALL DML statement. When SESAM/SQL analyses the statement, it checks the authorization associated with the password.

The first step is to open a table or section of a table (a logical file) by specifying a valid password in the Open statement. If users fail to specify the correct password within the maximum permitted number of attempts when issuing an Open statement interactively, the system locks them out. Only system administrators can cancel the lock.

Once opened, the table or logical file can be edited using CALL DML statements, provided the same password is specified.
The password governs which data can be accessed and how that data can be accessed. If an attempt is made to access an attribute for which rights have not been granted, SESAM/SQL responds as if the attribute does not exist in the database and rejects the statement with a status code.
CALL DML statements issued with an invalid password are likewise rejected with a status code.

SEPA offers the following functions:

- Executes all maintenance activities on the password catalog:
  change, revoke or reassign access rights.
- Outputs information on all passwords (name, access rights).

Password catalogs for password-protected CALL DML tables are not backed up separately.

Data access control by means of data encryption

The cryptographic functions ENCRYPT and DECRYPT of SESAM/SQL enable you to encrypt and decrypt the sensitive data of a database.

Sensitive data is protected against unauthorized access by encryption. Only the users who know the “key” can decrypt the data. The sensitive data of a database which is operated in a insecure environment can also be protected in this way.

SESAM/SQL uses the Advanced Encryption Standard (AES) of Rijndael with a 128-bit (16byte) key in Electronic Codebook Mode (ECM). General information on the AES is available on the internet at: http://csrc.nist.gov/

Detailed information on the cryptographic functions of SESAM/SQL is provided in the “ SQL Reference Manual Part 1: SQL Statements”.

Key management

The security of encrypted data depends mainly on the security of the keys used. Keys must be redundant and stored at different places, if necessary also on different media. Their storage strategy is part of the security policy.

The AES uses the same key for encryption and decryption (symmetric encryption method). If the key is known, sensitive data can be decrypted directly because only the key is secret, but not the encryption method. The key cannot, however, be changed any number of times as all the encrypted data would have to be decrypted and encrypted again using the new key.

When the key is lost, encrypted data can no longer be decrypted. For security reasons, SESAM/SQL leaves no internal trace of the key or of unencrypted values.

For security reasons, it may be necessary to save “old” keys to read encrypted data from backup copies.

Saving encrypted data

Encrypted data is only saved in its encrypted form in SESAM/SQL. This applies not only for the spaces of the database but also for the indexes, the logging files (DA-LOG, TA-LOG, WA-LOG) and the temporary files of the utility functions.

CAUTION!

However, in the main memory of the DBH and of the client applications both the unencrypted data and the key are still (after ENCRYPT or DECRYPT has been used) visible in plain text. Dumps of such areas contain confidential data and must be treated with particular care.

After the DBH and the client applications have terminated, the main memory areas can no longer be accessed. Dumps and cursor files (see "Cursor files for retrieval statements", `SES*.CURSOR.*` file names) should be deleted not just logically but also physically (e.g. using the BS2000 command DELETE-FILE ...,OPTION = *DESTROY-ALL).

In the BS2000 user ID in which the DBH is running it can happen (e.g. because of administration intervention) that files (e.g. CO-LOG file, traces, dumps) are created which may contain keys or the encrypted data in decrypted form. Consequently this BS2000 user ID and also the administration access to the DBH should also be protected.

Other analysis tools can also make keys and data visible in plain text and store them. Particular notice should be taken of these. For analysis purposes you can create a log of the SQL statements and their user variables using the SESCOSP utility routine, see the “ Database Operation” manual.

NULL values

NULL values become NULL values again when they are encrypted.

Privileges and encryption

Privileges permit access to tables, to columns and (via views) to rows of a table, see section “Access protection based on privileges in SQL”. They also define what operations (e.g. INSERT) are permissible on these objects. Privileges have a greater functional scope than encryption, but they do not cover all security aspects.

The encryption of sensitive data complements the privileges, e.g. in the following cases:

- Privileges can permit access to sensitive data.
  Particular roles (e.g. the universal user or the owner of a table) have all the privileges for an object.
  However, encryption only permits such users to decrypt sensitive data if they also know the key.
  ENCRYPT() enables individual columns of a table or even individual values to be protected against unauthorized access.
- Privileges control access in the SQL system only.
  They do not prevent access using other means (e.g. the BS2000 command SHOW-FILE). Encryption, on the other hand, prevents unauthorized reading using any other means.
- Privileges apply for all SQL statements of a transaction.
  When encryption is implemented using different keys, sensitive data with different security requirements can also be encrypted differently, even within one and the same transaction.
- In the case of privileges users identify themselves by the application which was used or by the access to the operating system which was used. SESAM/SQL can, for example, not distinguish between two users with the same access. In the case of encryption they can, however, differentiate because of their knowledge of the keys.

Protection of person-related data by means of anonymization

The legal provisions for data protection stipulate that personal data

- may only be used and processed for the purpose for which it was collected
- must be retained in as small a scope as possible
- may not be stored for longer than is necessary for normal processing

In the case of deviations from these requirements, personal data must be anonymized so that it cannot be traced back to the natural person to which the data originally belonged.

Quote: §3 (6) Data Protection Act of the Federal Republic of Germany:“**Anonymization** is the changing of person-related data in such a manner that the individual details concerning personal or material conditions can no longer be assigned to a particular or ascertainable natural person or only with a disproportionately large investment of time, costs and labor.”

It is common practice to generate test databases from productive databases. With regard to data protection, however, person-related data must be protected when this is done.

With the utility function ALTER DATA FOR TABLE, SESAM/SQL provides a function which supports the anonymization of data and prevents any conclusions about the original context from being drawn. From the viewpoint of data protection, continued use may then also be made of this anonymized test data.

The anonymization of the data is not logged. No function is available to undo data anonymization. The algorithm used cannot be ascertained by comparing the data before and after it has been anonymized.

The data itself is not changed in this process; only the assignment of the column values to the individual rows in the table is changed. The value range and also the frequency distribution of the various column values are retained.

The assignment of the column values to the various rows in the tables is implemented differently for each column and each time the function is called. Columns which are connected logically can also be interchanged together.

Example

The column values of the *personal* table are shuffled. The logical connection of the columns Salutation, First Name and Gender, and of the columns City and ZIP Code are retained.

```
ALTER DATA FOR TABLE personal
SHUFFLE VALUES FOR COLUMN (Salutation, First Name, Gender),Last Name,
                          (ZIP Code, City), Street, House Number,
                           Telephone, Date of Birth, Place of Birth
```

Logging security-relevant events with SAT

SESAM/SQL logs security-relevant events using the component SAT (Security Audit Trail) of the software product SECOS (Security Control System).

To do this, SESAM/SQL transfers log records (SATLOG records) to SAT (provided that the SAT logging in SESAM /SQL is enabled). SAT stores these log records into a protected logging file (SATLOG file). The SATLOG file can be analyzed with the help of the SAT evaluation routine SATUT. SATUT creates usefully edited SAT log files and/or result lists.For more information about SAT and the SAT evaluation routine SATUT, refer to the “ Security Control System - Audit” manual.

SESAM/SQL provides the SESAM system administrator with the following options for enabling and disabling SAT logging in SESAM/SQL:

- DBH option SECURITY, refer to the “ Database Operation” manual, chapter “DBH start statements and options”.
- Administration statement SET-SAT-SUPPORT or OPT, SAT, see the “ Database Operation” manual, chapter “DBH and SESDCN administration”.

In order to log SESAM events, both the SAT logging in SESAM/SQL and the SESAM events of the SAT preselection in SAT must be enabled.

If SAT logging is enabled, SESAM/SQL transfers SATLOG records to SAT for the following events:

- Start or end of a SESAM-DBH task or a service task
- End of a process
- Intervention by administration in a DBH session
- Manipulation of the database structure with DDL or SSL
- Execution of utility statements
- Changes in user accesses and access rights

DML accesses are **not** logged with SAT.

The SATLOG records of SESAM/SQL are described in the appendix to the “ Database Operation” manual, in the section “Layout of the log records for SAT”.

Backup concept

The backup concept in SESAM/SQL comprises the following components:

- the transaction concept, which guarantees data consistency during operation, and
- the recovery concept, in which various different measures are used to recover data consistency when errors occur.

This chapter describes

- the transaction concept
- transaction logging as the basis for transaction recovery and system recovery
- the handling of transactions during system recovery
- possible error situations and suitable recovery measures
- media recovery, based on SESAM backup copies and logging
- the use of replications
- backup-specific files.

Transaction concept

A transaction is a sequence of related statements which transfer a database from one consistent state to another consistent state.

Transactions are either executed completely or not at all.

Example

One thousand euros are to be transferred from account 1789 to account 1564.

This requires the following steps:

- Determine the current balance in account 1789.
- Check whether the balance in account 1789 is greater than 1,000 euros.
- Adjust the balance in accounts 1789 and 1564.

The transaction has the following structure:

```
Start of transaction               --
   Search account 1789              |
   Check whether balance > 1000     |
   ...                               > Transaction bracket
   Adjust balance in account 1789   |
   Adjust balance in account 1564   |
End of the transaction             --
```

Transaction in application programs

Transactions are opened and closed in different ways:

SQL transaction:

> Transaction begins when the program starts or when the preceding transaction ends; it begins with the SQL statement that initiates the transaction. The statement that commits a transaction is COMMIT WORK; the statement that rolls back a transaction is ROLLBACK WORK.

CALL DML transaction:

> You define the beginning and the end of a transaction in your program with CALL DML statements. The transaction begins with the “begin transaction” statement (BTA) and ends with the “end transaction” statement (ETA) or when DBH performs an internal rollback of the transaction.

SQL and CALL DML transactions may not be combined within a single transaction.

However, in a CALL DML transaction certain SQL statements may be issued. Exceptions are the COMMIT WORK and ROLLBACK WORK statements as well as the SQL statements used for queries and modifications which must also not be used in an SQL transaction (see the “SQL Reference Manual Part 1: SQL Statements”).

Applications with linked-in DBH may contain only CALL DML or SQL transactions.

A transaction may contain statements that pertain to one or more tables.

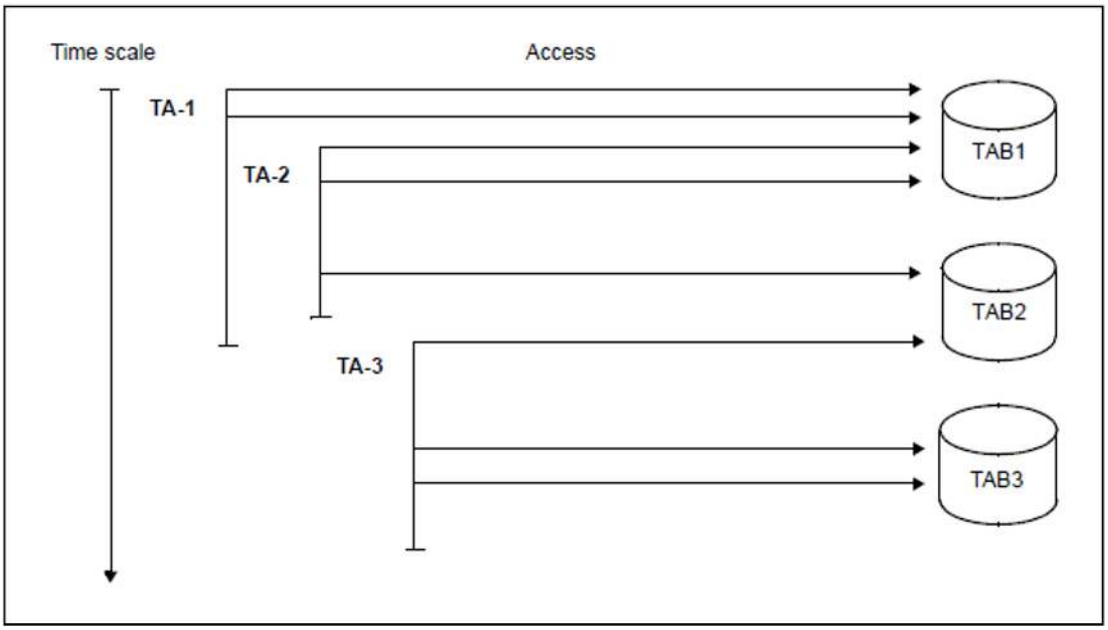

Figure 17: A number of different transactions concurrently access several different tables

SESAM/SQL allows a maximum of 32,767 transactions to concurrently access between 1 and 254 databases. CALL DML allows up to 254 tables (see OLD-TABLE-CATALOG in the “Database Operation” manual).

Locking in SESAM/SQL transactions

Locking ensures the consistency of processed data.

CALL DML	A transaction exclusively reads all the records if the associated logical file was opened with open mode “X” and the additional parameter “readnolock” has not been specified. Shared reading is only possible if the associated logical file was opened with open mode “R”.
SQL	Query operations are conducted in shared mode; update operations are conducted in exclusive mode.

This excludes the possibility of corruption of the data.

Modifications in connection with CALL DML

SESAM/SQL allows the following modifications to locking in connection with data retrieval in CALL DML (see the “CALL-DM Applications” manual):

- Reads are carried out without locking (this can lead to a non-repeatable read).
- The exclusive lock is ignored (this can lead to a dirty read).

Modifications in connection with SQL

The locking method and modifications to it are mapped to the isolation level in SQL. SQL users explicitly or implicitly specify an isolation level for each transaction.

- implicitly at conversion time or by means of the configuration file
- explicitly by means of the SET TRANSACTION statement or the pragma ISOLATION LEVEL.

This also determines the degree of transaction concurrency (see section “SQL transaction”).

Wait states in connection with transactions

If a transaction attempts to access a row that has been locked by another transaction, the transaction cannot execute the statement: The transaction is placed in a wait state. At the same time, the transaction that caused the row to be locked is registered in the table of active transactions.

Table of active transactions	
Transaction	is locked by transaction
A	
B	A
C	B
D	A

Table 45: Transactions in a wait state

The waiting transaction can only continue with processing once the transaction that locked the row has been committed or rolled back. The SESAM/SQL DBH monitors waiting transactions for this reason and reactivates them once the transaction that locked the row has completed.

The lower-level DBH-Option TRANSACTION-SECURITY can be used to modify the behavior of the DBH when dealing with locking transactions and the escalation of transaction locks. During normal operation it is possible to modify these parameters with the administration command MODIFY-TRANSACTION-SECURITY (see the “Database Operation” manual).

Two situations can arise in which automatic reactivation is impossible or is deferred for an unnecessarily long period: deadlock and longlock.

Deadlocks

A deadlock occurs when two or more transactions lock each other. If this occurs, neither transaction can be completed.

> The administration statement SET-SESSION-DIAGNOSIS enables you to have additional information about the users and the objects concerned output to SYSLST when a deadlock occurs, see the “Database Operation” manual.

The following table shows three deadlocked transactions. The number of I/O operations required by different types of access varies: reading a data block involves a single I/O operation; writing a data block involves two I/O operations.

Table of active transactions		
Transaction	is locked by transaction	I/O operations
A	C	100
B	A	20
C	B	60
D		5

Table 46: Three deadlocked transactions

The SESAM/SQL DBH automatically clears deadlocks.

Whenever a transaction is locked, the SESAM/SQL DBH checks whether the lock causes a deadlock.

If it does, the SESAM/SQL DBH rolls back the deadlocked transaction that has carried out the least I/O operations to date (in table 46, transaction B). The unlocked transactions can continue processing. In table 47, this applies to transaction C.

Once the deadlock has been cleared, the situation is as follows:

Table of active transactions		
Transaction	is locked by transaction	I/O operations
A	C	100
C		60
D		5

Table 47: Transaction table after the deadlock is cleared

The transaction that has carried out the smallest numbers of I/O operations is always selected as the transaction to be rolled back.

Longlocks

The SESAM/SQL DBH detects transactions that have suspended processing for long periods of time and are locking resources required by other transactions. The SESAM/SQL DBH automatically rolls back transactions of this kind. System administrators specify a rollback criterion that defines how long inactive transactions are allowed to lock other transactions (the LOCK-TIME parameter in the TRANSACTION-SECURITY DBH option or the MODIFY-TRANSACTION-SECURITY administration command, see the “Database Operation” manual).

Transaction logging

Transaction logging includes mechanisms for automatic error handling, and restores data consistency via a recovery mechanism if an error occurs. This kind of operation is referred to as transaction recovery. In addition, transaction management provides the basis for the recovery measures used by external and internal restarts. Transaction logging guarantees the physical and logical consistency of the database.

Physical consistency

Transaction logging can guarantee physical consistency since the SESAM/SQL-DBH buffers the physical database block before the first change to the physical before image (PBI) and backs this image up if the data block is prematurely displaced into the restart log file (WA-LOG file). If the data block is not prematurely displaced prior to the end of the transaction then no PBI is written.
Saving the after image (AI), that results from any changes also helps guarantee physical consistency.

Logical consistency

Logical consistency is ensured by writing the parts of a row that are to be changed to the transaction log file as a logical before image (LBI).

Consistency point

The DBH adds a consistency point to the current transaction log file for each statement that causes the end of a transaction. If a number of concurrent transactions complete within a short time of one another, a group commit is written for them.

Permanence of the effects of a transaction

At the end of a transaction, the physical and logical before image blocks of the closing transaction are released simultaneously. The transaction can no longer be rolled back and is therefore permanent. The databases manipulated by the transaction are in a logically consistent state with regard to the transaction, as are the relevant after images.

transaction log files

Transaction logging (TA logging) is fundamental to a session-oriented recovery procedure.

The SESAM/SQL DBH creates the relevant transaction log files for use in operation with transaction logging enabled:

TA-LOG files

Two TA-LOG files are used alternately. Information is stored in these files that is used to protect the DBH session. The TA-LOG files are uniquely assigned to each SESAM/SQL DBH. Among other things, the TA-LOG files contains the following:

- the logical before images
- the physical after images
- information on the consistency points that are used to perform an automatic restart after a system failure.

WA-LOG file

Specific information that ensures that the DBH session can be controlled and is protected and that the DBH can restart is stored in this file (see section “DBH-specific files”). Each SESAM/SQL DBH has its own WA-LOG file. Among other things, the WA-LOG file contains the following:

- the DBH options which the DBH uses for the restart
- list of spaces involved in the restart
- the physical before images
- information that describes the progress of the restart process
- information on synchronization with openUTM

DDL-TA-LOG file

In SQL applications, this file is assigned to the relevant space. It serves to protect longrunning DDL and SSL statements that execute in the service task and takes some of the load off the TA-LOG files.

After the statement has terminated successfully, it is deleted again if it was not created by the user.

SESAM/SQL always creates the DDL-TA-LOG file on the DBH user ID even if the database resides on a DB user ID. The copies are created on the storage medium (disk) and the primary and secondary assignments are made in accordance with the information in the DDL-TA-LOG media records in the media table.

The following default values apply for primary and secondary assignment of the DDL-TA-LOG file: 1536 (primary assignment) and 384 (secondary assignment).

Restart

SESAM/SQL carries out a restart when operations are resumed after errors have occurred.

During a restart, the DBH rolls back the transactions that were still active at the time the error occurred to the last valid consistency point. This returns all the databases involved to a consistent state. For the restart to succeed, the DBH has to have been logging transactions and the transaction log files TA-LOG1, TA-LOG2 and WA-LOG must be available for the time of the crash. If one of these files is not available or if the files were written at different times, no restart is possible. Depending on whether the current DBH session is interrupted or not, the DBH carries out an external or internal restart.

- External restart:
  The DBH carries out this type of restart after a system failure if it is loaded after an abort session. If the system fails, transaction logging allows the SESAM/SQL DBH to restart automatically. The databases are treated as follows:
  - The status of the last consistency point is recovered on the basis of the physical before image blocks. Any database write jobs that had not been executed at this point are now implemented on the basis of the after images (physical repair)
  - Following this, all transactions that were active at the time of the last consistency point are rolled back on the basis of the logical before image (logical repair).
  - After the restart, the databases remain open and can once again be used by application programs.
- Internal restart:
  Restart after a minor internal error. The DBH carries out an internal restart without interrupting system operation. In the case of an internal restart, physical and logical repair is performed once the internal memory areas have been reinitialized. The session then continues.

The time taken by the restart depends primarily on the period required for physical and logical repair. The following control options are available to reduce the time up to renewed DBH availability after the restart:

- You can influence the frequency with which the after image blocks are written to the database during normal database operation.
- You can delay logical reset until normal operation recommences after a restart.

The more frequently modified blocks are written to the database during normal operation, the less often this operation has to be performed during physical repair on the basis of the after images following a restart. This reduces the time required before normal operation can be resumed after an interruption. However, excessively frequent database writes during normal operation can have a negative impact on performance. You should therefore monitor the I/O rates during normal operation and adapt the settings during normal operation if necessary (see the “Database Operation” manual).

If you delay the logical reset until the start of normal operation then the reset operations are performed in parallel with the normal user transactions or jobs and are synchronized as in normal operation. For a detailed description of this operation, refer to the “Database Operation” manual.

Potential error situations and appropriate recovery measures

The following error situations can arise when working with a SESAM/SQL database:

- computer failure (e.g. due to power outage)
- device failure (e.g. a hard-disk failure due to a defective read/write head)
- operating system software error that does not corrupt data
- operating system software error that has falsified the data
- database system software error that does not corrupt data
- database system software error that has falsified the data
- errors in the application software that cause incorrect or incomplete data to be stored in the database

SESAM/SQL takes care of error situations caused by computer failure, and error situations resulting from operating system software errors through its transaction logging features and the SESAM/SQL DBH's ability to perform a restart; these mechanisms work well, provided data has not been falsified and database system files have not been destroyed (see section “Transaction logging” and section “Restart”).

SESAM/SQL attempts to limit error situations resulting from a defective hard disk to the spaces located on the disk in question. In addition, the BS2000 function DRV (Dual Recording by Volume) makes it possible to handle error situations attributable to hard-disk errors and to render them invisible to SESAM/SQL.
When a new hard disk is added as a mirror disk, the data must be reconciled (i.e. rendered consistent) using means provided by BS2000. By contrast, after a system failure, areas of the database which have a differing status must not be reconciled by the database administrator, because SESAM/SQL's restart capabilities take care of this automatically.

Generally, all failure and error situations can be taken care of in principle with the aid of media recovery (see section “Media recovery”). However, the SESAM/SQL DBH's restart capabilities and BS2000's DRV function render the application available again much more quickly in such error situations.

In the following error situations, media recovery is the only means of restoring a defective SESAM/SQL database:

- operating system software error that has falsified the data
- database system software error that has falsified the data
- errors in the application software that lead to incorrect or incomplete data in the database
- destruction of database system files due to hardware faults

Media recovery

With the aid of media recovery, the database administrator can repair a damaged SESAM/SQL database caused by hardware failures where other recovery measures would fail.

Media recovery is based on the following concept:

- The database administrator creates SESAM backup copies at carefully selected points in time. He/she can create the backup copies on disk or on magnetic tape cartridge using ARCHIVE or HSMS (utility statement COPY).

  SESAM/SQL does not just allow the whole of the database to be backed up - it also allows smaller units to be backed up. The smallest backup unit supported is a single user space. The time the backup is performed and the choice of appropriate backup units have a considerable effect on the time required by any subsequent recovery measures.
- SESAM/SQL logs all changes made to the database after the SESAM backup in log files (CAT logging, DA logging).
- The repair of the database or a user space is achieved by resetting to an appropriate SESAM backup copy and then applying the modifications to this copy that have been logged in the log files (CAT-LOG files and DA-LOG files) since.

BS2000 user IDs in the case of media recovery

SESAM backup copies and logging files may either be stored on the DBH user ID or on a DB user ID.

SESAM/SQL always tries to first create the SESAM backup copies or logging files on the DB user ID. However, this is only possible if the DBH user ID is defined as co-owner of the DB files on the DB user ID or if the database administrator has already created the corresponding files using CREATE-FILE on the BS2000 user ID, see section “Database files and job variables on foreign user IDs”.

It must be possible to create all the files necessary for a backup in one user ID, the DB user ID or the DBH user ID.

RECOVER and REFRESH also assume that the files are stored on the DB user ID. Only if SESAM/SQL does not find copies there, does SESAM/SQL search on the DBH user ID. The same procedure applies for the logging files.

If backup files are present both on the DB and DBH user IDs, the files on the DB user ID are used.

Files and tables used for media recovery

SESAM/SQL uses the following database-specific files together with tables stored in the catalog space (catalog tables) for media recovery:

- PBI file (in the case of disk copy and online backup using ARCHIVE) or an HSMS work file (in the case of online backup using HSMS)
- CAT-REC file
- CAT-LOG files
- DA-LOG files
- RECOVERY_UNITS catalog table
- DA_LOGS catalog table

The database administrator specifies the file attributes and the storage media for the unavailable database-specific files in the media table.

Media table

The media table is located in the catalog space and contains entries for each databasespecific file type, including a description of the file attributes (the size of the primary and secondary assignments, shareability) and information on the storage media on which the files belonging to the file type in question are to be created. Each entry of this kind is known as a media record.

Edit media table

The first media record for the CAT-REC file and the first media record for the CAT-LOG files are added to the media table with the utility statement CREATE CATALOG when the catalog space is created (see section “Creating the database's catalog space”). The database administrator adds the first media record for DA-LOG files and the first media record for the PBI file or the HSMS work file to the media table with the utility statement CREATE MEDIA DESCRIPTION.

Using the utility statement ALTER MEDIA DESCRIPTION, the database administrator can update the media table by adding or deleting individual media records. In addition, they can use this statement to modify the description of the file attributes. All the media records for a specific file type can be deleted using the utility statement DROP MEDIA DESCRIPTION.

Analysis of the media table by SESAM/SQL

Before SESAM/SQL creates a file it first searches for this file. In the case of logging files, it observes the appropriate rules (see section “BS2000 user IDs in the case of media recovery”). If a database-specific file is to be created and the file is not found, SESAM/SQL analyzes the media table and creates the file in accordance with the information contained in the first media record for the relevant file type. If the media table does not contain an entry for the file type in question, SESAM/SQL always first tries to create the file on the catalog ID which is assigned to the DB user ID, see section “Database files and job variables on foreign user IDs”.
Otherwise, SESAM/SQL creates the file on the DB user ID. Details of the actual defaults that SESAM/SQL uses are provided in the description of the database-specific files below.

SESAM/SQL continues to create files on the storage medium specified in the first media record for the relevant file type until there is no more space on the device. Only then does SESAM/SQL analyze the second media record for this file type and create the next file of this type on the storage medium specified in the second media record. SESAM/SQL continues in this way until it has worked its way through the media table, in other words until it has reached the final media record for the relevant file type. Using one of the clauses of the utility statement CREATE MEDIA DESCRIPTION, the database administrator can specify whether or not SESAM/SQL should request additional information on storage media for the relevant file type at the console once it has worked its way through the table.

For the PBI file or the HSMS work file, the media table is only analyzed with COPY. If RECOVER is issued for an online backup which was created using ARCHIVE, the PBI file is stored on the default pubset of the DBH user ID. If a DMS error occurs when processing the PBI file, for example, because the file cannot be extended, the COPY statement is aborted.

PBI file

SESAM/SQL needs a PBI file in order to be able to ensure that SESAM backup copies (in the case of disk copy and online backup using ARCHIVE) have a consistent status when created online with the utility statement COPY).
In the case of online backup using HSMS, an HSMS work file is used.

The contents of PBI file or the HSMS work file

In the PBI file, SESAM/SQL logs the physical before images (PBIs) of blocks modified by concurrent SQL statements and/or CALL DML statements during the copy operation.

Once a SESAM backup copy has been created online on disk, SESAM/SQL immediately adds the logged PBIs to the SESAM backup copy.

If backup is performed on magnetic tape cartridge using HSMS, HSMS is responsible for administering the before images. The HSMS work file contains the before images of blocks that are changed in the space to be backed up. HSMS administers the blocks changed over the course of the backup and reads the block to be backed up from the HSMS work file or from the space. Once the backup is complete, HSMS deletes the work file.

In the case of a SESAM backup copy on magnetic tape cartridge using ARCHIVE, SESAM/SQL also backs up the PBI file on magnetic tape cartridge. In this case the file is only read in at the time of repairing or resetting.

Storage medium, primary and secondary assignments

SESAM/SQL creates the PBI file or the HSMS work file as a temporary file that only exists for as long as it takes to create the SESAM backup copies online. The copies are created on the storage medium (disk) and the primary and secondary assignments are made in accordance with the information in the PBI media records in the media table.

The following default values apply for primary and secondary assignment of the PBI file or HSMS work file: 576 (primary assignment) and 72 (secondary assignment).

BS2000 file name

The PBI file or the HSMS work file have the following file name in BS2000: *user_id.catalog.time_stamp.01*

user_id	BS2000 user ID
catalog	Name of the database
time_stamp	time when the copy was created
01	counter

CAT-REC file

The CAT-REC file (catalog recovery file) serves as a control file for the recovery of the catalog space and the database as a whole. Different variants of the file exist when you work with logging files:

- as the CAT-REC file (original);
  this is created with CREATE CATALOG or with the first COPY of the database provided that the database was created without logging files.
  This variant describes the current state of the original.
- as the CAT-REC copy;
  this is created with COPY CATALOG[_SPACE] or CHANGE-CATLOG and is a defined restart point for the use of replications. Every COPY or CHANGE-CATLOG overwrites any existing copy.
  Up to this version the modifications can be applied to the replication.
- as the CAT-REC file of the replication;
  this is created with CREATE REPLICATION. The file contains all the records up to and including the backup copy used to create the replication.

  This file is needed in two situations:
  - If a REFRESH REPLICATION is to be performed. The difference in the contents of the CAT-REC copy and the CAT-REC file of the replication forms the content of the REFRESH REPLICATION job.
  - If with RECOVER CATALOG USING/TO REPLICATION an original database is repaired, reset or entered in the SQL database catalog using the replication. The replication of the CAT-REC file then becomes the CAT-REC file of the original.

Contents of the CAT-REC file

The CAT-REC file contains records with the following information:

- Identification block
  This contains the identification record and the database name.
- CREATE CATALOG record
  This contains information on the catalog space from BS2000's point of view, e.g. the primary and secondary assignments, the BS2000 password, and the storage medium.
- Records on the SESAM backup copies of the catalog space
  Each record contains information on a specific SESAM backup copy of the catalog space; this information includes the SESAM backup's time stamp, the SESAM backup copy's version number, the name of the SESAM backup copy and the storage medium for the SESAM backup copy.
- Records on the CAT-LOG files
  Each record contains information on a specific CAT-LOG file; this information includes the creation date, the version number of the associated SESAM backup copy of the catalog space, and a sequence number which indicates the CAT- LOG file's number in the sequence of CAT-LOG files for the SESAM backup copy in question.

Each time the catalog space or the database is backed up, SESAM/SQL adds a record pertaining to the relevant SESAM backup copy of the catalog space to the CAT-REC file.
Each time the CAT-LOG file changes (see section “CAT-LOG files”), SESAM/SQL adds an appropriate record for this CAT-LOG file to the CAT-REC file.

Using the utility monitor, the database administrator can view information on the content of the CAT-REC file. He /she can also maintain the CAT-REC file online or offline (see the “Utility Monitor” manual).

The utility statement der MODIFY enables the database administrator to maintain the CAT-REC file online (see the “SQL Reference Manual Part 2: Utilities”).

Storage medium, primary and secondary assignments

In the case of a database used with logging activated), SESAM/SQL creates the CAT-REC file during the execution of the utility statement CREATE CATALOG. In the case of a database used without logging, SESAM/SQL creates the CAT-REC file when the first SESAM backup copy is made of the entire database or of the catalog space. The file is created on the storage medium (disk) and the primary and secondary assignments are made in accordance with the information contained in the first CAT-REC media record in the media table.

The default value 12 applies for both primary and secondary assignment of the CAT-REC file.

BS2000 file names

The CAT-REC file has the following file name in BS2000:
user_id.catalog.CAT-REC
user_id.catalog.CAT-REC.COPY
user_id.replication.CAT-REC.REPL

user_id	BS2000 user ID
catalog	Name of the database
replication	name of the replication

Backing up the CAT-REC file

The dual recording by volume or dual copy technique is to be used to maintain parallel copies of the CAT-REC file.

A copy of the CAT-REC file is created by COPY CATALOG[_SPACE] or CHANGE-CATLOG. Every COPY or CHANGE-CATLOG overwrites any existing copy. In the event of a RECOVER, the original database can be reset to the status of the CAT-REC copy.

In the case of COPY CATALOG[_SPACE] on magnetic tape cartridge using HSMS, you can specify in the HSMS parameter file whether or not the CAT-REC file should be backed up, too.

CAT-LOG files

CAT-LOG files are log files for the catalog space (see section “Logging”).

Contents of the CAT-LOG file

In CAT-LOG files, SESAM/SQL logs all the changes in the catalog space. Changes in the catalog space are caused by SQL statements used to administer the storage structure, by utility statements, by SQL statements used to administer user entries, and by SQL statements for the definition and administration of schemas. In particular, SESAM/SQL logs all changes to the RECOVERY_UNITS and DA_LOGS catalog tables in CAT-LOG files. Moreover, SESAM/SQL documents each copy made of the database and each copy made of the catalog space in the CAT-LOG file. The creation of a copy of the database or the catalog space automatically triggers a change of CAT-LOG file. The SESAM backup copies serve as resumption points when applying logged modifications during the repair of the database or the catalog space (see section “Recovering the database, catalog space and user spaces”).

Storage medium, primary and secondary assignments

SESAM/SQL creates the first CAT-LOG file during the execution of the utility statement CREATE CATALOG. The file is created on the storage medium (disk) and the primary assignment is made in accordance with the information contained in the first CAT-LOG media record in the media table (the secondary assignment is always 0). SESAM /SQL creates subsequent CAT-LOG files in accordance with the information given in the media table.

The following default values apply for primary and secondary assignment of the CAT-LOG files: 768 (primary assignment) and 0 (secondary assignment).

BS2000 file name

CAT-LOG files have the following file name in BS2000:
user_id.*catalog*.*version*.C.*nnnn*

user_id	BS2000 user ID
catalog	Name of the database
version	six-digit version number of a SESAM backup copy of the catalog space
nnnn	serial number of the CAT-LOG file, which indicates that it is the *nnnn* th created since the SESAM backup copy with the number version for the catalog space. Gaps may occur in the consecutive numbering.

Opening the CAT-LOG file

If, when opening the CAT-LOG file at open time, the storage media defined via CREATE CATALOG or ALTER MEDIA DESCRIPTION for the CAT-LOG file are full or cannot be accessed, then CC occurs. After this CC, only read access to the database is possible. The CC changes the entry in the SQL database catalog from ACCESS=WRITE to ACCESS=READ.

To ensure that the file can be opened correctly, the user must create memory space on the relevant medium or make the medium accessible. If this not possible, issue a CREATE FILE command with the name of the CAT-LOG file (name from the error message in the SYSLST file). The user must use the SESADM administration program to reset the entry in the SQL database catalog to ACCESS=WRITE.

Switchover to a different CAT-LOG file

SESAM/SQL switches to a different CAT-LOG file in the following situations:

- When a new SESAM backup copy of the whole database or the catalog space is created. In this case, the new CAT-LOG file is given the version number of the new SESAM backup copy of the catalog space and the serial number 0001, e.g. *catalog*.000017.C.0001.
- The serial number of the DA-LOG file overflows.
  In this case the new CAT-LOG file is given the version number incremented by one and the serial number 0001, e.g. *catalog*.000018.C.0001.
- The space in the CAT-LOG file is exhausted.
- An error occurs in the BS2000 DMS.

In the latter two cases, the new CAT-LOG file retains the same version number as its predecessor and the serial number is incremented, e.g. *catalog*.000017.C.0002.

In addition, the CHANGE-CATLOG administration statement can be used to switch the CAT-LOG file and the DA-LOG files to the relevant database(s) (see the “Database Operation” manual). In this case, the version number also remains unchanged and the serial number is incremented.

DA-LOG files

DA-LOG files are log files used in connection with user spaces (see section “Logging”). These log files can be output using the SEDI70 utility (see the “Database Operation” manual).

Contents of DA-LOG files

In DA-LOG files, the SESAM/SQL DBH logs all changes to user spaces. Changes to user spaces occur as a result of SQL statements that change data and as a result of CALL DML statements.
In addition, SESAM/SQL uses DA-LOG files to document the creation of copies of the whole database or the catalog space and copies of user spaces. The creation of a copy of the database or catalog space automatically causes a change of DA-LOG file. The SESAM backup copies serve as resumption points when applying logged modifications during the repair of the database or the user spaces (see section “Recovering the database, catalog space and user spaces”).

Storage medium, primary and secondary assignments

SESAM/SQL creates the first DA-LOG file when the first change is made to a user space. The file is created on the storage medium (disk) and the primary assignment is made in accordance with the information in the first DA-LOG media record in the media table (the secondary assignment is always 0). SESAM/SQL creates subsequent DA-LOG files in accordance with the information given in the media table.

The following default values apply for primary and secondary assignment of the DA-LOG files: 768 (primary assignment) and 0 (secondary assignment).

BS2000 file name

DA-LOG files have the following name in BS2000:
user_id.*catalog*.*version*.D.*nnnn*

user_id	BS2000 user ID
catalog	Name of the database
version	six-digit version number of a SESAM backup copy of the catalog space
nnnn	is the serial number of the DA-LOG file, which indicates that it is the *nnnn*th such file created since the SESAM backup copy with the number version for the catalog space. Gaps may occur in the consecutive numbering.

Opening the CAT-LOG file

If, when opening the DA-LOG file, at open time the storage media defined via CREATE MEDIA DESCRIPTION or ALTER MEDIA DESCRIPTION for the DA-LOG file are full or cannot be accessed, then CC occurs. After this CC, only read access to the database is possible. The CC changes the entry in the SQL database catalog from ACCESS=WRITE to ACCESS=READ.

To ensure that the file can be opened correctly, the user must create memory space on the relevant medium or make the medium accessible. If this not possible, issue a CREATE FILE command with the name of the DA-LOG file (name from the error message in the SYSLST file). The user must then use the SESADM administration program to reset the entry in the SQL database catalog to WRITE.

Switchover to a different DA-LOG file

SESAM/SQL switches to a different DA-LOG file in the following situations:

- When a new SESAM backup copy of the whole database or the catalog space is created. The DA-LOG file is given the version number of the new SESAM backup copy of the catalog space and the serial number 0001, e.g. *catalog*.000017.D.0001.
- The space in the DA-LOG file is exhausted.
- An error occurs in the BS2000 DMS.
- A user space has been repaired
- If the storage medium in use is a magnetic tape cartridge, the DA-LOG file is also switched when the DBH starts up or restarts.

In the latter four cases, the new DA-LOG file retains the same version number as its predecessor and the serial number is incremented, e.g. *catalog*.000017.D.0002.

When the serial number *nnnn* of the DA-LOG file has reached the value 9999 and SESAM/SQL switches the DA-LOG file, the new DA-LOG file is given the version number incremented by one and the serial number 0001, e.g. *catalog*.000018.D.0001. In addition, a new CAT-LOG file with the version number incremented by one and the serial number 0001 is created, e.g. z.B. *catalog*.000018.C.0001

In addition, the CHANGE-DALOG administration statement can be used to switch the DA-LOG files to the specified database or databases (see the “Database Operation” manual). In this case, the version number also remains unchanged and the serial number is incremented by 1.

RECOVERY_UNITS catalog table

The RECOVERY_UNITS catalog table is a base table created by SESAM/SQL in the database's catalog space. RECOVERY_UNITS is used to administer SESAM backup copies of user spaces.

The contents of RECOVERY_UNITS

Each record in RECOVERY_UNITS contains information on a SESAM backup copy of a user space; the information includes the name of the user space, the time stamp assigned to the SESAM backup copy, and details of the DA-LOG file associated with the SESAM backup copy.
SESAM/SQL adds an appropriate record to the RECOVERY_UNITS table whenever the database administrator creates a SESAM backup copy of a user space with the utility statement COPY. The database administrator is responsible for maintaining the RECOVERY_UNITS table (see the section “Maintenance of the metadata of SESAM backup copies”).

The database administrator can view the contents of the RECOVERY_UNITS catalog table with the aid of the utility monitor (see the “Utility Monitor” manual).
The utility statement der MODIFY enables the database administrator to maintain the catalog table RECOVERY_UNITS (see the “SQL Reference Manual Part 2: Utilities”).

DA_LOGS catalog table

The DA_LOGS table is a base table created by SESAM/SQL in the database's catalog space. DA_LOGS is used to administer the DA-LOG files.

The contents of DA_LOGS

Each record in DA_LOGS contains (among other things) the name of a DA-LOG file, the time stamp assigned to this DA-LOG file, and information on the user spaces for which changes are logged in this DA-LOG file. In combination with the RECOVERY_UNITS catalog table, the DA_LOGS catalog table supplies all the information that SESAM/SQL requires in order to repair user spaces.
Whenever SESAM/SQL switches to a new DA-LOG file, it adds an appropriate record to the DA_LOGS catalog table. The database administrator is responsible for maintaining the DA_LOGS table (see the section “Maintenance of the metadata of SESAM backup copies”).

The database administrator can view the contents of the DA_LOGS catalog table with the aid of the utility monitor (see the “Utility Monitor” manual).
The utility statement MODIFY enables the database administrator to maintain the catalog table DA_LOGS (see the “SQL Reference Manual Part 2: Utilities”).

Create SESAM backup copies

The database administrator creates SESAM backup copies using the utility statement COPY. He/she can create the backup copies on disk or on magnetic tape cartridge using ARCHIVE or HSMS. When doing this it is important, for security reasons, to make sure that the SESAM backup copies are not located on the same disk as the database's catalog space and user spaces.

Selecting the backup unit

You can choose any of the following SESAM backup copies:

- The whole SESAM/SQL database.
  If you choose this option, the database's catalog space and all user spaces are backed up at the same time.
- The database's catalog space
- Single user space
- A space set (a backup unit consisting of a number of different user spaces)

A space set is identified by a time stamp, which is included with each user-space entry in the RECOVERY_UNITS catalog table. The time stamp indicates the point in time at which the copy was created; this time must be the same for all user spaces in a space set. Spaces with the same time stamp are created by a backup with a COPY CATALOG statement or when a number of spaces are backed up by a joint COPY statement.

When choosing a backup unit, the database administrator should take the following into account:

- The time it takes to create the copy.
  The time it takes to create the copy depends on the volume of data to be backed up.
- The time required for repair or reset.
  The time required to perform a repair depends on the volume of data in the SESAM backup copy to be read in and the scope of the changes that have been made since the backup was made. The time required for reset depends purely on the volume of data in the SESAM backup copy to be read in.
- When selecting a space set, care must be taken to ensure that no relationships that are verified by SESAM/SQL exist between the user spaces in the space set in question and user spaces outside the space set (e.g. referential integrity, indexes). The same applies to an individual user space.
- The space requirements for SESAM backup copies.
  The space requirements depend on the volume of data to be backed up.
- When using COPY CATALOG, it is possible to exclude spaces from the backup if they simultaneously meet both the following conditions:
  - The spaces are pure index spaces and
  - logical data saving is deactivated for these spaces.

  In the case of RECOVER CATALOG .. GENERATE INDEX ON NO LOG INDEX SPACE, no backups are read in for these spaces. Instead they are reset and recreated.

Specifying the times at which SESAM backup copies are to be made

The times at which SESAM backup copies are to be created are chosen on the basis of

- the backup plan tailored specifically to the database application by the database administrator
- the accomplishing of specific database maintenance functions in connection with which the creation of a backup copy is appropriate or, in certain cases, a necessity.

Creating SESAM backup copies in accordance with the backup plan

Generally, the database administrator periodically creates SESAM backup copies of the whole database, the catalog space, space sets and individual user spaces.

When choosing the period between two consecutive backups of the same backup unit, the database administrator should base his/her choice on aspects such as the time required to create a backup copy (which depends on the volume of data in the backup unit). For example, one might perform a backup of the whole of the database once a month, and backups of space sets and user spaces on a weekly basis.

Additional factors with a bearing on the backup period are the frequency with which changes are made, as well as the speed with which the data is available again after an error situation (i.e. short repair times). The database administrator therefore usually back up user spaces that are changed frequently and need to be recovered quickly more often than other user spaces.

Creating SESAM backup copies in connection with maintenance functions

When certain kinds of database maintenance task are carried out, it makes sense (or is, indeed, essential) to create SESAM backup copies of the database or the relevant user spaces. The exact situations in which this applies are described in the relevant sections.

Immediately following the execution of the utility statement CREATE CATALOG, the database administrator should create a SESAM backup copy of the catalog space as an initial resumption point for any repair measures that might prove necessary. Similarly, a SESAM backup copy of the relevant user space should be created after the execution of every CREATE SPACE statement, because the recovery of a user space without an appropriate SESAM backup copy is only possible in connection with the repair of the whole database.

BS2000 file name for the SESAM backup copies

The names of the SESAM backup copies are as follows:

- for catalog space: *user_id.catalog*.CATALOG.*version*
- for user spaces: *user_id.catalog.space*.*version*
- for the CAT-REC file: *user_id.catalog*.CAT-REC.COPY

user_id	BS2000 user ID
catalog	name of the database for which the catalog space or user space was backed up
space	name of the backed up user space
version	version number

The version number is a six-digit integer. The first SESAM backup copy of the catalog space is numbered 000002. The version number for backup copies of the catalog space is incremented by one each time a new backup copy of the catalog space is created. The version number of the first SESAM backup copy of any user space is 000001, and is incremented by one for each new backup copy of the user space in question. Gaps may occur in the continuous numbering sequence if COPY statements are rejected because of error.

When a SESAM backup copy of the whole database or a space set is created, the SESAM backup copies of user spaces created in the process may be assigned different version numbers. Each SESAM backup copy of a relevant user space is given the appropriate nexthighest version number.

When the catalog space is backed up the CAT-REC file is automatically backed up as well. The CAT-REC copy name does not contain a version number. This means that the file is overwritten each time COPY CATALOG [_SPACE] is used.

Read accesses to SESAM backup copies

The user can read-access a SESAM backup copy of a complete SESAM/SQL database under a separate DBH. This reduces the workload on the current session with the original database.

The user can add the SESAM backup copy to the SQL database catalog with the aid of the ADD-SQL-DB-CATALOG-ENTRY statement. This is done by specifying the number of the SESAM backup copy of the catalog space in the COPY-NUMBER operand.

Only DML read access is permitted, i.e. utility statements are not permitted either. The SESAM backup copy which is read accessed remains available without any changes for recoveries that may become necessary at some later stage.

Foreign copies and replications as backup copies

As well as SESAM backup copies, you can also use foreign copies (see section “Using foreign copies of a database”) and replications (see section “Database replication”) for media recovery.

Logging

SESAM/SQL logs all changes which have been made in the database since a specific SESAM backup copy was created in logging files (see section “Files and tables used for media recovery”).

If the database is located on a DB user ID and the logging files are also to be stored on this user ID, you must make the necessary preparations, see section “Database files and job variables on foreign user IDs”.

It is important to differentiate between the following:

- SESAM/SQL logs changes that affect the database's catalog space (i.e. changes arising as a result of utility statements, or of SQL statements used to administer the storage structure, to administer user entries, or to define and administer schemas) in CAT-LOG files (CAT logging).
- SESAM/SQL logs changes to user spaces (i.e. changes arising as a result of SQL statements that change data, or CALL DML statements) in DA-LOG files (DA logging).

Enabling logging for a database

When a database is created with the utility statement CREATE CATALOG , the database administrator specifies whether changes to the database are to be logged. Logging is then carried out for the whole of the database, in other words in the catalog space and the associated user spaces. Also, logging can be enabled by activating LOG in COPY CATALOG.

In order to avoid loss of data through system or disk errors, the database should generally be used with logging enabled. Database operation without logging is only to be recommended in the case of test databases or databases for temporary data, provided data loss can be compensated for in a different way, and in the case of databases used primarily or exclusively for retrieval purposes.

Enabling logging for user spaces

If logging is enabled for the database, it is also carried out by default for each user space created with CREATE SPACE. However, the database administrator can opt to disable logging when creating a user space. In addition, he /she can use ALTER SPACE to disable logging for a space originally created with logging enabled.If logging is not enabled for a database, logging cannot be carried out for the database's user spaces.

It can be useful to define a user space without enabling logging if the user space contains nothing but indexes or tables of temporary data.
If an error occurs, the database administrator can recover defective indexes with RECOVER INDEX (see section “Recovering indexes”).

In the case of COPY CATALOG, pure index spaces which are not present in the logical data backup can be excluded from the COPY backup. When a subsequent RECOVER is performed, it is necessary to specify the clause GENERATE INDEX ON NO LOG INDEX SPACE. No backups are then read in for the index spaces. Instead they are reset and recreated.

When deciding whether to operate a user space in which just indexes are located with or without logging, the database administrator should remember that operation without logging is accompanied by a performance gain. However, the regeneration of indexes that are defective or contain errors with RECOVER INDEX takes longer than the recovery of indexes in the context of a general repair process. Administrators should therefore choose the option that best suits the real-life application in question.

Disabling or interrupting logging for user spaces

SESAM/SQL logs each interruption to logging for a user space in the catalog table RECOVERY_UNITS by adding an appropriate entry with a time stamp.

The disabling or suspending of logging by the database administrator

The database administrator can temporarily disable logging for a user space with the aid of the SQL statement ALTER SPACE. This can be useful if a large update run needs to be accelerated.

The database administrator must ensure that the status of the user space as it was when logging was disabled, i.e. immediately prior to the update run disabling logging, can be recovered if an error occurs. Repair can then be carried out simply by repeating the update run. The database administrator can achieve this as follows:

- If logging has been suspended for the first time since the creation of the most recent SESAM backup copy of the user space and an error occurs, the database administrator can restore the status of the user space that was current at the time logging was suspended by issuing the utility statement RECOVER TO with the time stamp for the suspension of logging.
- If it is not the first time that logging has been suspended since the creation of the most recent SESAM backup copy of the user space, the database administrator must create a SESAM backup copy of the user space with the utility statement COPY immediately before suspending the logging. This SESAM backup copy can be reset with the utility statement RECOVER TO if an error occurs.

After the update run, the database administrator should create another SESAM backup copy of the user space using the utility statement COPY, and re-enable logging for the user space with COPY at the same time. This SESAM backup copy then serves as a starting point for subsequent recovery measures in this user space.

You can also create a foreign copy instead of a SESAM backup copy and logging can be activated with PREPARE-FOREIGN-COPY, see “Database Operation” manual.

Suspension of logging by SESAM/SQL in its own utility statements

SESAM/SQL automatically suspends the logging of the relevant user spaces when executing the utility statements LOAD OFFLINE , IMPORT TABLE, RECOVER INDEX, RECOVER ADJUST, REORG ... NEW ROW_IDS and MIGRATE if a MIGRATE CALL DML ONLY TABLE is not involved. The database administrator should therefore create SESAM backup copies of the relevant user spaces immediately before executing these utility statements. If an error occurs during a LOAD OFFLINE, IMPORT TABLE, MIGRATE, RECOVER INDEX, RECOVER ADJUST or REORG ... NEW ROW_IDS, it will be possible to reset the status that was valid immediately before the utility statement executed, and the statement can be repeated. In the event of a RECOVER following an error in IMPORT TABLE execution, the table may sometimes still be created. However, in this case it must be deleted with DROP TABLE before IMPORT TABLE can be repeated. If a MIGRATE fails to execute correctly, the database administrator must use the SQL statement DROP TABLE to delete manually any table already created by the failed MIGRATE before attempting a second MIGRATE.

The database administrator must again create SESAM backup copies of the relevant user spaces immediately after a LOAD OFFLINE, IMPORT TABLE, MIGRATE, RECOVER INDEX, RECOVER ADJUST or REORG ... NEW ROW_IDS; these backup copies serve as a starting point for continued logging. If no such copies are created, these user spaces will have the state “copy pending”. They will be locked to update statements until the database administrator creates the relevant SESAM backup copies.

Specifying storage devices for logging

The recovery of the catalog space or user spaces after a disk error is only possible if the SESAM backup copies, the CAT-REC file, and the CAT-LOG and DA-LOG files are located on storage devices other than the ones on which the catalog space and the user spaces are located.

When creating the catalog space with the utility statement CREATE CATALOG, the database administrator also defines the storage group and, thus, the storage medium on which SESAM/SQL creates the CAT-REC file and the CAT-LOG file (see section “Creating the database's catalog space”). Once the catalog space is created, the database administrator can define additional storage groups with the SQL statement CREATE STOGROUP.

The database administrator allocates the storage medium used for the DA-LOG files, and any other storage medium possibly required for the CAT-LOG files, via the media table. CAT-LOG files and DA-LOG files can only be created on disks; disks are identified by means of the relevant storage group.

Administering SESAM backup copies and log files

Administering SESAM backup copies of the catalog space and CAT-LOG files

The CAT-REC file contains an appropriate entry for each SESAM backup copy of the catalog space. Each entry of this kind is followed by the entries for the associated CAT-LOG files.

The database administrator can edit the CAT-REC file, i.e. he/she can delete entries as a means of administering the SESAM backup copies of the catalog space. This is done with the utility statement MODIFY or using the utility monitor (offline update, see the “Utility Monitor” manual).
Once an entry has been deleted from the CAT-REC file, the corresponding SESAM backup copy or CAT-LOG file is unknown to SESAM/SQL and is no longer available for recovery purposes.

It is advisable to delete SESAM backup copies and log files that are no longer needed under BS2000.

Administering SESAM backup copies of user spaces and DA-LOG files

The catalog table RECOVERY_UNITS contains an appropriate entry for each SESAM backup copy of a user space. Information on DA-LOG files is stored in catalog table DA_LOGS.

The database administrator can maintain the RECOVERY_UNITS and DA_LOGS catalog tables with the aid of the utility statement MODIFY. Once an entry has been deleted from RECOVERY UNITS and DA_LOGS, the corresponding SESAM backup copy or DA_LOG file is unknown to SESAM/SQL and is no longer available for recovery purposes.

It is advisable to delete SESAM backup copies and log files that are no longer needed under BS2000.

Recovering the database, catalog space and user spaces

There are two different types of recovery: repair and reset. The database administrator performs both with the utility statement RECOVER.
Depending on the possible backup units, the database administrator can choose between the following units for repair and reset:

- The entire SESAM/SQL database, including the catalog space and all user spaces
- The database's catalog space
- Individual user space, space list or space set

A unit of several user spaces with a common time stamp can be designated as a space set. Spaces with the same time stamp are created by a backup with a COPY CATALOG statement or when a number of spaces are backed up by a joint COPY statement. The recovery of a space set as a unit is only possible if logging was not interrupted for any of the user spaces in this space set.
A space list designates a number of user spaces from the backup of an entire database or a space set which are grouped together for a joint RECOVER statement. The spaces in a space list must all have the same time stamp. The recovery of the spaces in a space list is only possible if logging was not interrupted for any of the user spaces in this space list.

To reduce the recovery times, you can also restrict the number of spaces included in RECOVER:

- If you specify the SCOPE PENDING clause in RECOVER then the user spaces are repaired if they are identified as defective when opened.
  SCOPE PENDING is not permitted for foreign copies or for RECOVER SPACESET ... TO or RECOVER SPACE *space list* TO.

  In the case of RECOVER CATALOG, the catalog space is always repaired and the user spaces are only repaired if they are identified as defective when opened.

  In the case of RECOVER CATALOG ... TO, the catalog space is always reset. The user spaces are only reset if they are identified as defective when opened or if an inconsistency between catalog space and user space is detected from the point of view of the catalog. This type of inconsistency exists if the user space has been modified since the time of the backup to which the space is being reset.
- Administer indexes in index spaces, i.e. in spaces which contain only indexes and are not included in logical data saving. You use two clauses to work with these index spaces.
  - GENERATE INDEX ON NO LOG INDEX SPACE
    You can specify this clause in RECOVER CATALOG. When you specify this, the spaces that contain only indexes and are not involved in the logical data backup are reset and the indexes are reconstructed.
  - NO INDEX
    You can specify this clause when performing the reset of a user space, a space list or a space set. It is used to identify indexes as defective instead of reconstructing them when they have been made invalid through the reset of a space. This case occurs if the base table or the partitioned table and an associated index are located on different user spaces and not all the user spaces affected are reset simultaneously.

Repair

Repair is carried out using previously created SESAM backup copies of the database, the catalog space or a user space, a space list or a space set, together with the associated log files. The CAT-LOG files are the log files for the catalog space; the DA-LOG files are the log files for the user spaces.

The database administrator can use the INF mask of the utility monitor to output the related backup files in job variables. The database administrator can obtain information on existing SESAM backup copies of the catalog space from the CAT-REC file with the aid of the utility monitor. To find out what SESAM backup copies of user spaces are available, the database administrator checks the RECOVERY_UNITS catalog table, again with the aid of the utility monitor (see the “Utility Monitor” manual).
The repair returns the database, the catalog space or user space, a space list or space set to the state it was in prior to the occurrence of the error situation.

User spaces, the catalog space or the entire database can also be repaired by means of foreign copies (see section “Repair and reset using foreign copies”).

The repair of user spaces, the catalog space or of the entire database is also possible using a replication (see section “Using a replication to repair an original database”).

Repair of catalog space

When repairing the catalog space, SESAM/SQL uses the information from the CAT-REC file concerning the SESAM backup copy of the catalog space specified by the database administrator in RECOVER, and the information on the associated CAT-LOG files. SESAM/SQL then reads in this SESAM backup copy and applies the modifications logged in the CAT-LOG files.

The figure 18 outlines the repair of the catalog space.

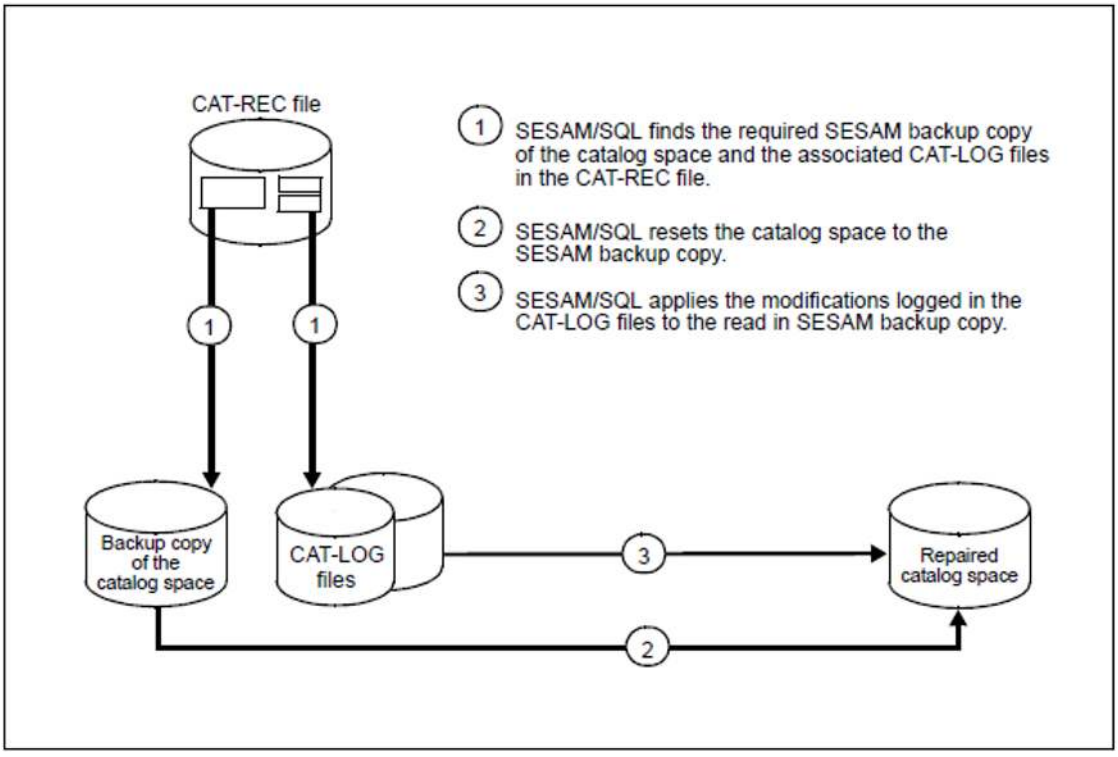

Figure 18: Repair of the catalog space

Repair of a user space

When repairing a user space, a space list or a space set, SESAM/SQL retrieves from the RECOVERY_UNITS catalog table the information on the SESAM backup copy of each user space specified by the database administrator in RECOVER. In the DA-LOGS catalog table, SESAM/SQL finds the DA-LOG files in which changes made to the user space since the copy was made are logged. SESAM/SQL then reads in this SESAM backup copy and applies the modifications logged in the DA-LOG files.

The figure 19 outlines the repair of a user space.

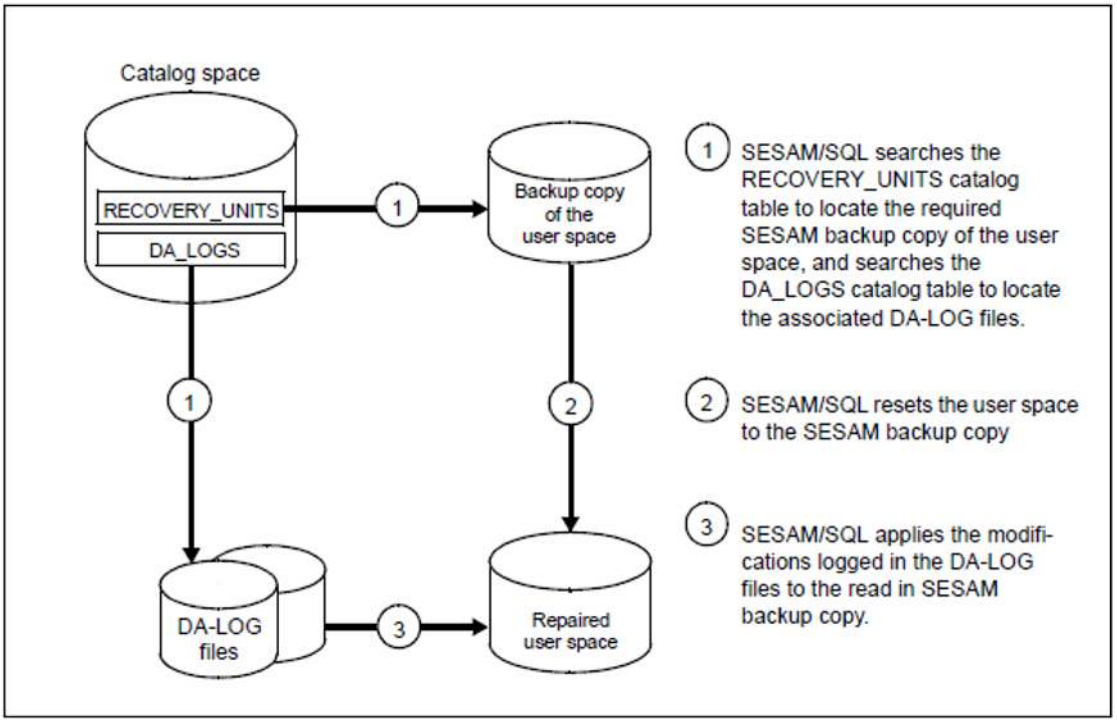

Figure 19: Repair of a user space

Repair of the entire SESAM/SQL database

During a repair of the entire of the database, SESAM/SQL begins by repairing the catalog space on the basis of the SESAM backup copy specified in RECOVER. Using the metadata from the recovered catalog, SESAM/SQL then repairs all the database's user spaces from the latest SESAM backup copy of each user space. This means that the consistency of the database is always guaranteed after an orderly repair operation. If, however, SESAM/SQL aborts the repair of a user space (say, because a DA-LOG file is missing or corrupt) and “freezes” the user space in its current state, the database administrator can opt to proceed with repair or to render the database consistent in its frozen state.

If SESAM/SQL has aborted a repair operation due a missing or corrupt DA-LOG file, the database administrator can complete the recovery with RECOVER RESTART if he/she provides a suitable intact DA-LOG file. To do this, the database administrator must execute the utility statement RECOVER RESTART for each user space.

They can restore the consistency of the database in its frozen state by executing the utility statement RECOVER ADJUST for each user space. When RECOVER ADJUST is executed, SESAM/SQL restores consistency by means of the same adaptation mechanisms as when resetting individual user spaces (see “Database consistency after a reset”).

Reset

Reset as a means of recovering individual user spaces, a space list, a space set, the catalog space or the entire database is always an attractive option whenever logging was deactivated either entirely or temporarily for the backup unit in question and consequently repair is not possible

Equally, reset to the status of the last SESAM backup copies of the spaces in question is the only way of eliminating a database error caused by a defective user program. This also applies even if logging was active. The selective applying of the modifications to the DA-LOG files to mask data modifications caused by the defective user program is not possible.

User spaces, the catalog space or the entire database can also be reset by means of foreign copies (see section “Repair and reset using foreign copies”).

It is also possible to reset user spaces, the catalog space or the entire database on the basis of a replication (see section “Using a replication to repair an original database”).

Reset of user spaces

The database administrator uses the RECOVER TO utility statement to reset a user space, a space list or a space set.

The reset is performed on the basis of a previously created SESAM backup copy of a user space, a space list or a space set. If you choose a space list or a space set, all the spaces must have the same backup time stamp. The reset process then returns the user space or the user spaces of the space list or space set to the status which applied immediately before the SESAM backup copies were created.

> **! CAUTION!**
>
> You cannot undo a RECOVER TO. A more recent state can no longer be established.

The database administrator determines what SESAM backup copies of user spaces are available by accessing the RECOVERY_UNITS catalog table with the utility monitor. The required SESAM backup copy can then be selected via the version number of the backup, the time stamp or the file name of the SESAM backup copy.

Resetting user spaces to a mark

A mark is a special entry in the catalog table RECOVERY_UNITS of the information schema. It has a time stamp and as a RECOVER_TYPE is given the string “MARK”. It represents a specified state of the database which can be restored on the basis of a previous backup by applying the changes logged in the logging files.

A mark is written when performing the utility functions LOAD OFFLINE, MIGRATE, IMPORT, RECOVER INDEX or REORG ... NEW ROW_IDS or for the SSL statement ALTER SPACE ... NO LOG. A database administrator can reset a user space to a mark with the utility statement RECOVER SPACE *space* USING *rec_unit* TO TIMESTAMP or in the utility monitor via the mask COP using this data. *rec_unit* can be a SESAM backup copy, a foreign copy or a replication here.

Example

You want to undo a LOAD OFFLINE for the space *space*.

The following steps are required:

a. Make the backup available.

b. In the catalog table RECOVERY_UNITS of the information schema, identify which entry contains the RECOVER_TYPE “MARK” and the RECOVERY_TIMESTAMP *before_load* with the time stamp of the LOAD.

c. Execute a RECOVER statement:
`RECOVER SPACE` *space* `USING` *rec_unit* `TO TIMESTAMP` *before_load*

If the functions LOAD OFFLINE, MIGRATE, IMPORT, RECOVER INDEX or REORG ... NEW ROW_IDS are aborted as a result of errors, this usually means that the mark has not yet been written. The space can then be reset to the state prior to the function call using `RECOVER SPACE` *space* `USING`. In the case of LOAD OFFLINE it is important to note that indexes can already be marked as defective at the time of abortion. These indexes are not be rebuilt by RECOVER SPACE *space* USING which means that RECOVER INDEX may still be required.

Reset of the entire SESAM/SQL database

The database administrator resets the entire database using the RECOVER CATALOG utility statement.

Two options are available here:

- Resetting to a SESAM backup copy which was created earlier using RECOVER CATALOG TO:

  This reset restores the database to the status which existed at the time the SESAM backup copy was created.

  In this case SESAM/SQL first resets the catalog space on the basis of the SESAM backup copy specified in RECOVER CATALOG TO. The metadata of the reset catalog space is then used to repair all the database's user spaces on the basis of the last created SESAM backup copy for the individual user spaces. As a result, the consistency of the database is always guaranteed following a correctly performed repair.

- Resetting to a (freely) selectable time using RECOVER CATALOG [USING ...] TO ANY *timestamp*:

  This reset restores the database to the status which existed at the specified time. When ANY is not specified, *timestamp* must identify the time of a SESAM backup copy of the catalog space. The SESAM backup copy must be entered in the CAT-REC file.

  SESAM/SQL reads in the specified SESAM backup copy of the catalog space (USING specified) or the most recent SESAM backup copy of the catalog space before the specified time. SESAM/SQL then applies the changes to the CAT-LOG files which were created subsequent to this SESAM backup copy up to the specified time.

  The metadata of the reset catalog space is used to ascertain and read in all the user spaces of the database on the basis of the most recently created SESAM backup copy of the various user spaces. SESAM/SQL then applies the changes to the DA-LOG files which were created subsequent to this SESAM backup copy up to the specified time.

> **! CAUTION!**
>
> A reset to the state of a previously created SESAM backup copy performed using RECOVER TO cannot be undone.

Reset of the catalog space

The database administrator can performs the reset of the catalog space using the RECOVER CATALOG_SPACE TO utility statement.

Resets are performed on the basis of a previously created SESAM backup copy for the catalog space. The reset sets the catalog space to the state obtaining at the time of the creation of this SESAM backup copy.

> **CAUTION!**
>
> If the catalog space is reset separately then it is not usually possible to access the individual user spaces since their modification time stamps no longer match those recorded in the catalog.

RECOVER CATALOG_SPACE TO leaves the user spaces unchanged and these have to be repaired separately.

The database administrator can use the utility monitor to identify which SESAM backup copies of the catalog space are available on the basis of the CAT-REC file.The required SESAM backup copy can then be selected via the version number of the backup, the time stamp or the file name of the SESAM backup copy.

> **CAUTION!**
>
> A reset to the state of a previously created SESAM backup copy performed using RECOVER TO cannot be undone. All records following the chosen backup copy are deleted in the CAT-REC file. A more recent state can no longer be established. However, you can back up the CAT-REC file and the CAT-LOG files outside of SESAM/SQL before reset.

Database consistency after a reset

The consistency of the whole of the database is not necessarily guaranteed after the reset of individual user spaces, a space list or a space set.

Consistency during reset involves the following:

- The metadata in the catalog space (table definitions and index definitions) must match the structure of the user data in the reset user space.
- The user data in a table must match the associated indexes.
- The integrity constraints defined for the table must be fulfilled.

If the catalog space is reset separately then the user spaces remain unchanged and must be repaired individually. It is not usually possible to access the user spaces since their modification time stamps no longer match those recorded in the catalog.

If individual user spaces, a space list or a space set are reset, SESAM/SQL forces consistency between the metadata in the catalog space and the user data immediately after the reset by employing the adaptation mechanisms described below.

> **CAUTION!**
>
> Table data can be lost when adapted to the catalog metadata. If the database administrator wishes to ensure that no data is lost during reset as a result of these adaptation mechanisms, he/she should always backup and reset the complete database

Consistency of the catalog space and user spaces

The metadata that describes a user space (the table definitions and index definitions) in the catalog space must be consistent with the structure of the user data in that user space. This consistency requirement is usually not fulfilled if the user space is reset to a SESAM backup copy, and SQL statements used to administer the memory structure or define and administer schemas (statements that have an effect on the user space) have been executed since the SESAM backup copy was created.
For example, a base table or an index may have been created in a user space since the SESAM backup copy was created, and these are then no longer in the user space after a reset. However, the associated metadata still exist in the catalog space. It is also possible that a base table or an index might have been deleted in the user space, in which case the relevant metadata will also have been deleted from the catalog space. Once the user space has been reset, it will contain a base table or index for which no metadata exists in the catalog space.

To monitor the consistency between the catalog space and the user spaces, SESAM/SQL maintains a internal table in each user space; this table contains a description of the user space that corresponds to the relevant metadata in the catalog space. SESAM/SQL uses this table to detect discrepancies between a user space and the associated description in the catalog space, and adapts the user space to the description in the catalog space as follows:

- If the catalog space contains metadata for tables and indexes that do not exist in the user space, SESAM/SQL re-creates these tables and indexes in the user space. The tables do not contain data; the indexes have the correct structure and are likewise empty.
- If tables and indexes exist in the user space but there is no corresponding metadata in the catalog space, SESAM/SQL deletes the tables and indexes in question from the user space.
- If tables' definitions and, thus, their descriptions in the catalog space have been modified, SESAM/SQL deletes the tables in question and re-creates them in the user space in line with the modified definitions. These new tables are empty.

If data is lost when SESAM/SQL restores consistency between the catalog space and the user spaces, the database administrator can restore the original data to the newly created tables, which are initially empty:

- He/she begins by unloading the data from a SESAM backup copy with the utility statement UNLOAD OFFLINE ... FROM COPY_FILE (see the “SQL Reference Manual Part 2: Utilities”).
- He/she then loads this data into the newly created tables with the utility statement LOAD.

Consistency between base tables and the related indexes

During reset, SESAM/SQL guarantees consistency between a base table and the indexes defined for it by rebuilding the indexes, provided the base table or the partitioned table and a related index are located in different user spaces and the database administrator has not reset all the user spaces affected.
If the table and the related indexes are in the same space list or space set and the database administrator has reset the space list or space set, consistency is guaranteed. SESAM/SQL does not rebuild the indexes in this case.

Consistency between base tables and integrity constraints

During the reset of a user space, SESAM/SQL checks whether the user space contains any base tables for which integrity constraints have been defined. If this is the case, SESAM/SQL checks whether the integrity constraints currently defined in the catalog space for the base tables are still fulfilled.

If an integrity constraint is violated, SESAM/SQL puts the user space in the state “check pending” (see section “Space state after the execution of utility statements”).

It is up to the database administrator to correct the violation of the integrity constraints and to free the user space from the “check pending” state. He/she does this with the utility statement CHECK CONSTRAINTS and SQL statements that query and update data, which he/she issues with the Pragma CHECK OFF).

Recovering databases in various situations

Suitable measures for recovery allow a defective SESAM/SQL database to be recovered in a variety of error situations. Generally, the various RECOVER statements will recover a database without difficulty. Other problematic situations occur which require special treatment before the database can be recovered. Sometimes, data will be lost in these cases. SESAM/SQL always, however, ensures consistency between the catalog space and the user spaces.

Recovering user spaces in various situations

The table 48 describes not only the normal situation for RECOVER SPACE, but also a number of problematic situations and the corresponding measures which allow a user space to be repaired or reset.

Situation	**Measures**	**Result/ comment**
Logical data backup activated, SESAM backup copy and associated DA-LOGs available	Correct repair of the space: RECOVER SPACE	Space is repaired
DA-LOG file not available RECOVER SPACE aborted	If DA-LOG can be made available, continue with RECOVER RESTART	Space is repaired
	If DA-LOG file cannot be made available, terminate repair with RECOVER ADJUST	Data lost as a result of DA-LOG which is not available and any subsequent DA-LOGs
SESAM backup copy not available	If another SESAM backup copy is available with the associated DA-LOGs: repair using the status of this SESAM backup copy with RECOVER USING	Space is repaired

	If the entry for the SESAM backup copy has already been deleted with MODIFY: reset to the status of this SESAM backup copy with RECOVER TO COPY_FILE. Tape backups must first be read in using HSMS or ARCHIVE and renamed.	Data lost as a result of DA-LOGs to which modifications could not be applied
Space cannot be accessed by RECOVER	Use administration statements to place the database in the FREE status and then in the ACTIVE status again or stop and start the DBH again. Save the faulty space for diagnostic purposes: 1. either rename the faulty space with BS2000 resources 2. or copy the faulty space with the BS2000 COPY function and then delete the original space Repeat RECOVER	Space is repaired
Space faulty, no SESAM backup copy available	Recover the empty space with RECOVER SPACE TO COPY_FILE '*DUMMY'. The space is then accessible again. Carry on working with the space, load data.	At first complete loss of all data in the user space. You can continue to work with the space.
	Delete space with DROP SPACE FORCED.	Complete loss of all data in the user space. Consistency between user spaces and metadata is restored.

Table 48: Possible results for RECOVER SPACE

Recovering the database or the catalog space in various situations

The table 49 describes not only the normal situation for RECOVER CATALOG or RECOVER CATALOG_SPACE, but also a number of problematic situations and the corresponding measures which allow repair or reset of the database or catalog space.

Situation	**Measures**	**Result/ comment**

Logical data backup activated, SESAM backup copy and associated CAT-LOG and DA-LOGs available	Execute RECOVER CATALOG_SPACE or RECOVER CATALOG SCOPE PENDING clause: Only defective spaces are repaired.	Catalog space or database is repaired
Logical data backup deactivated for pure index spaces but activated for all other spaces; SESAM backup copy and associated CAT-LOG and DA-LOGs available for all spaces in the logical data backup	Execute RECOVER CATALOG ... GENERATE INDEX ON NO LOG INDEXSPACE.	Database is repaired and indexes are rebuilt
Problems on repairing spaces with RECOVER CATALOG	First execute RECOVER CATALOG_SPACE, then recover the individual spaces with RECOVER SPACE	see table 48
CAT-LOG file no longer available	If CAT-LOG file can be made available: Repeat RECOVER CATALOG_SPACE or RECOVER CATALOG	Catalog space or database repaired
	If the CAT-LOG file cannot be made available: 1. Reset to the last SESAM backup copy with RECOVER ... TO. 2. Check whether measures to create a new CAT-REC file would be advisable. For this, the software customer service should be consulted. Perform RECOVER CATALOG_SPACE or RECOVER CATALOG if necessary.	1. Data loss, as no changes logged as CAT-LOGs are applied. 2. Data loss, as only the changes logged as CAT-LOGs prior to the missing CAT-LOG file can be applied.
SESAM backup copy not available	If another SESAM backup copy and the associated CAT-LOG and DA-LOG files are available: repair using this SESAM backup copy with RECOVER USING.	Catalog space or database is repaired
	If an entry for a suitable SESAM backup copy has already been deleted from the CAT-REC file: Check whether measures to create a new CAT-REC file are sensible. In this case, contact your software customer service department.	Complete data loss as repair is only possible for files recorded in CAT-REC

Catalog Space cannot be accessed by RECOVER CATALOG[_SPACE]	1. Use administration statements to place the database in the FREE status and then in the ACTIVE status again 2. or stop and start the DBH again Save the faulty catalog space for diagnostic purposes and then delete it Add the catalog again to the SQL database catalog or restart the DBH and repeat RECOVER CATALOG[_SPACE].	Catalog space is repaired

Table 49: Possible results for RECOVER CATALOG and RECOVER CATALOG_SPACE

Recovering the database if the CAT-REC file is faulty

Wherever possible, the CAT-REC file should be mirrored (see “Backing up the CAT-REC file”). If the CAT-REC should nevertheless become damaged, you must contact your software customer service department.

Recovering indexes

The database administrator can recover or rebuild defective indexes with the utility statement RECOVER INDEX).

Optionally, he/she can recover or rebuild

- a specific index, specified by means of its index name,
- all indexes that belong to a specific table,
- all indexes that belong to the tables in a specific user space.

Database replication

A replication is a copy of a database which can be used for recovery or as a shadow database.

Replications can be executed under a different name in the same DBH as the original or in another DBH under an optional name. Replications can be added to the SQL database catalog in multiple DBHs for read access.

The database administrator can create and update replications of an original database from SESAM backup copies (catalog backups on disk or magnetic tape cartridge) or from a foreign copy (see section “Creating replications from foreign copies”) at any intervals. If necessary, he/she can deletes spaces from a replication or extends the replication with new spaces.

A replication which contains all the spaces of the original database (catalog space and all user spaces) is known as a complete replication. A partial replication contains the catalog space and at least one of the user spaces.
A complete replication can become a partial replication. Conversely, a partial replication can be extended to become a complete replication.

Replications and partial replications are available for read access. If a replication contains a partitioned table, all the partitions of the table must be contained in the replication. A partial replication must always be complete in itself, i.e. the partial replication must contain all the indexes for all the tables which belong to the partial replication.

A complete replication is created from a complete catalog backup via the utility statement CREATE REPLICATION.

A partial replication is created by:

- the utility statement CREATE REPLICATION ... FOR SPACE ... from a complete catalog backup or a catalog backup without NO LOG index spaces
- the utility statement CREATE REPLICATION ... from a catalog backup without NO LOG index spaces
- the utility statement REFRESH REPLICATION ... FOR SPACE ... from a complete replication or a partial replication, for example, when logging was interrupted for a replication space in the original and this space can no longer be updated in the replication
- the SQL statement CREATE SPACE which is used to create a new space in the original

The database administrator can extend a replication by adding user spaces of the original to the replication using the utility statement REFRESH SPACE on the basis of a backup. These spaces can be new to the replication or they may have previously belonged to the replication (“previous replication spaces”). The replication must be updated to such an extent that the backup used is known in the replication.

You can find information about current and previous replication spaces via the utility monitor, mask COP.4.6 (see “Utility Monitor” manual).

In order to be able to update a replication with REFRESH REPLICATION, all the replication spaces must be included in the logging and logging must have been interrupted (e.g. using LOAD OFFLINE). This also applies to spaces which are to be added to a replication using REFRESH SPACE.

As part of the media recovery, the database administrator can also use a replication as an alternative to SESAM backups to repair or reset an original database:

- He/she can repair user spaces, the catalog space or an entire database with the aid of a replication (see section “Using a replication to repair an original database”). Under certain circumstances this may be faster than a normal RECOVER.
- He/she can use the replication to reset user spaces, the catalog space or an entire database to the status of the replication (see section “Reset the original database using replications”).

- He/she can create a new original database from the replication (see section “Creating a separate original database from the replication”).

Create a replication

- Creating a replication from a SESAM/SQL backup copy

  The database administrator uses the utility statement CREATE REPLICATON to create a replication from a SESAM backup copy generated using COPY CATALOG. The replication is identical to the original database at the time of backup.
  The CAT-REC file which is specified in CREATE REPLICATION identifies a backup copy. It has to contain the required database backup as the last backup. The CAT REC copy generated when COPY CATALOG was executed meet this requirement. The replication's CAT-REC file which was created in the course of CREATE REPLICATION contains the status of the original at the time COPY CATALOG was executed.

  The backup used by CREATE REPLICATION must always be a full database backup on magnetic tape cartridge or disk created with COPY CATALOG.

- Creating a replication from a non-SESAM backup copy

  The database administrator can create a replication or a partial replication from a foreign copy. For information on generating a foreign copy, see section “Using foreign copies of a database”.

You can create a partial replication by using the FOR SPACE parameter to specify the required user spaces. The replication always contains the catalog space. The created replication can only be subsequently updated if all the spaces of the replication are handled using the Logging facility in the original.

After the replication has been created it can be immediately used for recovery. In addition, it is automatically added to the DBH's SQL database catalog.

Like any other database, the replication can be entered in the SQL database catalog using the ACCESS=REPLICATION parameter in ADD-SQL-DATABASE-CATALOG-LIST on start-up or can be added using the same parameter with the administration statement ADD-SQL-DB-CATALOG-ENTRY while the DBH is running (see the “Database Operation” manual). The original database and replication can use the same logical name provided they run under different DBHs.

A replication can be added to SQL database catalogs of different DBHs for recovery.

> The following applies to SESDCN: Since the relation between the original name and the replication name is established in the DBH, they cannot both be included in distribution if the original and the replication are used in 2 DBHs simultaneously and if both are accessed via the same logical database name.

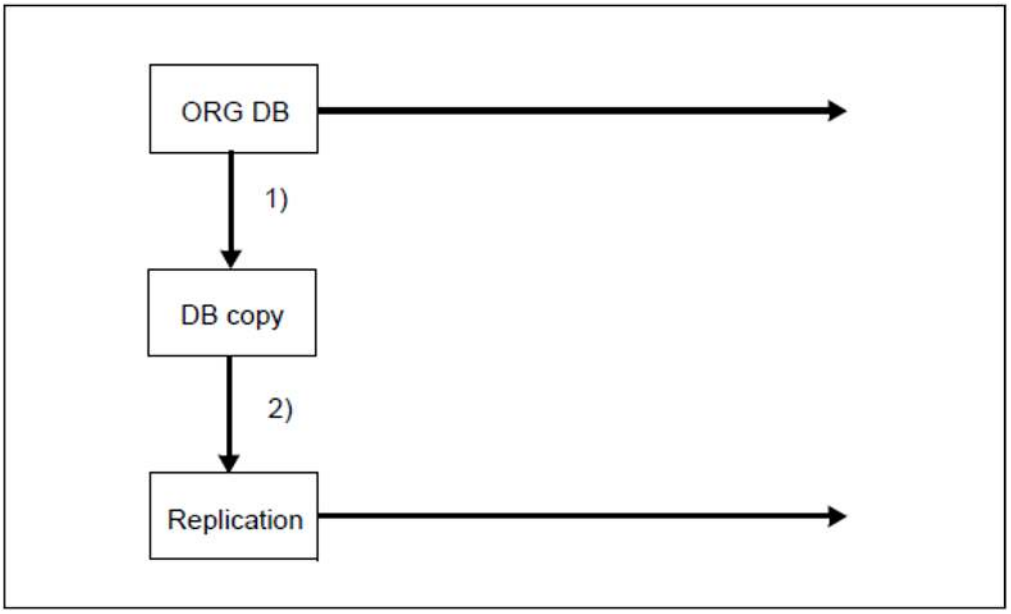

Figure 20: Create replication

1. Creating a SESAM backup copy of the original database
2. Creating a replication from the copy

Retrieval using a replication

An important area of application for replications and partial replications is to reduce the burden on DBHs in the OLTP operation of read-only applications which do not necessarily need the latest version of the data (e.g. for evaluations and statistics).

Retrieval using partial replications

The database administrator can decide to use only a partial replication. This partial replication contains only those parts of the database which are required by the current application. A partial replication requires less space and can be created and updated more quickly.

A partial replication must always be complete in itself, i.e. the partial replication must contain all the indexes for all the tables which belong to the partial replication.

SESAM/SQL responds with the SQL status code “index or space inaccessible” if indexes or tables which are not part of the partial replication are accessed.

Updating and extending replications

The database administrator can use the REFRESH REPLICATION utility statement to update a replication or a partial replication:

- REFRESH REPLICATION
  updates the whole replication or partial replication by applying the modifications logged in the logging files.
- REFRESH REPLICATION FOR SPACE
  only updates the specified spaces and the catalog space of the replication by applying the modifications logged in the logging files. The replication will then only contain these spaces and will become a partial replication.

A replication should be updated in the following cases:

- If a recovery application needs access to the most recent values.
- If the replication is to be used as a shadow database to allow particularly fast repair.

The database administrator can extend a replication by adding user spaces of the original to the replication using the utility statement REFRESH SPACE on the basis of a backup. These spaces can be new to the replication or they may have previously belonged to the replication (“previous replication spaces”). The replication must be updated to such an extent that the backup is known in the metadata of the replication.
You can find information about current and previous replication spaces via the utility monitor, mask COP.4.6 (see “Utility Monitor” manual).

During REFRESH REPLICATION or REFRESH SPACE, read applications cannot access the replication, since exclusive access is necessary for the update process. The replication may be added only to the SQL database catalog of the DBH for which the utility statement has been executed.

Updating the replication (REFRESH REPLICATION)

REFRESH REPLICATION updates the whole replication with all the user spaces that are current in the replication.

REFRESH REPLICATION FOR SPACE updates the catalog space of the replication and all specified user spaces. If the replication contained more user spaces than specified, these will be logically deleted from the replication (“previous replication spaces”). The replication will become a partial replication of the original replication. Previous replication spaces are not physically deleted.

The database administrator must make available all the logging files from a certain period and the CAT-REC copy of the original database. The job variable SESAM.*replication*.NEXT-REPL-LOG contains the number and the subnumber of each of the files to be made available first. The job variable is created in CREATE REPLICATION and updated in each case in REFRESH REPLICATION.

All the user spaces of a replication must be included in the logging in the original database. Spaces, for which logging was interrupted in the original, can no longer be updated. In this case, the replication can be reduced using REFRESH REPLICATION FOR SPACE if the affected space is not specified in the space list in FOR SPACE. Logging is interrupted by the statements ALTER SPACE NO LOG, LOAD OFFLINE, IMPORT TABLE, RECOVER INDEX, RECOVER TO and MIGRATE, see also “Suspension of logging by SESAM/SQL in its own utility statements”.

A replication can also be reduced with REFRESH REPLICATION FOR SPACE if spaces of the replication are invalid or are no longer required in the replication.
Spaces become invalid through DROP SPACE in the original, RECOVER SPACE USING REPLICATION RENAME, RECOVER SPACE TO REPLICATION or RECOVER SPACE USING REPLICATION TO *mark*.

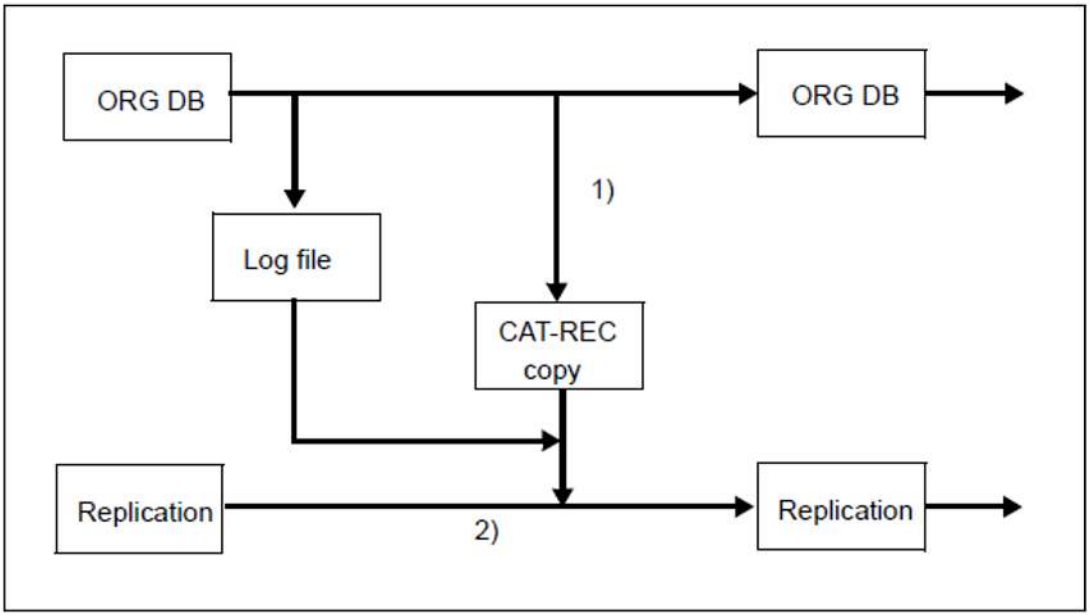

Figure 21: Update a replication

1. Switching the log files and creating a CAT-REC copy
2. Updating the replication by means of logging files and the CAT-REC copy

Procedure

At a given moment, the replication is to be updated to the status of the original database. To do this, use CHANGE-CATLOG or COPY CATALOG_SPACE to swap the CAT-LOG and DA-LOG files in the original. This creates a CAT-REC copy.

> For the following reasons it is advisable to issue the CHANGE-CATLOG statement at a time when no transactions are being performed:
> Transactions which are still active are rolled back in the replication at the end of the update.
>
> If transactions have been rolled back, all logging files from the start of the rolled back transaction onwards must be processed when REFRESH REPLICATION is subsequently called. This is the only way that a defined state can be obtained in the replication.
> REFRESH REPLICATION should therefore be avoided if long-lasting transactions are open. In the worst case, sometimes several times, the modifications may be applied to these transactions, these transactions then may be rolled back and, then again, the modifications may be applied to these transactions.

REFRESH REPLICATION restarts from the replication's CAT-REC file. The difference from CAT-REC copy determines which CAT-LOG files must be incorporated in the catalog space. After the CAT-LOG files have been incorporated, the current RECOVERY_UNITS and DA-LOGS catalog tables of the processed catalog space can be used to determine to which DA-LOG files in the user spaces the modifications have to be applied. Then the modifications are applied to the DA-LOG files.

The replication is identical to the original at the time the CAT-REC copy was used.

The numbers of the oldest logging files which are required for the next REFRESH REPLICATION are entered in a job variable.

In the same way as for RECOVER in the service task, the catalog space and the user spaces are updated in a DBH which the modifications are applied to.
The replication is locked if an error occurs during the applying of the modifications, e.g. because of a missing log file, a memory bottleneck or because the service task has terminated. Once the cause of the error has been eliminated, the replication can be completed using REFRESH REPLICATION.
If REFRESH REPLICATION aborts because the DBH is terminated or because it is executing an internal restart, it is not possible to repeat REFRESH REPLICATION. In this case, the replication is marked as defective and has to be recreated.

CREATE SPACE in the original automatically changes an existing replication to a partial replication since the new space is not yet present in the replication.

Extending the replication (REFRESH SPACE)

The database administrator can extend a replication by adding the specified user spaces to a replication using the utility statement REFRESH SPACE. These spaces can be new to the replication. They may also have previously belonged to the replication (“previous replication spaces”), but were then deleted from the replication because logging was interrupted or because they were invalid.

REFRESH SPACE adds the specified spaces from a SESAM backup copy (disk or magnetic tape cartridge) or a foreign copy to the replication and updates it to the current state of the replication by applying the modifications logged in the logging files.

REFRESH SPACE does not check whether specified spaces are already contained in the replication. Spaces that already exist are overwritten. If a replication space is recreated, the medium of the replication catalog space will be used.

The replication must be updated for REFRESH SPACE to such an extent that the backups used are known in the replication. Backups that were created at a time before the backup from which the replication was created can also be used.
If several spaces are specified in REFRESH SPACE, all the backups must have the same time stamp.
Modifications logged in the logging files are applied to spaces by REFRESH SPACE to update them to the current state of the replication. Logging must not have been interrupted between the backup used and the state of the replication on the affected spaces in the original.

The database administrator must make available all the DA-LOG files that were created from the time of the backup to the current state of the replication. You can determine which files are required via the RECOVERY_UNITS in the INFORMATION_SCHEMA.

Using a replication to repair an original database

- RECOVER CATALOG USING REPLICATION
- RECOVER CATALOG_SPACE USING REPLICATION
- RECOVER SPACE USING REPLICATION

RECOVER CATALOG USING REPLICATION

The database administrator can use the utility statement RECOVER CATALOG USING REPLICATION to repair an entire original database with a complete replication. Partial replications can be used to repair corresponding parts of the original.

The version which is to be repaired is determined by the specified CAT-REC file. The specified CAT-REC file may not be older than that of the replication.
If you specify a CAT-REC file which is older than the current CAT-REC file of the original, you are effectively resetting the database (see section “Reset using RECOVER CATALOG USING REPLICATION”).

The CAT-REC file of the original database is overwritten when RECOVER USING REPLICATION is used. The database administrator must save the version of the original before calling RECOVER, if he/she plans to access it again.

Using a replication for repair is only possible if the original database's catalog space and user spaces have always been processed using the Logging facility.
A regularly updated replication ensures greatly accelerated repair of the original database. If the replication is used to repair the original database, the modifications have to be applied only to those logging files which differed between in original and the replication at the time the error occurred.
The more frequently a replication is updated, the more up-to-date it is and hence the quicker the repair can be performed.

If only a partial replication exists, only the catalog space and the user spaces contained in this partial replication can be repaired. The other spaces might then need to be repaired using RECOVER SPACE (see “Recovering user spaces in various situations”).

You can only use a replication for repair if the conditions which permit a REFRESH REPLICATION are satisfied in the original. This means that the logical data backup must not have been interrupted and no space in the replication may been deleted from the original using DROP SPACE

If you use the COPY function, the replication is retained and can be used for further processing. If you use the RENAME function, the replication becomes the original database and is no longer available as a replication. When using RENAME, the replication and original must be located on the same BS2000 user ID and the files of the original must no longer exist since it is otherwise no longer possible to recatalog the files.

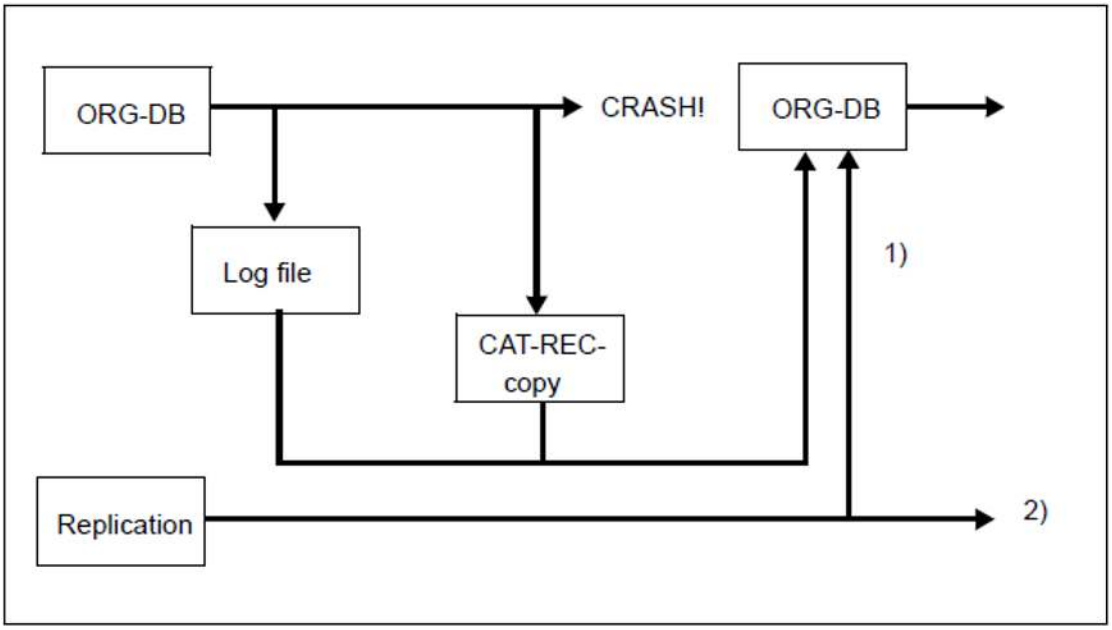

Figure 22: Repairing the original database using a replication

1. You can use RECOVER CATALOG USING REPLICATION to repair a database from the replication, the logging files and the current CAT-REC file.
2. The COPY option leaves the replication unchanged.

Procedure

If the original database's CAT-REC file was not physically destroyed when an error occurred, it can be used for the repair. If a CAT-REC file no longer exists, the latest copy must be identified.

> In order to recover the current version of the original, the CAT-REC file of the original must be used for repair. If you use an older CAT-REC copy, the catalog is reset.

The database administrator must provide the required logging files for repair. If the replication is processed from the same DBH as the original database, it must be placed in the FREE status before repair. This is not necessary if different DBHs are used. However, no parallel REFRESH REPLICATION of the replication may be performed.

If a RECOVER USING REPLICATION where the current CAT-REC file was not specified is aborted, it is possible that the CAT-REC file of the original may have already been overwritten. To make sure that no data is lost, this file should be backed up before RECOVER is called. This is not automatically performed by SESAM/SQL.

Partial replication without index space

If the partial replication is not used for recovery, the indexes to the tables are not necessary. This makes it possible, for example, to use only spaces with tables for the partial replications. If this partial replication is used for repair, the spaces with the primary data are recovered. The spaces containing the indexes can then be recreated using RECOVER TO *DUMMY.

It is therefore not necessary to log index spaces outside the partial replication. These spaces are reset to their old backup status or are recreated with RECOVER TO *DUMMY, in which case all indexes are recreated. However, for time reasons this is only recommended for small tables or where a small number of indexes are present.

RECOVER CATALOG_SPACE USING REPLICATION

The utility statement RECOVER CATALOG_SPACE USING REPLICATION allows the database administrator to repair the catalog space of the original. The replication can be a complete replication or any partial replication as the catalog space is always contained in the replication.

The same information as given in section “RECOVER CATALOG USING REPLICATION” applies to the logging files and the CAT-REC file, but only the CAT-LOG files are required.

In the case of RECOVER CATALOG_SPACE USING REPLICATION with RENAME the catalog space of the replication is subsequently no longer available. The replication must be recreated.

RECOVER SPACE USING REPLICATION

The utility statement RECOVER SPACE USING REPLICATION allows the database administrator to repair one or more user spaces of the original. The replication can be a complete replication or any partial replication. The user spaces to be repaired must be current in the replication.

The same information as given in the section “RECOVER CATALOG USING REPLICATION” applies to the logging files, but only the DA-LOG files are required. Information there regarding the CAT-REC file is of no significance here because the catalog space is not affected.

In the case of RECOVER SPACE USING REPLICATION with RENAME, the affected user spaces will subsequently no longer be available in the replication. These spaces can be added to the replication again with REFRESH SPACE.

Reset the original database using replications

- RECOVER CATALOG TO REPLICATION
- Reset using RECOVER CATALOG USING REPLICATION
- RECOVER CATALOG_SPACE TO REPLICATION
- RECOVER SPACE TO REPLICATION
- RECOVER SPACE USING REPLICATION TO mark

RECOVER CATALOG TO REPLICATION

The database administrator can use the utility statement RECOVER CATALOG TO REPLICATION to reset an entire original database with a complete replication to the status of the replication.

During this process, the CAT-REC file and the metadata are also adapted. The database administrator must save the status before RECOVER, if he/she wants to return to this status later.

When an original has been reset, all old log files are overwritten. In other words the status before the reset can no longer be reached with RECOVER.

If you use the COPY function, the replication is retained and can be used for further processing. If you use the RENAME function, the replication becomes the original and is no longer available as a replication. Replication and original must be located on the DBH user ID and the files of the original must no longer exist as otherwise you cannot recatalog the files.

If a partial replication is used, only the spaces contained in the partial replication are reset. These are the catalog space and the user spaces which are current in the partial replication. All other user spaces of the original must then be adjusted to the reset state. In the case of NO LOG index spaces, this can take place automatically by means of RECOVER CATALOG TO REPLICATION by specifying GENERATE INDEX ON NO LOG INDEX SPACE. All other spaces must be adjusted individually with RECOVER SPACE.

Reset using RECOVER CATALOG USING REPLICATION

A special case occurs when RECOVER CATALOG USING REPLICATION is used for a reset, and a CAT-REC file is specified which is as old or younger than the replication but older than the current version of the original. The original is not reset exactly to the state the catalog had when the CAT-REC copy was created, but one more CAT-LOG record will be applied. In this way it is possible to get closer to or even achieve the current state with the last CAT-REC copy if the CAT-REC file of the original is defective.

Example

A CAT-REC file is created and the new, already changed CAT-LOG file is entered. REFRESH REPLICATION takes account of this fact and does not evaluate this last entry since this file normally represents the current and therefore open CAT-LOG file.

However, if you specify this CAT-REC copy for RECOVER USING REPLICATION then this last entry is evaluated which, in this case, means that to all the entries, which have been logged in this CAT-LOG file and the DA-LOGS contained in it, the modifications are applied. If the CAT-LOG has not been changed, to anything the modifications are applied. If the CAT-LOG has been changed, the applying of the modifications ends at the point where the change occurred.

RECOVER CATALOG_SPACE TO REPLICATION

The utility statement RECOVER CATALOG_SPACE TO REPLICATION allows the database administrator to reset the catalog space of the original to the state of the replication. The replication can be a complete replication. It can also be any partial replication as the catalog space is always contained in the replication.

All user spaces of the original must then be individually adjusted to the reset state using RECOVER SPACE.

In the case of RECOVER CATALOG_SPACE TO REPLICATION with RENAME the catalog space of the replication will subsequently no longer be available. The replication must be recreated.

RECOVER SPACE TO REPLICATION

The utility statement RECOVER SPACE TO REPLICATION allows the database administrator to reset user spaces of the original. The replication can be a complete replication or any partial replication. The user spaces to be reset must be current in the replication.

The same information as given in resetting spaces with SESAM backups or foreign copies applies to building indexes on index spaces. The NO INDEX clause can be used to prevent the generation of indexes on dependent index spaces; the affected indexes are then defective.

In the case of RECOVER SPACE USING REPLICATION with RENAME, the affected user spaces will subsequently no longer be available in the replication.
In the case of RECOVER SPACE USING REPLICATION with COPY, the affected user spaces will be retained in the replication, but they cannot be updated any further in the replication because logging was interrupted as a result of the resetting in the original.

The following steps are necessary to add reset spaces to the replication again:

1. Copy spaces in the original
2. Create new CAT-REC.COPY file with the administration statement CHANGE-CATLOG
3. Update replication with REFRESH REPLICATION FOR SPACE without specifying the reset spaces
4. Extend the replication by adding the reset spaces with REFRESH SPACE from the previously created backup

RECOVER SPACE USING REPLICATION TO mark

The utility statement RECOVER SPACE USING REPLICATION TO *mark* allows the database administrator to reset a user space of the original and to apply modifications logged in the logging files up to the specified mark. A mark represents the time of an interruption to logging. It is created, for example, in LOAD OFFLINE before the user data is loaded.

The same information applies as for RECOVER SPACE TO REPLICATION.

Creating a separate original database from the replication

You can use both RECOVER CATALOG USING REPLICATION and RECOVER CATALOG TO REPLICATION to create a new original database. This new database must be entered in the SQL catalog list. RECOVER CATALOG USING/TO REPLICATION creates all the spaces of the new database from this replication provided these were current in the replication.

Procedure

The database administrator must use the administration statement ADD-SQL-DB-CATALOG-ENTRY to enter the new database in the DBH.
It is assigned the status LOCKED, because it does not exist yet.
The database is generated on the basis of the replication using RECOVER CATALOG TO/USING REPLICATION.

The following points must also be taken into consideration by the system administrator:

- He/she must change and adapt the storage groups in the new database.
- He/she must use COPY CATALOG to create separate SESAM backup copies, possibly followed by COPY SPACE since the old copies can no longer be accessed. He/she must then use the utility monitor to remove the metadata of old SESAM backup copies for the catalog space from the CAT-REC file. MODIFY RECOVERY must then be used to delete old references to SESAM backup copies for the user spaces
- If the new original has been created from a partial replication, the missing spaces must be created with RECOVER... TO '*DUMMY' or explicitly deleted using DROP SPACE FORCED.

For further information on how to proceed, see section “Using a replication to repair an original database” and section “Reset the original database using replications”.

Using a replication as a shadow database

The term “shadow database” refers to a replication of an original database that is maintained in parallel with the original. Using shadow databases, you can:

- free a DBH from pure read accesses during OLTP operation
- repair an original database
- reset an original database
- create an independent database

In this section, the illustration below will be used to describe how you can maintain a replication as a shadow database. For an explanation of the individual steps, refer to the next page.

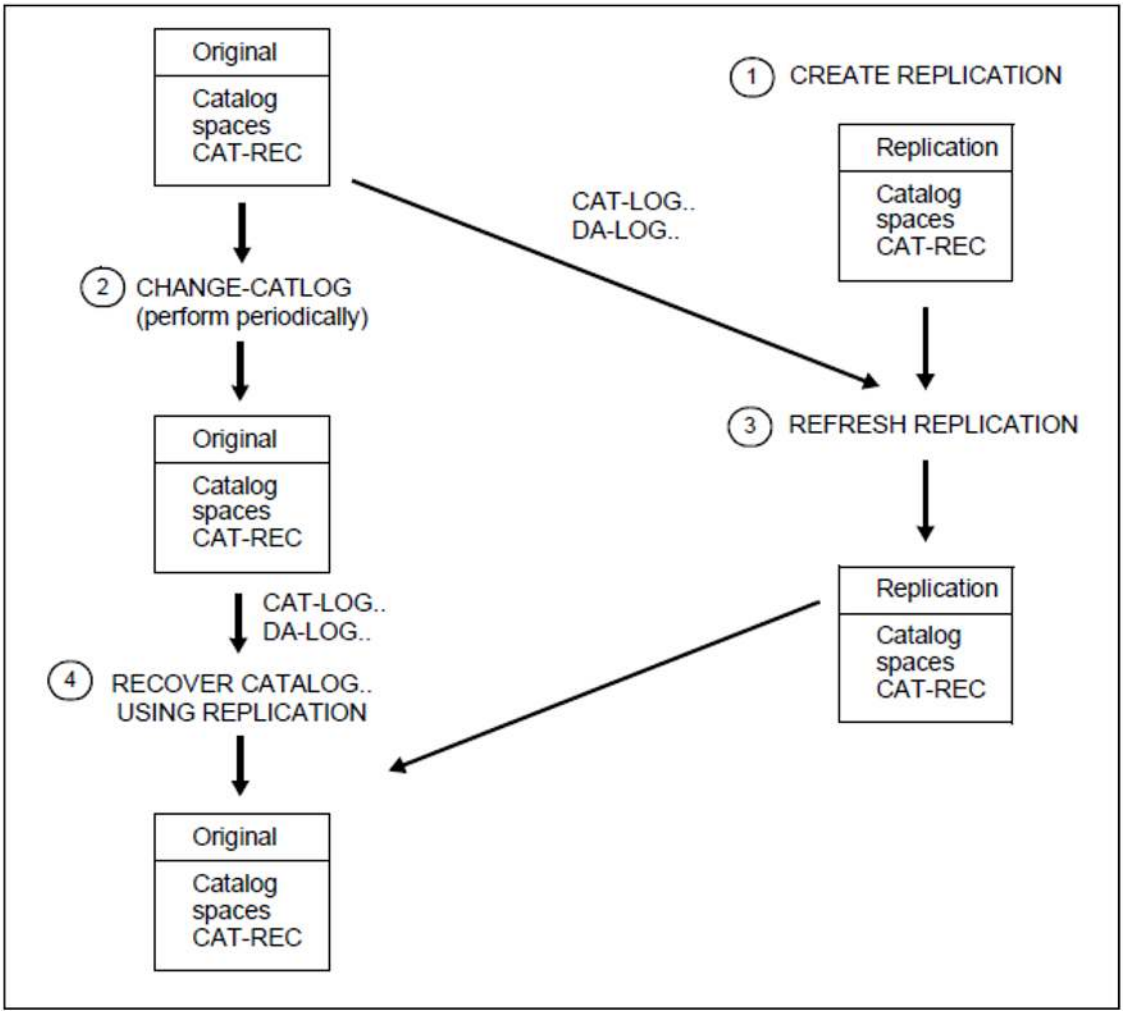

Figure 23: Using a replication as a shadow database

1. Use CREATE REPLICATION to create a replication from a SESAM backup copy or a foreign copy of the original database.

   If you want to use a replication as a shadow database you must update it at regular intervals. This process is described in steps 2 to 4:

2. Start a transfer of the log files from the original database at appropriate intervals.
3. Use the utility statement REFRESH REPLICATION to apply the modifications to the log files in the replication.

   If the original database is defective, repair it using the replication:
4. Use the utility statement RECOVER CATALOG..USING REPLICATION to repair the original database. When you do this, the replication serves as the backup copy. With this operation the modifications are applied to the log files created since the last update of the replication.

You can shorten the down time of the original database by using the RENAME parameter. When you do this, the replication is renamed and thus becomes an original database. The original database thus continues to be fully available and you can now create a new replication to be used as the shadow database.

You can further reduce the down time by repairing only the defective spaces with the utility statement RECOVER SPACE USING REPLICATION RENAME or the catalog space with RECOVER CATALOG_SPACE USING REPLICATION RENAME.
The user spaces will then be missing from the replication. These can be added to the replication with REFRESH SPACE. Alternatively, you can also recreate the replication.

Backup-specific files and catalog tables

File(s)	Used for	Assignment
TA-LOG1/2	Transaction logging	576/48
DDL-TA-LOG	Transaction logging in DDL	1536/384
WA-LOG	Restart	1320/48
DA-LOG	logging for user spaces	768/0
CAT-REC	controls recoveries; the file is present as ○ original ○ copy for updating a replication ○ replication	12/12
CAT-LOG	logging for the catalog space	768/0
PBI	the physical before image	576/22
RECOVERY_UNITS catalog table	administering SESAM backup copies of user spaces	
DA_LOGS catalog table	administering DA-LOG files	

Table 50: Backup-specific files

In contrast to the file assignment sizes shown in the table, in the event of a RECOVER, the files TA-LOG1/2 and WA-LOG are always created at the size which applied when the original catalog was processed.

Database Operation

The Data Base Handler, which is abbreviated to SESAM/SQL DBH or just DBH, is the central component for the control, execution and monitoring of database operation. Access to SESAM/SQL databases is impossible without the DBH.

The starting and termination of the DBH and the monitoring and administration of the current DBH session are tasks taken care of by SESAM/SQL system administrators.

This chapter provides basic information on database operation. It is intended specifically for readers who are interested in SESAM/SQL system administration.

The sections in this chapter cover the following subjects:

- the tasks and potential applications of the DBH
- communication between the application program, the DBH and the database
- The combining of applications in configurations
- Distributed processing with SESAM/SQL DCN
- DBH files
- means of increasing throughput
- means of monitoring and control available to system administrators.

The Data Base Handler

Overview

A SESAM/SQL application program is not able to access the data in a database directly. Instead, access to the data in a database is provided by the Data Base Handler (DBH). In conjunction with connection modules, the DBH provides an interface between the application program and the database.

The DBH performs the following tasks:

- it analyzes the statements from the application programs, determines the optimum access method and executes the statements
- it passes the results of database operations back to the application program
- it monitors all database activity
- it controls the service tasks for utility functions
- it logs important information on database activity in DBH-specific log files)
- it reports any errors that occur

Working with multiple databases

The SESAM/SQL-DBH supports operation with multiple databases. It can handle as many as 254 databases in parallel. Which means that each individual application program can access as many as 254 databases. If this proves to be insufficient in large applications, implementation of the SESDCN distribution component should be considered (see section “Distributed processing with SESAM/SQL DCN”).

Working with multiple requesters

Multiple requesters can work simultaneously with the SESAM/SQL database. Simultaneous access to the database by multiple requesters is managed and coordinated by the DBH.
SESAM/SQL allows an optional number of requesters. In timesharing, a requester is either a batch or an interactive program, a terminal or a pair consisting of the user ID and the terminal.

System architecture

SESAM/SQL is available in 3 product versions:

- with independent DBH there is
  - the standard edition of SESAM/SQL with single-task operation
  - the enterprise edition with multitask operation
- with linked-in DBH, there is the SESAM/SQL-LINK version.

The figure 24 compares linked-in DBH with independent DBH.

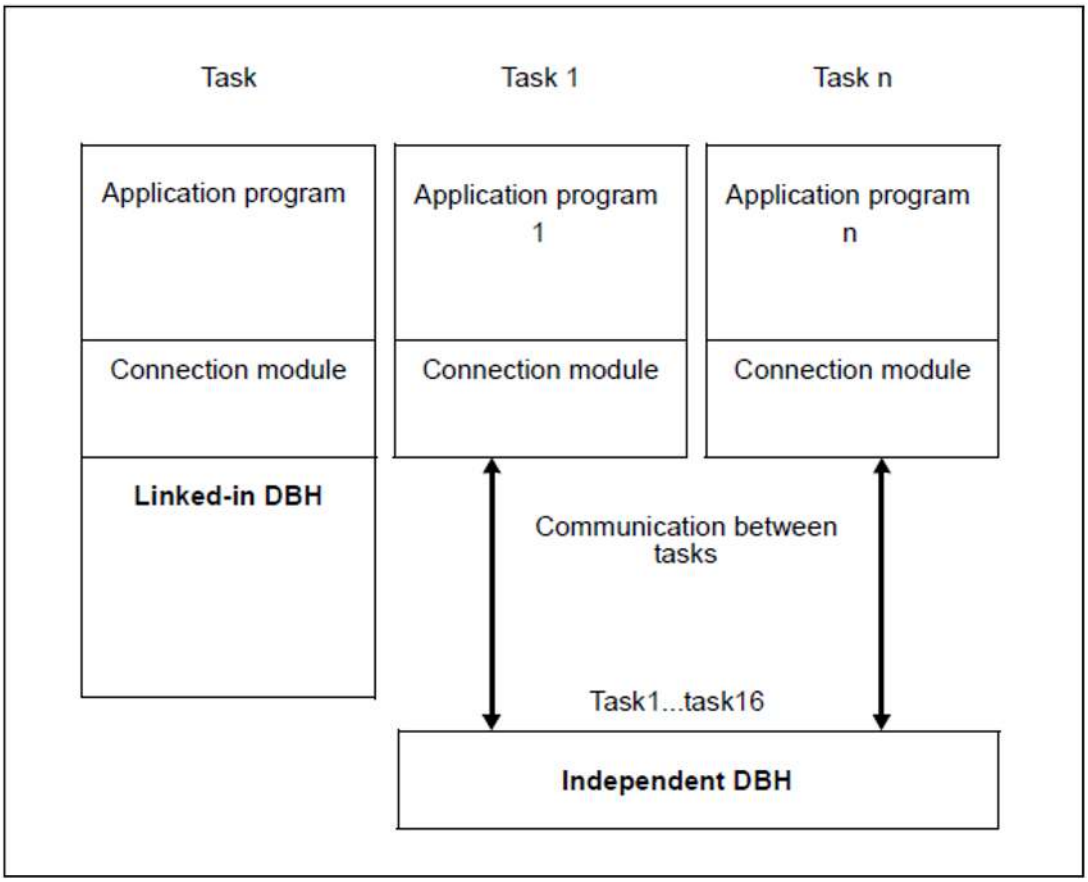

Figure 24: Comparison of linked-in DBH and independent DBH

Independent DBH

The figure 25 shows independent DBH and its interfaces. Independent DBH can access multiple databases and work with multiple requesters, as shown in this illustration.

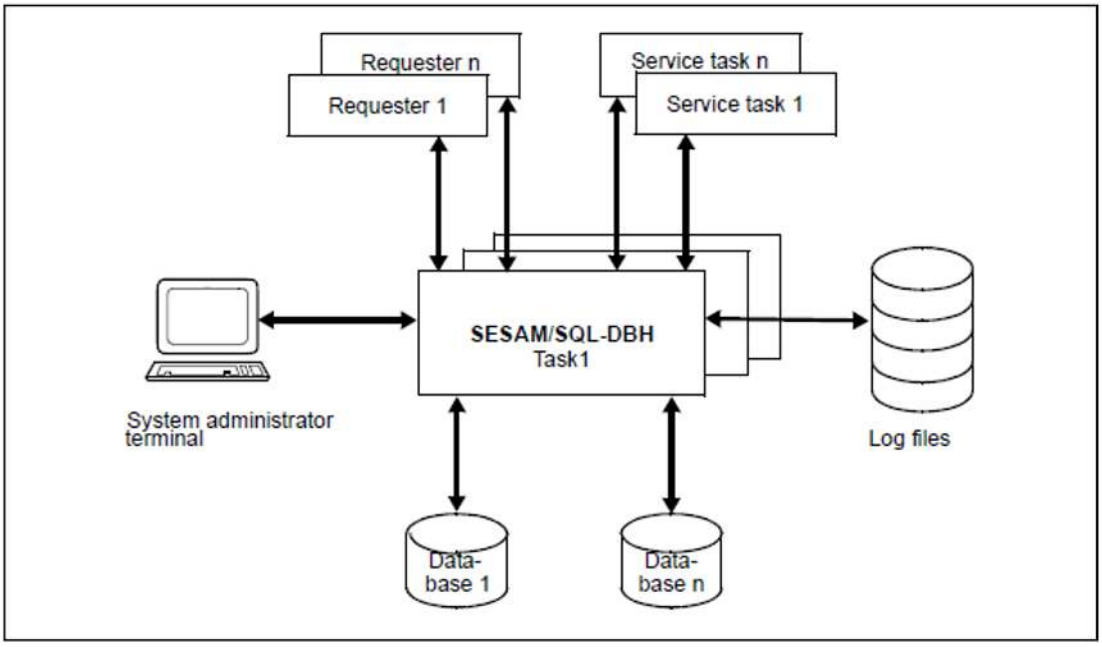

Figure 25: The SESAM/SQL-DBH and its interfaces

The **independent DBH** consists of one or more tasks (up to 16) (multitasking only possible with the enterprise edition). The tasks of this task family must be regarded as a unit, i.e. they all have the same functionality and are started and terminated together. For the system administrator, the task family presents itself with a “single-system image”. This means:

- The administrator starts SESAM/SQL in a task which, in turn, starts the other DBH tasks belonging to the task family according to the specifications made in the DBH-TASKS load option. The DBH is operational only when all DBH tasks have been started.
- The administrator only communicates with the task explicitly started, but administration statements always effect all tasks. It is not possible to address an individual task within the task family.
- In the course of an active session, all outputs from SESAM/SQL are made via the explicitly started DBH task.
- SESAM/SQL automatically distributes the load to the individual DBH tasks. The system administrator cannot influence load distribution.

Linked-in DBH

The figure 26 illustrates **linked-in DBH** and its interfaces. Linked-in DBH can access multiple databases, but in contrast to independent DBH, it can only work with one requester, as shown in this illustration. It only consists of one task.

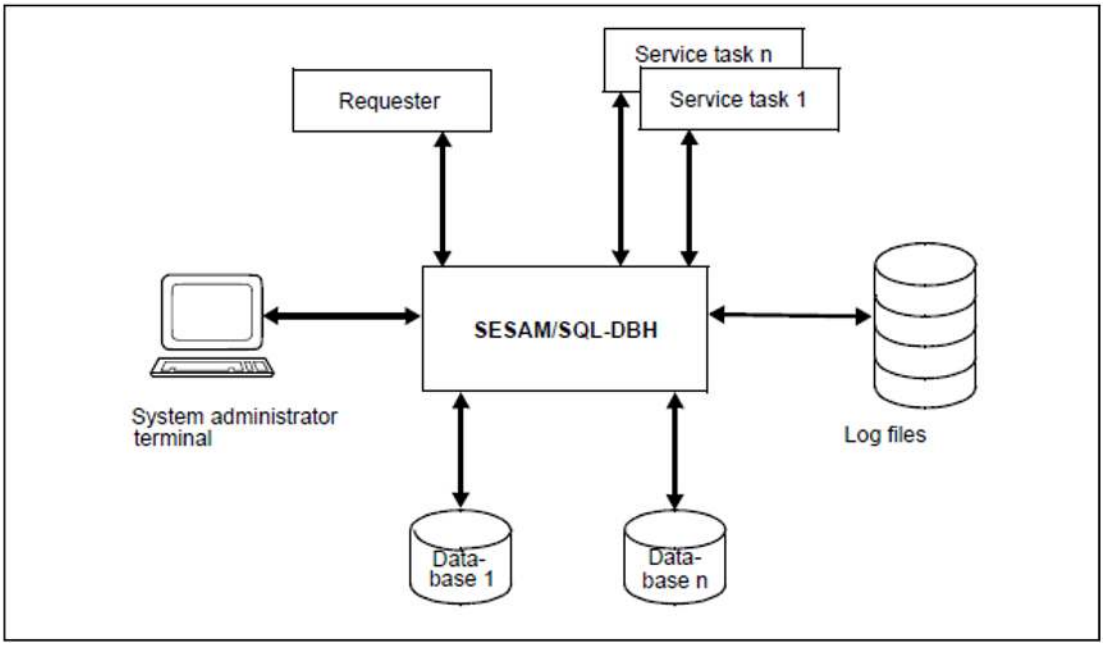

Figure 26: The SESAM/SQL-LINK-DBH and its interfaces

The linked-in DBH works exclusively with requests from a single application program and is linked directly to the application program. When the first statement is issued, the SESLINK connection module automatically loads the modules of the linked-in DBH from the SESAM/SQL module library and functions as a mediator between the application program and the DBH for the duration of the program run (see section “Connection modules”).
The program and the DBH share a common task. This means that no time is spent on handling communication between tasks when a database operation is requested. In addition, because the linked-in DBH only needs to handle requests from a single application program, it requires only a relatively short time to perform database operations. The linkedin DBH is only slower than the independent DBH if the update rate is high; this is because only one thread is generated for the linked-in DBH (see section “Multi-thread operation”).
A prerequisite for operation with the **linked-in DBH** is that the add-on product SESAM/SQL-LINK is installed. It constitutes the option of choice when one or more databases are to be accessed by a single application.

Means of control available to the SESAM/SQL system administrator

The starting and termination of the DBH and the monitoring of the current DBH session are the responsibility of the SESAM/SQL system administrator. A DBH session is the period between the starting of the DBH and its normal termination.

System administrators assign the databases to the DBH by means of entries in SQL database catalog lists or CALL DML table catalog lists (see the “Database Operation” manual). The DBH can access these databases in the course of the session.

There are three means of registering the required databases:

1. by means of DBH start statements
2. by means of the administration statements ADD-SQL-DB-CATALOG-ENTRY and ADD-OLD-TABLE-CATALOG-ENTRY during the current session (see the “Database Operation” manual)
3. by means of the utility statement CREATE CATALOG during the current session; this statement is used to enter newly defined databases (see the “SQL Reference Manual Part 2: Utilities” manual).

When starting the DBH, the system administrator can parameterize it by specifying DBH options. The system administrator can adapt resources, limit values, and work rules to the requirements of the current DBH session.

The SESAM/SQL system administrator can also have important information of the DBH session transferred automatically by email, see section “Sending important information of the DBH session by email”.

If necessary, system administrators can intervene in the current DBH session to control operations with the aid of administration statements and adapt the settings of the DBH options to changed conditions. The current DBH options can be saved in a file and used for the next DBH session.

The DBH administration commands RECONFIGURE-DBH-SESSION and RELOAD-DBH-SESSION are available to the SESAM/SQL system administrator for increasing the availability of the independent DBH, for importing a correction version while DBH operation is in progress, and for dynamic reconfiguration of the DBH session (see the “Database Operation” manual).
Both administration statements are executed without interrupting the DBH session. From the user viewpoint no DBH failure occurs.

Connection modules

Every application program that accesses SESAM/SQL databases has to be linked with a connection module. The connection module sets up communication between the SESAM/SQL Data Base Handler (DBH) and takes care of the transfer of data between the application program and the DBH.

In TIAM and DCAM applications, the application program passes its statements directly to the connection module in a subroutine call. The connection module converts these statements and passes them on to the DBH. The DBH executes the statements and transfers the results of database operations to the connection module, which in turn passes them back to the application program.

Each connection module can be used by XS-capable and non-XS-capable applications. It is even possible to switch addressing modes between two connection module calls within a program.

Part of an application program can run in XS mode while another runs in non-XS mode. Both parts can issue database calls.

In linked-in applications you must not mix SQL and CALL DML statements.

Connection module variants

The choice of connection module to be linked to an application program depends on whether the application program is to interact with the independent DBH or the linked-in DBH.

There is one connection module only per operating mode for application programs that interact with the independent DBH:
TIAM applications are linked with the connection module SESMOD,
DCAM applications are linked with SESDCAM, and UTM applications are linked with SESUTMC.

Application programs that interact with the linked-in DBH always have to be linked with the SESLINK connection module. SESLINK is already linked with the SEDI61L and SEDI63L utilities.

Connection module	DBH	Operating mode	Category
SESMOD	independent DBH	TIAM	DBCON
SESDCAM	independent DBH	DCAM (CALL-DML only)	
SESUTMC	independent DBH	UTM	
SESLINK	linked-in DBH		DBCONL

Table 51: Connection module variants

Connection module parameters

DBCON require various parameters in order to fulfill its functions. The DBH name and the configuration name are important connection module parameters. They assign the application program linked to the connection module to a DBH and a configuration. Beyond that, there are various other DBCON configuration parameters

TIAM, DCAM and UTM applications

Connection modules for TIAM and DCAM applications that work with the independent DBH (DBCON) are usually parameterized through the application program's configuration file (see section “The configuration file”).

The user enters the connection module parameters in the configuration file and assigns them before the application program linked with the connection module is started. The parameters are then transferred to the connection module in the course of the start procedure.

In cases in which the linked-in DBH is used, where the DBH is directly linked with the application program, the DBH name and the configuration name are contained in the DBH's configuration file as DBH start parameters (see section “The configuration file for DBH start parameters”). No configuration parameters are used in connection with DBCONL.

The list below shows the connection module parameters for TIAM, DCAM and UTM applications, in each case in alphabetical order.

- It begins with connection module parameters used in connection with TIAM, DCAM and UTM applications.
- These are followed by connection parameters that are only used in connection with both DCAM and UTM applications or with DCAM applications only. Connection module parameters that apply only to UTM applications are described in section “Starting a SESAM/SQL UTM application”.

All parameters are DBCON parameters.

Connection module parameters for TIAM, DCAM and UTM applications

Spaces are **not** permitted in parameter lines.

CCSN=*ccs-name*	Coded character set with which the user program works.

ccs-name: Name of the coded character set as defined in BS2000 (system component XHCS) or *NONE if no coded character set is to be sepcified for the user program.
Default: *NONE

If a coded character set is used (CODE-TABLE *ccs_name* clause in the CREATE CATALOG or ALTER CATALOG statement), the database's CCS name must match the user program's CCS name when the user program accesses the database. User program accesses with CCSN=*NONE or accesses by user programs from SESAM/SQL < V5.0 are rejected with SQLSTATE.

This check does not take place for the utility statements CREATE/ALTER CATALOG, CREATE/REFRESH REPLICATION and RECOVER CATALOG [SPACE].
If no CCSN was specified for the database, this check does not take
place either.

CNF=*k* Name of the configuration in which the application program is to operate

k A..Z, 0..9, ?
Default: ?

DIAG-DUMP=*(diag)* Criterion for the generation of a diagnostic dump

diag: {SQLSTATE=*state*|STATUS=*status*}

state: SQLSTATE with class and subclass (5 bytes);
status: CALL DML status with subnumber (4 bytes)

SQLSTATE can be partially qualified by entering "***" for the subclass.
CALL DML status can be partially qualified by entering "**" for the subnumber.

The first time the relevant error message is issued (SQLSTATE or CALL DML status), a user task dump will be output (see the "Database Operation" manual)

ISOL-LEVEL= *level* Default value for the isolation level of SQL transactions. Only effective in SQL applications.

level: {READ-UNCOMMITTED | READ-COMMITTED | REPEATABLE-READ | SERIALIZABLE}
Default: SERIALIZABLE

Valid for all transactions of the ESQL application program if the isolation level was not specified by the appropriate SQL statement in the program.

This parameter is only evaluated by TIAM and UTM applications.

NAM=*x*

Name of the DBH with which the application program is to work

x: A..Z, 0..9, ?

Default: ?

NOTYPE

The message SESAMxx not available ... is suppressed

NVT

The application can only run locally and it can only communicate with the DBH entered in the NAM parameter. This also applies when
SESDCN is loaded.

PREFETCH-BUFFER=*buffersize*

Size in KB of the buffer used for the prefetch.
Value range: 0 <= buffersize <= 4096

This parameter is only evaluated by TIAM and UTM applications.

PRIO-CHECK=*t*

Interval in seconds between two consecutive checks of the BS2000 task priority of the application program.

10 <= t <= 3600

Default: 300

PRIO-CHECK=OFF

The check of the BS2000 task priority of the application program will be deactivated following the first check.

SQLOPT-KEEP-JOIN-ORDER={YES | NO}

Default: NO

NO

The SESAM/SQL optimizer decides on the order of the join operands in the case of multiple joins.

YES

For the order of the joins, the SESAM/SQL optimizer keeps as closely as possible to the order of join operands defined in the SQL statement. This applies for explicit joins (e.g. table_1 JOIN table_2 ON search_condition).

SQLOPT-PREFERRED-JOIN-METHOD={SORT-MERGE | NESTED-LOOP | NONE}

Default: NONE

NONE

The SESAM/SQL optimizer decides on the join algorithm.

SORT-MERGE
NESTED_LOOP

The SESAM/SQL optimizer uses the specified join algorithm as far as possible.

.

> ¡ Use these two SQL notes with care! They apply for all applications which work with this configuration file. However, pragmas and annotations of the SQL statements have priority over these SQL notes of the configuration file.

TRACE,TYPE=*type*

Logging of the call or message trace is activated when the application program starts.

type: {CALL|MSG | (CALL,MSG)}
[,OUTPUT={SYSOUT | SYSLST | (SYSOUT,SYSLST)}]
Default for output (OUTPUT): SYSLST

Refer to the “Database Operation” manual for further information on call and message tracing.

DCAM- and UTM-specific connection module parameters

PUF=*p*

Maximum message length of DCAM or UTM applications in bytes
1 <= *p* <= 64000
Default: 4096

A message may be:

- the request (statement) of an application program to a SESAM/SQL DBH
- the response of a SESAM/SQL DBH to the application program

Since the length of SQL messages is difficult to estimate, we recommend you choose the value 64000 for *p* for applications with SQL statements.

TOTAL-APPL=*a*

Maximum number of SESAM/SQL user applications in the configuration (this only applies to applications that do not operate in
distributed mode)
1 <= *a* <= 128
Default: 64

TOTAL-USERS=*u*

Maximum number of SESAM/SQL user in the configuration (this only applies to applications that do not operate in distributed mode)
1 <= *u* <= 16000
Default: 128

The TOTAL-APP and TOTAL-USERS parameters apply to the entire configuration. The first application started determines the size of the user pool on the basis of these parameters. Any applications that are started subsequently simply make use of this pool. Any parameters specified in these applications are ignored.

DCAM-specific connection module parameters

NAME-APPL=*appl* The name of the DCAM application.

appl: 1-8 printable characters of any kind

Default: TSN=tsn

REQUEST-USERS=*r* The maximum number of concurrent asynchronous calls per task in
DCAM applications.
1 <= *r* <= 128
Default: 7

START Cold start for DCAM applications.
Default: warm start

The combining of applications in configurations

A number of separate SESAM/SQL applications can execute concurrently on a single computer; a separate DBH and its associated application programs are involved in the execution of each of these applications.

In distributed processing in which application programs work with more than one DBH, different DBHs, application programs and distribution components (SESDCNs) can be involved in the applications (see section “Distributed processing with SESAM/SQL DCN”).

In order to avoid conflicts, system administrators can combine applications that belong together in configurations, thereby isolating them from other configurations. For example, it is possible to combine test applications and production applications in separate configurations so that they can work completely independently of one another.

A configuration can contain the following components:

- In non-distributed processing, one or more DBHs with the relevant databases and one or more application programs
- In distributed processing, all the DBHs that belong together (each with the relevant databases), plus application programs and SESDCNs
- In addition, SESAM/SQL defines a pool to accommodate the administrative data used in communication and in distributed processing for each configuration with at least one transaction application, and for each configuration engaged in distributed processing.

All the components that are to be combined in a single configuration must be located on the same computer. Each DBH and each application program and, in distributed processing, each SESDCN, too, must be assigned to just one configuration at execution time.

Configuration name

The configuration name is used to assign the individual components to their respective configurations. The application program obtains the configuration name via a connection module parameter in the configuration file; the DBH and SESDCN obtain the configuration name via a DBH or DCN option.

The configuration name must be unique on the computer. Configurations located on different computers can use the same names. However, it is better to use configuration names that are unique across all systems in order to make it possible to perform an SESDCN recovery on a computer other than the one on which SESDCN was cold-started (see the “Database Operation” manual).

Communication between configurations

The individual configurations operate fully independently of one another. Without the distribution component SESDCN, an application program can only communicate with one DBH, namely with the one whose name was assigned as a connection module parameter and which belongs to the same configuration as the application program. With SESDCN, the configurations work independently of one another, but application programs can send requests to DBHs that do not belong to their own configuration, even if they are located on a different computer.

The figure 27 on the next page shows two computers, each with two configurations:

- Configuration M on computer 1 consists of application programs, SESDCNA, DBH 1, DBH2 and the pool. It works independently of configuration Z on the same computer, but it can communicate with configuration J and configuration Y on the other computer.

- Configuration Z on computer 1 consists of application programs and DBH3. It communicates with no other configurations.
- Configuration J on computer 2 consists of application programs, SESDCNX, DBH6 and the pool. It communicates with configuration Y on the same computer, and with configuration M on computer 1.
- Configuration Y on computer 2 consists of application programs, SESDCNB, DBH7, DBH8 and the pool. It communicates with configuration J on the same computer, and with configuration M on computer 1.

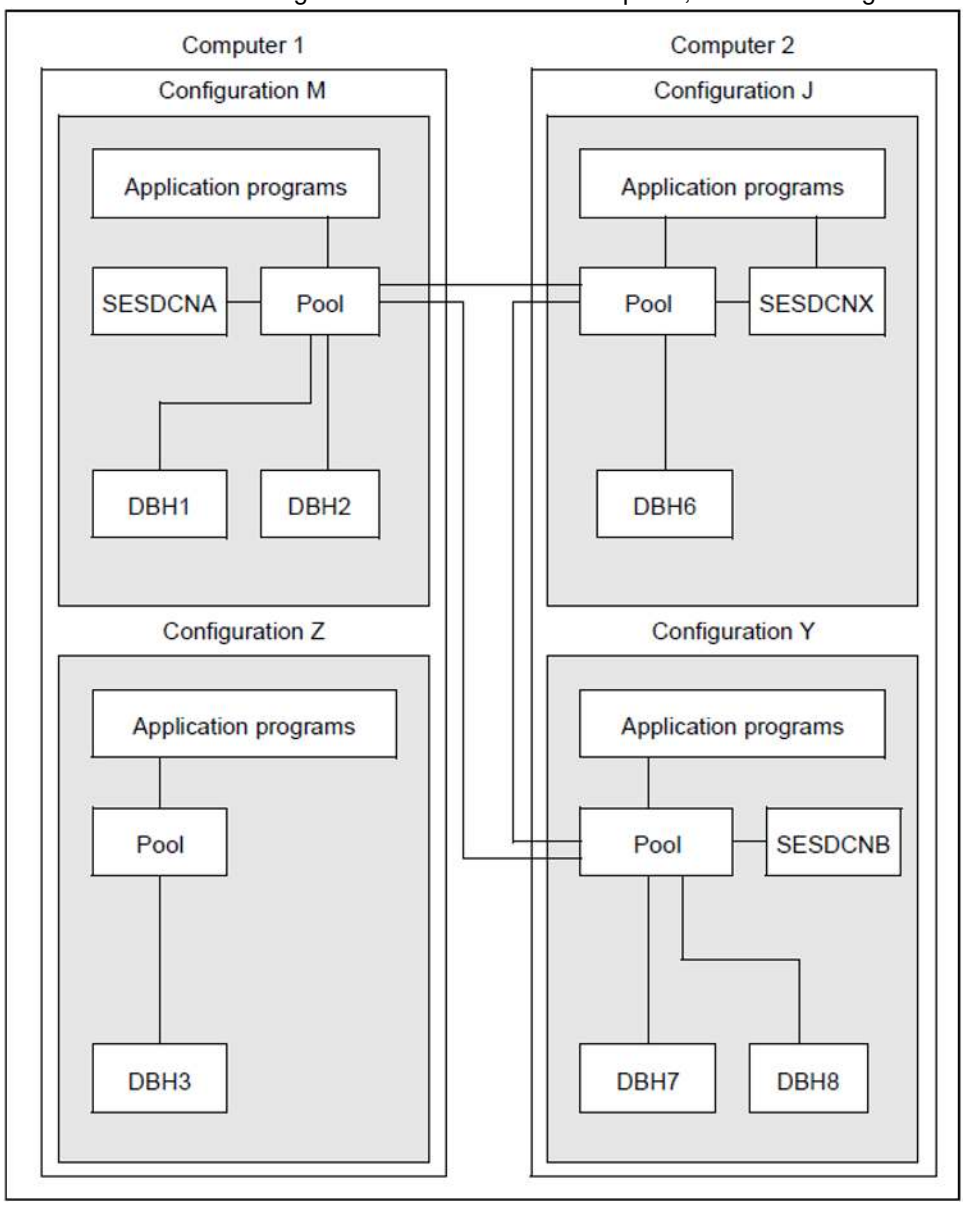

Figure 27: Example of communication between different configurations

Communication with virtual hosts

When operating a SESAM configuration you can also use a virtual host. In this case, SESAM/SQL uses the virtual host rather than the real host as its communication partner.

As a result, you can make the distribution rules and SQL system accesses independent of real host names.

To enable SESAM/SQL to use a virtual host, you must enter a corresponding statement for the application SES091 *cnf* in the file $TSOS.SYSDAT.BCAM.APPLICATIONS (*cnf* is the configuration name).

Example

Entry with *cnf = P* in the $TSOS.SYSDAT.BCAM.APPLICATIONS file:

```
NEA SES091P VIRHOST
```

After this SESAM/SQL will use the virtual host name as the host name in the user identification (<user-identification>). Without this assignment the user identification will continue to be via the real host name.

> If the entry was made in the $TSOS.SYSDAT.BCAM.APPLICATIONS file, the allocation file (of applications to virtual hosts) may not be modified dynamically (see the APPLICATION-TABLE parameter in the BCAM commands DCSTART and DCOPT).

In TIAM and DCAM applications the virtual host is used automatically.

In UTM applications, the parameter HOST NAME in the MAX option must be supplied with the name of the virtual host during UTM generation. As a result, the UTM applications can also use the virtual host and communicate with the SESAM configuration. This is absolutely necessary because, just like in the situation with real hosts, all the partners in a configuration must use the same host.

The virtual host must not be deactivated while a SESAM configuration is operating on it. If this does occur, this will be identified as a failure of the communication partners even if the real host is still active.

Distributed processing with SESAM/SQL DCN

In distributed processing with SESAM/SQL-DCN, an application program can work with more than one SESAM /SQL DBH during a single session. The application program can work with the different DBHs within the same configuration, in more than one configuration on the same computer, or in more than one configuration on more than one computer. Cross-computer interaction between the application and different DBHs allows the application program to access databases located on other computers. Here, one refers to distributed databases.

In distributed processing, the computers involved are referred to as home or remote computers. The terms “home” and “remote” refer to the application program's view of the system.

Home computer

The computer on which the application executes is referred to as the home computer.

Remote computers

In a computer network, all computers other than the application's home computer are referred to as remote computers.

Applications for distributed processing

Distributed processing based on SESAM/SQL-DCN can be used in the following applications:

- When distributed databases are used, it is possible to work with databases on different computers within one session. The data can be stored on the computer on which it is required most frequently. The SESAM/SQL database is the distribution granularity.
- Configurations can communicate with one another (on one computer, or on other computers across a network).

The following features characterize distributed processing based on SESAM/SQL-DCN:

- The application programs' independence of database locations:
  The location of a database is irrelevant for the application programs. It is determined internally by the system and is not addressed explicitly by the application programs. The programs are therefore not affected by changes to the location. In particular, application programs used to date in a non-distributed context can be used in a distributed context without modification. The distribution of the databases is something that is transparent to the application programs.
- Two-phase commit at the end of a transaction:
  Multi-computer transaction logging guarantees the security of transaction-oriented network-wide processing of data. This allows updates to multiple databases within one transaction, as well as automatic recovery after a system failure (see section “Restart”).
- Deadlock and longlock detection:
  Deadlocks and longlocks are recognized and resolved by the system beyond the boundaries of a single computer.

Distributed databases offer the following advantages:

- Higher performance:
  Throughput may be improved by processing user requests on different computers.
- Higher level of availability:
  Several data-processing systems that can, if necessary, be operated stand-alone, ensure that the failure of one computer does not affect all of the data-processing procedures.
- Adaptable organization:
  Working procedures need not be centered on a single computer center.
- Flexibility in terms of expandable capacity.

Operating modes of a SESAM/SQL-DCN application

Distributed processing with SESAM/SQL-DCN can be carried out in timesharing mode (TIAM) and in transaction mode. Timesharing mode can be implemented through the use of the Universal Transaction Monitor openUTM or the data communication system DCAM.

SESDCN distribution component

The distribution component SESDCN is the central element in the product SESAM/SQL-DCN. SESDCN has to be loaded in all configurations involved in distributed processing. The configurations in question can all be located on the same computer, or they can be located on different computers. If an application accesses a database which is under the control of a DBH and the DBH belongs to a different configuration from the application program, the access is referred to as a remote access. By contrast, if an application program only accesses a database which is under the control of a DBH that belongs to the same configuration as the application program itself, the access is referred to as local access.

The distribution component SESDCN is responsible for the following different tasks:

- building and administration of distribution rules
- receipt and forwarding of remote accesses to the appropriate SESAM/SQL-DBH
- administration of database operations (e.g., the execution of administration statements and the detection of locks)
- monitoring of local DBHs and application programs
- coordination and execution of SESDCN recovery
- Deadlock detection

Distributed Data Base Handler

A SESAM/SQL DBH must be loaded in each configuration or on each computer in or on which an application program accesses one or more databases. The DBH controls and administers database accesses. The DBHs engaged in distributed processing are referred to collectively as the Distributed Data Base Handler, or DDBH for short.

Distribution of statements

In distributed processing with SESAM/SQL-DCN, the connection module DBCON sends SQL and/or CALL DML statements issued by an application program during a session to the various DBHs for processing.

Each statement that does not complete a conversation or transaction is sent in full to a specific DBH by the connection module. A statement is therefore only processed by one DBH. This technique is known as function shipping. Only statements that terminate the current transaction (COMMIT, ROLLBACK in SQL, ETA in CALL DML) are sent to all DBHs involved in that transaction.

Defining distribution in the distribution rules

When statements are distributed, the application program does not see which DBH is processing what statement. In distributed processing, just as in non-distributed processing, the application program need only be aware of the database in which it addresses a specific table, for example.

What the application program needs no know is

- which computer the database is stored on,
- which configuration the database is assigned to,
- which distribution components the connection module DBCON is sending the statements to (during remote access only), and
- which DBH is processing the statements.

The connection module obtains all this information itself through the distribution rules which define the “access path” for each database; this path comprises the processor name, the configuration name, the DCN name and the DBH name. Because the connection module uses the database name as a search criterion, the database name must be unique throughout the distributed application, i.e. it must be unique network-wide.

Distribution rules are defined and administered using the distribution component SESDCN. System administrators define them using the SESDN control statements ADD-DISTRIBUTION-RULE-LIST and ADD-NETWORK-LINK-LIST. SESDCN builds the distribution rules on the basis of the statements is receives. Using administration commands, system administrators can also tune the distribution rules during live operation. SESDCN is administered via the same interfaces as the DBH.

Increasing throughput by loading more than one SESDCN

In most cases it suffices to load SESDCN once per configuration. However, if run-time response slows because SEDCN receives a large number of remote requests, the overall throughput in connection with remote accesses can be increased by loading more than one SESDCN. The transmission components are used more efficiently if remote requests are processed concurrently.
If more than one SESDCN is loaded, it is important to remember that only one SESDCN can be assigned to one DBH.

Interaction between SESAM/SQL-DBH and SESAM/SQL-DCN

Distributed processing with SESAM/SQL-DCN can be used to work with different configurations locally on one computer or with different configurations on other computers, too. Communication between the application program, SESDCN and the Distributed Data Base Handler works in both cases according to the same principle, which is outlined in figure 28. This means that the configurations A and B in figure 28 can be assumed to be on the same computer or on different computers.

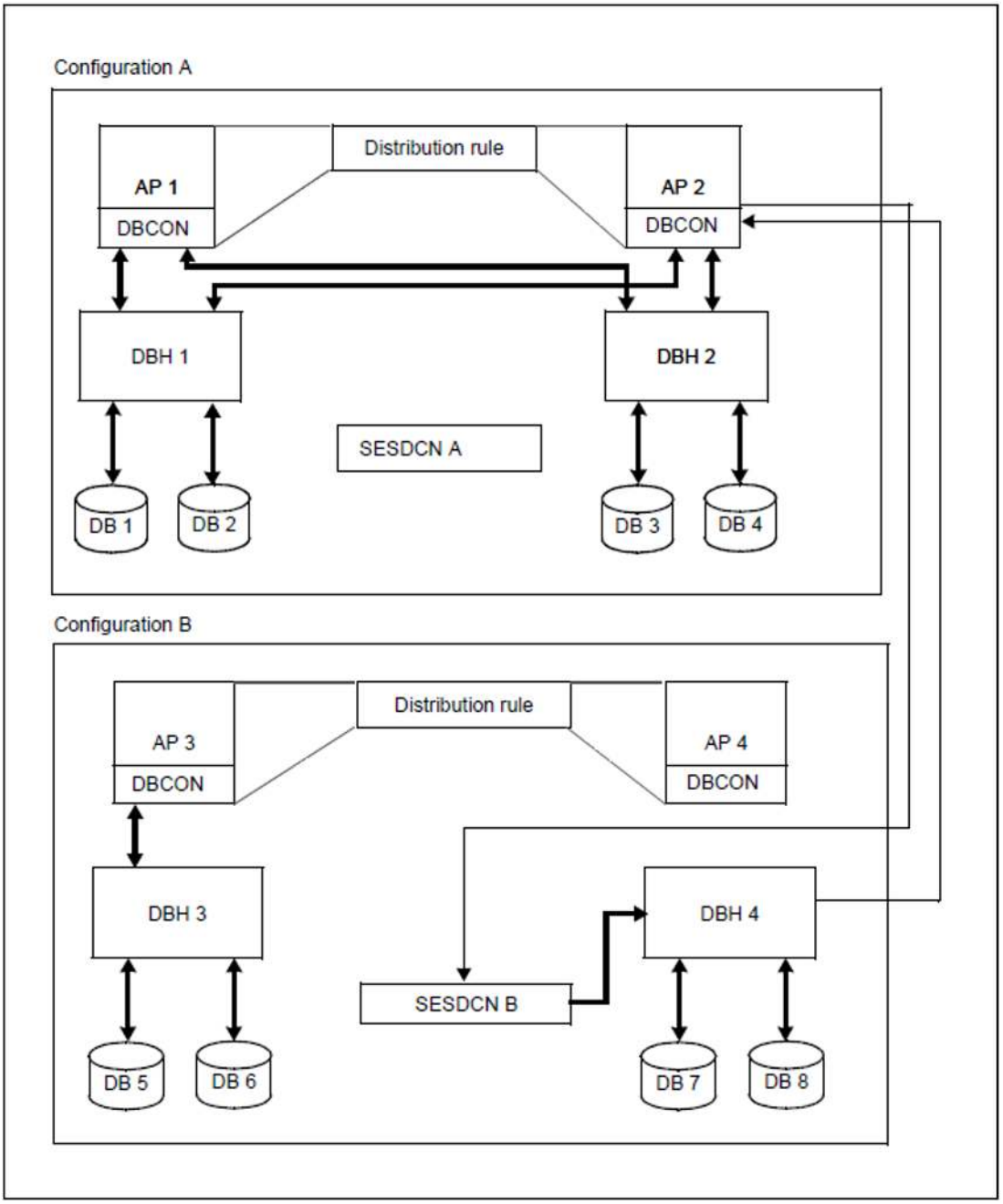

Figure 28: Distributed processing

Every application program must be linked to a DBCON connection module. The connection module establishes the link needed to work on the desired database between the application program and the DBH.

Here, there are two specific cases to differentiate between:

- Local access (the application program and the database are part of the same configuration):
  In this case the connection module sends the statement to the DBH specified in the distribution rules. For example, in figure 28 above, the application program AP 1 sends a statement to DB 3. Local access is represented by a continuous line.
- Remote access (the application program and the database are part of different configurations):
  In this case the connection module sends the statement to the SESDCN distribution component specified in the distribution rules. At the same time, the connection module tells the distribution component which DBH is specified in the distribution rules as being responsible for working on the database. The distribution component then sends the statement to the relevant DBH. Once the statement has executed, the DBH sends the response directly to the connection module of the application program that issued the request.
  This is the case, for example, with the application program AP 2 and the database DB 8 in figure 28. The remote access is represented by a dashed line.

Data security in distributed processing

This section examines aspects of the backup concept that apply specifically to distributed processing with SESAM /SQL-DCN. Cross-configuration transaction logging ensures that no data losses or inconsistencies will be incurred by the failure of a component in the computer network.

Distributed transactions

A transaction is regarded as distributed if more than one DBH is involved in processing it. A distributed transaction is processed in three steps (see figure 29).

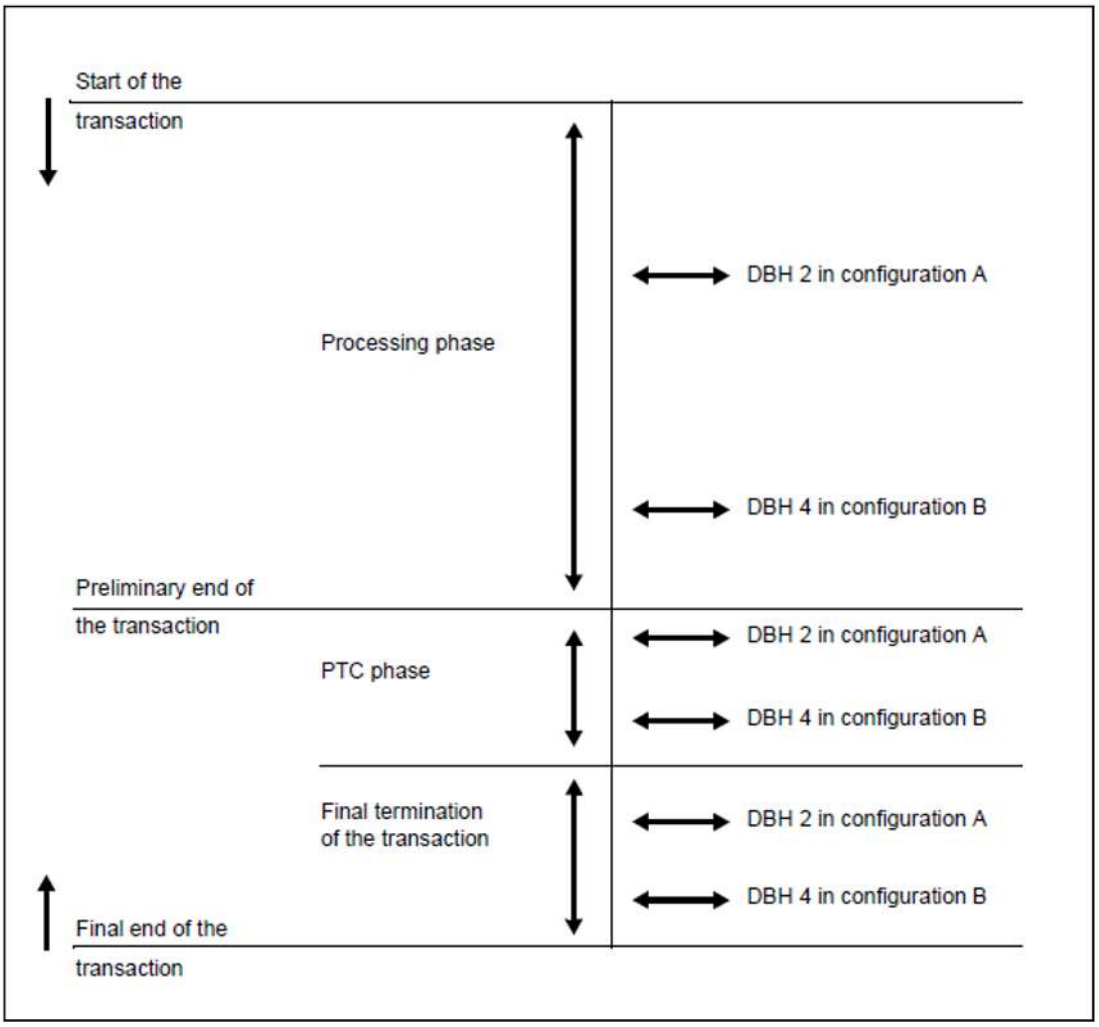

Figure 29: Execution of a distributed transaction

”Configuration A”,
“Configuration B” and
“DBH 2” and
“DBH 4” in figure 29

refer to the situation illustrated in figure 28.

Processing phase

The first SQL statement that introduces the transaction, or the BTA statement in CALL DML, and the transaction's other SQL or CALL DML statements are executed.

PTC phase (Prepare To Commit)

The processing phase within the transaction has finished. The changes made are logged to guard against system failure, but they can still be reversed. The final end of the transaction cannot be finally completed until “prepared to commit” has been confirmed by all the DBHs involved.

“Final termination of the transaction” phase

The transaction is irrevocably terminated: All changes are committed or reset.

The master DCN and the backup file

Master DCN

If a computer crashes, SESDCN has to be restarted in order to maintain transaction consistency (see “SESDCN restart”). The coordination and execution of the restart is SESDCN's responsibility. SESDCN also monitors the creation of backup files. If SESDCN was loaded more than once in the configuration, the SESDCN loaded first carries out this task. The SESDCN that controls the restart is referred to as the master DCN.

The backup file

A backup file containing the restart data has to be available for a restart after a system failure to succeed.

The master DCN checks whether the backup file already exists.

- If no corresponding file exists yet, the backup file is created under the file name specified via the link name SESDLG the first time the master DCN is started or, if no assignment exists under this link name, under the default file name in the user ID in which the master DCN is loaded.
  The file can be supplied with a password using the SET-DCN-OPTIONS statement SESDLG-PASSWORD. Password protection applies both to read and to write accesses.
- If a backup file already exists and is not yet password-protected, password protection can only be achieved using the BS2000 command MODIFY-FILE-ATTRIBUTES. Password protection should cover both read and write access here, too. If it opens an existing backup file, the master DCN determines whether a restart is necessary.

The naming conventions, the access method and the default assignment for SESDCN's backup file are as follows:

Standard name	SES.DLG*c*
Link name	SESDLG
Access method	PAM-shared (or SHAREUPD)
Primary/secondary assignment	9 / 24

The name of SESAM/SQL-DCN's backup file includes the configuration name “*c*”, which assigns it to the configuration to which the master DCN belongs.

System administrators can also create the backup file under a name of their choice using the SDF command CREATE-FILE, and can assign it with the link name SESDLG.

SESDCN restart

The backup file contains information indicating whether the last SESDCN session terminated correctly. The next time SESDCN starts, the master DCN opens the backup file and checks whether a restart is necessary. A prerequisite for restarting SESDCN is that the DBHs involved in distributed processing are capable of restarting. An SESDCN restart constitutes the logical continuation of the previous SESDCN run; as a result, all the load options used in the previous run apply again.

The handling of distributed transactions

The way in which SESDCN handles distributed transactions during a restart depends on the phase of processing in which the previous SESDCN run was aborted, and on the result of the DBH restart performed prior to the SESDCN restart.

The way in which the DBH handles the transactions varies, depending on the time at which the system failed:

- The DBH rolls back a transaction during a restart if the transaction was in the processing phase and PTC phase had not yet been completed from the DBH's point of view.
- Transactions that had completed the PTC phase when the system failed remain in this state when the DBH restarts. The decision on whether to roll back or terminate such transactions is not made until SESDCN restarts and the information in the backup file is evaluated.

Restarting SESDCN on a remote computer

SESDCN can be restarted on the computer on which it was originally started (the cold-start computer) or on a different computer. A prerequisite for performing the restart on a different computer is that configuration names that are unique network-wide were assigned when the system was generated.

In order to perform the restart on another computer, the whole configuration has to be moved to that computer. This means that all the application programs, SESDCNs, DBHs, the backup file, and all the DBH-specific files that belong to the configuration have to be moved to that computer. All the required tasks must be restartable on the new computer. In addition, the system environment for the DBH must be the same on the new computer, i.e. the DBH user IDs must be the same on the new and old computers. In the same way, the DB user IDs must be the same on the new and old computers.

Updating the distribution rules

If the SESDCN restart takes place on a different computer, the location of the restart configuration within the computer network changes from the point of view of the other configurations. For this reason, the restart configuration's master DCN automatically replaces the old processor name in the distribution rules for the remote computers it knows and can access with the name of the computer on which the restart takes place. This ensures that communication is still possible with the configurations on these machines.

If not all the remote computers in the network are accessible to the master DCN at the time of the SESDCN restart, the master DCN cannot change all the distribution rules automatically. If this is the case, system administrators have to update some of the distribution rules by issuing the appropriate administration statements (see the “Database Operation” manual).

If a number of different configurations perform their SESDCN restarts on different computers after a crash, a full update of the distribution rules by the relevant master DCNs is likewise impossible. In this case, system administrators have to update a number of distribution rules with SESDCN administration statements.

Cross-version distributed processing

The SESAM/SQL versions V4.0 and higher can communicate with each other without restriction and with a maximum message length of 64,000 bytes.

In the case of cross-version distributed processing, the application and the DBH must always be located in different configurations. The configurations may be located on the same computer or on different computers.

User program accesses with CCSN=*NONE or accesses by user programs from SESAM/SQL < V5.0 to a database with CCSN are rejected with SQLSTATE. Cross-version distributed processing is only possible for databases without CCSN (CODE_TABLE has the value _NONE_).

DBH file handling

Operations involving the DBH are performed via files each of which has a separate task. You can use the configuration file to assign start parameters to the DBH. The CATID list and MAIL parameter file contain control information for the DBH. During the session, the DBH also creates files in which it saves information relating to its work. Among other things, these files make it possible to safeguard operating steps, trace user operations and influence the behavior of the DBH during operation.

Specifying a CATID list

With SESAM/SQL-Server, it is possible to restrict internal file searches to certain CATIDs. To do this, you enter the required CATIDs in a file in list form. You can then assign this file to the DBH using the ADD-FILE-LINK command and the link name SESAMCID.

The CATID list is evaluated the first time a file is accessed. Any changes to the file take effect when the DBH is restarted or the administration statement MODIFY-CATID-LIST is issued. The CATID list is output using the administration statement SHOW-CATID-LIST (see the “Database Operation” manual).

To create a CATID list, you must use a SAM file with variable record length. Each record may contain only one CATID which is 1 to 4 characters in length. Only the characters A ... Z and 0 ... 9 are permitted (no colons or spaces). A maximum of 50 CATIDs can be evaluated. No wildcards may be specified.

SESAM/SQL does not check that the contents of the CATID file are correct. If the file contains syntactically incorrect entries or CATIDs that are not known to the BS2000 system then an error message is output to SYSLST. If the CATID file is empty then only the default pubset is used.

The CATID of the default pubset for the user ID must always be present in the list. If the user does not specify the CATID of the default pubset then it is automatically entered by SESAM/SQL.

If a new file is to be created, SESAM/SQL uses the following procedure to check whether this file already exists:

- SESAM/SQL searches the file for all the CATIDs in the CATID list. If the file is found for one of the CATIDs then it is not newly created but used as it is.
- If the CATID of the file that is to be created is not in the CATID list, an error message is output.
- If the CATID of the file that is to be created is present in the CATID list and the file is not present for any CATID in the CATID list then it is newly created.

If the file searched for cannot be assigned unambiguously, a message to this effect is issued.

You should assign the utility monitor the same CATID list as the DBH to ensure that the same data is accessed in all cases.
You assign the CATID list to the utility monitor using the ADD-FILE-LINK command and the link name SESAMCID. The CATID list is evaluated the first time a file is accessed. Changes in the CATID list do not take effect until the utility monitor is restarted. You use the administration statement MODIFY-CATID-LIST to edit the DBH's CATID list. This command does not affect the CATID list that was assigned to the utility monitor.

The configuration file

A configuration file is a BS2000 file created by the user. In the configuration file the user can enter start parameters for the DBH. Users can also assign start parameters for SESDCN, for the utility monitor and for the connection module of the application program (DBCON) via the configuration file.

SESAM/SQL differentiates between several types of configuration file:

1. the configuration file for DBH start parameters
2. the configuration file for SESDCN start parameters
3. the configuration file for the DBCON or utility monitor start parameters

A global configuration file can be used to group together the configuration parameters for all or several different components.

All the files are SAM files which can be assigned any appropriate name and in which users specify the relevant parameters.

In 1. and 2. above, comment lines must start with “//REMARK”; in 3. they must start with “REMARK”. Strings of any alphanumeric characters can be used in comment lines. The end of the comment is the end of the line. Comment lines must not come between continuation lines, otherwise they will be interpreted as part of the statement.

Before starting the DBH, the user assigns one of the following files:

- either the configuration file for DBH start parameters with the link name SESCONF or via the system file SYSDTA
- or a global configuration file with the command CONNECT-SESAM-CONFIGURATION.

The same process is performed for SESDCN, DBCON and the utility monitor.

During the startup procedure, the parameters entered are passed to the DBH or to DBCON or to the utility monitor.

If a parameter to be evaluated in the configuration file contains a syntax error. The file is closed and the component started issues its own error message.

The configuration file for DBH start parameters

This type of configuration file serves as an input file for the DBH's start parameters and contains nothing but DBH start statements and options. They specify the resources, limit values and work rules for the current session.

The individual DBH start parameters' syntax and meanings are described in section “Controlling and monitoring the session”.
Information on the entry in the configuration file and a detailed description of DBH start parameters are provided in the “Database Operation” manual.

You can create a DBH configuration file conveniently with all DBH options and its current values in the active DBH session using the administration statement SAVE-DBH-OPTIONS, see the “Database Operation” manual.

Global configuration file

This type of configuration file groups the configuration parameters for several components into a single file, e.g. all the DBHs, SESDCNs and DBCONs for a configuration. The appropriate framework must have been put in place when SESAM/SQL was installed (see the “Database Operation” manual).

The parameters for each component are grouped into blocks in the file. Each parameter block is introduced by a one-line entry CONFIGURATION-LINK=*component*. *component* identifies the component to which the parameter block applies. You assign the file to the component for which the file contains parameters with the command CONNECT-SESAM-CONFIGURATION (see the “Database Operation” manual).

The rules applying to configuration files for individual SESAM/SQL components such as the utility monitor or SESDCN also apply to the use of the different parameters in the global configuration file.

The particularities relating to the entry of start parameters for SESDCN, DBCON and the utility monitor are described in the section "The configuration file for DBH start parameters ".

Example of a global configuration file

```
//REMARK LOAD OPTIONS FOR THE DBH *********************************
CONFIGURATION-LINK=SESDBB1
//SET-DBH-OPTIONS-
//    DBH-IDENTIFICATION=*PARAMETERS(-
//        CONFIGURATION-NAME=Z-
//       ,DBH-NAME=X-
//    )-
//   ,ADMINISTRATION=*PARAMETERS(-
//        ACCOUNTING=*PARAMETERS(-
.
.
.
//REMARK K_2_DBH_NEW_1 *******************************************
//ADD-SQL-DATABASE-CATALOG-LIST-
//    ENTRY-1=*CATALOG(-
//        CATALOG-NAME=AUFTRAGKUNDEN-
//       ,USER-ID=KENN1-
//       )-
.
.
.
//REMARK LOAD OPTIONS FOR USER PROGRAMS **********************
CONFIGURATION-LINK=SESDBB2
REMARK CONFIGURATION DATA ***************************************
CNF=Z
NAM=X
SQLOPT-PREFERRED-JOIN-METHOD=SORT-MERGE
PUF=64000
TOTAL-USERS=00128
REMARK A DB PROCESS IS ASSIGNED EXACTLY ONE UTM PROCESS ******
UTMVG=JA
```

Start parameters for SESDCN

Here you define the distribution rules and access paths for the current SESDCN session.

The individual SESDCN start parameters' syntax and meanings are described in section “SESDCN control statements and options”. Information on the entry in the configuration file and a detailed description of SESDCN start parameters are provided in the “Database Operation” manual.

Alongside a global configuration file, you can also use a specific configuration file as the start parameter input file for SESDCN. This type of configuration file then contains only SESDCN control statements and options.

Start parameters for DBCON and the utility monitor

You can assign start parameters for the connection module (DBCON) of an ESQL or CALL DML application, or for the utility monitor via a global configuration file or via a specific configuration file. This type of configuration file can contain parameters for the utility monitor, for DBCON or both.

Attention should be paid to the following when entering parameters in configuration files of this type:

- Only one parameter must be entered per line or record.
- The parameter must begin in column 1 and must not contain blanks.
- Each parameter must only occur once in the file.
- Parameters can be entered in any order.

A DBCON or the utility monitor only ever evaluates its own specific parameters. To make it possible to differentiate, utility monitor parameters begin with the prefix “SEE”; DBCON start parameters have no prefix. In addition to the parameters flagged with "SEE", the utility monitor also reads the DBH name and the configuration name (connection module parameter NAM or CNF).

DBCON parameters

The DBH name and the configuration name are important DBCON parameters (connection module parameters). These two parameters assign the application to which the connection module is linked to a DBH and to a configuration (see section “Connection module parameters”).

Only applications that work with an independent DBH, not linked-in applications, have a configuration file containing connection module parameters. With linked-in applications, the key parameters (the DBH name and the configuration name) are contained in the linked-in DBH's configuration file as DBH options.

In TIAM and DCAM operations, users must assign the connection module parameters via the configuration file before starting the application, unless they wish to use the preset default parameters. CALL DML application users can also pass certain connection module parameters via special CALL DML calls (see the “CALL-DM Applications” manual). However, the configuration file offers an advantage in terms of flexibility because it allows the application program to be assigned to a different DBH or configuration if necessary, without needing to change the source code or re-link the application. Configuration parameters for a number of components can be combined in a single global configuration file.

The same configuration file can be used for TIAM applications, UTM applications and DCAM applications. If the configuration file for a TIAM application contains DCAM-specific or UTM-specific connection module parameters, these are ignored. The same applies to DCAM or UTM applications

The SQL notes for TIAM and DCAM applications are listed with their respective syntax and meanings starting.

Utility monitor parameters

The utility monitor parameters (the so-called configuration data) control how the utility monitor executes.

The authorization key for working with the utility monitor or the authorization key for administration is a mandatory parameter and has to be specified by users via the CALL interface. There are also various optional parameters that can be specified.

If users do not specify a configuration file, they must supply the configuration data (or at least their authorization to use the utility monitor) when the monitor executes.

You should enter the DBH name and the configuration name (connection module parameter NAM or CNF) in the configuration file since these specifications are displayed in the CNF - CONFIGURATION screen even though they cannot be edited there. If you do not enter the DBH name, the configuration name and the CCS name (connection module parameter CCSN) in the configuration file, the utility monitor is started with the defaults for the DBH name and configuration name (see the “Database Operation” manual).

The utility monitor's most important configuration data is listed below. For a full description of the configuration data, see the “Utility Monitor” manual.

SEE-AUTHID=authorization

Authorization key for working with utility monitor.
This parameter is mandatory.

SEE-INST-LOGGING={ON | OFF}

Enables/disables logging in the statement file.
Default: OFF

SEE-EXECUTE={ON|OFF}

Causes/prevents the execution of logged statements.
Default: ON

SEE-ERROR={ON|OFF}

Specifies the reaction to DBH error messages output while the statement file is being processed (abort or continue).
Default: ON

SEE-COPY={ON|OFF}

Enables/disables the automatic backup of database objects.
Default: ON

SEE-ADMIN=password

Administration password or password for the SESADM administration program.This is a mandatory parameter for administration via the CALL DML interface.

MAIL parameter file

The MAIL parameter file is a BS2000 SAM file which the user creates and assigns to the DBH via the link name SESMAIL at startup, see section “Sending important information of the DBH session by email”.

Detailed information on the structure of the MAIL parameter file and on email output of the DBH is provided in the “Database Operation” manual.

DBH-specific files

DBH-specific files are session-related files which the DBH creates and which are permanently associated with just that DBH.

The default ID for DBH-specific files is the same as the startup ID for the DBH. The system administrator, however, can ensure that DBH-specific files other than the cursor files are created under a different ID. In this event, he/she must assign the appropriate link names to the files before starting the DBH.

The media catalog

The DBH's media catalog contains storage information for the following DBH-specific files:

- the transaction log files (2 TA-LOG files)
- the restart log file (WA-LOG file)
- the cursor files

The media catalog describes the volumes on which these files are to be stored, as well as the catalog IDs under which they are to be stored, and the storage statements used to store them.

The information in the media catalog is only evaluated if a file has not already been created at the start of the session.
The DBH-specific files are only created on the specified medium if they are not already located on a different medium at the start of the session.

System administrators create the media catalog with the DBH option MEDIA CATALOG. The media catalog does not exist in the form of a file. It is only valid for the duration of a DBH session and is maintained in the DBH's memory.

Overview of DBH-specific files

The table below provides an overview of all DBH-specific files. The placeholders in the file names have the following meanings:

ssss Session identifier; this defaults to the BS2000 task sequence number (TSN)

iiii Four-digit start value for the file counter

c Configuration name (one byte)

n DBH name (one byte)

st Service task ID (two bytes, between 01 and 64); this is only supplied within a service task.

Default file name	Link name	Primary/ secondary assignment	Access method	Meaning
SESAM.CO-LOG.*ssss.iiii*	----------	192/24	PAM/ BTAM	File for request logging (independent DBH)
SESAM*cn*.CURSOR.0001 SESAM*cn*.CURSOR.0002	----------	9/24or as specified in media catalog	PAM	Work file for intermediate and final results of retrieval statements (independent DBH)
SESLK*cn*.CURSOR.0001 SESLK*cn*.CURSOR.0002	----------	9/24or as specified in media catalog	PAM	Work file for intermediate and final results of retrieval statements (linked-in DBH)
SESAM*cn*.TA-LOG1 SESAM*cn*.TA-LOG2	TALOG1 TALOG2	As specified in media catalog	PAM	Transaction log file (independent DBH)
SESLK*cn*.TA-LOG1 SESLK*cn*.TA-LOG2	TALOG1 TALOG2	As specified in media catalog	PAM	Transaction log file (linked-in DBH)
SESAM*cn*.WA-LOG	WALOG	As specified in media catalog	PAM	Restart log file (independent DBH)

SESLK*cn*.WA-LOG	WALOG	As specified in media catalog	PAM	The restart log file (linked-in DBH)

Table 52: DBH-specific files

If the DBH-specific files are not to be created on the storage medium specified in the media catalog, the system administrator can override these specifications by setting up the files with the CREATE-FILE command before the DBH is started.

> The prefix SESLK **cn** is replaced by SESLK *cnst* in the case of TA-LOG, WA-LOG and cursor files of a linked-in DBH in *a single* service task.

Transaction log files TA-LOG

The transaction log files (the TA-LOG files) are DBH-specific files in which the SESAM/SQL DBH logs information that is important in the event of a system failure and makes it possible to roll back transactions (see section “Transaction logging”). Specifically, the DBH logs information in the TA-LOG files that guarantees the physical and logical consistency of the data in the database.

This includes:

- after images, i.e. blocks, or rows or parts of rows in a block which have been created through modifications to data
- logical before images, i.e. parts of rows within a database block as they were prior to the first update
- consistency points set by the DBH at each “end transaction” statement.

The DBH creates two TA-LOG files for each DBH session with enabled transaction logging. The TA-LOG files are only ever not created if the system administrator has disabled transaction logging with the relevant DBH option. Storage information pertaining to the TA-LOG files is stored in the DBH's media catalog.

The TA-LOG files use the following standard names:

DBH variant	Standard name	Link name
independent DBH	SESAM*cn*.TA-LOG1 SESAM*cn*.TA-LOG2	TALOG1 TALOG2
linked-in DBH	SESLK*cn*.TA-LOG1 SESLK*cn*.TA-LOG2	TALOG1 TALOG2

Table 53: Standard names of the TA-LOG files

Using the RECOVER utility function also results in the “SESLK*cnst*.TA-LOG1” and “SESLK*cnst*.TA-LOG2” file being created.

Additional information on naming conventions, the default assignment and the access method associated with the TA-LOG files is provided in the table “DBH-specific files”.

> TA-LOG files are I/O-intensive; in exceptional cases I/O bottlenecks can occur. TA-LOG files should therefore be located on a separate device which is as fast as possible.
> The size of the write unit for the TA-LOG files depends on the disk type. SESAM/SQL uses the maximum possible input/output length (64 to 160 KB). Information on the maximum input/output length in half pages (2KB) can be obtained using the BS2000 command
>
> ```
> /SHOW-PUBSET-CONFIGURATION PUBSET=<catid>,INFORMATION=*PUBSET-FEATURES.
> ```

A separate transaction log file exists for SQL statements that define and administer schemas and administer storage structures. Its default name is:
:*catid*:*userid*.*catalogname*.*spacename*.DDLTA.

The file is a log file which is assigned to the space on which the SQL statement operates. It contains the physical before images of the above SQL statements. The file is deleted at the end of the transaction if it was not created b y the user.

Restart log file WA-LOG

The restart log file (the WA-LOG file) is a further DBH-specific backup file that serves to control a restart. It contains the following information:

- DBH options
- information on consistency points
- physical before images
- information on the progress of the restart
- information on openUTM synchronization.

The WA-LOG file uses the following standard name:

DBH variant	Standard name	Link name
independent DBH	SESAM*cn*.WA-LOG	WALOG
linked-in DBH	SESLK*cn*.WA-LOG	WALOG

Table 54: Standard name for the WA-LOG file

When the utility function RECOVER is used, the file SESLK*cnst*.WA-LOG is also created.

The DBH creates a WA-LOG file for each DBH session with transaction logging. A WA-LOG file is not created only if the system uses the appropriate DBH option to disable transaction logging. Storage information pertaining to the TA-LOG files is stored in the DBH's media catalog.

Additional information on naming conventions, the default assignment and the access method associated with the WA-LOG files is provided in the table “DBH-specific files”.

Cursor files for retrieval statements

A cursor file is a DBH-specific work file that is required for processing retrieval statements.

Cursor files can have two meanings in SESAM/SQL:

- on the one hand, SESAM/SQL-DBH stores the intermediate results of retrieval statements in cursor files.
- on the other hand, multiple users can also store the results of CALL DML search queries in a cursor file and subsequently further restrict them. Such CALL DML-specific cursor files are identified by a file identifier (see the “CALL-DM Applications” manual).

Storage information pertaining to the internal cursor files is contained in the DBH's media catalog.

SESAM/SQL creates a maximum of two cursor files. The internal cursor files have the following standard names:

DBH variant	Standard name
independent DBH	SESAM*cn*.CURSOR.0001
	SESAM*cn*.CURSOR.0002
linked-in DBH	SESLK*cn*.CURSOR.0001
	SESLK*cn*.CURSOR.0002

Table 55: Standard names for internal cursor files

The final results of CALL DML search queries and of SQL statements referring to a cursor are stored in the file with the suffix “0001”. In addition, this file can be used as required as an internal temporary work area for storing intermediate results.

The intermediate results of secondary index analyses for CALL DML and SQL retrieval statements are stored in the file with the suffix “0002”.

Both files are created only if actually required.

Additional information on naming conventions, the default assignment and the access method associated with the “internal” cursor files is provided in the table “DBH-specific files”.

The CO-LOG file for request logging

The request log file (CO-LOG file) is a DBH-specific log file which is used to log requests (see section “Evaluating request logging with SESCOSP”).

The DBH opens the CO-LOG file as soon as system administrators enable request logging. If system administrators disable and then re-enable request logging within the same session, the DBH closes the current CO-LOG file and then opens a new one.

The CO-LOG file uses the following standard name:

DBH variant	Standard name
independent and linked-in DBH	SESAM.CO-LOG.*ssss.iiii*

Table 56: Standard name of the CO-LOG file

Using the DBH option SESSION-LOGGING-ID, system administrators can assign their own values for the session ID (*ssss*) and the file counter (*iiii*).

Additional information on naming conventions, the default assignment and the access method associated with the CO-LOG file is provided in the table “DBH-specific files”.

Database files and job variables on foreign user IDs

SESAM/SQL enables you to store the database in a user ID other than the DBH user ID, namely the DB user ID. If the catalog is located on the DB user ID, when you run some utility or DDL statements attempts will be made to locate files and job variables to be created on the DB user ID.

To enable this, the database administrator must make the following preparations. There are two possibilities:

- To define the co-ownership for the DBH user ID in the DB user ID
- To create files and job variables with BS2000 commands

To define the co-ownership for the DBH user ID in the DB user ID

This is the recommended procedure. As a prerequisite, the software product SECOS must be in use.

In the DB user ID the database administrator defines the DBH user ID as co-owner of the objects concerned (files and job variables). This gives the DBH user ID the same rights for the objects concerned as the DB user ID. These rights also include the right to create an object and to specify a password for it.

Example

The DBH user ID <dbh-id> shall have the right to create, administer and delete files for the catalog <db-cat> on the DB user ID <db-id>.

Solution

<db-id> defines a condition guard <db-cond>, that grants <dbh-id> timewise unlimited access:

```
/create-guard <db-cond>,user-inf='access conditions for DBH'
/add-access-conditions guard-name=<db-cond>, -
/                      subjects=*user(user-identification=<dbh-id>)
```

Next, <db-id> defines a co-owner protection rule in the active rule container SYS.UCF. This indicates that the access conditions for the files with sample name “<db-cat>*” are specified in the protection guard <db-cond>.

```
/add-coowner-protection-rule rule-container-guard=sys.ucf, -
/            protection-rule=rule1, -
/            protect-object=*parameters(name=<db-cat>*,-
/                                       condition-guard=<db-cond>)
```

You can define co-ownership for job variables (e.g. SESAM. *replication*.NEXT-REPL-LOG) in the same way. The active rule container for job variables is named SYS.UCJ.

> The co-ownership for each BS2000 catalog ID (cat-id) must be defined separately.

For more information on the SECOS “Co-owner protection” function, refer to the “Security Control System - Access Control” manual, chapter “Guards – protection for objects”.

To create files and job variables with BS2000 commands

The database administrator uses the BS2000 command CREATE-FILE to create a catalog entry on the DB user ID for each of the files concerned.
Job variables are created on the DB user ID with the BS2000 command CREATE-JV.

The files and job variables must be shareable (operand USER-ACCESS=*ALL-USERS) and have write access (operand ACCESS=*WRITE).

Files and job variables with BS2000 password

Where co-ownership has been defined, SESAM/SQL creates files with the BS2000 password specified.

In cases where the database administrator creates the files and job variables, he/she must specify the BS2000 password required (e.g. when creating the files and job variables).

Relevant statements and files

The following statements can create or delete files in the DB user ID:

Statement / Process	**Files**
SQL statements	
CREATE SPACE	user space
DROP SPACE	Delete user space
ALTER TABLE	Exception file
Utility functions	
CHECK FORMAL	Exception file
COPY (on disk) COPY CATALOG[_SPACE] COPY CATALOG[_SPACE] LOG	Backup files of catalog space and user spaces CAT-REC copy, CAT-LOG file, DA-LOG file CAT-REC file and CAT-REC copy, if they are being created for the first time (catalog previously without logging)
CREATE CATALOG	Catalog space, CAT-LOG file, (CAT-REC file)
CREATE INDEX	Work file in the case of parallel index creation
CREATE REPLICATION	Catalog space, user spaces, CAT-REC file of the replication
EXPORT TABLE	export file
IMPORT TABLE	Work file in the case of parallel index creation
LOAD	Exception file

RECOVER CATALOG RECOVER CATALOG_SPACE RECOVER SPACE	Catalog space, user space, CAT-LOG file, DA-LOG file catalog space, CAT-LOG file User space, DA-LOG file
REFRESH SPACE	user space
REORG SPACE REORG CATALOG_SPACE	Work file
UNLOAD	Output file, error file
Administration statements of SESADM or administration commands	
CHANGE-CATLOG or CAW	CAT-LOG file, DA-LOG file, CAT-REC copy
CHANGE-DALOG or DAW	DA-LOG file
DML statements	
First modifying access	DA-LOG file

Table 57: Files which can be created in the DB ID

> For more detailed information, refer to the descriptions of the statements mentioned above. The DDL-TA-LOG file is always created on the DBH user ID. SESAM/SQL only deletes files which SESAM/SQL itself has created. Files which were created by a user are not deleted.

The Data Base Handler's buffers and containers

The Data Base Handler's (DBH) buffer and container technology represents an important function for increasing throughput. Buffers and containers are main-memory areas that are used to store data temporarily.

The SESAM/SQL DBH's most important buffers and containers are

- User data buffer - for user data
- System data buffer - for system access data
- log buffers, used for transaction logging and logical data backups
- the cursor buffer, used for the intermediate results or retrieval statements
- the plan buffer, used for SQL access plans
- the transfer container, used for SQL scans and inquiry and response areas of logical files
- the work container, used for internal statement formats

The aim behind the buffer and container technology is to minimize hard-disk I/O: write jobs that would normally be carried out block by block can be collected and written to the disk in one operation.

Since main memory is not available in unlimited quantities, not all buffered data blocks can be kept resident in memory. The data blocks in the buffer areas and the work container are therefore displaced when a buffer overflows. The DBH's buffer administration feature decides which blocks are to be displaced; the decision is based on criteria intended to achieve the best possible throughput. For example, the DBH sets up separate buffer areas for data with differing displacement behavior, but also differentiates when displacing blocks within the individual buffer areas. In doing so, it avoids the displacement of blocks containing frequently required information in favor of blocks containing information required less frequently.

The sections that follow describe which data is stored temporarily in the individual buffers and containers.

User data buffer and system data buffer

The SESAM/SQL DBH administers a user data buffer and a system data buffer which are available for storing user data and system access data respectively. In each case, the administrative unit and access unit is one PAM block with a size of 4kBytes.

From the point of view of the DBH, the blocks in the user data buffer are all equally important, whereas the system data buffers have a hierarchical structure. The DBH therefore administers the system data buffer blocks in different priority classes according to their importance in terms of access optimization.

System administrators can set the respective buffer sizes when they start the DBH by setting the relevant DBH options accordingly.

Displacement from main memory

If there is insufficient space in the buffer to store new data, the standard procedure is for blocks to be displaced according to the LRU principle (the “least recently used” principle). The oldest blocks (i.e. the blocks that have not been accessed for the longest period of time) are always overwritten. This means that blocks that are constantly required are kept resident and given priority over other blocks.

Log buffers

TA-LOG buffer

When operating with transaction logging enabled, the DBH creates what is known as the TA-LOG buffer. The TA-LOG buffer is implemented as an alternating buffer, in other words it consists of two buffers into which the DBH alternately writes backup information. As soon as a buffer is full, its contents are transferred to the current TA-LOG file (see section “Transaction log files TA-LOG”). While the information is being transferred, the DBH writes the backup information to the other buffer.

CAT-LOG and DA-LOG buffers

For each database, the DBH maintains a so-called CAT-LOG buffer and a DA-LOG buffer in which it temporarily stores the statements for CAT logging or DA logging before they are transferred to the CAT-LOG or DA-LOG file).

Cursor buffer

The DBH stores the intermediate results produced by retrieval statements in the cursor buffer.

If the size chosen for the cursor buffer is too small and the buffer overflows, the data in the cursor buffer is written to one or more internal cursor files (see section “Cursor files for retrieval statements”).

To keep access to the cursor files to a minimum, system administrators can adjust the size of the cursor buffer for the DBH session using a DBH option, and thereby optimize the handling of retrieval statements.

The DBH statistics supplied by the SESAM/SQL utility SESMON help administrators to choose an appropriate size for the cursor buffer (see the “Database Operation” manual).

Plan buffer

The SESAM/SQL DBH reserves an area of main memory specially for storing SQL access plans.

SQL access plan

An SQL access plan is an evaluation rule for an SQL statement.

When a static SQL statement is used, an SQL access plan is created the first time the statement is issued. If the statement executes successfully, the plan is not deleted immediately, but stored in the plan buffer. If the same SQL statement is executed again, the relevant plan is already available and need not be generated again; this leads to a considerable performance gain.

A dynamic SQL statement generates an SQL access plan in connection with the EXECUTE IMMEDIATE or PREPARE statements (see the “SQL Reference Manual Part 1: SQL Statements”). When the EXECUTE IMMEDIATE statement is used, the associated plan is generated and executed immediately. When the PREPARE statement is used, the plan remains in the plan buffer at least until the end of the transaction so that follow-up statements can also use it (see also the “Performance” manual).

If the same dynamic SQL statement is processed with EXEC, IMMEDIATE or PREPARE in a subsequent transaction, the associated plan is reused, if it is still in the plan buffer.

Size of the plan buffer

The plan buffer comprises a primary buffer used by the DBH to store SQL access plans, and a secondary buffer, which contains administrative information. The size of the plan buffer depends on two factors:

- the DBH option SQL-SUPPORT and its operand PLANS, which defines the minimum number of concurrent SQL access plans
- the DBH option COLUMNS, which defines the area size for retrieval statements.

System administrators can tune both settings to suit the requirements of the current session (see the “Database Operation” manual).

If the plan buffer is too small, the SQL access plans are displaced according to the LRU principle: The plan that has been used the least recently is overwritten by a new plan.

The SESAM/SQL utility SESMON supplies operating statistics for the plan buffer. The “SQL INFORMATION” screen displays a selection of information, including the size of the plan buffer and details of how it is currently used.

Transfer container

When SQL access plans are processed, SQL scans are produced. These are subareas of an evaluation rule for an SQL statement.
During the processing of CALL DML applications, inquiry and response areas are requested for each logical file with the OPEN statement.

In order to have main memory available for the inquiry and response areas of SQL scans or logical files, the SESAM /SQL DBH maintains the transfer container, which is administered in fixed units of 256 bytes each.

Transfer container size

At the beginning of the DBH session, the DBH creates a transfer container of a specific size. The basic transfer container size is 64Kbytes, but this can be increased using a DBH option. Frequent requests for storage during a session can cause the transfer container to be become seriously fragmented, with the result that the DBH can ultimately no longer complete the current requests. When this occurs, the transfer container is reorganized. If this does not produce the desired effect, the DBH increases the size of the transfer container up to a given maximum.

Using a DBH option, system administrators can specify a suitable size for the basic and the maximum container size setting. During DBH operation they can use the administration statement MODIFY-STORAGE-SIZE to modify the maximum size to suit the requirements of the current session. In this context, useful information is provided in the DBH statistics supplied by the SESAM/SQL utility SESMON in its “SYSTEM INFORMATION” screen (see the “Database Operation” manual).

Work container

The SESAM/SQL DBH analyzes the statements in the application program to check that they are lexically, syntactically and semantically correct. The DBH takes a correct statement or a subarea of an SQL statement and generates an optimized format known as the internal statement format from it.

Based on a CALL DML statement, the DBH generates the internal statement format directly.

With an SQL statement, the DBH initially generates an SQL access plan, in other words a statement rule for the SQL statement. An SQL access plan consists of at least one but usually several subareas known as SQL scans. The optimized format of a scan is what finally constitutes the internal statement format.

Internal statement formats are placed in the work container so that they can be accessed by follow-up statements. A follow-up statement can, for instance, be the SQL statement FETCH or continuation of processing after a previously locked transaction has been unlocked.
This approach considerably reduces the amount of effort that has to be invested in analysis and optimization.

Using a DBH option, system administrators can specify a suitable size for the work container when the DBH session is started. During DBH operation they can use the administration statement MODIFY-STORAGE-SIZE to modify the maximum size to suit the requirements of the current session. Useful information is provided in the DBH statistics supplied by the SESAM/SQL utility SESMON in its “SYSTEM INFORMATION” screen (see the “Database Operation” manual).

Additional means of increasing throughput

In addition to differentiated buffer technology (see section “The Data Base Handler's buffers and containers”) the SESAM/SQL DBH implements numerous additional measures for increasing throughput. These include not just internal optimization measures, but also measures in which system administrators can intervene and which they can control.

The sections that follow describe the different means of increasing throughput which system administrators can tune:

- Multitasking
- Multi-thread operation
- Priority control
- Flexible processing of retrieval statements
- Relocation of CPU-intensive actions

Multitasking

The SESAM/SQL database system (Enterprise Edition) allows the throughput to be increased on a multiprocessor system by loading one DBH with multiple tasks (DBH option DBH-TASKS). This allows the CPU performance available to the DBH to be utilized to a better effect.
The tasks of a DBH all process a common database.

The system administrator works with a single-system image, i.e. the system administrator manages the DBH as a whole and not a specific task. Any output issued by the active system is issued in the task which the system administrator started and it is that task that the system administrator communicates with.

The DBH automatically distributes the load to the DBH tasks. The tasks are used relative to their load, i.e. the system attempts to carry out processing with as few tasks as possible.

The system administrator can monitor the task loads in the “TASKS” mask in the SESAM/SQL utility SESMON.

Multi-thread operation

The SESAM/SQL database system executes as a nonprivileged program in BS2000. As with any other user task, the SESAM/SQL DBH is interrupted when certain events occur - for example, when a time slice ends, or during I/O operations with WAIT.

Asynchronous I/O

Database processing is generally highly I/O-intensive. In order to prevent the DBH's work from being interrupted too frequently, SESAM/SQL splits up all I/O operations in such a way that the DBH can perform other tasks while it is waiting.

Asynchronous I/O, in other words the splitting up of I/O operations, is especially useful for multi-thread operation: in multi-thread operation, the DBH can process different requests concurrently so that one request can make use of the time in which another request is waiting for the completion of an I/O operation.

Information on the current progress of a request is stored in special save areas, the thread areas. This information allows the DBH to continue processing a request that has been interrupted and is waiting once the relevant I/O operation has completed.

Handling queues

If a request is interrupted because an I/O operation has been triggered, the associated data is generally logically and physically inconsistent. By applying suitable locks, the DBH prevents concurrent requests from accessing this data.
Processing of a request is interrupted if a conflict arises in such a lock or in a transaction lock.
Processing a request is also interrupted in DBH if a service task is currently working for the request.
Interruption of a request in DBH means that the relevant thread waits until the system resources it requires are available.

Controlling multi-thread operation

Using the DBH option THREADS, system administrators can activate multi-thread operation when starting the independent DBH.

The independent DBH can administer up to 1024 threads. It uses the threads on a loadoriented basis, i.e., not all threads are permanently in use, and this helps to limit the administrative overhead.

Only a single thread is generated in the linked-in DBH.

The DBH option THREADS allows system administrators to specify the maximum number of concurrent active threads.

The value of the DBH option THREADS is compared internally with the value of the DBH-TASKS option. With independent DBH, the value must be larger than or equal 2 times the number of tasks.
With linked-in DBH, the DBH option THREADS is set to 1.

Priority control

The SESAM/SQL DBH normally processes requests with varying BS2000 priorities in the order in which they are received. System administrators can modify this approach by enabling priority control with the DBH option REQUEST-CONTROL when they start the independent DBH. Priority control is unavailable in connection with the linked-in DBH.

Priority classes and weights

Using the independent DBH's priority control capability, it is possible to relativize BS2000 priorities for requests and to adapt them to the requirements of the DBH session. To do this, system administrators divide the full range of BS2000 priorities (priorities 30 through 255) into three priority classes. By assigning a weight to each priority class, they can control the processing sequence for requests in the different priority class in accordance with requirements.

Intervention in operations

If necessary, system administrators can intervene in and tune priority control during the current session with the aid of different administration statements (see section “Administration statements and commands”).

Flexible processing of retrieval statements

Under certain circumstances the processing of retrieval statements can be very timeconsuming and can block other requests. In order to avoid slowing performance to too great a degree, the DBH modifies its search strategy to suit the situation in such cases.

Processing via secondary indexes

The DBH interrupts the processing of retrieval statements via secondary indexes and continues it sequentially if it sees no advantage in performing searches by means of secondary indexes.

Sequential processing

The DBH stops the sequential processing of retrieval statements as soon as the number of logical accesses to the user data exceeds a defined limit (see the “Database Operation” manual).
Statements interrupted as a result of this are queued until processing can continue. The request remains assigned to the thread. The thread is not available for use by other requests.

If user data is accessed too frequently from the DBH's point of view (in other words, a set limit is exceeded), the DBH terminates processing completely.

Controlling the processing of retrieval statements

Using the DBH option RETRIEVAL-CONTROL, system administrators can adapt the criteria by which the DBH chooses its search strategy to suit the requirements of the session. For example, they can define a limit for the number of logical accesses to user data which, when reached, should cause the DBH to abort sequential processing. Administration statements are also available which they can use to adjust the DBH's search strategy to current requirements during the current DBH session (see section “Administration statements and commands”).

Relocation of CPU-intensive actions

CPU-intensive actions, such as the sorting of intermediate result sets, can take a considerable length of time. To prevent itself from being blocked during such periods, the SESAM/SQL DBH relocates CPU-intensive actions of this kind in separate tasks provided for service requests.

Service tasks are available for sorting intermediate result sets, for example, but they can also be used for certain utility functions (see the “SQL Reference Manual Part 2: Utilities”). The utility statements COPY, LOAD OFFLINE and UNLOAD OFFLINE are relocated in service tasks; the same applies to parts of utility functions, such as the restoring or updating of a SESAM backup copy, the building of an index, or the reorganization of user data.

A service task is not assigned to a specific type of request; it can carry out all types of service request. This means that it is possible to keep the number of available service tasks low and to make optimum use of them.

At the end of a session, the DBH terminates all the service tasks. Active or incomplete service requests are aborted.

Controlling the utilization of service tasks

System administrators can specify specifically for each session the number of tasks that are to be available at the start of a DBH session and the maximum number of tasks that can be started during the course of the session using the DBH option SERVICE-TASKS. In the linked-in DBH a maximum of one service task is available.

The “SERVICE TASKS” screen in the SESAM/SQL utility SESMON provides system administrators with a means of monitoring service task utilization.

System administrators can use the administration statement MODIFY-SERVICE-TASKS to adjust the number and attributes of the service tasks during ongoing operation.

Start commands for SESAM/SQL programs

All SESAM/SQL programs are started using SESAM start commands.

It is assumed that you have installed SESAM/SQL and CRTE with IMON or that you have stored the CRTE and SESAM libraries under the standard file names described.

The start commands for all SESAM/SQL programs are defined in the SDF syntax file supplied with SESAM/SQL-Server (see also the “Database Operation” manual):

Start command	**Program**
START-SESAM-DBH	SESAM
START-SESAM-DCN	SESDCN distribution component
START-SESAM-ADMINISTRATION	SESADM administration program
START-SESAM-PERF-MONITOR	SESMON
START-SESAM-TUNING-TRACE-EVAL	SESCOSP
START-SESAM-RETRIEVAL-DIALOGUE	SEDI61
START-SESAM-CDML-DIALOGUE	SEDI63
START-SESAM-LOG-FILE-EVAL	SEDI70
START-SESAM-UTILITY-MONITOR	Utility monitor SESUTI

Table 58: Start commands

The start commands are described in the corresponding descriptions for starting the programs.

> For reasons of compatibility, the earlier command START-PROGRAM may also be used. For more information, refer to the appendix of the “Database Operation” manual.

The start commands perform the following:

- Ascertaining and assigning the system files
  The start commands ascertain the system files required to start the program and assign the files by means of corresponding link names.
  The names of the required system files are determined using the following algorithm: The system attempts to determine the name of the system file using IMON. SESAM/SQL searches for the files of the recent version. The files of the most recently installed version are searched for by CRTE. If the system does not succeed the standard file name is used.
  The files concerned and the standard names that apply to them are listed below:

File	Link name	Standard name
CRTE library	BLSLIBxx	$.SYSLNK.CRTE for /390 servers $.SKULNK.CRTE for x86 servers
SESAM module library	SESAMOML	$.SYSLNK.SESAM-SQL.<ver> for /390 servers $.SKULNK.SESAM-SQL.<ver> for x86 servers
Procedure for CONNECT-SESAM-CONFIGURATION command		$.SYSSPR.SESAM-SQL.<ver>.RUN-CFG
Procedure for the start commands		$.SYSSPR.SESAM-SQL.<ver>.RUN-STA
FHS format library for SESMON	MAPLIB	$.SYSFHS.SESAM-SQL.<ver>.MON.E
FHS format library for the utility monitor (German/English)	MAPLIB	$.SYSFHS.SESAM-SQL.<ver>.UTI.D
	MAPLIB	$.SYSFHS.SESAM-SQL.<ver>.UTI.E
Help text file for the utility monitor (German/English)	SEEHELP	$.SYSMAN.SESAM-SQL.<ver>.UTI.D
	SEEHELP	$.SYSMAN.SESAM-SQL.<ver>.UTI.E

Table 59: Default names of the system files

- Determining the operating mode
  The programs are started in the interactive mode when the started via the start commands. If the user wants to run a program in the batch mode, then the user must call the start command in an ENTER procedure.

Controlling and monitoring the session

System administrators can parameterize the DBH and the SESDCN when they start them. They can also monitor the current session and intervene in operations if necessary.

The sections that follow describe important functions for controlling and monitoring DBH and SESDCN sessions:

- DBH start statements and options
- SESDCN control statements and DCN options
- administration statements and commands for the DBH and SESDCN
- Sending important information of the DBH session by email
- request logging with SESCOSP
- evaluation of SESAM/SQL operations with SESMON
- DA-LOG formatting by SEDI70

DBH start statements and options

The purpose of DBH start statements and options is to parametrize the DBH. They specify the resources, limit values and work rules for the current session.

Entering DBH start parameters

System administrators specify the DBH start statements in a specific sequence when starting the SESAM/SQL DBH. After the first start statement, SET-DBH-OPTIONS, they can add DBH options as necessary. The DBH reads the start statements and options from SYSDTA.

There are different ways of entering the DBH start statements and options:

- entry within the procedure used to start the DBH
- entry via a file to which system administrators assign the SYSDTA system file before starting the DBH
- entry directly at the console (in interactive mode)
- entry via the configuration file which system administrators assign via the link name SESCONF or the CONNECT-SESAM-CONFIGURATION statement.

The format of DBH start statements and options follows the same rules as SDF (**S**ystem **D**ialog **F**acility), see the “SDF Dialog Interface” manual). SDF supports the entry of DBH start statements and options directly in a guided dialog, analyzes each statement entered, and transfers it for further processing to the DBH.

DBH start statements

The DBH start statements initiate the parametrization of the DBH. They cause the DBH options to be read and entries to be added to the SQL database catalog list and the CALL DML table catalog list. DBH start statements are entered as three SDF statements:

1. SET-DBH-OPTIONS
2. ADD-SQL-DATABASE-CATALOG-LIST
3. ADD-OLD-TABLE-CATALOG-LIST

The first DBH start statement must be the SET-DBH-OPTIONS statement. The other two statements are optional because entries in the SQL database catalog list or CALL DML table catalog list can also be made using administration statements.

DBH options

DBH options parameterize the DBH and define the main characteristics of a session. The system administrator can modify most of the DBH options (with the exception of CONFIGURATION-NAME and DBH-NAME) during operation. The current DBH options can be saved in a file and used again in the next DBH session.

DBH options are subdivided into higher- and lower-level DBH options.

- Higher-level DBH options, such as ADMINISTRATION or STORAGE-SIZE, relate to a particular subject area.
- Lower-level DBH options each deal with part of the subject area of their higher-level option.

All the higher-level DBH options are assigned one or more lower-level options. System administrators enter the DBH options having entered the DBH start statement SET-DBH-OPTIONS.

There are default settings for all DBH options. The system administrators only need to enter those options for which they wish to specify values other than the defaults. However, before they can specify their own values for one or more lower-level options, they must first enter the associated higher-level option. System administrators can use administration statements (MODIFY-...) to modify many DBH options during ongoing operation.

The DBH options available to system administrators are listed in an overview on the pages that follow. For a detailed description of DBH options, see the “Database Operation” manual.

> ¡ When entering DBH options via SYSDTA, ensure that the comment lines are started by “//REMARK”. The specifications “/REMARK” and “REMARK” will be interpreted as commands and this will lead to an SDF error message. Comments must not be put inside statements.

Overview of the DBH options

Higher-level option	Lower-level option	**Brief description**
DBH-IDENTIFICATION		**Identifies the DBH**
	CONFIGURATION-NAME	Defines the configuration name
	DBH-NAME	Defines the DBH name
ADMINISTRATION		**Administers the DBH**
	ACCOUNTING	Activates accounting
	ADMINISTRATOR	Assigns administration authorization
	MSG-OUTPUT	Controls the outputs of the DBH
	SECURITY	Prevents unauthorized access
CPU-RESOURCES		**Controls the CPU utilization**
	DBH-TASKS	Specifies the number of the DBH tasks
	SERVICE-TASKS	Specifies the number of tasks for service requests
FILE-RESOURCES		**Specifies settings for files**
	MEDIA-CATALOG	Creates a media catalog
	SESSION-LOGGING-ID	Identifies session-specific files

LINKED-IN-ATTRIBUTES	CODED-CHARACTER-SET	**Specifies attributes of the linked-in DBH** Specifies a coded character set which is used by the user program.
RECOVER-OPTIONS	SYSTEM-DATA-BUFFER USER-DATA-BUFFER MEDIA-CATALOG	**Specifies settings for RECOVER or REFRESH runs** Dimensions the system-data buffer Dimensions the user-data buffer Creates the media catalog
STORAGE-SIZE	CURSOR-BUFFER SYSTEM-DATA-BUFFER TRANSFER-CONTAINER USER-DATA-BUFFER WORK-CONTAINER	**Creates buffer and container size** Dimensions the cursor buffer Dimensions the system data buffer Dimensions the transfer container Dimensions the user data buffer Dimensions the work container

SYSTEM-LIMITS		**Specifies limit values**
	COLUMNS	Enlarges the area for retrieval statements
	OLD-TABLE-CATALOG	Specifies the maximum number of entries permitted in the CALL DML table catalog
	SPACES	Specifies the maximum number of simultaneously accessible spaces
	SQL-DATABASE-CATALOG	Specifies the maximum number of entries permitted in the SQL database catalog
	SQL-SUPPORT	Specifies limit values for the SQL interface Makes SQL scans or logical files available
	SUBORDERS	Specifies the number of concurrent system
	SYSTEM-THREADS	threads
	THREADS	Specifies the maximum number of concurrent threads
	USERS	Specifies the maximum number of concurrent users
SYSTEM STRATEGIES		**Defines the processing strategy**
	REQUEST-CONTROL	Activates priority control
	RESTART-CONTROL	Controls the duration of availability in the event of a restart
	RETRIEVAL-CONTROL	Influences the search strategy of the DBH
	TRANSACTION-SECURITY	Activates transaction management

Table 60: Higher- and lower-level DBH options

SESDCN control statements and options

SESDCN control statements and options parameterize the distribution component SESDCN and define distribution rules (see section “Distributed processing with SESAM/SQL DCN”).

SESDCN control statements

System administrators enter the SESDCN control statements when starting SESDCN. They enter them via the same interfaces as DBH start statements and options (see “Entering DBH start parameters”).

SESDCN control statements are entered as three SDF statements:

1. SET-DCN-OPTIONS
2. ADD-DISTRIBUTION-RULE-LIST
3. ADD-NETWORK-LINK-LIST

The first statement causes the DCN options to be read. The two subsequent statements are used to define the distribution rule. The distribution rule assigns an “access path” to each database to be accessed in the distributed-processing scenario. The access path consists of the processor name, the configuration name, the DCN name and the DBH name.

DCN options

System administrators enter the DCN options immediately after the SESDCN control statement SET-DCN-OPTIONS.

DCN options define the main characteristics of SESDCN operation. Since all DCN options are preset to defaults, system administrators need only specify those options that need to be set to something other than their respective defaults.

The following overview provides a brief description of the DCN options. For a detailed description of DBH options, see the “Database Operation” manual.

> ¡ When entering DCN options via SYSDTA, ensure that the comment lines are started by “//REMARK”. The specifications “/REMARK” and “REMARK” will be interpreted as commands and this will lead to an SDF error message. Comments must not be put inside statements.

DCN-OPTION	**Short description of function**
ADMINISTRATOR	Assigns administrator authorization
COLDSTART	Requests a cold start.
DCN-IDENTIFICATION	Assigns a DCN name and configuration name.
REMOTE-ACCESS	Permits remote access by other computers
SESDLG-PASSWORD	Assigns a password.
SYSTEM-LIMITS	Specifies limit values for the permitted number of users and transaction applications, and defines reset criteria for transactions.

Table 61: DCN options

Administration statements and commands

Administration statements and commands are available that allow system administrators to intervene in active operations:

- DBH-specific administration statements and commands allow system administrators to administer the DBH session.
- During distributed processing based on SESAM/SQL-DCN), system administrators can use SESDCN-specific administration statements and commands to administer the distribution component SESDCN.

If system administrators require specific information on database operations, they can run the SESAM/SQL utility SESMON. SESMON's DBH statistics provide information that system administrators can use to monitor the DBH session and determine optimum settings for the DBH options. The SESDCN statistics provide information SESDCN operations (see section “Outputting operational data with SESMON”).

Entering administration statements and commands

The administration statements and commands can be entered by the following means:

- through the administration program SESADM, which runs as a process in its own right, but which can also be accessed as a subroutine of the utility monitor
- through the BS2000 command INFORM-PROGRAM
- through CALL DML statements in a CALL DML program.

You can also access the administration program SESADM from the World Wide Web.
SESADM operation is described in the “Database Operation” manual.

Input format

Depending on the interface through which administration is to take place, system administrators must use administration statements or administration commands.

- Administration statements are for administration with SESADM. The administration program SESADM reads the input through the SDF dialog interface. Administration statement syntax therefore complies with SDF rules (e.g. SHOW-INACTIVE-SQL-USERS).
- When performing administration tasks using INFORM-PROGRAM or within a CALL DML program, administration commands in ISP format (e.g. USER,INACT) are available to the system administrator.

Administration statements offer an almost identical functional scope to administration commands. For this reason, only administration statements are described in the following overviews. Refer to the “Database Operation” manual for a brief description of the relevant administration commands.

Output of administration statements and commands

Large database configurations also supply extensive outputs, e.g. in the SHOW statements of SESADM. The statement MODIFY-OUTPUT-MODE in the DBH and DCN menus enables you to direct the output of administration statements to SYSOUT/SYSLST or to a temporary file.
The information outputs of most SHOW statements can be output in S variables and processed further in S procedures.

Under SESADM all outputs are complete. In the event of administration using INFORM-PROGRAM or a CALL-DML program an output section (a so-called response) has a maximum size of 32000 Byte. Larger output volumes are output in multiple responses. You can call the successor responses using the administration command NEXT.

Result of the administration statements

In interactive mode status messages and error messages are sent to SYSOUT and SYSLST, and in batch mode to the console and SYSLST. In addition, the message number of the last response of the DBH or of SESDCN to an administration statement is recorded in the S variable SESAM-RESULT and in the job variable #SESAM.SESADM.JV. This permits automatic administration with SESADM. See the “Database Operation”.

DBH administration statements

The tables below provide an overview of the DBH administration statements.

A detailed description of the syntax and meaning of the DBH administration statements is provided in the “Database Operation” manual. The manual also matches administration statements with equivalent administration commands.

The three tables are divided up as follows:

- The table 62 describes all administration statements that provide user-specific information.
- The table 63 contains all administration statements that display or change the DBH start statements and options issued (see section “Controlling and monitoring the session”).
- The table 64 contains all other statements that can be used to control the SESAM/SQL operation.

Administration statement	Brief description
SHOW-USER-SPACES	Displays the spaces used by the user and thus locked
SHOW-CALL-DML-SUBORDERS	Displays the number of active CALL DML OPEN requests of selected users
SHOW-CATALOG-USERS	Displays the number of active users of selected databases
SHOW-INACTIVE-SQL-USERS	Displays all inactive SQL users
SHOW-SPACE-USERS	Displays all active users of a selected space together with relevant additional information
SHOW-TRANSACTIONS	Displays all open transactions of selected users with relevant additional information
SHOW-USERS	Displays all active users together with relevant additional information
SHOW-CATID-LIST	Displays the current CATID list

Table 62: DBH administration statements for user-specific information

Administration statement	Brief description
ADD-OLD-TABLE-CATALOG-ENTRY	Adds an entry to the CALL DML table catalog list

ADD-SQL-DB-CATALOG-ENTRY	Adds an entry to the SQL database catalog list
MODIFY-ADMINISTRATION	Changes the administration authorization
MODIFY-CATALOG-ACCESS-RIGHTS	Changes the access rights for the specified database
MODIFY-REQUEST-CONTROL	Changes the parameters for priority control
MODIFY-RESTART-CONTROL	Influences the duration of any restart
MODIFY-RETRIEVAL-CONTROL	Changes the criterion for interrupting or canceling retrieval statements
MODIFY-SQL-SORT-LIMIT	Changes the limit value for the number of sort records found in a cursor table
MODIFY-SUBORDER-LIMIT	Changes the limit value for the maximum number of SQL scans and/or logical files of CALL DML requests
MODIFY-MSG-OUTPUT	Changes the DBH output
MODIFY-OLD-TABLE-CATALOG-LIMIT	Changes the maximum number of entries in the CALL-DML table catalog
MODIFY-RECOVER-OPTIONS	Changes the options for subsequent RECOVER or REFRESH runs
MODIFY-SECURITY	Changes the maximum permissible number of password violations
MODIFY-SERVICE-TASKS	Changes the number and attributes of service tasks
MODIFY-SESSION-LOGGING-ID	Changes the identification of session-related files
MODIFY-STORAGE-SIZE	Changes the maximum size of transfer and work containers
MODIFY-TRANSACTION-SECURITY	Changes the transaction security parameters
RECONFIGURE-DBH-SESSION	Changes DBH options dynamically
RELOAD-DBH-SESSION	Reloads DBH modules
REMOVE-OLD-TABLE-CATALOG-ENTRY	Removes an entry from the CALL DML table catalog list
REMOVE-SQL-DB-CATALOG-ENTRY	Removes an entry from the SQL database catalog list
REUSE-OLD-TABLE-CATALOG-ENTRY	Creates a valid reference to an already existing table entry in the CALL DML database catalog list
REUSE-PARTITIONS	Restores the availability of partitions

SAVE-DBH-OPTIONS	Saves the current DBH options
SET-ACCOUNTING-PARAMETER	Controls logging of request accounting for the RAV procedure
SET-REQUEST-CONTROL	Turns priority control on and off
SET-USER-INACTIVE-TIME	Sets a time period after which a user's open but inactive transactions are rolled back
SHOW-DBH-MEDIA-CATALOG	Displays the currently valid DBH option MEDIA-CATALOG
SHOW-DBH-OPTIONS	Displays the currently valid DBH options except for MEDIA CATALOG
SHOW-OLD-TABLE-CATALOG-ENTRIES	Displays the entries in the CALL DML table catalog list
SHOW-PARTITIONS	Displays the availability of partitions
SHOW-SQL-DB-CATALOG-ENTRIES	Displays the entries in the SQL database catalog

Table 63: Administration statements for DBH start statements and options

Administration statement	**Brief description**
ABORT-LOCK-SEQUENCE	Terminate lock sequence of another user
ASSIGN-SYSLST	Switches a SYSLST file
BEGIN-LOCK-SEQUENCE	Starts a lock sequence
CANCEL-STATEMENT	Cancels a DML statement
CHANGE-CATLOG	Switches a CAT-LOG file and DA-LOG files
CHANGE-DALOG	Switches DA-LOG files
CLOSE-SPACE	Closes user space physically
COMMIT-PTC-TRANSACTION	Commits a PREPARE-TO-COMMIT transaction
CREATE-DUMP	Creates a main-memory dump
END-FOREIGN-COPY	Removes the “copy pending” state from spaces after foreign copy.
END-LOCK-SEQUENCE	Terminate lock sequence
HOLD-TRANSACTION-ADMISSION	Suspends admission of further transactions
MODIFY-CATID-LIST	Updates the CATID list

PREPARE-FOREIGN-COPY	Closes a database in order to be able to create a foreign copy
RELEASE-USER-RESOURCES	Releases all a user's resources
RESUME-TRANSACTION-ADMISSION	Permits resumption of transaction admission
ROLLBACK-PTC-TRANSACTION	Rolls back a user's PREPARE-TO-COMMIT transaction
ROLLBACK-TRANSACTION	Rolls back a user's transaction
SET-DBH-MSG-TRACE	Controls the logging of DBH messages
SET-DIAGNOSIS-DUMP-PARAMETER	Controls the creation of a dump
SET-SESSION-DIAGNOSIS	Controls deadlock analysis
SET-SQL-DB-CATALOG-STATUS	Changes the status of a database
SET-TUNING-TRACE	Controls request logging
STOP-DBH	Terminates a DBH session

Table 64: Other DBH administration statements that control database operation

SESDCN administration statements

The following table provides an overview of all the administration statements that are available for the administration of SESDCN. A detailed description of the syntax and meaning of the DBH administration statements is provided in the “Database Operation” manual. Here you will find also the administration statements with the corresponding administration commands.

Administration statement	**Brief description**
ADD-DISTRIBUTION-RULE-ENTRY	Adds a database to the distribution rule
CREATE-DUMP	Creates a main-memory dump
HOLD-TRANSACTION-ADMISSION	Suspends admission of further transactions
HOLD-USER-ADMISSION	Suspends admission of further users
MODIFY-ADMINISTRATION	Changes the administration authorization
MODIFY-DISTRIBUTION-RULE-ENTRY	Changes the host name in the distribution rule
REMOVE-DISTRIBUTION-RULE-ENTRY	Removes a database entry or database entries from the distribution rule
RESUME-TRANSACTION-ADMISSION	Permits resumption of transaction admission
RESUME-USER-ADMISSION	Resumes admission of new users

ROLLBACK-TRANSACTION	Rolls back a user's transaction
SET-USER-CALL-TRACE	Controls the logging of the statements of selected users
SET-USER-MSG-TRACE	Controls the logging of SESAM messages of selected users
SHOW-DISTRIBUTION-RULE-ENTRIES	Displays active databases entered in the distribution rule
SHOW-TRANSACTIONS	Displays all open transactions of selected users with relevant additional information
SHOW-USERS	Displays all active users together with relevant additional information
STOP-DCN	Terminates SESDCN

Table 65: SESDCN administration statements

Sending important information of the DBH session by email

The SESAM/SQL system administrator can also have important information of the DBH session transferred automatically by email.

Detailed information on the structure of the MAIL parameter file and on email output of the DBH (DBH options, administration statements and commands) is provided in the “Database Operation” manual

Prerequisites for email output of the DBH

The mail service of the software product interNet Services must be available.

The email is actually sent by the DBH with the mail transmitter (SEND-MAIL interface) of the software product interNet Services.

MAIL parameter file

The MAIL parameter file is a BS2000 SAM file which the user creates and assigns to the DBH via the link name SESMAIL at startup, see section “MAIL parameter file”.

In the MAIL parameter file you enter the recipient(s), the sender, and relevant message numbers for email outputs of the DBH.

Without the MAIL parameter file you can (initially) not use the email output of the DBH.

The MAIL parameter file can be modified while the DBH session is active. However, the (modified) parameters become effective only when the administration statement MODIFY-MAIL-PARAMETERS is entered. The administration statement SHOW-MAIL-PARAMETERS enables you to display the current MAIL parameters.

Options and statements for controlling email output of the DBH

You set the scope of the DBH’s email output with the DBH options MSG-OUTPUT and SERVICE-TASKS. The administration statements MODIFY-MSG-OUTPUT and MODIFY-SERVICE-TASKS or the administration commands OPT,MSG-OUTPUT and OPT,SVT allow you to modify the scope of the DBH’s email output while the DBH session is active.

With the MODIFY-OUTPUT-MODE statement you can define additional output by email as the output destination for all the following administration statements.

Evaluating request logging with SESCOSP

SESCOSP is a utility which outputs information on logged requests.

If request logging is enabled, the DBH collects data in a request log file (CO-LOG file). The data is formatted and output by the utility SESCOSP.

Request logging allows system administrators to track and assess events in the period that was logged. The data shows which statements were executed for which programs, and which spaces and databases were addressed. SESCOSP reports can thus help to identify a statement which is critical with regard to performance.

System administrators enable request logging with the administration statement SET-TUNING-TRACE (see the “Database Operation” manual).
Request logging, however, increases the load on the DBH.
A description of how to use SESCOSP is provided in the “Database Operation” manual.

Outputting operational data with SESMON

The performance monitor SESMON collects data on current database operations, computes statistics based on a variety of criteria, and outputs the information on screen, as printed lists, to a file or via the SESAM/SQL agent to a management platform. SESMON runs as a separate task and does not affect runtime behavior.

The utility provides system administrators with a wide range of information on DBH or SESDCN operations:

- the current settings of DBH and DCN options
- information on the size and contents of the container
- information on the progress of a DBH session restart
- information on SQL statements and SQL access plans
- information on every active application
- the number of service tasks, and information on service task loading
- information on the progress of a RECOVER or REFRESH run
- accesses of spaces and cursor files
- information on queues
- the number of currently active transactions and their respective statuses
- Occupancy of the prefetch buffer

SESMON provides system administrators with documents

- to analyze the performance of SESAM/SQL
  System administrators can monitor the effects of different options on DBH and SESDCN response times.
- for an optimum program mix
  The information provided allows system administrators to determine the effects of concurrently executing application programs.
- for statistical purposes
  For instance, the data provides information on the frequency of individual database operations and the number of hard disk accesses in a specific period of time.

You can also access the performance monitor SESMON from the World Wide Web.

A description of how to use SESMON is provided in the “Database Operation” manual.

DA-LOG formatting by SEDI70

SEDI70 is a utility program for outputting modifications made in the user spaces.

If logging is activated, the DBH collates data on modifications made to the user data in a log file (DA-LOG file). The SEDI70 utility edits the data and outputs it in a variety of lists.

The information issued lists the SQL and CALL DML statements executed and the spaces and databases accessed. The DA-LOG file also logs the creation of SESAM backup copies for a user space.

A description of how to operate SEDI70 is contained in the “Database Operation” manual.

Building, loading and maintaining databases

This chapter describes

- the steps in which a SESAM/SQL database is built, and how the database structure can be modified
- how user data can be loaded and unloaded
- the maintenance tasks associated with a SESAM/SQL database

Statements and tools for building and maintaining a database

The utility monitor is a convenient tool that is used to build and maintain SESAM/SQL databases. It provides the database administrator with the means he/she requires in order to carry out all the tasks associated with building and maintaining a SESAM/SQL database through a menu-driven interface (see the “Utility Monitor” manual).

Alternatively, the database administrator can use ESQL programs to build and maintain SESAM/SQL databases. The following groups of statements are available for using this program:

- Utility statements
- SQL statements for managing the storage structure
- an SQL statement for administering user entries
- SQL statements for the definition and administration of schemas

The SQL statements are described in detail in the “SQL Reference Manual Part 1: SQL Statements” and the utility statements are described in the “SQL Reference Manual Part 2: Utilities”.

Building a database and modifying its structure

This section describes

- how the basic structure of a SESAM/SQL database is built through the creation of the catalog space, and
- how the basic structure of a SESAM/SQL database can be extended, and how the structure of a SESAM/SQL database can be modified.

Creating the basic structure of a SESAM/SQL database

The basic structure of a SESAM/SQL database is created with the utility statement CREATE CATALOG. When this statement is executed, SESAM/SQL creates the database's catalog space.

The table below presents an overview of the necessary activities and lists the associated utility statements. The right-most column indicates which main function to choose in the utility monitor.

Activity	**Statement and function**	**Main function in the utility monitor**
Create a catalog space for the database	CREATE CATALOG Defines the ○ BS2000 password for the database ○ DB identifier ○ Character set (code table) ○ the attributes of the catalog space ○ the storage group for the catalog space ○ the storage group for the CAT-REC file and the first CAT-LOG file ○ the universal user	Main function CREATE CATALOG (CRC) and continuation forms
Back up the catalog space	COPY Saves the database (catalog space); not absolutely necessary	

Table 66: Utility statements and the main utility monitor function used to create the basic database structure

Creating the database's catalog space

An authorized user creates the database's catalog space with the utility statement CREATE CATALOG.

System administrators grant the right to execute the utility statement CREATE CATALOG to a specific system user ID by making this user ID known to the appropriate SESAM/SQL DBH with the DBH option ADMINISTRATOR, or with the administration statement MODIFY-ADMINISTRATION (see the “Database Operation” manual).

The catalog space can be created either on the DBH user ID or on a DB user ID, see the section “Database files and job variables on foreign user IDs”.

The catalog space can be up to 4 TB in size on pubsets with “large files”. Otherwise it can be up to 64 GB in size.

The catalog space is where the database's metadata is stored. SESAM/SQL initializes the empty metadata by entering the information specified in connection with CREATE CATALOG, together with other administrative information (e.g. the views of the information schema). From the point of view of the operating system, the catalog space is an ordinary BS2000 file. The database also always has to be initialized with CREATE CATALOG. The catalog space's BS2000 file name is

:*catid*:*user_id.catalog*.CATALOG

The BS2000 user ID for the database

All the files in a SESAM/SQL database with the exception of logging and error files have to be cataloged on the same BS2000 user ID. SESAM/SQL enters the user ID for the database in the DBH's SQL database catalog (see the “Database Operation” manual).

The following is important in connection with the databases that are not located on the DBH user ID:

- SESAM backup copies on magnetic tape cartridges can only be created with HSMS. For this to be possible, the DBH ID must be defined as a co-owner of the DB files in the DB user ID. Repair or reset from a SESAM backup copy on tape is also only possible using HSMS.
- SESAM/SQL can only execute the SQL statement CREATE SPACE if the preparations have been made for creating the catalog space, see the section “Database files and job variables on foreign user IDs”.
- If logging files (CAT-LOG files, DA-LOG files) are to be stored on the DB user ID, preparations must have been made for creating the logging files, see the section “Database files and job variables on foreign user IDs”.
  If it is not possible to create logging files on the DB user ID, SESAM/SQL will create them on the DBH user ID.

  During repairs, SESAM/SQL first searches for the log files on the DB user ID and then on the DBH user ID.

Altering catalog space

The universal user can alter the catalog space using the SQL statement ALTER SPACE "CATALOG" (CATALOG in double quotes).

ALTER SPACE enables the free-space reservation to be altered and a new storage group to be specified for the catalog space. If the storage group is altered, the universal user must have the special privilege USAGE for the new storage group.

The alterations become effective the next time the catalog space is reorganized.

Backing up catalog space

Once the basic structure of the database has been defined, the database administrator should create a SESAM backup copy of the catalog space with the utility statement COPY. To do this, the database administrator requires the UTILITY privilege. Details of how to create SESAM backup copies using COPY are provided under section “Creating a SESAM backup copy with COPY”.

Extending the basic structure of a database and modifying the structure

The basic structure of a SESAM/SQL database simply comprises the catalog space and, if logging is enabled, the CAT-REC file and the first CAT-LOG file. The basic structure can be extended as follows to create an operational SESAM/SQL database capable of accommodating user data:

- Creating system entries
- Granting special privileges
- Defining storage groups for user spaces
- Inserting the first media record for DA-LOG and PBI files or the HSMS work file in the media table
- Creating user spaces
- Creating schemas

The structure of an operational SESAM/SQL database can be modified by means of the following measures to adapt it to current requirements:

- Creating and deleting system entries
- Granting and revoking special privileges
- Defining, modifying, and deleting storage groups for user spaces
- Maintaining the media table (inserting or deleting media records, modifying descriptions of file attributes in the media table)
- Creating, modifying, and modifying user spaces
- Creating, modifying, and deleting schemas
- Modify properties of the database

The above measures are explained in detail in the sections that follow.

The table below provides an overview of the activities that need to be carried out, and links the tasks with the relevant SQL or utility statement. The right-most column indicates the main functions in the utility monitor that the database administrator uses to complete each task.

Activity	Statement and function	Main function in the utility monitor

Create and delete system entries	CREATE USER Assign authorization key (for SQL user) CREATE SYSTEM_USER Create a system entry DROP USER Delete an authorization key and the associated system entries DROP SYSTEM_USER Delete a system entry	Main function CREATE CATALOG (CRC) and continuation forms and Main function ALTER CATALOG (ALC) and continuation forms
Grant and revoke special privileges	GRANT Grant special privilege REVOKE Revoke special privilege	
Create, modify and delete storage groups	CREATE STOGROUP Create a storage group ALTER STOGROUP Modify a storage group DROP STOGROUP Drop a storage group	Main function SSL and continuation forms
Add the first media record for DA-LOG and PBI files to the media table	CREATE MEDIA DESCRIPTION Add the first media record for the databasespecific DA-LOG and PBI files respectively to the media table.	Main function CREATE CATALOG (CRC) and continuation forms
Maintain the media table	ALTER MEDIA DESCRIPTION Add or delete media records; modify the description of file attributes DROP MEDIA DESCRIPTION Delete all the media records for a specific file type.	Main function ALTER CATALOG (ALC) and continuation forms
Create, modify and delete user spaces	CREATE SPACE Create user space ALTER SPACE Alter user space DROP SPACE Delete user space	Main function SSL and continuation forms

Create, modify and delete schemas	CREATE SCHEMA Create a schema Alter a schema DROP SCHEMA Delete a schema	Main function CREATE SCHEMA (CRS) and continuation forms Main function ALTER SCHEMA (ALS) and continuation forms
Modify properties of the database	ALTER CATALOG Modify properties	Main function ALTER CATALOG (ALC) and continuation forms

Table 67: SQL/utility statements and main utility monitor functions for extending and modifying the structure of a database

Create and delete system entries

In order to work with a SESAM/SQL database with SQL, BS2000 users need an SQL user authorization key. The database's metadata must contain at least one appropriate system entry for each authorization key (see section “System entry”).

Create a system entry

The database administrator assigns an authorization key to an SQL user with the SQL statement CREATE USER.

The database administrator defines a system entry with the SQL statement CREATE SYSTEM_USER.

The relevant SQL user authorization key has to have been assigned with CREATE USER before a system entry can be created with CREATE SYSTEM_USER. The database administrator can create more than one system entry per authorization key. An SQL database user is represented by all the system entries in the database's metadata that contains the SQL user's authorization key.

Delete a system entry

The database administrator deletes an SQL user's authorization key with the SQL statement DROP USER.

DROP USER deletes an authorization key and all its associated system entries. An authorization key cannot be deleted if it owns schemas, user spaces or storage groups, or if it is the grantor of a privilege. You cannot delete the authorization identifier of the universal user.

To delete individual system entries for an SQL user, the database administrator uses the SQL statement DROP SYSTEM_USER.

A system entry cannot be deleted if it is the universal user's only system entry.

Grant and revoke special privileges

Regardless of the number of system entries that have been created, only the universal user is initially authorized to work with the new database using SQL and utility statements. If other SQL users are to take on certain database administration tasks, these users have to have the necessary authorization (to issue utility statements, for example). Moreover, individual users must be granted the right to define schemas (see section “Creating, modifying and deleting schemas”). For this reason, the universal user is able to grant special privileges for the following activities with the SQL statement GRANT:

- the creation and deletion of SQL users (CREATE USER)
- the creation of a schema (CREATE SCHEMA)
- the execution of utility statements (UTILITY)
- the definition of storage devices for the spaces (CREATE STOGROUP)
- the creation of user spaces in a specific storage group (USAGE ON STOGROUP)
- all special privileges (ALL SPECIAL PRIVILEGES)

The universal user is authorized to grant special privileges with the exception of USAGE ON STOGROUP. The owner of this storage group is authorized to grant the special privilege USAGE ON STOGROUP for a given storage group.

With the SQL statement REVOKE, the grantor can revoke each special privilege granted with the SQL statement GRANT.

Further information on special privileges and the granting of privileges in general is provided in section “Access protection based on privileges in SQL”.

Defining, modifying and deleting storage groups

A storage group defines the storage media (private disks and the associated device type, or public disks) and a BS2000 catalog ID, which are needed in order to create user spaces and SESAM backup copies.

Create a storage group

The database administrator defines a storage group with the SQL statement CREATE STOGROUP.

Modify a storage group

The database administrator can modify the definition of a storage group with the SQL statement ALTER STOGROUP.

The ALTER STOGROUP statement only modifies the definition of the storage group. Existing user spaces that use the storage group's disks are not affected.

Disks removed from the definition are no longer used for new storage allocations to user spaces. Disks can be deleted explicitly with the aid of the DROP VOLUME clause, or implicitly by switching them from common disks (PUBLIC) to private disks, and vice versa.

Drop a storage group

The database administrator can delete an existing storage group with the SQL statement DROP STOGROUP.

DROP STOGROUP is rejected if one or more user spaces are still located on the storage media defined by the group.

Add the first media record for DA-LOG and PBI files to the media table

Information on storage media and attributes for database-specific files (the CAT-REC file, CAT-LOG files, DA-LOG files and PBI file or the HSMS work file) is stored in the media table in so-called “media records”. The first media record for the CAT-REC file and the first media record for the CAT-LOG files is created when the utility statement CREATE CATALOG executes (see section “Creating the database's catalog space”).

The database administrator creates the first media record for DA-LOG files and the first media record for PBI file or the HSMS work file with the utility statement CREATE MEDIA DESCRIPTION.

The execution of the statement CREATE MEDIA DESCRIPTION for a specific file type is rejected if the media table already contains an entry for the relevant file type.

The media table can be maintained (i.e., further media records can be added or media records can be removed) with the aid of the utility statements ALTER MEDIA DESCRIPTION and DROP MEDIA DESCRIPTION.

Maintain the media table

Adding or deleting media records, and modifying the description of file attributes

Using the utility statement ALTER MEDIA DESCRIPTION, the database administrator can modify the description of the file attributes in the media table for a database-specific file type. In addition, ALTER MEDIA DESCRIPTION can be used to add or delete media records in the media table.

Any modifications you make with ALTER MEDIA DESCRIPTION only affect new files of the specified file type. This statement has no effect on existing files.

Delete all the media records for a specific file type.

Using the utility statement DROP MEDIA DESCRIPTION, the database administrator can delete all the media records for a certain database-specific file type from the media table.

DROP MEDIA DESCRIPTION has no effect on files whose attributes are described in the media records that are to be deleted, i.e. existing files are not deleted.

Creating, modifying and deleting user spaces

User spaces store the user data in a SESAM/SQL database, i.e. the user tables and the indexes defined for those tables. From the point of view of the operating system, user spaces are ordinary BS2000 files.

Create user space

If the database, to which a user space is to be assigned, is stored on a DB user ID, the database administrator must make preparations for creating the new user space on this BS2000 user ID, see section “Database files and job variables on foreign user IDs”.

The database administrator creates a user space with the SQL statement CREATE SPACE. A maximum of 999 user spaces can be created per database.
A user space can be up to 4 TB in size on pubsets with “large files”. Otherwise it can be up to 64 GB in size.

Alter user space

The owner of a user space can modify the space with the SQL statement ALTER SPACE.

ALTER SPACE can be used to modify the free-space reservation and to disable DA logging. In addition, a new storage group can be specified for the user space in the USING clause. To do this, the owner of the user space requires the special privilege USAGE ON STOGROUP for the new storage group (see section “Special privileges”).
A change to the free-space reservation takes effect the next time the user space is reorganized. The change takes effect before this:

- in the case of IMPORT TABLE, if tables are imported together with their user data,
- in the case of LOAD OFFLINE, if data is loaded into an empty table or if new data is appended to existing data.

A change to the storage group takes effect the next time the user space is reorganized.

Delete user space

The owner of a user space can delete the space with the SQL statement DROP SPACE.

Free-space reservation for the catalog space and for user spaces

The free-space reservation feature makes it possible to specify the extent to which blocks in the catalog space or a user space are to be filled with data. Choosing a suitable setting for free-space reservation can improve performance during updates.

SESAM/SQL takes into consideration the reservation of free space when user data is loaded, when a table is imported with its user data, when partition boundaries are changed, when columns are added, changed or deleted and when column values are interchanged

Spaces with a size of more than 64 GB

A user space (and also the catalog space) which is created (with CREATE SPACE, CREATE CATALOG or RECOVER TO) with SESAM/SQL on a pubset which supports “large files” can be up to 4 TB in size (“large space”). Otherwise it can be up to 64 GB in size.

The maximum space size can be ascertained from the SYS_INFO_SCHEMA. For this purpose the SYS_SPACE_PROPERTIES view (in the utility monitor via the SNF function) displays the highest possible page number with MAX_POSSIBLE_PAGE:

- 1,073,741,822 (X'3FFFFFFE') for spaces up to 4 TB
- 16,777,214 (X'00FFFFFE') for spaces up to 64 GB

Spaces from earlier SESAM/SQL versions can be migrated to “large spaces” using the utility statement REORG SPACE provided the spaces are located on a pubset which supports large files. After migration you should create a SESAM backup.

If no SESAM backup is created after migration, the space in ongoing operation exceeds the 64 GB limit and the space subsequently needs to be recovered, a backup of the “old” space must be used. The recovery will be aborted because the 64 GB limit was exceeded while the logging files were being recovered.

You can bypass this problem by initially generating a replica from the backup of the “old” space. The replica is then reorganized with REORG SPACE, and the recovery is subsequently implemented with the replica (RECOVER ... USING REPLICATION).

Creating, modifying and deleting schemas

Schemas contain metadata that describes the formal structure of base tables, indexes, views and routines, and metadata that describes the privileges.

Create a schema

The database administrator grants the special privilege CREATE SCHEMA to SQL users with the SQL statement GRANT. SQL users with this privilege can then create schemas and, thus, objects for those schemas (see section “Special privileges”). These authorized users create schemas with the SQL statement CREATE SCHEMA.

When creating a schema, users authorized to use CREATE SCHEMA can define SQL objects (base tables, views, indexes and routines) and can grant table and EXECUTE privileges by including the SQL statements CREATE TABLE, CREATE VIEW, CREATE INDEX, GRANT and CREATE PROCEDURE in CREATE SCHEMA.

Using the SQL statement GRANT, schema owners can pass on table and EXECUTE privileges for objects in their schemas to other users (see section “Table privileges”).

Alter a schema

The owner of the schema can adjust it to current requirements:

- Create, modify or delete base tables.
  Schema owners can create base tables with the SQL statement CREATE TABLE, and can modify and delete existing base tables with the SQL statements ALTER TABLE and DROP TABLE respectively. For further details on creating, modifying and deleting base tables, see section “Base table”.
- Create or delete views.
  Schema owners can create views with the SQL statement CREATE VIEW, and can delete views with the SQL statement DROP VIEW. For further details on creating and deleting views, see section “View”.
- Create or delete indexes.
  Schema owners can create indexes with the SQL statement CREATE INDEX, and can delete existing indexes with the SQL statement DROP INDEX. For further details on creating and deleting indexes, see section “Index”.
- Create or delete routines.
  Schema owners create routines with the SQL statements CREATE PROCEDURE and CREATE FUNCTION, and delete existing routines with the statements DROP PROCEDURE and DROP FUNCTION. For further details on creating and deleting procedures, see section “Routine”.
- Grant or revoke table and EXECUTE privileges.
  Schema owners grant privileges for tables and routines in their schemas with the SQL statement GRANT. Privileges are revoked with the SQL statement REVOKE. For further details on granting and revoking privileges, see section “Access protection based on privileges in SQL”.

Delete a schema

Schema owners can delete a schema with the SQL statement DROP SCHEMA.

Modify properties of the database

The SQL statement ALTER CATALOG can be used to modify the database's coded character set (CCS).

When `CODE_TABLE _NONE_` is specified, no coded character set is used for the database. This corresponds to the behavior of SESAM/SQL before V5.0.

Exporting and importing base tables

EXPORT TABLE exports a base table from a SESAM/SQL database to a BS2000 file. This BS2000 file is known as an export file. The export file is created with a coded character set name (CCSN) in accordance with the database's character set. When it is imported the export file's CCSN is checked against the database's coded character set.

The table's metadata is stored in the export file. User data can also be exported. IMPORT TABLE can then be used to create a new base table from this export file. The new base table can also be a partitioned table.

You can use the EXPORT/IMPORT TABLE statements to perform the following tasks:

- To copy a base table within a database or from one database to another:
  Using an export file is an easy way to copy a base table since the structure of the target table is defined via the metadata in the export file. Significantly fewer operating steps are required compared to the use of the pair of statements LOAD/UNLOAD because, when LOAD is used, it is necessary to reconstruct the table structure. You can specify a search constraint in the EXPORT TABLE statement to define which user data is to be transferred to the new table.
  However, unlike the files used with LOAD/UNLOAD, the export file cannot be processed manually.
  The export file is considerably smaller than a file generated using UNLOAD (see section "Unloading user data with UNLOAD").
- To move a base table from one space to another:
  To do this, you first export the base table and then delete it with DROP TABLE. You can then import into a space of your choice using IMPORT TABLE. Views, reference constraints and access rights for this table may have to be adapted as appropriate.
- To convert a base table to a partitioned table or to modify the number and boundaries of partitions of a partitioned table:
  To do this, you first export the base table and then delete it if required with DROP TABLE. It can be imported as a new, partitioned table with the same or a different name, specifying the (modified) partition structure with IMPORT TABLE.
  The user-friendly utility statement ALTER PARTITIONING FOR TABLE is also provided for this purpose, see section "Changing the partitioning of a base table".
- To restructure a base table:
  A base table can be split into a number of different new tables.
  To do this, you must export it and then import it as often as required under different names. These new tables are initially identical but can be edited using ALTER TABLE in order to achieve the required subdivision.
- To re-assign the row numbers in a base table:
  You can re-assign the row numbers of base tables with primary keys.
  To do this, you must export the table, delete it and then re-import it into the same space while specifying NEW ROW_IDS.
- To reorganize a table:
  You can use the IMPORT TABLE statement to reorganize a table since IMPORT TABLE takes account of the space's block utilization. To do this, you must export the table, delete it and then re-import it into the same space.
  A table can also be reorganized with REORG ONLINE TABLE.

- To archive a single base table:
  Export files created with EXPORT TABLE can be used as backup copies of individual tables. They can also be saved to magnetic tape cartridge. Unlike backups created using the COPY statement, it is not the granular space but the higher-resolution granularity table that is used. A compact backup is created because less administrative information has to be saved. It is, however, not possible to run a RECOVER on this type of backup. EXPORT files can only be processed by IMPORT TABLE.

> ¡ An export file which contains a table with columns of the data type NATIONAL CHARACTER (VARYING) can no longer be imported with SESAM/SQL V4.0 or earlier.

Exporting base tables with EXPORT TABLE

The database administrator can use the utility statement EXPORT TABLE to write a base table which can be partitioned from a SESAM/SQL database to a variable record-length SAM file.

This creates a so-called export file that can only be interpreted by IMPORT TABLE. The exported table and the new table generated on the basis of the export file have the same structure.

The export file is created with a coded character set name (CCSN) in accordance with the database's character set.

The following metadata of the database is stored in the export file:

- the metadata of the base table and all the associated columns
- the metadata for all the indexes relating to this base table
- the metadata for all the integrity constraints with the exception of the reference constraints defined in this base table

You can use a WHERE clause to restrict the volume of user data that is to be transferred to the export file. The export file cannot be processed manually.

Importing a base table with IMPORT TABLE

The database administrator can use the IMPORT TABLE statement to load a new base table using the information contained in the export file into a schema in a SESAM/SQL database. The CCSN of the export file must match the database's coded character set (CODE_TABLE). If no coded character set is used for the database, the export file's CCSN is ignored. The new base table can also be a partitioned table.

A base table which can be partitioned is generated in the specified database from the export file's metadata:

- The table is created.
- The space resp. the definition of the partitions is taken from the statement. The spaces must already have been created for partitioned tables.
- The table type is taken over from the exported table.
- All the columns of the exported table are taken over. The data type, default attributes and the sequence of rows are retained.
- If there is a primary key then this is also taken over. A primary key must exist for partitioned tables.

You can decide whether or not the user data stored in the export table is to be taken over. If you decide to import the user data you can also insert the corresponding indexes or integrity constraints from the export file into the new table.

Interrupting logging in the case of IMPORT TABLE

If a base table is located in a user space for which logging has been activated, SESAM/SQL logs all DML or DDL statements that update the table contents or table definition in DA-LOG or CAT-LOG files.
If data is loaded into this table by means of IMPORT TABLE then logging is interrupted. If integrity constraints are also loaded into the table using IMPORT TABLE then it is guaranteed that these integrity constraints are fulfilled.
If logging has been interrupted then, after IMPORT TABLE, the database administrator must create a backup copy of the user space in which the table is located. Otherwise the user space will remain in the state “copy pending” and can only be worked on with utility or retrieval statements (see the “SQL Reference Manual Part 2: Utilities”).

Loading and unloading user data

With LOAD/UNLOAD, you can

- load user data from an input file into a base table
- unload user data from a base table or a view to an output file
- transfer user data from one base table or a view to another

Loading or unloading user data offline or online

When you execute the two statements you can choose between the operating modes OFFLINE and ONLINE. The main differences concern the performance, lock behavior and error handling.

The behavior of the OFFLINE operating mode is like that of the conventional LOAD. Processing takes place in a service task. The user space is locked exclusively against changes.

In the ONLINE operating mode better performance is achieved in processing small files. The user space is not locked exclusively against changes.

Detailed information on OFFLINE/ONLINE is provided in the description of the utility statements in the “SQL Reference Manual Part 2: Utilities”.

Loading user data with LOAD

The database administrator can load user data from an input file into a base table in a SESAM/SQL database with the utility statement LOAD. The input file is a SAM file of variable record length. Any of the following files can be used as the input file:

- an input file whose format is specified by SESAM/SQL (standard format)
- an input file whose format is specified by the user (user defined format)
- a transfer file
- a file in the LOAD_FORMAT
- an input file in delimiter format
- an input file in CSV format

The coded character set name (CCSN) of the input file is used for checking against the database's coded character set.
In DELIMITER_FORMAT, in CSV_FORMAT and in the self-defined format, strings that are to be loaded are converted to the database's coded character set if necessary.

LOAD error file

Users can specify the name of an error file in the LOAD statement's USING FILE clause.

The error file is only created or used by SESAM/SQL when an error occurs. LOAD appends new information to an existing error file. The entries relating to different LOAD statements are separated by two header lines.

The error file has the following name (not including the USING FILE clause): *catalog.space.*EXC.L

catalog	Name of the database.
space	Name of the user space containing the table into which the load operation is to be performed. In the case of a partitioned table the space name of the first partition is used. The error file contains the error information for all partitions in the table.
EXC.L	Default value for the name of the error file for LOAD statements.

Temporarily disabling logging during LOAD OFFLINE

If a base table is located on a user space for which logging has been enabled, SESAM/SQL logs all DML and DDL statements that modify the table contents or the table definition in DA-LOG and CAT-LOG files. Logging is interrupted when data is loaded into this table with LOAD OFFLINE. In the case of LOAD ONLINE, logging continues. The database administrator must therefore create a SESAM backup copy of the user space on which the table is located with the utility statement COPY immediately after LOAD OFFLINE has completed and the CONSTRAINT CHECK has succeeded. Otherwise the user space will remain in the state “copy pending” and can only be worked on with utility or retrieval statements (see the “SQL Reference Manual Part 2: Utilities”).

Unloading user data with UNLOAD

Using the utility statement UNLOAD, the database administrator can write user data from a table in a SESAM/SQL database to a SAM file of variable-length records. The database administrator can choose

- whether SESAM/SQL specifies the format (standard format)
- whether he/she specifies the format for the output file him-/herself (user defined format)
- whether the output file is to be created in delimiter format
- whether the output file is to be created in CSV format
- whether the output file is to be created as a transfer file
- whether the output file is to be created in the UNLOAD_FORMAT that will allow it to be used as an input file with the utility statement LOAD.

The output file is created with coded character set name (CCSN). If necessary, the data to be output in DELIMITER_FORMAT, in CSV format and in self-defined format is converted to the output file's coded character set. In CSV format the column names can also be output (in the first line).

You can output entire tables (UNLOAD TABLE) or also data from selected columns of the table (UNLOAD DATA).

UNLOAD ONLINE enables you to unload user data from base tables or views. The data which is to be output can also be restricted by a search condition, and a collation sequence can be defined for the output file.

UNLOAD OFFLINE enables you to unload user data from base tables. In addition, you can output data:

- a user space that could not be repaired FROM SPACE clause, user space is in the state "recover pending").
- a backup copy (FROM COPY_FILE clause)

Error file in the case of UNLOAD

In the case of UNLOAD, SESAM/SQL creates a new file if required or updates an existing file. The file has the following name:

catalog.*space*.EXC.U	in the case of UNLOAD OFFLINE and
file.EXC.U	in the case of UNLOAD ONLINE

catalog is the name of the database.

space Name of the user space containing the table into which the records are to be loaded. In the case of a partitioned table the space name of the first partition is used. The error file contains the error information for all partitions in the table.

file Name of the file which is specified in the INTO FILE clause.

New entries are appended to an existing exception file. The entries from different UNLOAD statements are separated by a header line. If an exception file contains at least one record, a warning is issued.

Transferring user data between base tables

The utility statements UNLOAD and LOAD make it possible to transfer user data from one base table or view to another base table.

For example, to transfer data from base table A to base table B, users on table A unload the data from table A with UNLOAD, specifying the LOAD_FORMAT option. SESAM/SQL applies a structure to the output file that allows the users on table B to use it as an input file for B by executing LOAD with the UNLOAD_FORMAT option.

If A is a base table that is to be unloaded and for which a primary key has been defined, the output file created by an UNLOAD with LOAD_FORMAT is sorted on its primary-key values. If the primary key in table B (into which data is to be loaded) matches the primary key in table A, the loading operation can be accelerated by using LOAD SORTED to load the data into B.

The columns from which data is to be unloaded and into which data is to be loaded in the two tables between which the data is transferred must be defined identically.

Verifying the correctness of the data

SESAM/SQL offers the database administrator the option of verifying the correctness of the data in the following ways:

- through a check that determines whether the integrity constraints have been fulfilled, i.e. a check to determine whether the content of the data is correct, and
- through a check of the formal correctness of tables and indexes.

Check integrity constraints

During database operations, SESAM/SQL carries out plausibility checks to make sure that only those changes are made to the data that are compatible with the integrity constraints.

Utility statement CHECK CONSTRAINTS

The database administrator can verify compliance with integrity constraints with the utility statement CHECK CONSTRAINTS.

The following can be checked with CHECK CONSTRAINTS:

- selected integrity constraints
- all integrity constraints defined in one or more base tables
- all integrity constraints defined for the base tables of one or more user spaces

The database administrator can opt to check the integrity constraints for all specified tables, or just for tables in the “check pending” state (see section “Space state after the execution of utility statements”).

Integrity constraints always need to be checked with CHECK CONSTRAINTS if the data has changed after the following utility statements:

- LOAD, if LOAD was executed without the CONSTRAINT CHECK clause.
- RECOVER, if the current status of the database was not recovered.

In these cases, SESAM/SQL places the tables affected by the respective utility statement or the user space processed in the “check pending” state.

Information on how to deal with tables that remain in the “check pending” state after execution of the utility statement CHECK CONSTRAINTS is provided under “Using SQL statements to work with tables in a “check pending” state”.

Checking the formal correctness of tables and indexes

The database administrator can check the formal correctness of user data and indexes using the utility statement CHECK FORMAL. The check on the formal correctness essentially consists of a check on SESAM/SQL's internal blocks and tables.

With CHECK FORMAL, the database administrator can check a single base table, a single index, or all the base tables and indexes in a user space.

CHECK FORMAL is also permitted in connection with replications and SESAM backup copies present in the SQL database catalog list. CHECK FORMAL can also be used for foreign copies if the foreign copy is present as a database in the SQL database catalog list. To prevent it from losing its property of being a foreign copy, it must be added with ACCESS=READ.

The database whose spaces, tables and indexes are to be checked must be added to the DBH's database catalog with the ADMIN attribute so that defective spaces can be marked as such.

If the database is added to the SQL database catalog with ACCESS=READ, as a backup copy or as a replication then a defective space is not marked as such. In this case, references to the error can only be found in SQLSTATE and in the error file.

CHECK FORMAL error file

SESAM/SQL creates an error file for CHECK FORMAL or appends information to the existing file if CHECK FORMAL identifies inconsistencies.

The file name in the BS2000 user ID for the DBH is: *catalog*.*space*.EXC.C

catalog	Name of the database
space	Name of the user space for which the formal check is to be performed In the case of CHECK FORMAL TABLE for a partitioned table, an error file with the space name of the partition is is created for each errored partition.
EXC.C	Default value of the name of the error file for CHECK FORMAL statements

SESAM/SQL enters a record in the error file for each discovered inconsistency. SESAM/SQL only creates/uses the error file if inconsistencies are genuinely identified.

Working on locked user spaces

A utility statement may leave a user space in a state in which its further processing may be restricted (see chapter “Utility concept”).

Details of the utility statements that can be executed in connection with a user space in a specific state and the state the user space assumes as a result of the statement are provided in the “SQL Reference Manual Part 2: Utilities”.

- Spaces in a “copy pending” state require COPY.
- Spaces in a “load running” state require RECOVER and then possibly RECOVER INDEX.
- Spaces in a “recover pending” state require RECOVER.
- Spaces in a “check pending” state require CHECK CONSTRAINTS. If CHECK CONSTRAINTS reports integrity violations, the violations must be cleared.

Using SQL statements to work with tables in a “check pending” state

Tables in a user space that is in a “check pending” state) can only be worked on by their owners, using SQL statements that manipulate data. Using this approach, all records that violate an integrity constraint can be adapted or deleted. A prerequisite here is that these SQL statements are issued with the pragma CHECK OFF. Syntactically, the pragma CHECK OFF is an ordinary SQL comment. However, it allows SQL statements to be executed on tables for which integrity constraints have been violated. A pragma begins within an SQL statement with the string “--%” and is terminated with an end-of-line character. The CHECK OFF pragma is described in detail in the “SQL Reference Manual Part 2: Utilities”.

A further means of correcting violations of integrity constraints is for the owners of the tables in question to delete the violated integrity constraints with the SQL statement ALTER TABLE ... DROP CONSTRAINT.

Once all violations of integrity constraints have been deleted, the database administrator must once again execute the utility statement CHECK CONSTRAINTS in order to make the affected user spaces or tables available to the authorized groups of persons again.

Maintaining a database

The following maintenance tasks need to be carried out in connection with a SESAM/SQL database:

- the reorganization of spaces and base tables on user spaces
- the carrying out of tasks associated with media recovery
- Migrate databases and tables
- Querying metadata

Reorganizing spaces and base tables

Using the utility statement REORG, the database administrator can reorganize the catalog space, individual user spaces or also individual base tables in user spaces. During the reorganization process, SESAM/SQL ensures that storage areas that belong together logically are located next to one another physically.

The purpose of reorganization

The blocks of the catalog space, of the individual user spaces and of the base tables are concatenated in logical order. When processing begins, the physical sequence (the ascending sequence of block numbers) and the logical sequence (the content of the chained blocks) match.
During database operations, SESAM/SQL can create new blocks if the insufficient space is available for new data in the existing blocks. The logical sequence of new and old blocks is maintained through chaining, but it ceases to match the physical sequence. This can seriously reduce access speed when a space is accessed sequentially. Reorganization ensures that the logical and physical block sequences match.

During reorganization, SESAM/SQL takes free-space reservations into consideration that the database administrator has defined previously with the statements CREATE CATALOG, CREATE SPACE, or ALTER SPACE in the PCTFREE clause for the space in question (catalog space or user space).

Utility statement REORG [CATALOG_]SPACE

If the database administrator has previously specified a new storage group for the user space to be reorganized using the SQL statement ALTER SPACE, the user space is moved physically to this storage group during reorganization, unless otherwise stipulated by the COPY specification or the specification of a work file.

SESAM/SQL reorganizes the space into a work file. The work file can be given. Otherwise SESAM/SQL uses a standard work file whose name is formed from the name of the space file together with the suffix .REORG. The standard work file can also be created by the user; it is then used as the standard work file with the defined file features.

You can issue the sequence of statements EXPORT TABLE, DROP TABLE and IMPORT TABLE to restructure a space that contains tables and indexes in such a way that the tables and indexes are located in separate spaces. First, the relevant table is exported to an export file along with the associated indexes. The table itself is deleted with DROP TABLE. You then import the table from the export file back to the original space using IMPORT TABLE. You can also specify a separate index space for the indexes (see section “Importing a base table with IMPORT TABLE”).

A simpler procedure is possible if indexes have been created on a mere table space by SESAM/SQL according to unique constraints. These indexes can be relocated to other spaces by explicitly creating them on other spaces with CREATE INDEX. SESAM/SQL then relocates the unique constraint (UNIQUE) to the explicitly defined index and deletes the implicitly defined one.

Utility statement REORG ONLINE TABLE

SESAM/SQL reorganizes a base table by modifying and copying the blocks in the user space. As no exclusive transaction locks are required for this purpose, other DML applications can both read and modify the base table.

In the case of partitioned tables a single partition can also be reorganized. The partition is defined by the ON SPACE clause. Otherwise all partitions of the base table are reorganized one after the other.

Tasks associated with media recovery

The concept of media recovery is explained in section "Media recovery". The current section examines in more detail the tasks that need to be carried out in the context of media recovery and presents the utility statements used by the database administrator to carry out the individual tasks.

Creating a SESAM backup copy with COPY

With COPY utility statement, the database administrator can create SESAM backup copies of the whole of the SESAM/SQL database or parts of the database, such as the catalog space or user spaces. SESAM/SQL creates SESAM backup copies either on magnetic tape cartridge or on disk. A SESAM backup copy can be created in online mode or in offline mode. It is also possible to create a backup from mirror units of a local or remote storage system using HSMS. In addition, the database administrator can have the formal correctness of the SESAM backup copy verified and can enable logging for a user space or a catalog space once the copy has been created. Spaces which only contain indexes and are not involved in the logical data backup can be excluded from the backup.

If the database is located on a DB user ID, SESAM/SQL will always first try to create the SESAM backup copy on the DB user ID. This is only possible if the preparations have been made for creating the files, see the section “Database files and job variables on foreign user IDs”. Otherwise the copies will be created on the DBH user ID.

If logging is enabled, COPY CATLOG and COPY CATLOG_SPACE always lead to a change of CAT-LOG file and DA-LOG file.
If logging is not enabled, SESAM/SQL creates the CAT-REC file according to the specifications in the media table the first time the COPY CATALOG or COPY CATALOG SPACE statement is issued.
For each COPY CATALOG and COPY CATALOG_SPACE, a CAT-REC copy is created on the same media where the CAT-LOG file is located. No metadata is stored for these additional copies. If the CAT-REC copy is to be located on the DB user ID, the above applies. CAT-REC copies from previous COPYs are overwritten.

SESAM backup copies on magnetic tape cartridge can be created using the COPY ... USING DIRECTORY utility statement or with the software products HSMS or ARCHIVE (see the “SQL Reference Manual Part 2: Utilities”). If an HSMS parameter file is present then the name specified for COPY ... USING DIRECTORY is interpreted as an HSMS archive. Otherwise the backup is performed using ARCHIVE.

Creating a SESAM backup copy online or offline

When COPY is executed, SESAM/SQL puts all the spaces (the catalog space and user spaces) acted on by COPY in a transaction-free state.

- The following effects have to be taken in consideration when using COPY ONLINE: When COPY ONLINE is used, SESAM/SQL again permits SQL query and update statements and CALL DML statements once the creation of the copy has begun. SESAM/SQL rejects all other SQL statements and all utility statements and issues an SQL status code.
  If the online backup is to take place in an HSMS archive, the database files to be backed up must not be located on private disk.
  All the blocks in the user spaces that are to be modified during the copy operation are logged by SESAM/SQL prior to modification as so-called physical before-images (PBI) in the PBI file (in the case of disk copy and ARCHIVE) or they are logged by HSMS in the HSMS work file. This file is always created on the user ID in which the DBH is running. Once the copy operation is complete, SESAM/SQL terminates PBI logging and permits all statements access to the relevant user spaces.
  When a SESAM backup copy is created on disk, SESAM/SQL immediately copies the logged before-images to the SESAM backup copy in order to ensure that the SESAM backup copy matches the relevant user space in its transaction-free state immediately prior to the actual copy operation.
  When a SESAM backup copy is created on magnetic tape cartridge using ARCHIVE, SESAM/SQL also saves the PBI file on a separate magnetic tape cartridge. In this case, the file is not restored until a repair or reset is carried out with the utility statement RECOVER.
  In the case of backup in an HSMS archive, the block to be backed up is einther read from the database file or from the HSMS work file and is then backed up. A specific SESAM/SQL DBH can execute more than one COPY ONLINE statement at a time. However, these must refer to spaces in different databases.
- When COPY OFFLINE is used, SESAM/SQL retains all update requests for the user spaces or the catalog space affected by the copy operation until the COPY OFFLINE is complete.
  The SESAM backup copy matches the relevant space or spaces in the transaction-free state immediately prior to the actual copy operation.

Creating SESAM backup copies from mirror units using HSMS

`COPY ... USING DIRECTORY` *hsms_archive_name* `{BY_ADD_MIRROR_UNIT | BY_SRDF_TARGET}`

is a very efficient means of backing up database files that are stored on a mirror unit in an HSMS archive on disk or on magnetic tape cartridge. Depending on the location of the mirror unit, two cases must be differentiated (explained here taking the Symmetrix function TimeFinder/Mirror as an example):

1. BY_ADD_MIRROR_UNIT
   In this case the TimeFinder/Mirror function (Symmetrix Multi Mirror Facility) of the Symmetrix systems is used during ongoing operation. Database operation runs on the so-called normal unit in the local symmetrix. An additional disk, the so-called additional mirror unit, is used in the local symmetrix for data mirroring. The normal unit and additional mirror unit together are referred to as the multi-mirror pair.

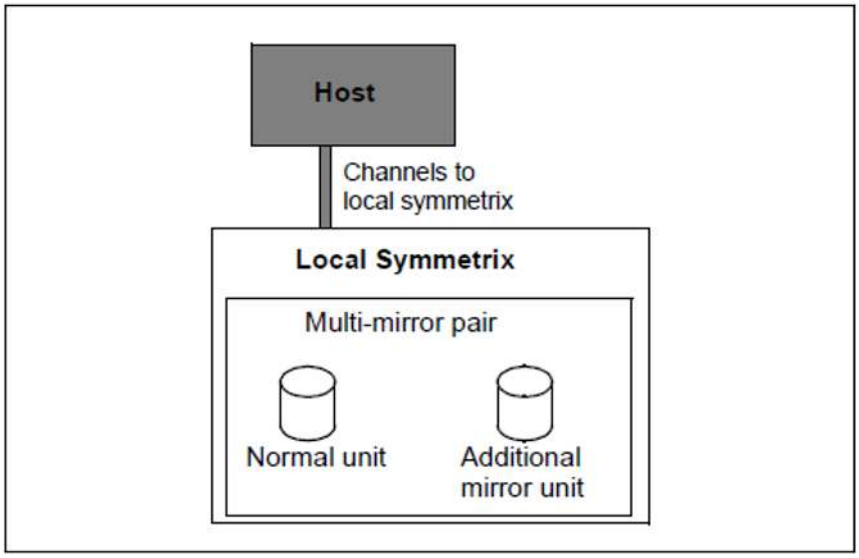

Figure 30: Configuration with TimeFinder

2. BY_SRDF_TARGET
   In this case the SRDF (Symmetrix Remote Data Facility) function of the symmetrix systems is used during ongoing operation. Database operation runs on the so-called source unit in the local symmetrix. An additional disk, the so-called target unit, is used in the remote symmetrix for data mirroring. The source unit and target unit together are referred to as the remote copy pair.
   The remote copy pair must be operated in synchronous mode.
   The HSMS archive must be located on the same system to which the source unit is connected.

   To create a SESAM backup copy it is necessary that the target unit in the remote symmetrix is assigned an additional mirror unit. In other words the target unit and additional mirror unit are both located in the remote symmetrix and there form a multimirror pair (TimeFinder/Mirror).

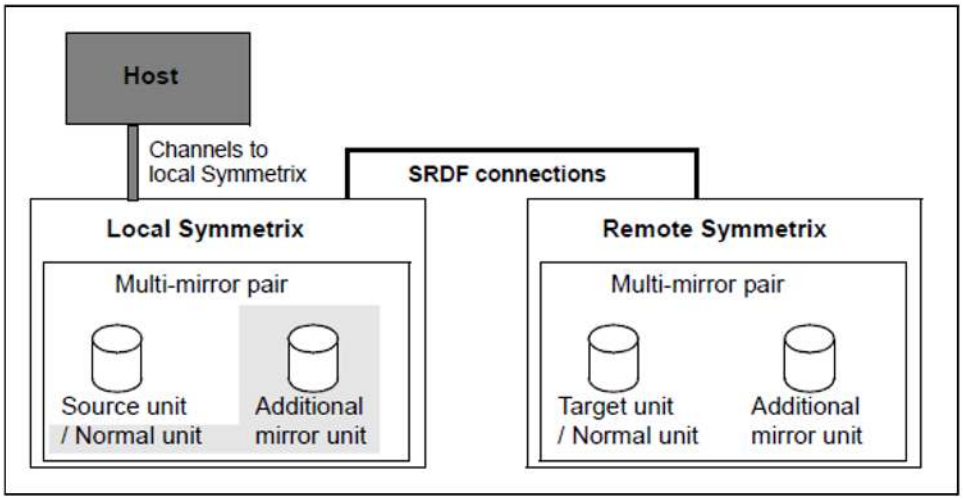

Figure 31: Configuration with SRDF and TimeFinder

Note: To permit further data mirroring in the local symmetrix the source unit can be supplemented by an additional mirror unit (TimeFinder/Mirror). A SESAM backup copy could also be created from this using BY_ADD_MIRROR_UNIT (against a gray background in the figure, of no significance in the further description).

In both cases the SESAM backup copy is created from the additional mirror unit using HSMS. The database files must be located on the same additional mirror unit.

When the term mirror units is used in the description below, additional mirror units (symmetrix systems) or clone units (ETERNUS DX, symmetrix or CLARiiON CX systems) are also meant.

You can find more detailed information on these storage systems and on their replication functions in the “SHC-OSD (BS2000)” manual.

Procedure for backup from mirror units using HSMS

Backup takes place in three phases:

- Firstly the mirror unit is split from the normal unit (case 1 in the previous section) or target unit (case 2). DML read access to the database is possible.
- The specified database files are backed up from the mirror unit to the specified HSMS archive.
  The database on the normal unit (case 1) or source unit (case 2) is processed by the DBH. Mirroring to the target unit (case 2) is continued. Read access to the database (in the case of COPY ... OFFLINE) or even DML update access (in the case of COPY ... ONLINE) is possible during the backup process.
- Once the backup to the HSMS archive is complete, the mirror unit and the normal unit (case 1) or target unit (case 2) are synchronized again. Read and write access is also possible during this time.

SESAM/SQL uses the HSMS concurrent COPY (CCOPY) function when executing the COPY statement. HSMS uses the software product SHC-OSD. For backup in an HSMS archive SESAM/SQL builds the HSMS statement BACKUP-FILES with the parameter WRITE-CHECKPOINTS=*YES in order to write restarting points. This means that the HSMS backup run can continue following an abortion with the HSMS statement RESTART-REQUESTS (you can find further information about this in the “HSMS (BS2000)” manual, volume 1).

The advantage of this process is that the database administrator does not have to worry about either splitting or synchronizing the mirror unit.

For this function the DBH task requires one of the privileges TSOS or HSMS-ADMINISTRATION.

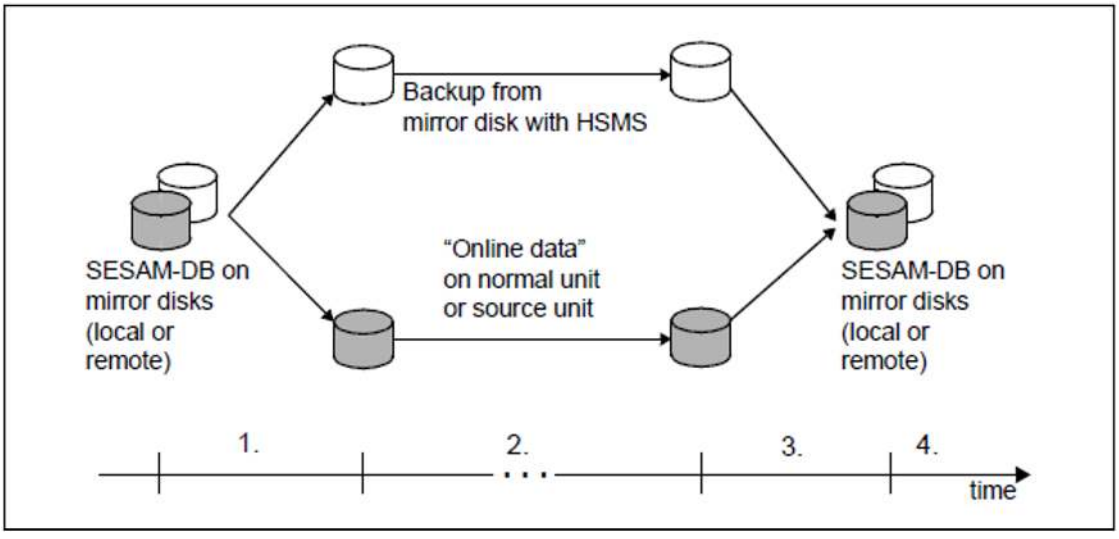

Figure 32: Procedure for backup from additional mirror units using HSMS

1. Initialization phase: time from the start of the COPY statement to the separation of the mirror disks. During this phase only read access is possible for the spaces to be backed up. Write access is only possible at the end of the initialization phase. SESAM/SQL builds the HSMS statement BACKUP-FILES.
2. Backup and operation phase: time during which HSMS backup and database operation run in parallel. HSMS backs up the database files from the mirror unit to an HSMS archive. The database on the normal unit or the source unit is processed in parallel to this by the DBH. DML update statements are permitted again in COPY ONLINE.
3. Synchronization phase: time from the end of the HSMS backup to the completion of the mirror disk synchronization. Following the HSMS backup, the mirror disks are synchronized again. DML read and update statements are possible again.
4. The backup was successful if the backup to the HSMS archive was successful and the formal check with the CHECK FORMAL parameter specified did not result in an error. Successful backup is recorded in the RECOVERY_UNITS.

Notes about backup with the CHECK FORMAL parameter

The CHECK FORMAL parameter can be specified for an online backup from mirror units using HSMS.
If the source unit or normal unit belongs to a shared pubset, the SESAM/SQL DBH must run on the server which is represented as the master side for the shared pubset. The formal check of the spaces to be backed up is performed on the split additional mirror unit parallel to backing up the files to the HSMS archive.

Otherwise the parameter may not be specified for an online backup to magnetic tape cartridge.

If the parameter CHECK FORMAL is specified for an offline backup to magnetic tape cartridge, the formal check of the spaces to be backed up will first be performed and then the request is given to HSMS. This means that the spaces to be backed up will be locked until the formal check is complete. Separation of the disks only begins after this.

When such requirements apply, you should therefore favor online backup from additional disk units using HSMS.
In other cases the following procedure is recommended for an offline backup: The backup should be created with NO CHECK FORMAL. The database administrator should then read in the backup again from the HSMS archive, add the database to the SQL database catalog of another DBH and perform the formal check.

If an error is detected in a formal check, this means the backup was not successful; the original space is not flagged as “defective”, however.

Maintenance of the metadata of SESAM backup copies

The RECOVERY_UNITS and DA_LOGS catalog tables are located in the database's catalog space. RECOVERY_UNITS contains entries for the SESAM backup copies of the user spaces; DA_LOGS contains information on the DA-LOG files.
Refer to section “Files and tables used for media recovery” for a detailed description of the RECOVERY_UNITS and DA_LOGS tables.

The CAT-REC file (catalog recovery file) contains recovery unit records for the catalog space and the associated CAT-LOG records.
Refer to section “CAT-REC file” for a detailed description of the CAT-REC file.

Using the utility statement MODIFY, the database administrator can do the following:

- Delete records in the catalog table RECOVERY_UNITS which pertain to the SESAM backup copies of a specific user space.

  At least the last two records are alwas retained. One of the remaining records refers to the SESAM backup copy of the user space that was created last. Otherwise all remaining records, or records of a specific age, are deleted for this user space.
- Delete records in the catalog tables RECOVERY_UNITS and DA_LOGS independently of their assignment to specific user spaces.

  At least the last two records per user space are retained in RECOVERY_UNITS. One of the remaining records refers to the SESAM backup copy of the user space which was created last. Otherwise all remaining records, or records of a specific age are deleted from RECOVERY_UNITS.
  In DA_LOGS, only those records are not deleted that pertain to DA-LOG files that follow on from SESAM backup copies for which RECOVERY_UNITS still contains records after a MODIFY.

  Foreign copies represent a special case here. When you work with foreign copies, no new records are created in the RECOVERY_UNITS table. You can specify the UNRESTRICTED clause in order to delete records from the DA_LOGS catalog table. Records are then deleted from DA_LOGS whether or not there is a last RECOVERY_UNITS record.

  > ! UNRESTRICTED deletes all the records with the specified age in the RECOVERY_UNITS and DA_LOGS, up to but excluding the last in each case. This record is retained for technical reasons.
  > A RECOVER is no longer possible if the required files have been deleted in the catalog tables RECOVERY_UNITS and DA_LOGS. The oldest record in DA_LOGS required for a foreign copy of a space is the record that was created in the space when it was closed after the last update. If the foreign copy is generated during or after a session in which the space was not opened or was used for read access only then the restart point is not updated in the space.

- In the current CAT-REC file, delete recovery unit records and CAT-LOG records which refer to SESAM backup copies of the catalog space.

  The latest recovery unit record and the associated CAT-LOG record are retained in the CAT-REC file so that a recovery is still possible with SESAM/SQL. All other records or records of a particular age can be deleted.

  Foreign copies represent a special case here. When you work with foreign copies, no recovery unit records are created in the CAT-REC file. CAT-LOG records cannot be deleted without the corresponding recover unit record.

Maintaining the CAT-REC file

The CAT-REC file contains entries pertaining to the SESAM backup copies of the catalog space, and information on the CAT-LOG files. Refer to section “Files and tables used for media recovery” for a detailed description of the CAT-REC file.

Using the utility monitor, the database administrator can view information on the content of the CAT-REC file. He /she can also modify CAT-REC files online or offline and therefore delete records on SESAM backup copies that are no longer required from a CAT-REC file (see the “Utility Monitor” manual).

The utility statement MODIFY enables the database administrator to modify the current CAT-REC file online (see the “SQL Reference Manual Part 2: Utilities”).

The CAT-REC file should always be mirrored using dual recording by volume or dual copy. It is possible to back up the CAT-REC file when the database is no longer added to an SQL database catalog. Changes made after this backup would, however not be automatically logged, so that the CAT-REC file could only be recovered to the status of this SESAM backup copy when required.

Deleting SESAM backup copies and log files in BS2000

When the database administrator uses MODIFY to delete records from the RECOVERY_UNITS and DA_LOGS catalog tables, the associated SESAM backup copies of the user spaces and the DA-LOG files are not deleted. The same applies in principle to SESAM backup copies of the catalog space and to CAT-LOG files that are no longer required for which the database administrator has deleted the relevant entries from the CAT-REC file with the utility statement MODIFY or using the utility monitor.

It is recommended that SESAM backup copies and log files that are no longer required be deleted from the BS2000 system. Files created on disk can be deleted with the command DELETE-FILE ...,OPTION=*DESTROY-ALL. SESAM backup copies on magnetic tape cartridge constitute part of the associated HSMS archive or ARCHIVE directory and must therefore be deleted with the aid of the software products HSMS or ARCHIVE (see the “HSMS (BS2000)” or “ARCHIVE (BS2000)” manuals respectively).

Repair and reset

Repair and reset are means of recovering a SESAM/SQL database after error situations have arisen that cannot be corrected by other means (see section “Potential error situations and appropriate recovery measures”). The database administrator carries out repair and reset using the utility statement RECOVER.

It is also possible to use foreign copies instead of the SESAM backup copies created with COPY when performing the repair or reset of user spaces, the catalog space or the entire database (see section “Repair and reset using foreign copies”).

Repair and reset of user spaces, the catalog space or the entire database are also possible with the aid of a replication. For information on this topic, refer to section “Database replication” or section “Database replication”.

Repair and reset are described in detail in “Media recovery”.

Rebuild indexes

Using the utility statement RECOVER INDEX, the database administrator can rebuild or restore indexes, see section “Recovering indexes”.

Migrate databases and tables

Using the utility statement MIGRATE, the database administrator can

- migrate a SESAM/SQL Version 1.1 database or a database from an earlier version to a base table of the current version
- migrate a CALL DML/SQL table to an SQL table.
- Migrate a CALL DML only table to a CALL-DML/SQL table.

Querying metadata

SESAM/SQL provides the database administrator with access to the database's metadata through information schemas.

In an ESQL program information can be queried using SQL statements (see the "SQL Reference Manual Part 1: SQL Statements" manual) or with the help of the utility monitor (see the "Utility Monitor" manual).

The tables for the information schemas are described in the “SQL Reference Manual Part 1: SQL Statements”.

Changing the partitioning of a base table

The utility statement ALTER PARTITIONING FOR TABLE enables the user to

- add a partition to a base table with primary key (ADD)
- alter the partition boundaries of a partition (ALTER)
- delete a partition (DROP)

A partition can also be added to a non-partitioned base table with primary key, which then becomes a partitioned base table.
When the penultimate partition of a partitioned table is deleted, the table automatically becomes a non-partitioned table (again).

Purpose of the partitioning

A strongly fluctuating data set in a base table means that the data records stored must be handled flexibly. One way to distribute the data to different user spaces is to partition a base table, see section “Partitioned table”. The partitioning of a base table is specified when it is defined with the CREATE TABLE statement. The partitions are frequently formed according to time criteria, e.g. according to the months of the year.

The user can react conveniently to changed requirements for the distribution of the data in a base table using the utility statement ALTER PARTITIONING FOR TABLE.

Database replication

When you work with replications you can perform the following functions:

- Create replication
- update replications or parts of the replication
- Extend the replication by adding user spaces to the replication
- repair of user spaces, the catalog space or the entire database with the aid of replications

The table below presents an overview of the necessary activities and lists the associated utility statements. The right-most column indicates which main function to choose in the utility monitor.

Activity	**Statement and function**	**Main function in the utility monitor**
Create replication	• create a copy of the database • CREATE REPLICATION	Main function COPY & RECOVER / REPLICATION (COP) with continuation forms Form COP.6
Update a replication	• define starting point by means of job variables • make log files available • REFRESH REPLICATION	Main function INFORMATION-SCHEMA Form INF.5 Main function COPY & RECOVER / REPLICATION (COP) Form COP.7
Extend the replication by adding user spaces to the replication	• define starting point using the RECOVERY_UNITS • provide DA-LOG files • make SESAM backup or foreign copy available • REFRESH SPACE	Main function INFORMATION-SCHEMA Form INF.5 Main function COPY & RECOVER / REPLICATION (COP) Form COP.7
Repair, reset or recreate of user spaces, the catalog space or the entire database using replications	• define starting point by means of job variables • make log files available • RECOVER USING/TO REPLICATION	Main function INFORMATION-SCHEMA Form INF.5 Main function COPY & RECOVER / REPLICATION (COP) Forms COP.2.n.3 (n=1,2,4,5)

Table 68: Utility statements and main functions of the utility monitor for the use of replications

You will find a detailed description of how to work with the replication of a database in section “Database replication”.

Using foreign copies of a database

A foreign copy is a copy of user spaces, a catalog space or the entire database created in any way of your choosing. SESAM/SQL allows you to create foreign copies without shutting down the database.

The following sections describe:

- the preconditions that have to be satisfied for the generation of a foreign copy
- how you can generate a foreign copy
- what a foreign copy can be used for

Preconditions for a foreign copy

Before a foreign copy can be generated for individual spaces or from the entire database, the SESAM/SQL system administrator must close the spaces or the database logically or physically in the DBH.

> ! **CAUTION!** If you copy a space without first having closed it logically or physically then the backup cannot be used with RECOVER. If you try to run RECOVER with a backup of this type, SESAM/SQL will issue an error message.

You can close a space or a database in one of the following ways:

- close the spaces of the database logically with the administration statement PREPARE-FOREIGN-COPY

  If you logically close the spaces of a database using the administration statement PREPARE-FOREIGN-COPY then the spaces update is interrupted by means of a transaction lock and the buffer contents are written to a file. However, the files remain open in SESAM/SQL. No update is possible until the foreign copy has been completed. This is achieved by issuing the administration statement within a lock sequence. The transaction lock that prevents updates is then retained until the end of the lock sequence.

  The administration statement PREPARE-FOREIGN-COPY allows you to logically close the whole database or individual user spaces. The spaces addressed with a PREPARE-FOREIGN-COPY form a space set and have the same backup time. PREPARE-FOREIGN-COPY also allows you to enable logging for the specified user spaces.

  If you close the database only logically then you can only generate a foreign copy using tools that are suitable for backing up open files. This includes additional mirror unit division. In contrast, the BS2000 command COPY-FILE is only possible if the database or space has previously been closed.
- close the space physically with the administration statement CLOSE-SPACE

  Although CLOSE-SPACE physically closes a space, it does not protect it against further accesses. To achieve access protection, you must issue the administration statement within a lock sequence.
- close the database physically with the administration statement SET-SQL-DB-CATALOG-STATUS ... STATUS=FREE or with the administration statement PREPARE-FOREIGN-COPY ... CLOSE=YES

  After the database has been closed physically, foreign copies are also possible which require the database files to be physically closed (e.g. SNAPs).
- shut down the DBH with the administration statement STOP-DBH

The precise syntax and functions of the individual statements are described in detail in the “Database Operation” manual.

When you close the database, restart information for the applying of the modifications to the log files is entered in each space. The restart information contains:

- in the case of the catalog space, the CAT-LOG entry in the CAT-REC file up to which changes in the catalog space are present
- in the case of user spaces, the associated DA-LOG entry in the catalog table DA_LOGS up to which changes in the spaces are present

A foreign copy of a database must consist of the CAT-REC file, the catalog space and the user spaces.
The space from which the foreign copy is generated may not have the state “recover pending”, “check pending” or “load running”.

Creating a foreign copy

A foreign copy is a copy of a space, a catalog space or an entire database that was not created using SESAM tools. Below are descriptions of how to create a foreign copy using:

- the replication functions of the storage systems and SHC-OSD
- the BS2000 command COPY-FILE

Generating a foreign copy using replication functions

The following replication functions of the storage systems can be used to create foreign copies under SESAM/SQL:

- TimeFinder/Mirror (Symmetrix):
  Complete copy of the data on additional mirror units, also called **B**usiness **C**ontinuance **V**olume (BCVs)
- TimeFinder/Clone (Symmetrix):
  Directly available copy of the data, implemented as a complete copy (with `/ACTIVATE-CLONE ..., COPY-COMPLETE-DATA=*YES`)

  TimeFinder/Mirror and TimeFinder/Clone can also be employed in conjunction with SRDF.

  The product TimeFinder/Snap cannot be used to create foreign copies as it does not create a complete copy of the original.
- SnapView Clone (CLARiiON CX) :
  Directly available copy of the data, implemented as a complete copy (with `/ACTIVATE-CLONE ..., COPY-COMPLETE-DATA=*YES`)

  The SnapView Snap product cannot be used to create foreign copies as it does not create a complete copy of the original.
- Equivalent Copy (EC, ETERNUS DX):
  Directly available copy of the data, implemented as a complete copy (with `/ACTIVATE-CLONE ..., COPY-COMPLETE-DATA=*YES`)

  The product Produkt SnapOPC+ cannot be used to create foreign copies as it does not create a complete copy of the original.

Foreign copies from mirror units are created by means of the following general operator actions:

- Synchronizing the original and mirror disks
- Separating the original and mirror disks
- Creating a foreign copy of the mirror disk (separate processing)
- If required, resynchronizing the original and mirror disks

The required commands are described in detail in the “SHC-OSD (BS2000)” manual. There you will also find figures which illustrate how a disk is separated and synchronized.

Procedure for generating foreign copies with replication functions

The diagram illustrates how SESAM/SQL, the replication functions of the disk storage systems, and BS2000 interact when foreign copies are created. For explanations.

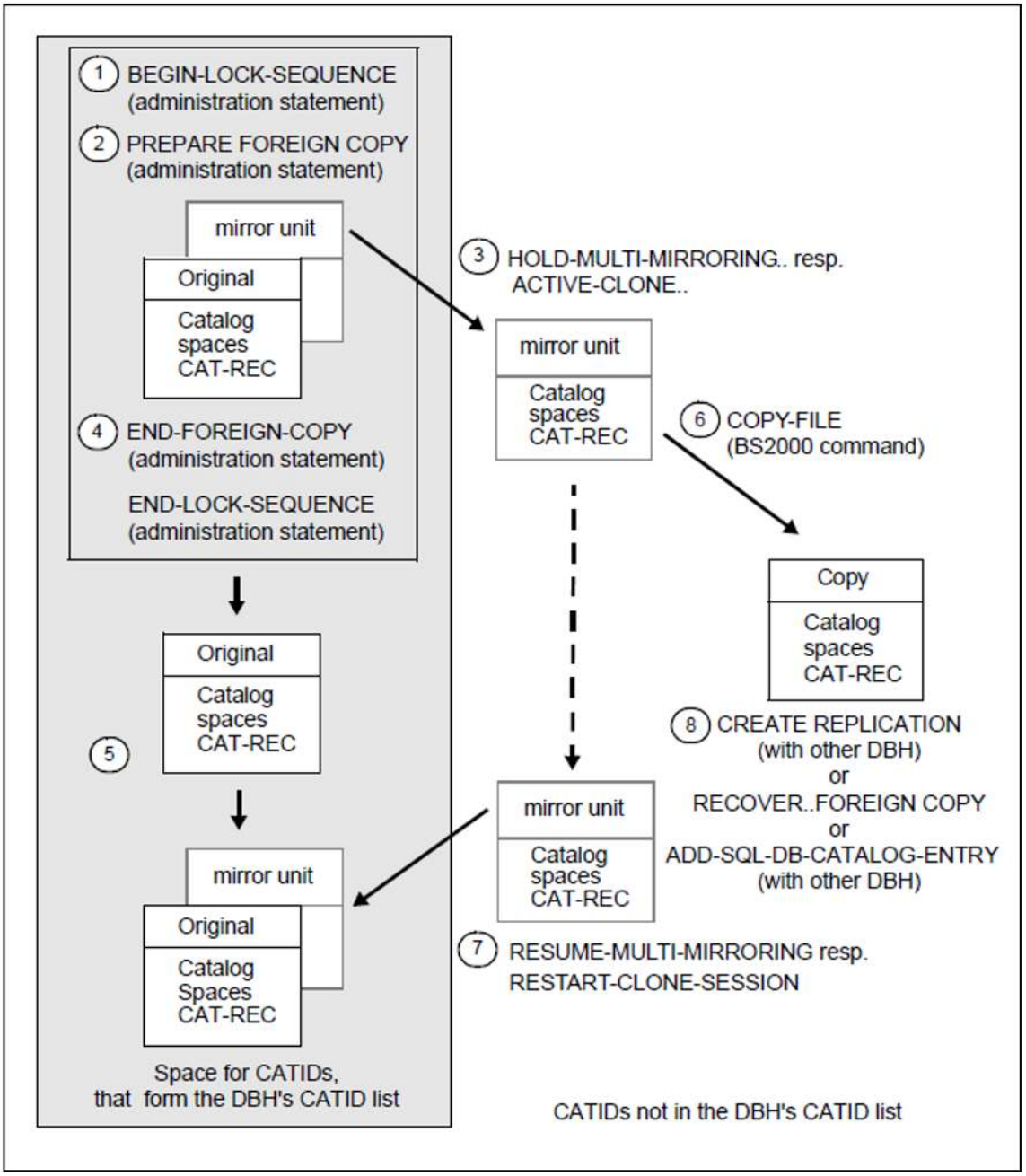

Figure 33: Creating a foreign copy using mirror units

1. Use the administration statement BEGIN-LOCK-SEQUENCE to open a lock sequence.
2. Close the database at logical level with the administration statement PREPARE-FOREIGN-COPY. This interrupts the space update by means of a transaction lock. The transaction lock to prevent updates is retained until the end of the lock sequence. The database consists of the CAT-REC file, the catalog space and the user spaces.
3. Use the SHC-OSD command HOLD-MULTI-MIRRORING to separate the multi-mirror pair on the Symmetrix system. The ACTIVATE-CLONE command is used to separate a clone unit in TimeFinder/Clone and SnapView /Clone.

   You use the NEW-PUBSET parameter to modify the CATID of the mirror unit. The mirror unit now corresponds to a copy of the database with a different CATID but the same name.

- BS2000 requires a modification to the CATID since the Additional Mirror / Clone Unit is to be imported into the same BS2000 system as the original database. The Additional Mirror / Clone Unit can be imported unchanged into another BS2000 system.
- SESAM/SQL requires unique names, that are independent of the CATID, within the set of CATIDs supervised by the DBH.
  To ensure that names are unique, you must assign the mirror unit a CATID that is not present in the CATID list of the DBH which is used to address the original database.

If you have not assigned a CATID list to the DBH or if the copy is to be located at a CATID in the CATID list, then you must change the name of the copy in COPY-FILE (step 6).

4. Use the administration statement END-FOREIGN-COPY to remove a “copy pending” status resulting from a utility statement that preceded the foreign copy. You can also change the database status after PREPARE-FOREIGN-COPY with physical closure of the database files.
5. Use the administration statement END-LOCK-SEQUENCE to terminate the lock sequence and remove the transaction lock blocking updates to the original database. The database will be logically opened the next time it is accessed.
6. Use the BS2000 command COPY-FILE to create a copy of the mirror unit. Please remember that SESAM/SQL requires unique names independently of the CATID itself within the CATIDs considered by the DBH.
   - To ensure that names are unique, you must assign the copy a CATID that is not present in the CATID list of the DBH which is used to address the original database.
   - When you use COPY-FILE, you can choose a new name for the copy. The name only needs to be modified during the copy process if you have not assigned a CATID list to the DBH or if the copy is to be located at a CATID in the CATID list.
7. You can use the command RESUME-MULTI-MIRRORING to reconstruct the multimirror pair on the Symmetrix system. A clone session is restarted in TimeFinder/Clone and SnapView/Clone using the RESTART-CLONE-SESSION command.

   After this a new foreign copy of the original database can be generated.
8. Depending on the intended usage, you must perform one of the following operations in order to use the copy of the mirror unit as a foreign copy for SESAM/SQL applications:
   - CREATE REPLICATION
     You assign the DBH a CATID list that contains the CATID of the copy but not that of the original database. You can now create a replication of the original database using the utility statement CREATE REPLICATION.
   - RECOVER ... FOREIGN COPY
     First of all you must copy the files in the foreign copy itself to the corresponding spaces in the original database. You can then repair or reset the original database with the utility statement RECOVER ... FOREIGN COPY.
   - ADD-SQL-DB-CATALOG-ENTRY
     You assign the DBH a CATID list that contains the CATID of the copy but not that of the original database. You enter the foreign copy as a duplicate of the original database in the SQL database catalog of this DBH.

If the copy is created from the mirrorunit using HSMS then you do not need to perform steps 3 - 7 yourself. HSMS runs these provided that you have specified the parameter CONCURRENT-COPY=*YES(WORK-FILE-NAME=*BY-ADD-MIRROR-UNIT) in the BACKUP-FILES statement.

Physically opened files whose consistency has been ensured by means of PREPARE-FOREIGN-COPY can be backed up with HSMS.

SESAM backup copies can also be created from mirror units, see “Creating SESAM backup copies from mirror units using HSMS”.

Sample procedure

In the procedure below an Additional Mirror / Clone Unit is split using SESADM:

```
/ASSIGN-SYSDTA  TO-FILE=*SYSCMD
/START-SESADM
//START-DBH-ADMINISTRATION ...
//BEGIN-LOCK-SEQUENCE
//PREPARE-FOREIGN-COPY SELECT=*LOGICAL(CATALOG-NAME=...)
//HOLD-PROGRAM
/HOLD-MULTI-MIRRORING ... bzw. /ACTIVATE-CLONE ...
/RESUME-PROGRAM
//END-FOREIGN-COPY SELECT=*LOGICAL(CATALOG-NAME=...)
//END-LOCK-SEQUENCE
//END
```

The database files on the split mirror unit can now be backed up.

Creating a foreign copy with the BS2000 COPY-FILE command

To create a foreign copy using the tools available in BS2000 you must perform the following steps:

1. The SESAM/SQL system administrator causes the DBH to release the original database by issuing the administration statement SET-SQL-DB-CATALOG-STATUS ... STATUS=FREE (see the “Database Operation” manual).
2. The database administrator uses the command COPY-FILE to create copies of all space files (i.e. the catalog space and all user spaces and the CAT-REC file) at BS2000 level. The file names of the copies must conform to SESAM/SQL conventions and must have the following form:

Catalog space	:*catid*:*user_id*.*catalog*.CATALOG
User spaces	:*catid*:*user_id*.*catalog*.*space*
CAT-REC file	:*catid*:*user_id*.*catalog*.CAT-REC
catid	is the BS2000 catalog ID.
user_id	is the BS2000 user ID, which must be the same for all copied spaces.
catalog	is the physical database name of the database duplicate.
space	is the name of the copy of a user space. *space* must be identical for the original and the duplicate of each user space.

3. With the administration statement SET-SQL-DB-CATALOG-STATUS ... STATUS=ACTIVE the system administrator adds the original database to the SQL database catalog of the current DBH session again.

You can back up individual user spaces by closing the space with CLOSE SPACE or INTR CLOSE and then using COPY-FILE to perform the backup. To stop SESAM/SQL from immediately reopening the space on an access attempt, the statements should be issued as part of a lock sequence.

The administration statements PREPARE-FOREIGN-COPY and END-FOREIGN-COPY make it possible to cancel any “copy pending” state resulting from a utility statement that preceded the foreign copy. To do this, you should issue the statement
PREPARE-FOREIGN-COPY before step 1 and END-FOREIGN-COPY after step 3.

Sample procedure

In the procedure below a copy of the database is generated using the COPY-FILE command:

```
/ASSIGN-SYSDTA  TO-FILE=*SYSCMD
/START-SESADM
//START-DBH-ADMINISTRATION ...
//BEGIN-LOCK-SEQUENCE
//PREPARE-FOREIGN-COPY SELECT=*LOGICAL(CATALOG-NAME=...)
//HOLD-PROGRAM
/COPY-FILE ...
/RESUME-PROGRAM
//END-FOREIGN-COPY SELECT=*LOGICAL(CATALOG-NAME=...)
//END-LOCK-SEQUENCE
//END
```

The copied database files can now be backed up.

Creating replications from foreign copies

The database administrator can create replications from foreign copies in the same way as from SESAM backup copies (see section “Database replication”). Using a replication, you can:

- free a DBH from pure read accesses during OLTP operation
- repair an original database
- reset an original database
- create an independent database

If you create a replication or partial replication from a foreign copy then the space must not have the state “check pending” or “recover pending” when the foreign copy is created. The foreign copy is specified by your CAT-REC file. You can use the RENAME parameter in the utility statement CREATE REPLICATION to accelerate the procedure. In this case, the replication is not created by copying the foreign copy but by a renaming operation. However, you should remember that this statement cannot be repeated even if an error occurs. Instead, it is necessary to create a new foreign copy.

Repair and reset using foreign copies

When repairing or resetting with RECOVER, you can use foreign copies instead of the SESAM backup copies created using COPY. Foreign copies can be used to recover the following backup units:

- individual user spaces
- a space list
- the catalog space
- the entire database

A RECOVER on the basis of a foreign copy is generally performed in accordance with the same rules that apply to the same operation using SESAM backup copies (see section “Tasks associated with media recovery” and section “Recovering the database, catalog space and user spaces”). Special characteristics of the use of foreign copies for a RECOVER operation are described below.

Special characteristics of the use of foreign copies for RECOVER operations

The user must copy the files for the foreign copy to the corresponding spaces. To do this, the spaces must be closed. In this case, SESAM/SQL simply updates the log files on the basis of the restart information present in the spaces. This does not affect SESAM backup copies.

RECOVER for user spaces with foreign copy

If you want to repair user spaces using a foreign copy then these spaces must have been logged at the time this foreign copy was created. The DA-LOG entry that is recorded as the restart point in the foreign copy must still be present in the metadata. Logging must not have been interrupted since the foreign copy was created. Any spaces that do not meet these conditions remain unchanged. Such spaces can only be subsequently accessed if they have not been modified since the time the foreign copy was created. If this is not the case, an error message informing you that the space is inconsistent is issued when you attempt to access such a space.

All the user spaces that are processed jointly (space list specification), must have the same time stamp).

If a space was not logged at the time the foreign copy was created then it can only be reset to the status of the foreign copy.

RECOVER for a catalog space with foreign copy

If you want to repair a catalog space using a foreign copy then the restart point must still be present in the CAT-REC file. You can use any CAT-REC file state that was created during or after the generation of the foreign copy of the catalog space.

If the CAT-REC file is part of a foreign copy and is used for repair purposes then only a reset is possible to the status of this foreign copy. In this case, there are no further entries available to which the modifications could be applied.

RECOVER for a database with foreign copy

When the entire database is repaired, SESAM/SQL first repairs the database's catalog space followed by the user spaces.

If a database is repaired using a foreign copy in which the user spaces were backed up before the catalog space then any modifications will be applied to the spaces as required. If the user spaces were backed up later than the catalog space then the restart information is not present in the DA_LOGS table. A repair is not possible then.

If the whole database was not included in the logging upon creating the foreign copy, you can only reset to this state.

RECOVER for a database with foreign copy to a particular time

SESAM/SQL resets the entire catalog to the status at the specified time. When ANY is not specified, *timestamp* must identify the time of a SESAM backup copy of the catalog space. The SESAM backup copy must be entered in the CAT-REC file.

SESAM uses the foreign copy of the catalog space and applies the changes to the CAT-LOG files which were created subsequent to this SESAM backup copy up to the specified time. Subsequently the changes are applied to the DA-LOG files on the foreign copies in the user spaces up to the specified time.

Duplicating a database

You can create a duplicate of a SESAM/SQL database; these duplicates are fully functional databases which have an identical structure and identical contents to the original database. To use a full foreign copy as a database duplicate you must perform the following steps:

1. To make it possible to access the duplicate database, the system administrator adds an appropriate entry to the SQL database catalog with the administration statement ADD-SQL-DB-CATALOG-ENTRY. Here, the system administrator can assign the physical database name any appropriate logical database name that has not yet been assigned. The duplicate database can then be addressed using this logical database name.
2. If different storage media apply for the database duplicate than for the original database, the database administrator modifies the definitions of the storage groups and user spaces with the SQL statements ALTER STOGROUP and ALTER SPACE (see “SQL Reference Manual Part 1: SQL Statements”).
3. If the duplicate's database-specific files are to be created on storage media other than those used for the corresponding files in the original database, the database administrator modifies the duplicate's media table accordingly with the utility statements ALTER MEDIA DESCRIPTION, CREATE MEDIA DESCRIPTION, DROP MEDIA DESCRIPTION (see section “Add the first media record for DA-LOG and PBI files to the media table” and section “Maintain the media table”).
4. Using the utility statement COPY CATALOG, the database administrator creates a SESAM backup copy of the duplicate database (see section “Creating a SESAM backup copy with COPY”).
5. The database administrator uses the utility monitor to delete records from the CAT-REC file that are earlier than the new backup and still refer to the original.
6. Using the utility statement MODIFY, the database administrator deletes all information on SESAM backup copies of user spaces that still refer to the original from the duplicate's RECOVERY_UNITS catalog table (see section “Maintenance of the metadata of SESAM backup copies”).

! **CAUTION!** If the database duplicate is to be used at another server then the access rights must be changed before the duplicate is created. For this to be possible, you must enter the system accesses and authorizations that permit accesses from other server in the database's metadata.

Disk reorganization with SPACEOPT

The SPACEOPT software product (purchased separately) reorganizes volumes of a pubset in order to eliminate the fragmentation of the storage areas which develops over the course of time and to reduce the number of extents of the files.

SPACEOPT cleans up fragmentation by optimally relocating (reorganization) the file extents on the volumes of a pubset, see the “SPACEOPT (BS2000)” manual.

SPACEOPT will also reorganize open files. You should note the following:

- File reorganization involves the actual physical reorganization of file extents. You should therefore only use this function when the I/O load of the volume being reorganized is low.
- During file reorganization, SPACEOPT temporarily locks the corresponding BS2000 catalog entries. If a file is opened or expanded while reorganization is in progress, SPACEOPT will interrupt the reorganization and release the lock in order to allow the transaction to pass. After is has obtained the catalog lock, SPACEOPT resumes processing. This procedure is repeated by up to three times per file before the file is excluded from the reorganization.

Interoperation of SESAM/SQL and openUTM

In conjunction with the universal transaction monitor openUTM (BS2000), SESAM/SQL-Server is a powerful, fully integrated DB/DC system for restartable online transaction processing applications.

SESAM/SQL-UTM applications located on different systems can exchange jobs within distributed UTM transactions using openUTM-D. The processing job is transferred to the computer on which the appropriate programs and databases are available.

openUTM-D communicates with the remote systems and monitors the distributed transactions in conjunction with the transaction monitors on the distributed systems. When openUTM-D is used, the database system works either in local mode or in distributed mode, if SESAM/SQL-DCN is also used.

UTM applications

Architecture

The UTM connection module KDCROOT is interposed between the application program and the connection module in UTM applications (see figure 34).

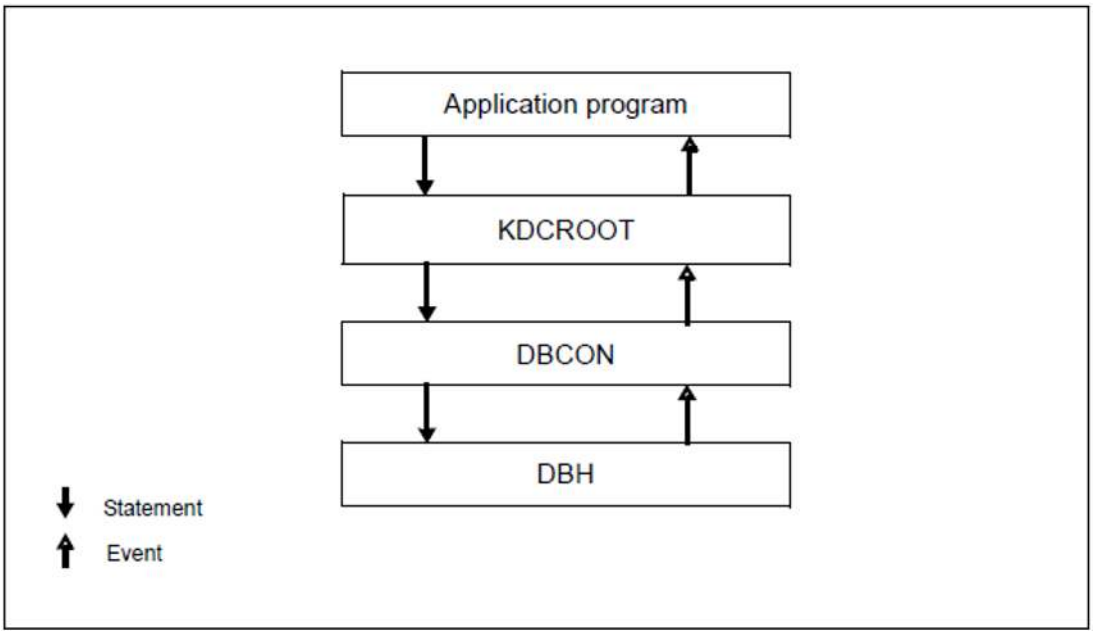

Figure 34: Data transfer between application program and DBH

Creating the KDCROOT connection program

When generating the SESAM/SQL-UTM application it must be specified that the UTM application is to communicate with the SESAM/SQL database system. This is done with the KDCDEF DATABASE statement, see below.

The KDCDB macro which is needed to translate the KDCROOT table module is located in the SESAM/SQL macro library for UTM applications. The other macros are in the UTM macro library. Both the macros from the UTM macro library and the KDCDB macro must be available for assembling.

KDCDEF statement DATABASE

DATABASE ENTRY=SESSQL,TYPE=SESAM [,LIB=*modlib*/LOGICAL-ID(SYSLNK)]	for SQL
DATABASE ENTRY=SESAM,TYPE=SESAM [,LIB=*modlib*/LOGICAL-ID(SYSLNK)]	for CALL-DML

LIB= If the parameter LIB= is set, the SESAM connection module SESUTMC will be dynamically loaded.

modlib Name of the BS2000 library containing the SESAM modules. This dynamically loads the connection module from the module library. Moreover, all other modules pertaining to the SESAM connection module are also dynamically loaded from this module library.

LOGICAL-ID (SYSLNK) The SESAM connection module will be searched for in the IMON installation path for the SESAM/SQL module libraries and loaded from there.

If SQL and CALL DML are to be used, both statements must be specified, see also “Example 2”.

Recommendations

- It is advisable to statically link the SESAM connection module to the UTM application module. In this case you must **not** specify the parameter LIB= in the KDCDEF statement DATABASE. The static link has the advantage that the generated UTM application will not have to be re-generated after a SESAM version change.
- If the SESAM connection module is not statically linked to the UTM application then LIB=LOGICAL-ID(SYSLNK) should be specified. In this case, the UTM application, with regard to SESAM/SQL, is not dependent on hardware- or version-dependent installation paths or library names. A prequisite for this is a correct installation with IMON.

Example 1

The SESAM connection module is statically linked and the application does not contain any CALL-DML program units. The following DATABASE statement is therefore required:

```
DATABASE ENTRY=SESSQL,TYPE=SESAM
```

Example 2

The SESAM connection module is statically linked and the application contains CALL-DML program units. In this case the following two DATABASE statements must be specified:

```
DATABASE ENTRY=SESSQL,TYPE=SESAM

DATABASE ENTRY=SESAM,TYPE=SESAM
```

Example 3

The SESAM connection module is dynamically loaded via the IMON installation path and the application contains CALL-DML program units. Two DATABASE statements with LIB parameter must therefore be specified:

```
DATABASE ENTRY=SESSQL,TYPE=SESAM,LIB=LOGICAL-ID(SYSLNK)

DATABASE ENTRY=SESAM,TYPE=SESAM,LIB=LOGICAL-ID(SYSLNK)
```

Assemble the KDCROOT connection program

Before assembling the KDCROOT connection program, assign the SESAM macro library. The syntax depends on the assembler you are using.

Example 1

Assigning the SESAM macro library in the case of ASSEMBH:

```
/ADD-FILE-LINK LINK-NAME=UTMLIB,FILE-NAME=utm_maclib
/ADD-FILE-LINK LINK-NAME=SESAMLIB,FILE=sesam_maclib
/START-PROGRAM FROM-FILE=$ASSEMBH
//COMPILE  -
//SOURCE=kdcroot
//,MACRO-LIBRARY=(*LINK(LINK-NAME=SESAMLIB)
.
.
```

Example 2

Assigning the SESAM macro library with ASSGEN:

```
/FILE sesam-maclib,LINK=ALTLIB2
/EXEC $ASSGEN
*COMOPT ALTLIB2
.
.
```

System access for SQL applications

In the openUTM system, UTM users identify themselves through their UTM system user ID. This UTM system ID consists of the symbolic computer name, the name of the UTM application and the KDCSIGN or LSES name (in the case of UTM-D on the side of the jobreceiving application).

In order to process a SESAM/SQL database with SQL, users needs an SQL user authorization code. The database must contain a system access for that authorization code.

When a system access is set up, a system user ID is assigned to an existing authorization code. The specifications for a UTM user or a BS2000 user can be entered as the system user ID.

An SQL user is represented by all system accesses in the database system which contain his or her authorization code.

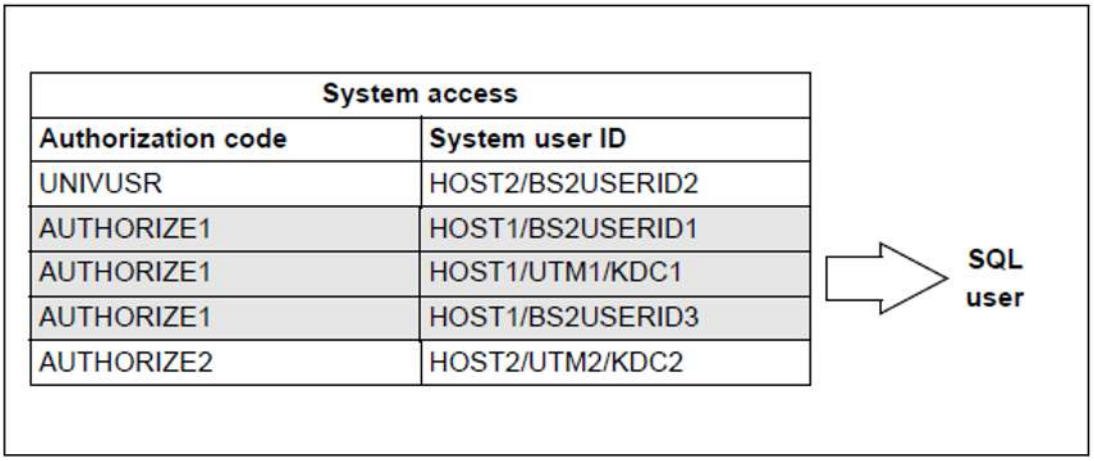

Figure 35: Example of system access by an SQL user

In section “Specifying an SQL user's authorization identifier” there is a description of how an SQL user identifies himself/herself to SESAM/SQL with an authorization code.

Linking a SESAM/SQL UTM application

When linked, the KDCROOT connection program and the various program units form an executable UTM application. SESAM/SQL is linked to the UTM application through the SESUTMC connection module.

The SESUTMC connection module needs to be linked to the UTM application if, as recommended, no library was specified for SESAM/SQL in the DATABASE statement when KDCROOT was generated, see also the “Recommendations”.

If the CALL DML interface is sorted, the SESORT connection module has to be linked to the UTM application.

If UTM is to load program units dynamically, this has to be specified when the UTM application is generated (KDCDEF).

How to link a UTM application is described in detail in the openUTM “ Generating and Handling Applications” manual.

Starting a SESAM/SQL UTM application

Connection module parameters

DBCON require various parameters in order to fulfill its functions. The DBH name and the configuration name are important connection module parameters. They assign the application program linked to the connection module to a DBH and a configuration. Beyond that, there are various other DBCON configuration parameters

Connection modules for UTM applications that work with an independent DBH (DBCON) are generally parameterized through the application program's configuration file (see section “The configuration file”).

The user enters the connection module parameters in the configuration file and assigns them before the application program linked with the connection module is started. The parameters are then transferred to the connection module in the course of the start procedure.

All parameters are DBCON parameters.

The connection module parameters that apply to UTM application are listed below in alphabetical order:

CCSN=*ccs-name*
: Coded character set with which the user program works.

ccs-name:
: Name of the coded character set as defined in BS2000 (system component XHCS) or *NONE if no coded character set is to be sepcified for the user program.
Default: *NONE

If a coded character set was specified for the database (CODE-TABLE *ccs_name* clause in the CREATE CATALOG or ALTER CATALOG statement), the CCS name of the database must match the CCS name of the user program when the user program accesses the database. User program accesses with CCSN=*NONE or accesses by user programs from SESAM/SQL < V5.0 are rejected with SQLSTATE.

This check does not take place for the utility statements CREATE/ALTER CATALOG, CREATE/REFRESH REPLICATION and RECOVER CATALOG [SPACE].

If no CCSN was specified for the database, this check does not take place either.

CNF=k
: Name of the configuration in which the application program is to operate

k
: A..Z, 0..9, ?

Default: ?

DIAG-DUMP=*(diag)*
: Criterion for the generation of a diagnostic dump.

diag: {SQLSTATE=*state*|STATUS=*status*}

state: SQLSTATE with class and subclass (5 bytes);
status: CALL DML status with subnumber (4 bytes)

SQLSTATE can be partially qualified by entering "***" for the subclass.
CALL DML status can be partially qualified by entering "**" for the subnumber.

The first time the relevant error message is issued (SQLSTATE or CALL DML status), a user task dump will be output (see the " Database Operation" manual)

ISOL-LEVEL=*level*

Default value for the isolation level of SQL transactions.
Only effective in SQL applications.

level: {READ-UNCOMMITTED | READ-COMMITTED | REPEATABLE-READ | SERIALIZABLE}
Default: SERIALIZABLE

Valid for all transactions of the ESQL application program if the isolation level was not specified by the appropriate SQL statement in the program.

This parameter is only evaluated by TIAM and UTM applications.

NAM=*x*

Name of the DBH with which the application program is to work

x: A..Z, 0..9, ?

Default: ?

NOTYPE

The message "SESAMxx not available ..." is suppressed

NOUNT

If a long-running search query was interrupted by SESAM/SQL, nostatus is reported to the CALL DML program unit.
If neither UNT nor NOUNT is specified, then the default is NOUNT.
This only refers to CALL DML statements.

NVT

The application can only run locally and it can only communicate with the DBH entered in the NAM parameter. This also applies when SESDCN is loaded.

PREFETCH-BUFFER=*buffersize*

Size in KB of the buffer used for the prefetch.
Value range: 0 <= *buffersize* <= 4096

This parameter is only evaluated by TIAM and UTM applications.

PRIO-CHECK=*t*

Interval in seconds between two consecutive checks of the BS2000 task priority of the application program.

10 <= t <= 3600

Default: 300

PRIO-CHECK=OFF
: The check of the BS2000 task priority of the application program will be deactivated following the first check.

PUF=*p*
: Maximum message length of DCAM or UTM applications in bytes
1 <= *p* <= 64000
Default: 4096

A message may be:

- the request (statement) of an application program to a SESAM/SQL DBH
- the response of a SESAM/SQL DBH to the application program

Since the length of SQL messages is difficult to estimate, we recommend you choose the value 64000 for *p* for applications with SQL statements.

TRACE, TYPE=*type*
: Logging of the call or message trace is activated when the application program starts.

type:
: {CALL|MSG|(CALL|MSG)}
[,OUTPUT={SYSOUT|SYSLST|(SYSOUT|SYSLST)}] Default for output (OUTPUT): SYSLST

Refer to the “ Database Operation” manual for further information on call and message tracing.

TOTAL-APPL=*a*
: Maximum number of SESAM/SQL user applications in the configuration (this only applies to applications that do not operate in distributed mode)

1 <= *a* <= 128

Default: 64

TOTAL-USERS=*u*
: Maximum number of SESAM/SQL user in the configuration (this only applies to applications that do not operate in distributed mode)

1 <= *u* <= 16000

Default: 128

UNT
: Status 16 is passed to the CALL DML program unit if a long-running search query was interrupted by SESAM/SQL.

Only effective for CALL DML statements!

UTMVG={JA | YES}	A DB process is assigned to just one UTM conversation. System resources are allocated to a conversation and not to a user. They cannot be utilized by the next conversation. SESAM/SQL automatically releases the resources when a conversation is concluded.openUTM conversation stacking means that DB conversations can be stacked without restrictions. Calls to SESAM/SQL (SQL or CALL DML statements) are not allowed in the conversation exit. UTMVG=YES must be set if database calls are to be issued in the first part of a UTM sign-on conversation.
UTMVG={NEIN | NO}	A DB conversation can cover several UTM conversations. The system resources are allocated to a user and not to a conversation. Therefore DB conversations cannot be stacked. SQL resources are bound to a UTM conversation and are automatically released at the end of the conversation. The CALL DML resources of a dialog conversation can be re-used in a subsequent dialog conversation. CALL DML calls are allowed in the conversation exits. SQL calls are never allowed in the conversation exit. Database calls are not allowed in the first part of a UTM sign-on conversation. If a further SESAM/SQL UTM conversation is opened, although the previous one is still open, openUTM will issue a corresponding error message. The default is UTMVG=NO.

> ¡ Special start parameters can be specified in a CALL DML application program. In openUTM applications, they are restricted to UNT and NOUNT (see the “ CALL-DM Applications” manual); NAM, NOTYPE and TRACE are not allowed.

Application start parameters

AT load time, the UTM application requires start parameters for UTM, SESAM/SQL and, where appropriate, for the FHS formatting system.

The UTM start parameters are entered as follows:

```
[.UTM] START startparameter[,...]
```

The UTM start parameters are described in the openUTM “ Generating and Handling Applications” manual.

The SESAM/SQL start parameters are specified in the configuration file. If the configuration file is empty, default values are used to parameterize the connection module.

If the FHS formatting system is used the start parameters must be specified as follows:

```
.FHS startparameter
```

The syntax and the meaning of the start parameters are described in the “ FHS (BS2000)” manual.

Example: starting a UTM application with ESQL-COBOL

Before starting a UTM application with ESQL-COBOL program units, you must allocate the SESAM/SQL module library. We recommend you dynamically load the required runtime systems at the UTM application start time.

Add the following statements to the start procedure of the UTM applications:

```
/ADD-FILE-LINK LINK-NAME=SESAMOML, FILE-NAME= sesam-modlib                    1.
.
.
/CONNECT-SESAM-CONFIGURATION TO-FILE= global-configuration-file, -
/                                                                             2.
CONFIGURATION-LINK= link-name
or
/ADD-FILE-LINK LINK-NAME=SESCONF,FILE-NAME= file-name                         3.
/START-PROGRAM FROM-FILE=*MODULE(LIBRARY= userlib ,ELEMENT= element-name -    4.
/                               ,RUN-MODE=ADVANCED -
/                               (ALTERNATE-LIBRARIES =YES) -
/                               )
.UTM utm-startparameters                                                      5.
.FHS fhs-startparameters                                                      6.
.UTM END                                                                      7.
.
.
```

1. Allocates the SESAM/SQL module library.
2. Allocates the configuration file.
3. Calls the DBL dynamic linking loader to start the UTM application.
4. Starts the UTM application
5. Specifies UTM start parameters. You will find a description of the individual start parameters in the openUTM “ Generating and Handling Applications” manual.
6. Specifies the start parameters for the FHS formatting system. You will find a description of the individual start parameters in the “ FHS (BS2000)”manual.
7. Terminates the entry of the UTM start parameters.

Transaction concept

An openUTM transaction begins when a program unit is started and ends when a synchronization point is set by terminating the program unit again (see the openUTM “ Concepts and Functions” manual).

The synchronization point is the point from which a system is restarted after the subsequent transactions have been rolled back.

Exception:
With some methods of termination, such as PEND KP, the program unit terminates without setting a synchronization point and without completing the transaction.

Only one database transaction is allowed to run in an openUTM transaction.

- In the case of CALL DML transactions, termination is initiated with an ETA statement. Following the ETA statement, no other DB statement is allowed within the openUTM transactions.
  The linked-in CALL DML statements “statement; ETA” and “ETA; BTA” are not allowed in openUTM applications.
- Neither COMMIT WORK nor ROLLBACK WORK is allowed in SQL transactions within openUTM transactions. PEND terminates the DB transaction.

When a SESAM/SQL-openUTM transaction is rolled back, both the DB transaction and the openUTM transaction are rolled back.

The SESAM/SQL-openUTM DB/DC system synchronizes the end of the openUTM transaction with the end of the DB transaction (see the openUTM manuals “ Generating and Handling Applications” and “ Concepts and Functions ”).

Synchronization proceeds as follows:

- In the case of CALL DML transactions:
  The DB transaction in the openUTM program unit is caused to stop. The DB transaction is not terminated immediately, since errors might still occur and stop the openUTM transaction from being completed.
  SESAM/SQL informs openUTM of the request to end the DB transaction and delays its termination until the openUTM transaction is ended.
  The openUTM program unit is still busy with jobs belonging to the openUTM transaction but not to the DB transaction, e.g. format or print output.
- In the case of SQL transactions:
  The end of the DB transaction and the end of the openUTM transaction are both instigated simultaneously (with a PEND).

The DB/DC transaction is completed successfully when the DB transaction has been completed successfully and the changes in the UTM areas have been committed. Locked system resources are freed again now.

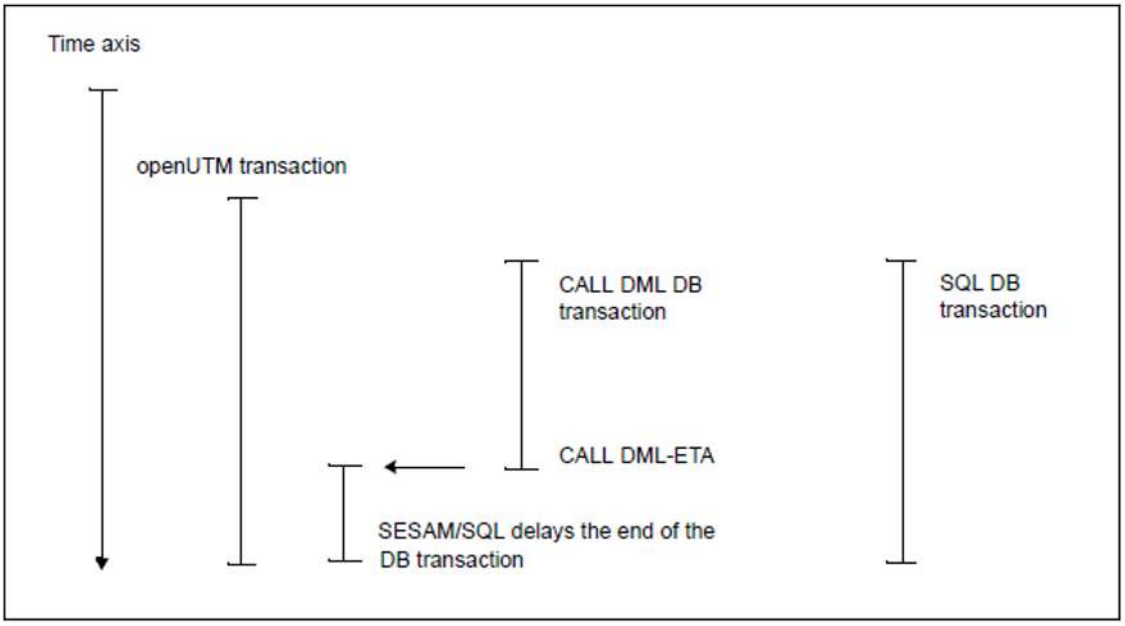

Figure 36: Synchronization of DB transaction and openUTM transaction

Restart

SESAM/SQL and openUTM perform a synchronized restart.

SESAM/SQL needs the data on the processing status of the DB transactions contained in the transaction log files (TA-LOG1 and 2, WA-LOG, DDL-TA-LOG) to do so. openUTM obtains information on open or completed openUTM transactions from the KDCFILE file (containing data on the progress of the openUTM application).

The SESAM/SQL-DBH should always be terminated using the (default) administration statement STOP-DBH UTM-SESSION-INFO=*KEEP (see the “ Database Operation” manual) to make sure that both SESAM/SQL and openUTM have all the information needed for a synchronized restart:
STOP-DBH UTM-SESSION-INFO=*DELETE rolls back all open transactions, closes all logical files and deletes the restart information. The UTM-SESSION-INFO=*KEEP parameter causes the WA-LOG file to remain active and the restart information it contains to be retained. Open transactions are also rolled back and the logical files are closed.

openUTM cold start and openUTM warm start

- An openUTM application that was terminated normally using KDCSHUT performs a *cold start* when the application is started again (see the openUTM “ Generating and Handling Applications” manual).
- If no orderly KDCSHUT was executed, a *warm start* is carried out the next time the application is started.
  A warm start is only successful if the DBH was loaded before. During the synchronization phase, the DBH must be able to supply information on transactions that were not terminated and that were PENDing when the system aborted. That is why the DBH must not be stopped using STOP-DBH UTM-SESSION-INFO=*DELETE.

Depending on how the previous session was terminated, SESAM/SQL and openUTM perform a cold start or a warm start:

End of the previous session		**Effect**	**Start of session**
SESAM/SQL	`/STOP-DBH UTM-SESSION-INFO= *DELETE`	DBH sets “not active” flag in the WA-LOG file, information is deleted	Cold start
openUTM	KDCSHUT N	KDCFILE contains restart information [1]	Cold start
SESAM/SQL	`/STOP-DBH UTM-SESSION-INFO= *DELETE`	DBH sets “not active” flag in the WA-LOG file, information is deleted	Cold start [2]
openUTM	aborted with error	KDCFILE contains restart information [3]	Warm start [2]
SESAM/SQL	`/STOP-DBH UTM-SESSION-INF0= *KEEP`	WA-LOG file remains active, information is updated	Warm start

openUTM	aborted with error	KDCFILE contains restart information [3]	Warm start
SESAM/SQL	aborted with error	WA-LOG file remains active, information is retained	Warm start
openUTM	aborted with error	KDCFILE contains restart information [3]	Warm start

Table 69: Cold start and warm start of an SESAM/SQL openUTM application

[1]KDCFILE may contain restart information on open transactions, but no information on transactions that were not terminated.

[2]Normally not executable because the SESAM/SQL DBH cannot respond to the Request status.[3]KDCFILE may contain restart information on open transactions and information on transactions that were not terminated.

[3]KDCFILE may contain restart information on open transactions and information on transactions that were not terminated.

Appendix

SESAM/SQL files and job variables

This section summarizes all names which SESAM/SQL uses to process BS2000 files and job variables.

The variable used in the tables below have the following meaning:

c	Configuration name (one byte)
n	DBH name (one byte)
st	service task ID (two bytes, between 01 and 64)
ssss	TSN (four digits)
iiii	start value (four digits)
##(*##*)	serial number (two or four digits)
version	serial number of copy (six digits)
catid	catalog identifier
userid	user identifier
dcnid	user identifier of DCN task
catalog	name of catalog
space	Name of the space
replication	name of replication
yyyy.mm.*ddhh:mm:ss*	Timestamp

SESAM/SQL files

The table below lists the session-related files which are created by the SESAM/SQL DBH

File default name	Meaning
:*catid*:$*userid*.SESAM.CO-LOG.*ssss.iiii*	request logging
:*catid*:$*userid*.SESAM*kn*.CURSOR.####	cursor files
:*catid*:$*userid*.SESLK*kn*.CURSOR.####	cursor files used with a linked-in DBH
:*catid*:$*userid*.SESAM*kn*.TA-LOG1 :*catid*:$*userid*.SESAM*kn*.TA-LOG2	transaction log files
:*catid*:$*userid*.SESLK*kn*.TA-LOG1 :*catid*:$*userid*.SESLK*kn*.TA-LOG2	transaction log files used with a linked-in DBH
:*catid*:$*userid*.SESAM*kn*.WA-LOG	log file for restart
:*catid*:$*userid*.SESLK*kn*.WA-LOG	log file for restart used with linked-in DBH

Table 70: Session-related files

Files created by a SESAM/SQL DBH which runs in a service task additionally have the service task id in their names, e.g.:
:*catid*:$*userid*.SESLK*kn****st***.TA-LOG1

File default name	Meaning
:*catid*:$*userid.catalog*.CATALOG :*catid*:$*userid.catalog.space*	Spaces of a database
:*catid*:$*userid.catalog*.CAT-REC	CAT-REC file
:*catid*:$*userid.catalog*.CAT-REC.COPY	copy of the CAT-REC file
:*catid*:$*userid.replikat*.CAT-REC.REPL	CAT-REC file of a replication
:*catid*:$*userid.catalog*.CATALOG.*version* :*catid*:$*userid.catalog.space.version*	backup of spaces
:*catid*:$*userid.catalog.yyyymmddhhmmss.##*	PBI file or HSMS work file
:*catid*:$*userid.catalog.version*.C.####	CAT-LOG file for catalog space
:*catid*:$*userid.catalog.version*.D.####	DA-LOG file for user space
:*catid*:$*userid.catalog.space*.EXC.L	error file for utility statement LOAD

:*catid*:$*userid*.*catalog*.*space*.EXC.U	error file for utility statement UNLOAD OFFLINE
:*catid*:$*userid*.*catalog*.*space*.EXC.C	error file for utility statement CHECK FORMAL
:*catid*:$*userid*.*catalog*.*space*.DDLTA	transaction log file for DDL

Table 71: Database-related files

The table below shows the file which is created by SESDCN.

File default name	Meaning
catid.*dcn-id*.SES.DLG*k*	SESDCN backup

Table 72: SESDCN-related file

The table below shows the utility monitor files.

File default name	Meaning
:*catid*:$*userid*.SESUTI.INPUTLOG.*yyyymmddhhmmss*	output file for statements
:*catid*:$*userid*.SESUTI.STDLOG.*yyyymmddhhmmss*	Default log file

Table 73: Utility monitor-related files

The table below contains an overview of the files used for tape backups

File default name	Meaning
:*catid*:$*userid*.*archive-directory-file.* ARC-PAR	ARCHIVE parameter file
:*catid*:$*userid*.SESAM*kn*.ARC-PAR	ARCHIVE parameter file
:*catid*:$*userid*.*hsms-archiv-name*.HSMS-PAR	HSMS parameter file
:*catid*:$*userid*.SESAM*kn*.HSMS-PAR	HSMS parameter file

Table 74: Utility monitor-related files

The table below contains an overview of the files required for diagnostic purposes

File default name	Meaning
:*catid*:$*userid*.SESAM*kn*.SYSLST	SYSLST log file of the DBH task

:*catid*:$*userid*.SESAM*kn*.SYSOUT	SYSOUT log of the DBH task
:*catid*:$*userid*.SESAM*kn*.SYSLST.SESST91	SYSLST log of the service task
:*catid*:$*userid*.SESAM*kn*.SYSOUT.SESST91	SYSOUT log of the service task

Table 75: Utility monitor-related files

S variables of SESAM/SQL

The table below lists the default names of the S variables which are processed by SESAM/SQL.

S variable default name	Meaning
S variables of the SESADM administration program:	
SESADM-RESULT	Last response of an administration statement (message key)

Table 76: Default names of the S variables

SESAM/SQL job variables

The table below lists the default names of the job variables which are processed by SESAM/SQL.

Job variable default name	Meaning
DBH-specific job variables:	
SESAM.SESDBH.*cn*	state of DBH session
SICHERUNGSINFORMATION.*cn*	TA-LOG state
Database-specific job variables for replication:	
SESAM.*replication*.NEXT-REPL-LOG [1]	contains the oldest CAT-LOG and DA-LOG files necessary for the first REFRESH REPLICATION
Job variables of utility monitor:	
SESAM.SESUTI.JV	state of utility monitor
#SESAM.SESUTI.JV	Status of a utility monitor (concurrent monitoring of multiple utility monitors which are running on the same system and user ID in different tasks)
#SESAM.RU.CATALOG	catalog space backup unit
#SESAM.RU.*space*	user spaces backup unit
#SESAM.RU.CAT-LOG	log file of catalog space backup unit
#SESAM.RU.DA-LOG	log file of user spaces backup unit
Job variables of the SESADM administration program:	
#SESAM.SESADM.JV	Last response of an administration statement (message key)

Table 77: Default names of job variables

[1]The layout of these job variables has changed incompatibly to SESAM/SQL V3.2 because of the 6-digit version number of the SESAM backup copy now used

All SESAM/SQL job variables are defined in the module SEZTXT in the SESAM/SQL module library. The associated source SEZTXT.ASS is supplied as a component of the library SIPANY.SESAM-SQL.<ver>.SPEZ. You can use this to edit the default names of job variables; for more information, refer to the “Database Operation” manual, section “Job variables”.

Maximum sizes of SESAM/SQL

In this section all the maximum sizes are listed which must be borne in mind in SESAM/SQL.

Maximum sizes for base tables

Maximum size of an unpartitioned base table	4 TB
Maximum size of a partition	4 TB
Maximum size of a partitioned base table	64 TB
Number of partitions in a table	16
Number of records in a partition of a partitioned table	268,435,454
Number of records in a partitioned table	4,294,967,264
Number of records in a non-partitioned table	4,294,967,294
Number of columns in a table with data type not equal to (N)VARCHAR	26,134
Number of columns in a table with data type (N)VARCHAR	1,000
Maximum dimension of a multiple column	255
Number of tables or partitions per space	30,000
Number of indexes per space	32,767

Table 78: Maximum sizes for base tables

Database files with their maximum sizes

The following table provides an overview of the database files and the database specific files with their maximum sizes.

File	Maximum size
• Catalog space • user space • SESAM backup copy on disk • Work file on disk for REORG	4 TB on pubsets with “large files”, otherwise 64 GB
• CAT-REC file, CAT-REC copy, CAT-REC.REPL • CAT-LOG and DA-LOG files • DDL-TA-LOG file • PBI file	64 GB
• Work file on tape for REORG • Work file (CRSI file) for CREATE INDEX and IMPORT TABLE • Input file for LOAD, Output file for UNLOAD • Export file for EXPORT/IMPORT TABLE • Error file for LOAD, UNLOAD, CHECK FORMAL or ALTER TABLE	Maximum size supported by the current BS2000 version

Table 79: Database files with their maximum sizes

Maximum values for working with the SESAM/SQL DBH

You can use DBH start statements and options to set limit values for certain parameters. The following table lists the maximum values for important parameters in a DBH session.

Maximum value	**Number**
Entries in the SQL database catalog	254
Entries in the CALL DML table catalog	254
Simultaneously accessible spaces	101 600
Concurrent users	32 767
Concurrent SQL access plans in the plan buffer	999 999
Parallel SQL cursors	999 999
Concurrent suborders (scans in the case of SQL and logical files in the case of CALL DML)	262 143
DBH tasks	16
Service tasks in the case of the independent DBH	64
CO-LOG files	9 999
Sorted rows in an SQL cursor table	2,147,483, 647
Spaces per database	1000
Threads in the case of the independent DBH	1024
Write threads in the case of the independent DBH	512
Maximum value	**Length**
Area for retrieval statements (in columns)	1024
Cursor buffer (in Kbytes)	1 500 000
System-data buffer (in Kbytes)	64 000 000
Transfer container (in Kbytes)	1 000 000
User-data buffer (in Kbytes)	64 000 000
Work container (in Kbytes)	1 000 000

Table 80: Maximum values for working with the SESAM/SQL DBH

Maximum values for working with SESAM/SQL-DCN

The following list provides an overview of the maximum values.
These are based on the system limits valid for distributed processing with SESAM/SQL-DCN:

Maximum values	System limits
Number of applications in a configuration	128
Maximum message length in bytes	64000
Number of entries in the distribution rule	340
Number of DBHs in a distributed transaction	13
Number of DBHs for which requests from a user task can be active concurrently	170
Number of DBHs that can be addressed per configuration for users working in a distributed system	25
Number of remote configurations that can be known in a configuration	406
Number of SQL scans or logical files that a user can work with simultaneously	Approximately USERS *0.75[1]

Table 81: Maximum values for working with SESAM/SQL-DCN

[1]USERS = permitted number of users in the SESDCN session (DCN option)

With dynamic memory management, in exceptional cases it is possible that the maximum number of SQL scans or logical files that a user can work with simultaneously is never reached.

Maximum values for data types

Data type	Range of values
Alphanumeric and national data types (length in digits)	
CHAR	1 <= *length* <= 256
NCHAR	1 <= *length* <= 128
VARCHAR	1 <= *length* <= 32,000
NVARCHAR	1 <= *length* <= 16,000
Numeric data types (Range of values)	
SMALLINT	-2^{15} <= *value* <= $2^{15}-1$
INTEGER	-2^{31} <= *value* <= $2^{31}-1$
NUMERIC	0 or $10^{-\text{-scale}}$ <= *value* <= $10^{\text{precision-scale}} - 10^{-\text{-scale}}$
DECIMAL	0 or $10^{-\text{-scale}}$ <= *value* <= $10^{\text{precision-scale}} - 10^{-\text{-scale}}$
REAL (21 binary digits)	0 or $5{,}4E^{-79}$ <= *value* <= $7{,}2E^{75}$
DOUBLE PRECISION (53 binary digits)	0 or $5{,}4E^{-79}$ <= *value* <= $7{,}2E^{75}$
FLOAT (21/53 binary digits)	0 or $5{,}4E^{-79}$ <= *value* <= $7{,}2E^{75}$
Time data types (Range of values)	
DATE	0001-01-01 <= *date* <= 9999-12-31
TIME	00:00:00.000 <= *time* <= 23:59:61.999
TIMESTAMP	0001-01-01 <= *date* <= 9999-12-31 and 00:00:00.000 <= *time* <= 23:59:61.999

Table 82: Maximum values for data types

Notes on migration

Databases originating from SESAM/SQL V2.x will be automatically migrated when they are accessed for the first time by a DBH of SESAM/SQL V9.1. In the event of migration the CCSN of a database from SESAM/SQL < V5.0 is changed to the value `_NONE_`. Spaces included in the logical data saving will be placed in the “copy pending” state by the migration. It is advisable to make a SESAM backup copy with COPY CATALOG OFFLINE. The first statement can also be COPY CATALOG_SPACE OFFLINE. The user spaces can be saved later on with COPY SPACE.

- Replications cannot be migrated. They must be created.
- SESAM backup copies cannot be migrated. They must be created.
  SESAM backup copies from SESAM/SQL as of V2.x can be opened for reading in SESAM/SQL V9.1.
  Databases originating from SESAM/SQL V1.x can only be migrated with the utility statement MIGRATE.
- Logging files from SESAM/SQL V9.0 cannot be processed in SESAM/SQL V9.1. RECOVER [USING] on the basis of a backup from SESAM/SQL V9.0 is therefore not possible in SESAM/SQL V9.1.

  However, it is possible to reset to a backup from SESAM/SQL as of V2.x with RECOVER SPACE ... TO. In this case the backup is not migrated, but after being read in the space thus created is migrated to SESAM/SQL V9.1. The space is placed in the “copy pending” state and must be saved.

  It is also possible to reset an entire database to a backup from SESAM/SQL as of V3.1 in several stages. Because the catalog space is initially in the “copy pending” state after the reset, the catalog space must first of all be saved before the user spaces can be reset.

  1. Reset the catalog space with RECOVER CATALOG_SPACE ... TO. The catalog space is migrated.
  2. Save the catalog space with COPY CATALOG_SPACE.
  3. Reset the user spaces with RECOVER SPACESET, specifying the time stamp of the catalog backup you want to reset to. Individual spaces can also be reset if only these are required.
  4. Save the spaces in the log. Because of the migration during the reset these have been placed in the “copy pending” state.

A return migration from SESAM/SQL V9.1 to SESAM/SQL V9.0 is possible. For this, please contact your service center.

You will find detailed information on the topic of migration in the Release Notice for SESAM/SQL V9.1.

Glossary

This glossary contains definitions of the most important terms used in the SESAM/SQL manuals.

Terms which appear in *italics* in the explanatory text are explained elsewhere in the glossary.

The “Synonym(s)” line refers to terms with similar or identical meanings that are used in other documentation, but not in SESAM/SQL manuals.

The word “See” accompanying a term refers the user to the term that is actually used in the SESAM/SQL manuals.

In documentation on relational databases, different terms are often used to denote the same thing. Relation, for instance, is also referred to as table, tuple as row, and attribute as column.

In the SESAM/SQL manuals, the terms “table”, “row” and “column” are used. The only exception being the “CALL DML Applications” manual, which uses “attribute” instead of “column”. (This term was used prior to SESAM/SQL V1.1.)

abstract table

A table whose individual rows are not stored persistently. Abstract tables always output the current values at present as determined by SESAM/SQL based on values from other tables. In SESAM/SQL, abstract tables form the basis for views in the information schema.

access authorization

See *privileges*

access handle

This term occurs in the context of *SESAM CLI* calls. An access handle is required when reading or writing a *BLOB value* sequentially. In SESAM/SQL, this sequential processing involves repeatedly issuing the appropriate SESAM CLI calls. During the process, the access handle manages internal information on which BLOB value is currently being processed, and up to which segment of the BLOB value processing has progressed. An access handle is generated using the SESAM CLI call SQL_BLOB_VAL_OPEN and closed using SQL_BLOB_VAL_CLOSE.

access mode of transaction

This is declared in the SQL statement SET TRANSACTION and specifies whether rows can be read (READ ONLY) or updated (READ WRITE) within a *transaction*.

accounting

User-related accounting feature that covers all the services incurred in the course of a session, such as the number of logical and physical file accesses.

activity

Function of the utility monitor used to create or change a contiguous collection of database objects (catalog, schema, table).

additional mirror unit

Additional *mirror disk* in a Symmetrix disk system which can be split off for other purposes (backups, test operations etc.) without impairing current operation.

Synonym: Business Continuance Volume (BCV). In other publications also referred to as: BCV data volume.

administration command

Type of command used by the *system administrator* to monitor and control the *database handler (DBH)* and the SESDCN *distribution component*.

Unlike *administration statements*, administration commands are entered using the BS2000 command INFORM-PROGRAM or by including a *CALL-DML* statement in a CALL DML program. Otherwise, administration commands basically function in the same way as administration statements.

administration statement

Type of statement used by the *system administrator* to monitor and control the *database handler (DBH)* and the SESDCN *distribution component*.

Administration statements are entered using the SESADM administration program, which features an interactive menu interface.

AES (Advanced Encrpytion Standard)

A standard for encryption, originating from the USA, that has been adopted worldwide. It is a block cipher where fixed length blocks of plain text are encrypted into blocks of cipher text of the same length with a fixed algorithm. AES is a symmetric cipher which uses the same secret *key* for both *encryption* and *decrpyption* verwendet. In SESAM/SQL, both the block length and the length of the secret *key* is 16 bytes (128 bits, AES-128).

after image

Block after an update.

aggregate

Collection of atomic values. Within a row, an aggregate represents the range of a *multiple column* in part or in its entirety. The INSERT and UPDATE statements can be used to assign aggregates to multiple columns.

annotation

Special SQL comment. Provides information for executing an SQL or *utility statement*. Depending on its position an annotation only has an effect of one particular operation in the statement.

application

Realization of a task in one or more application programs working with SESAM/SQL.

asynchronous conversation

UTM conversation executing independently of the *user* who started it.

Asynchronous requests are appropriate for tasks and messages where the user does not require return messages for the next dialog step.

atomic column

Column that can only contain a single value, as opposed to a *multiple column* .

attribute

See *column*

attribute value

See *column*

authorization identifier (of an SQL user)

An authorization identifier is assigned privileges which are used to determine which operations (e.g. SELECT or UPDATE) a given user may carry out on the database. The authorization identifier is created by means of the SQL statement CREATE USER and assigned to a *system user identification* by means of CREATE SYSTEM_USER.

autonomous transaction

Autonomously executing *transaction* with its own thread and its own transaction context within a surrounding *transaction*.

base table

Table created using the CREATE TABLE statement. A base table is permanently stored in the *database*. Base tables are also generated as the result of *migration*.

Base tables can have different *table styles*.

Base tables can be partitioned (*partitioned tables*).

The number and *data type* of the columns as well as any *integrity constraints* can be defined using the *SQL* statement CREATE TABLE. ALTER TABLE can be used to modify them. The number of *rows* is not part of the table definition.

batch mode

Mode of operation where a user's request is specified in its entirety and can be handled at a different time from when it was specified. This mode should be distinguished from *interactive mode*.

before image

Block before an update.

Big Endian

A sequence of bytes in memory in a particular coding. In the case of a big endian the most significant byte is stored at the lowest storage address. Big endians are used by SESAM/SQL for the *UTF-16 code units* (each with 2 bytes).

Antonym: *little endian*.

BLOB (Binary Large Object)

A data type used in the storage of multimedia data content in databases, e.g. graphics, video, or sound.

BLOB object

In SESAM/SQL, *BLOBs* are assigned not only a value, but also numerous attributes.

BLOB table

Special base table used exclusively for storing *BLOBs*. This table is created using the SQL statement CREATE TABLE ... OF BLOB.

BLOB value

Actual value of a *BLOB object*. It consists of a sequence of bytes of variable length, up to a maximum of 2^{31}-1 bytes.

block

Physical data unit of 4096 bytes, which serves as the unit of access under SESAM/SQL.
Synonym: data block

block mode

Mode which can be activated for a cursor by the PREFETCH *pragma*. When block mode is activated, the first FETCH-NEXT statement causes the DBH to read several rows of the cursor table into a buffer (*prefetch buffer*). The first FETCH statement only returns the first transferred row to the user. Each further FETCH statement returns the next row from the buffer until the buffer is empty, in which case a further FETCH statement causes further rows to be transferred. Block mode can accelerate processing of a cursor considerably.

block utilization

See *free space reservation*

BS2000 system user ID

See *system user identification*

buffer

Memory area for buffering *blocks* , recovery information, and statement-specific data. The purpose of buffering is to minimize disk I/O operations.
The buffer management of the *DBH* controls the displacement of *blocks* in the case of a buffer overflow.

buffer management

See *buffer*

byte order mark

The Unicode character “zero-width no break space”, NX'FEFF'. It is sometimes used as the first character to show whether a string is available in *little endian* or *big endian* format in the encoding form *UTF-16*.

CALL DML

Collection of statements from a special type of *data manipulation language (DML)* whose functions are activated by a subroutine call included in the application program. CALL DML can be used in transactions.

CALL DML mode

The application program is in CALL DML mode when it processes CALL DML statements.

CALL DML table

See *table style*

CALL DML table catalog list

Contains an entry for each *table* of the CALL DML *table style* that is processed in the course of a *DBH session* . Among other things, the *DBH* is informed of which CALL DML table is assigned to which *database*. There must be an entry in the *SQL database catalog* for each database to which a CALL DML table has been assigned.

The *system administrator* creates the CALL DML table catalog using the *DBH start statement*. Entries can be added or removed using the appropriate *administration statements*.

Cartesian product

The Cartesian product of two *tables* results in a new table containing all possible concatenations of rows from the first table with rows from the second. The number of rows in the resulting table is the product of the number of rows in the two underlying tables.The Cartesian product can also be applied to more than two tables in the same way.

CAT logging

Recovery feature which ensures that all changes to the *catalog space* are recorded in the *CAT-LOG* file. Combined with *DA logging* , CAT logging is capable of recovering a defective database or space (i.e. *media recovery*).

catalog

Named collection of *schemas* of a *database* . In addition to the user-defined schemas, the database’s catalog space always contains the *information schemas* . The database’s *metadata* can be queried using the information schemas.

catalog ID (CATID)

identifies the pubset which contains the BS2000 or SESAM/SQL files (*spaces*, *database-specific files*, *DBH-specific files*). It precedes the database or file name in the form :catid.

catalog recovery file (CAT-REC file)

Database-specific file required for *media recovery*. It contains entries on backup data for the *catalog space*. These entries are created using the COPY *utility statement*.

In the catalog recovery file, each set of backups is numbered sequentially. If a backup is required for recovery using the utility statement RECOVER, it is identified by its sequence number.

A CAT-REC copy is created when COPY is used or when CHANGE-CATLOG is used for administration (*catalog*.CAT-REC.COPY). If a replication is created, a CAT-REC file of the replication is also created (*replication*.CAT-REC.REPL).

catalog space

The catalog space contains the metadata for all *schemas* of a *database* , i.e. the information schemas, user-defined schemas and other types of internal administration information. In addition to the catalog space, there are also *user spaces* .

CATID

See *catalog ID*

CATID list

List of *CATID*s specified by the user, which can be used to restrict a file search. This list can be assigned to the DBH and the utility monitor on startup using the link name SESAMCID.

CAT-LOG file

Database-specific file. Changes to the *catalog space* are recorded in the CAT-LOG files (*CAT logging*). The *media table* defines the media used for the CAT-LOG files.

check constraint

Integrity constraint used to limit the number of data values that satisfy a *search condition* for one or more than one *column* of a *table*. If a check constraint has been defined, rows that do not meet any search conditions specified in INSERT or UPDATE statements are not inserted or updated.

CLI (Call Level Interface)

See *SESAM CLI*

client/server architecture

In this type of architecture, components called “clients” request services from components called “servers”. This role distribution is reflected more or less strongly in its various possible implementations: remote presentation, remote data management, distributed applications or distributed databases.A client/server architecture requires appropriate means of communication between the connected systems. Distributed transaction processing requires global *transaction management*.

code point (Unicode Code Point)

In Unicode, each character is assigned a number, the so-called code point. In SQL a Unicode code point can be specified in the format U&'\xxxx' or U&'\+xxxxxx', where x is a hexadecimal digit. The code points lie in the range U&'\0000' through U&'\10FFFF'.

code unit

A code unit is the unit of Unicode coding. For example, a code unit in *UTFE* is 1 byte (NX'nn') long and in *UTF-16* 2 bytes (NX'nnnn') long, where n is a hexadecimal digit.

coded character set (CCS)

Rules which define the unambiguous assignment of characters in a character set with their representation in bits. A coded character set is identified by its name (*coded character set name*, CCSN).

coded character set name (CCSN)

In a SESAM/SQL database, the CCSN (CODE_TABLE parameter in the *SQL* statements CREATE CATALOG and ALTER CATALOG) specifies the EBCDIC character set with which values stored in columns of the data type [VAR]CHAR are to be interpreted. The CCSN of a SESAM/SQL application (CCSN parameter of the configuration file) specifies the EBCDIC character set with which the application interprets character strings. A terminal's CCSN (CODED-CHARACTER-SET parameter in the BS2000 command /MODIFY-TERMINAL-OPTIONS) specifies the EBCDIC character set with which characters are displayed on the terminal. The CCSN of a file (CODED-CHARACTER-SET parameter in the BS2000 command /MODIFY-FILE-ATTRIBUTES) specifies the character set with which the characters in the file are to be interpreted.

collation

Sorting sequence depending on the encoding of the characters. In the Unicode standard the sequence in which the Unicode characters are collated is defined with the help of the Unicode Default Collation Table (DUCET). This table contains a cardinality of the character at various levels. In SESAM/SQL collation according to the Unicode Default Collation Table is offered via the SQL function COLLATE().

collation element

Collation element of the Unicode Default Collation Table (DUCET). In SESAM/SQL the result of the SQL function COLLATE().

CO-LOG file

DBH-specific file that logs the requests.

The *system administrator* can activate, and specify a volume for, the CO-LOG file using an *administration statement*.

column

Part of a *table* . Each column is assigned a name and a *data type* and contains *column values* of this data type. Columns may be *atomic columns* or *multiple columns* .
The "CALL DML Applications" manual uses the term "attribute" instead of "column".
Synonym: attribute

column constraint

Integrity constraint enforcing the property of a *column* at table definition. The integrity constraint is specified in the definition of the relevant column.

column value

In the case of Dan atomic *column,* the column value is the atomic value of the column. In the case of a multiple value, the column value is an *aggregate* of values of the data type of the column.

The “CALL DML Applications” manual uses the term “attribute value” instead of “column value”.

Synonym: attribute

communication name

See *DBH name*

compound index

See *index*

compound key

See *compound primary key*

compound primary key

Primary key consisting of a more than one *column*.

Synonym: compound key

COMPOUND statement

A COMPOUND statement contains *procedure statements* which are executed in a common context. Common *local procedure variables*, common *local cursors* and common *local error routines* which are all declared in the COMPOUND statement apply for these procedure statements. A COMPOUND statement may not contain another COMPOUND statement. COMPOUND statements cannot be nested.

concurrent access

Attempt by two or more application programs to access the same *row* (record) at the same time. SESAM/SQL coordinates concurrent access by means of concurrent transactions.

concurrent transactions

See *concurrent access*

configuration

An identifiable group of *DBHs*, distribution components (see *SESAM/SQL-DCN*) and application programs residing on the same system, independent of other DBHs, distribution components and application programs. The *utility monitor* is assigned to the configuration as an application program.

A configuration must have a unique configuration name within the system. Global (inter-network) uniqueness is recommended, however.

SESAM/SQL applications can run concurrently and independent of each other if they belong to configurations of different names.

configuration file

BS2000 file created by the user. It contains either the start parameters for the DBH or the control statements for SESDCN or the start parameters for the connection module (DBCON) for an application program and /or the utility monitor.

A configuration file for the DBH contains exclusively DBH start statements and options.

A configuration file for SESDCN contains exclusively DCN control statements and options.

The configuration file for DBCON or the utility monitor can contain both start parameters for DBCON and for the utility monitor.

configuration name

See *configuration*

connection module

Component which is used to connect the *DBH* to the application program. The connection module must be linked to each application program.

consistency

A state in which the data in the database is free of errors. See also *consistency check*.

consistency check

The various components of SESAM/SQL carry out consistency checks within the field of their relevant activities. Any inconsistencies the components find are output to the console or the data display terminal (central error messages) and to SYSLST.

consistency level

replaced by the *isolation level*. The consistency level is supported only for reasons of compatibility to SESAM /SQL < V2.0.

consistency point

Point in time at which the database is in a consistent state. In a DBH session using *transaction management* , the *DBH* ensures that the database is in a consistent state at defined times. These consistency points are used by transaction management as rollback points for restart.

constant

See *literal*

constraint

See *integrity constraint*

continuation form

Form which is available in certain functions of the utility monitor. It enables branching and further processing.

conversation

Related sequence of transactions. For the time of a conversation, resources of the *SQL session* are available, e.g. stored cursors. These resources are not subject to any security measures, i.e. they are dropped at the end of the *DBH session*.

- A DB conversation contains 0 through n DB transactions.
- A UTM conversation contains 1 through n UTM transactions.
- A DB conversation can be assigned to precisely one openUTM conversation or it can extend over several UTM conversations.
- An SQL conversation corresponds to a DB conversation (if it is a pure SQL conversation) or to a DB sub-conversation (if it is part of a mixed conversation).

correlation name

Additional name that can be defined for a *table* or derived table using an *SQL* statement.Synonym: synonym

> The `DO` string is used in the FOR statement. It may not be specified as a correlation name. However, you can use the special name `"DO"`.

cross join

Join operation for which the *derived table* corresponds to the *Cartesian product* of the tables involved.

CSV file

Standardized format for the platform-independent exchange of tabular data (CSV: Comma Separated Values). Such files can be generated using a large number of software products (e.g. using SESAM/SQL or Microsoft EXCEL).

cursor

Pointer within a special type of *derived table*, the cursor table, that allows rows to be retrieved one at a time. A cursor name is defined when the cursor is declared. The cursor description also includes an indication whether the cursor table is *updatable* or subject to a particular order criterion.

cursor buffer

Memory area reserved for the intermediate results of retrieval statements. It is managed by the *DBH*. In the case of a buffer overflow, the DBH relocates the data in it to one or more internal *cursor files*.

cursor file

DBH-specific file that exists in two variations:

- The internal cursor file is used by the DBH to store the intermediate results of retrieval statements.
- The user cursor file is used by the user to store the intermediate results of CALL DML searches. The user cursor file is identified by a *file identifier*.

DA logging

Recovery method that uses the *DA-LOG files* as recovery media to log all changes effected via DML (see *data manipulation language*). DA logging can be applied either to the entire database or to individual spaces. Combined with *CAT logging*, DA logging is capable of recovering a defective database or space (i.e. *media recovery*).

DA-LOG file

database-specific file. Used to log all DML (see *data manipulation language*) changes made to a *database* or a *space* (*DA logging*).

The media required for the DA-LOG file are managed with the *media table*.

data block

See *block*

data definition language (DDL)

Generic term frequently used in database languages to refer to the statements used for the definition of *schemas*, *tables*, *privileges*, and *integrity constraints*. In the *SQL standard*, the statements used for the definition and management of schemas are referred to as “SQL schema statements”.

data manipulation language (DML)

Generic term frequently used in database languages to refer to the statements used for the retrieval and modification of data. The *SQL standard* does not use the term “data manipulation language”. Instead, it refers to “ *SQL* data statements” for the retrieval and modification of data in the narrow sense. It also classifies SQL statements into “SQL transaction statements” (for transaction management), “SQL session statements” (for session control), “SQL dynamic data statements” (for *dynamic SQL*), and the WHENEVER statement for ESQL error handling.

data protection

Protection against unauthorized access.

On the one hand, data protection refers to the protection against unrestricted capture, storage, processing and distribution of the personal data of individuals (also called “privacy”). A number of legal measures (along with commissioners assuring their implementation) have been instituted to deal with this aspect of data protection. On the other hand, data protection includes the measures required to implement it.

Under SESAM/SQL, protection against unauthorized access to *databases* is implemented using a *system entry* . The system entry is made up of the *authorization identifier* and the *system user identification*. In *SQL*, different users can be granted different types of access to the various objects of a database by means of *views* and *privileges*.

For tables of the CALL DML *table style*, access control is implemented using a *password*. On BS2000 system level, the files of the database system can be protected by a BS2000 password.

data type

The data type defines the permissible values for database objects (e.g. *columns*). In SQL, the CREATE TABLE or ALTER TABLE statements are used to define the data type of a column. SESAM/SQL supports numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, REAL, FLOAT, DOUBLE PRECISION), character data types (CHARACTER, CHARACTER VARYING), and date/time data types (DATE, TIME, TIMESTAMP) as well as data types for *vectors* . Each data type can have the *null value* .

database

Related collection of data that is processed, manipulated and administered by the database system.
In SESAM/SQL, a database consists of the *metadata* in the *catalog space* and the user data in the associated *user spaces* . A database is identified by its database name.

database administration

Database administration covers the following functions:

- generating a *database*
- loading and unloading data
- importing and exporting tables
- securing and recovering a database or parts of it; controlling backups
- reorganizing and checking *spaces*

The individual functions can be activated either:

- by including *utility statements* in the application program, or
- using the menus of the *utility monitor.*

database administrator

Person or group of persons responsible for *database administration*. Synonym: database administrator

database catalog

See *CALL DML table catalog list*, *SQL database catalog list*

database file

BS2000 file containing a *space*. Database files can have the following names:

catalog.CATALOG for the *catalog space*

catalog.space (for any other spaces)

where “catalog” stands for the database name specified in the CREATE CATALOG statement, and “space” for the name of the space created using CREATE SPACE.

database handler (DBH)

SESAM/SQL component that analyzes, executes and coordinates all the database accesses of a *DBH session*. The database handler (DBH) is available in two variants:

- independent DBH
  This type of database handler is an independent program system that supports *multi-user operation*. The independent DBH runs independently of the user program. The DBH can consist of several DBH and service tasks; the first DBH task is called the start task, and the others are called followup tasks.
- linked-in DBH
  This type of database handler is linked into, and exclusively processes the requests of, a single application program. The linked-in DBH executes under the same task as this application program.

Synonym: SESAM/SQL DBH, DBH

database management system

See *database system*Abbreviation: DBMS

database system

Software system which supports all tasks in conjunction with managing and controlling large data sets. The processes contained in the database system lead to stable, redundancy-free, and expandable data organization. The database system enables concurrent access by multiple users to the same *databases* and ensures that the data resources remain in a consistent state. Synonym: database management system

database-specific file

File containing database-specific information maintained for each database. Information on database-specific files is stored in the *media table*.The following are database-specific files: *DA-LOG file CAT-LOG file*, *catalog recovery file* and *PBI file*.

DB user ID

The BS2000 user ID, is not the same as the *DBH user ID*, but is the user ID under which the database, i.e. the *catalog space*, its CAT-REC file and the *user spaces* are stored. The *database* can be stored under a DB user ID or under the *DBH user ID.*

DB/DC system

Database system (DB) which is combined with a data communication system (DC) to permit the common use of processes and concurrent accesses. Global and synchronized security measures guarantee that the databases and files in the entire DB/DC system are in a consistent state at all times. SESAM/SQL features a fully synchronized DB/DC system under the control of the transaction monitor openUTM.

DBH

See *database handler*

DBH name

It uniquely identifies a *DBH* among the other DBHs in a *configuration*. Some *DBH-specific files* are also identified by the DBH name.

The *system administrator* defines the DBH name using a *DBH option*. Default name: '?'

Synonym: communication name

DBH option

DBH options are parameters for the *DBH*, defined by the *system administrator* at startup time. DBH options affect the limits, resources and execution rules for the current *DBH session*.

DBH session

Period of time between starting and terminating a *DBH*.

DBH start statement

DBH start statements initiate reading of the *DBH options* and the insertion of entries in the *SQL database catalog list* or the *CALL DML table catalog list*.

The *system administrator* enters the DBH start statements when starting the *DBH*.

DBH task

BS2000 task under which the DBH basic functions are run. In *multitasking* the DBH can be loaded with multiple DBH tasks. Some partial functions of SESAM/SQL can also be run under *service tasks*.

DBH user ID

BS2000 user ID under which SESAM/SQL DBH was started as database server.

DBH-specific file

File that is created by the *database handler (DBH)* for the duration of a session.
The following files are DBH-specific: *cursor file* , *TA-LOG files* , *WA-LOG file* , *CO- LOG file* and the file for logging the DBH message buffer.
Information on the storage devices for the cursor file, the temporary files, the TA-LOG and WA-LOG files is held in the *media catalog* .

DCN

See *SESAM/SQL DCN (distribution component)*

DCN option

DCN options are start parameters for the *SESAM/SQL-DCN* distribution component.

DDL TA LOG file

A DDL-TA-LOG file is used to protect long-running DDL and SSL statements that execute in a service task. The DDL-TA-LOG file takes some of the load off the TA-LOG files. This file is assigned to the relevant space and is deleted when the transaction terminates.

deadlock

State in which two or more *transactions* mutually lock each other.

decryption

The conversion of cipher text into plain text using an *encryption* algorithm with a *key*. In SESAM SQL, this is performed with the SQL function DECRYPT().

default value

See *SQL default value*

default value character

Character that has to be individually defined for each *column* not contained in the primary key in *tables* of the CALL DML/SQL table style. Each column of this type is assigned a *non-significant attribute* value by applying the appropriate default value character.

derived table

Table that is the result of a query expression.

descriptor area (SQL)

Data structure for the description of the input and output values of *dynamic SQL* statements. Each value and its type are represented in the item descriptor area. In the case of *multiple columns*, there are several entries for one value (*vector*).

diacritical mark

A mark linked to a basic character or symbol, e.g. accent, tilde.

direct update (CALL DML)

Update performed using the CALL DML interface.

dirty read

Phenomenon that can occur if a particular *isolation level* has been specified.

A *transaction* updates or adds a *row*. A second transaction ignores the *record lock* and reads the row before the first transaction has committed it. So the second transaction has read a row that can still be changed by the first transaction, i.e. that was not in its final state.

distributed data management

See *distributed databases*

distributed database handler

The entirety of *database handlers* taking part in *distributed processing*.

distributed databases

Type of data management which implies that the associated databases are distributed over several processors (distributed management) and are subject to *distributed processing*. Distributed databases enable application programs to use one database locally on the processor on which data is most often needed. This data can also be queried and changed within a transaction by application programs from other processors.

distributed processing

The distributed processing component of SESAM/SQL is *SESAM/SQL-DCN*.

Distribution processing enables an application program to process more than one *DBH* in one session in a transaction.

Intra-configuration or inter-configuration as well as intra-processor or interprocessor processing is possible. The application program does not know which DBH it is using at a particular point in time. The *distribution rules* control the assignment of DBHs to a database as well as the distribution of databases over the network.

Distributed processing is a major asset if the processed databases reside on different processors (*distributed databases*). A openUTM application can be distributed over several processors using UTM-D regardless of whether any databases are distributed under *SESAM/SQL-DCN*.

distributed transaction

See *global transaction*

distribution rule

For *distributed processing,* the distribution of the databases to the individual processors must be known to *SESAM/SQL-DCN.* Within each processor, each database must be assigned to a *configuration*.

In the distribution rules, the *system administrator* specifies which *database* resides on which processor, which *DBH* is to be used to access the database and which *distribution component* passes on database accesses by application programs from other configurations.

dyadic operator

Operator comprising two operands, e.g. the basic arithmetic operators +, -, * and /.

dynamic SQL

Dynamic SQL offers *preparable SQL statements* that can be compiled and executed while the SQL application is running, as well as statements for setting and getting *SQL descriptor areas*. Dynamic SQL can be applied in interactive query or update programs that use variables, for example.

ECM (Electronic Codebook Mode)

A method for encrpyting plain text greater than a block in length with a block cipher, such as the *AES* cipher used in SESAM SQL. In ECM, long plain text is divided into several blocks, and each of these blocks is encrypted separately, with the same *key*. The last of these blocks may have to be padded. In SESAM SQL, this method is used to encrypt values longer than 16 bytes.

If the plain text contains a repetition of some byte sequence such that two blocks of plain text are exactly identical, the corresponding blocks with cipher text will also be the same. Such repetitions can therefore be detected in the cipher text without knowledge of the *key*.

embedded SQL statement

SQL statement embedded in a host language (e.g. COBOL) program. It is started by means of EXEC SQL and ended by means of END-EXEC. This allows SQL statements to be clearly distinguished from host language statements and precompiled.

encryption

The conversion of plain text into cipher text, using an encryption algorithm with a *key*. In SESAM/SQL, this is performed with the SQL function ENCRYPT().

escape character

The escape character is defined in the ESCAPE clause of a LIKE *predicate*. It must immediately precede the character “%”, “_” or another escape character in the comparison value. The escape character disables the placeholder or escape character functions of the “%” or “_” character, or another escape character. As a result, these characters are interpreted as normal characters.

exception file

File which the user

- can specify when changing the *data types* of one or more *column*s of a *base table*. If conversion errors occur, SESAM/SQL writes the original column values together with the related error message to the specified exception file without alerting the user.
- can specify when loading a base table. It contains information about corrupted records in the used input file and the cause of the error.

File created if necessary by SESAM/SQL for CHECK FORMAL, LOAD, or UNLOAD. This file contains information about corrupted records and the cause of the error. SESAM/SQL creates a separate exception file for each of these three utility statements, which is then updated with each new error that occurs.

export file

BS2000 file in which the metadata and user data of a base table are stored when the table is exported. Using the appropriate *SQL* statements, the user can then decide which user data is to be transferred to a new table. The export file cannot be edited manually, however, and is used exclusively for importing a table.

external restart

Restart following a system crash. It is performed by the *DBH* as soon as it loaded after an abort session.

file identifier (CALL-DML)

Two-digit identifier for a *logical file*. It is assigned when a logical file is opened using the CALL DML open statement. It can also be used to identify a *cursor file*.

follow-up update (CALL DML)

Update that is formulated using a previously entered *direct update*, but has different *attribute values*.

foreign copy

A copy that was created by some BS2000 means and not by the SESAM statement COPY. A foreign copy can be used as the basis for carrying out a *recovery* or creating a *replication*.

foreign key

Column in a *table* referencing a particular column that is a PRIMARY KEY (see *primary key constraint*) or UNIQUE (see *uniqueness constraint*) in another table. A foreign key is defined by means of the *referential constraint* that establishes the connection between the foreign key and the referenced column. The table containing the UNIQUE or PRIMARY KEY column is the referenced column. The table to which the referential constraint is applied is the referencing table. Referential constraints can also be applied to combinations of columns.

form

A screen form used for input and output of data. Forms are used for the user interface of the *utility monitor* , the SESADM user interface, and the SESMON user interface.

form name

name of a form. It is displayed in the status section of the screen when using the *utility monitor*.

form short name

Three-digit mnemonic name of a form. It is displayed in the status section of the screen when using the *utility monitor*.

free space reservation

The free space reservation specifies what percentage of each *block* is to be left free following execution of a LOAD or REORG (*space) utility statement.* Free space reservation is defined using the PCTFREE parameter of the CREATE CATALOG or CREATE SPACE statement. The block utilization specifies the maximum percentage of a block that may be occupied following a LOAD or REORG operation.

function identifier (CALL DML)

The function identifier specifies the functions available for the time of validity of the current *file identifier*.

global configuration file

The global configuration file combines the configuration data for several compo- nents in a single file (see *configuration file*).

global transaction

A transaction which is distributed by SESAM/SQL-DCN over multiple *database handlers*.

group commit

Collective commit of a number of “end transaction” calls.

host variable

Variable in a host language (e.g. COBOL) referred to in an *embedded SQL statement.* Host variables are prefixed with a colon. They are declared in the DECLARE section.

independent DBH

See *database handler*

index

Tree-like access structure referencing the *rows* of a *table* assigned to a *column* or column combination. Using an *inverted list* , an index assigns rows containing these values or value combinations in the corresponding columns to each value
or value combination of the underlying column(s). The primary purpose of indexes is to speed up data retrieval.

An index is called a one-column index if it refers to only one column. It is called a compound index if it refers to a number of columns. An index has a unique name within the schema in which it is defined and is stored in a *space* . An index is created using the *SQL* CREATE INDEX statement.
SESAM/SQL requires an index for each column (combination) for which a UNIQUE constraint has been defined. If an index has already been defined for the corresponding column(s), it is also used for the UNIQUE constraint. Otherwise, the required index is created implicitly. The name of an index that has been created implicitly starts with UI, followed by a 16-digit number.
An index to which the columns of the *primary key constraint* apply is also called a primary index or primary key index. An index which is not a primary index is called a secondary index.

index browsing (CALL-DML)

Function of *CALL DML* used to determine the frequency of attribute values (see *column*).

index space

User space in which at least one *index* is stored. A *space* is defined as an index space of a specific *base table* when at least one *index* of that *base table* is stored in it.

indicator variable

Special type of *host variable* of the numeric *data type* SMALLINT, which is assigned to another host variable. The indicator variable indicates whether the other host variable contains the *null value* or whether data was lost during the transfer of character-string values.

Information schema

See *schema*

INFORMATION_SCHEMA

See *schema*

inner join

Join operation which returns all rows that satisfy the join condition as the *Cartesian product* in the *derived table*.

Integrity constraint

Rule that applies constraints to the value range of a *column* or a number of columns as determined by their *data type*. An integrity constraint limits the number of rows in a base table. It can be defined for a column (*column constraint*) or a *table* (*table constraint*). The following constraints are available in SESAM/SQL: *CHECK*, *UNIQUE*, *primary key*, *referential* and *NOT NULL constraints*. The database system enforces these constraints.

interactive mode

Mode in which a user request is the result of a sequence of steps executed inter- actively at the terminal. The interactive mode should be distinguished from *batch mode*.

internal restart

Restart following minor internal errors. The internal restart is performed by the *DBH* without interrupting the current session.

internal statement format

Optimized format of a *CALL DML* statement, generated internally from the relevant user statement by the *DBH* . Internal statement formats can be reused in follow-up statements.

inverted list

Reference list assigned to a *column* of a *table*. For each column value, it contains a reference to the *rows* in the table containing the corresponding values. There are also inverted lists for column combinations. Inverted lists are required for secondary indexes (see *index*).

Isolation level

The isolation level specifies the degree to which consistency can be affected by *concurrent* (updating) accesses of another transaction when reading rows in a *transaction*.

The following isolation levels can be specified: “READ UNCOMMITTED”, “READ COMMITTED”, “REPEATABLE READ” and “SERIALIZABLE”. By using the appropriate isolation level, it is possible to prevent the following phenomena: “ *dirty read*”, “ *non-repeatable read* and “ *phantoms*”.

join

For *SQL* : the join is effected over two or more tables using the values in the *join column(s)* . SESAM/SQL knows four types of join then *cross join* , the *inner join* , the *outer join* and the *union join* .
For *CALL-DML* : the join is realized by means of the *search with join* realisiert.

join attribute

See *join column*

join column

Column of a *table* whose values are compared with a column of the same type in the same or another table two at a time. If the comparison results in the truth value “true”, the row returned by the *join* is included in the *derived table*.

join expression

Query expression containing the keyword JOIN. It specifies the *tables* to be joined, the type of *join* (*cross* , *inner* , *outer* or *union join*) to be performed and the join condition.

key

A sequence of bits used in an algorithm for *encryption* or *decryption* . The *AES* cipher used by SESAM/SQL employs the same key for both operations (symmetric cipher). Thus the key has to be kept secret.
The key used by SESAM/SQL is an alphanumeric character string with16 bytes (128 bits).
There are ciphers with different keys for encryption and decryption in which only one of the keys has to be kept secret (public key cryptography).

linked-in DBH

See *database handler*

literal

Character string that represents a fixed value. SESAM/SQL supports alphanu meric and special literals, numeric literals and date/time literals.

Little Endian

A sequence of bytes in memory in a particular coding. In the case of little endian the least significant byte is stored at the lowest storage address. Antonym: *big endian*.

load option

See *DBH option, DCN option*

local cursor

With the definition of local cursors, cursors are defined which can be addressed in the *COMPOUND statement*.

local error handling routine

The definition of local error routines determines what response is made when, during processing of a *procedure statement* in the context of the *COMPOUND statement*, an SQLSTATE '00000' is reported.

local procedure variable

Local procedure variables are variables which can only be addressed in the *COMPOUND statement*. A data type and, if required, a default value is defined for them. They have no indicator variable.

lock sequence

Lock sequences play an important role in the administration of the DBH. A lock sequence can be specified in the form of a time period. During this time period all locks on database catalog lists and spaces requested by the user are activated. The lock sequence is initiated using the DBH administration statement BEGIN-LOCK-SEQUENCE and concluded using END-LOCK-SEQUENCE. This closing statement also implicitly releases all locks.

locking concept

See *transaction concurrency*

locking granularity

Unit of data to be locked, e.g. rows, records, tables.

logical after image

Record or part of a record within a *block* after it has been updated. Logical afterimages (LAIs) are logged to *DA-LOG files* and *CAT-LOG files* (as part of *media recovery*).

logical before image

Record or part of a record within a *block* before it has been updated. Moreover, the logical before-image (LBI) contains system-internal rollback information. LBIs are used by *transaction management*to roll back open *transactions*.

logical data saving

See *DA logging*

logical database name

Name used by the application program to access a SESAM/SQL database. The logical database name may be the name of an existing database. In this case, the logical database name is identical to the *physical database name*. f no SESAM/SQL database with the logical database name exists, the logical database name is assigned to an existing database using the physical database name in the *SQL database catalog list*.

logical file (CALL DML)

A portion of a CALL DML *table* defined by the user. It is the user-specific view of a CALL DML table.

A logical file is identified by a *file identifier*.

longlock

Status where a *transaction* has been inactive for a long period of time (lock time) and occupies resources. The *DBH* automatically releases longlocks by rolling back transactions that exceed the lock time value. The default lock time value is 4 minutes. This can be changed either by means of a DBH option, or during operation by means of an administration statement.

main function

Function which allows processing of a number of utility monitor forms as one unit.

media catalog

The media catalog contains storage information for certain *DBH-specific files* (cursor file, temporary file, TA-LOG file, WA-LOG file).

It indicates the volume, *catalog ID* and storage assignments for the files to be created. The media catalog is created by the *system administrator* using *DBH options*. The media catalog is not held in a file; it resides in the *DBH* for the course of a *DBH session*.

media recovery

Recovery measure designed to protect data against loss, e.g. due to a hardware error. Media recovery includes data backup and *DA logging* as well as recovering data to a previous or the current state.

Recovering data to a previous state conveys data to the state of consistency it was in at the time the backup copy was taken. Recovering data to the current state means that all the changes effected in the *database* or a *space* since the last copy was taken are applied. This conveys the database to the state it was in when the failure occurred.

In SESAM/SQL, the smallest recovery unit is the *space*.

media table

The media table contains the characteristics of and the storage devices for *database-specific files*. The media table is held in the *catalog space*. The user can query the contents of the media table using the views of the information schemas (see *schema*). When creating a database-specific file, the *DBH* refers to the information kept in the media table. The media table can be processed using the *utility statements* CREATE/ALTER /DROP MEDIA DESCRIPTION FOR.

metadata

Type of data required by the database system to manage the *user data*, e.g. to describe its structure. Metadata is held in the *catalog space*.

migration

Refers to the conversion of databases (base tables) created under SESAM/SQL V1.x into *base tables* of a SESAM/SQL database of the current version. Migration is effected using the *utility statement* MIGRATE. Depending on the characteristics of the database to be migrated, migration results in *tables* of different *table styles*.

mirror disk

Disk set consisting of at least two disks having identical contents.

mixed mode

Operating mode enabling SESAM/SQL databases to be processed both using the *SQL* and the *CALL DML* interface.

monadic operator

Operator with just one operand. The signs + and - are examples of monadic operators.

multi-db operation

Operating mode enabling each user to work with a maximum of 254 databases at the same time.

multiple attribute

See *multiple column*

multiple column

Column which can contain more than one value of the same *data type* for each row. Each of these values is called an occurrence.

Synonyms: multiple attribute, multiple field

multiple field

See *multiple column*

multitasking

The multitasking architecture can be used to load the *DBH* with multiple tasks in conjunction with high performance requirements. This allows the DBH load to be distributed to several processors in multi-processor systems.

multithreading

Method enabling the *DBH* to utilize the CPU more efficiently.

Multi-threading allows the DBH to process multiple requests at the same time by means of threads. Each thread contains information on the current status of a particular request. If a request has to wait for an I/O operation to be completed, the DBH uses the CPU to process another request.

The number of threads and thus the number of parallel requests can be defined by the *system administrator* using a *DBH option*.

multi-user operation

Operating mode enabling multiple users to work with one *DBH* at the same time.

noncharacter

Noncharacters are Unicode characters which are permanently reserved for internal purposes and may not occur in SQL data. They comprise the 66 Unicode characters U&'\FDDx', U&'\FDEx', U&'\+0xFFFE', U&'\+0xFFFF', U&'\+10FFFE' and U&'\+10FFFF', where x is a hexadecimal digit. In *SQL*, their use in literals or host variables results in an error.

non-repeatable read

Phenomenon that can occur if a particular *isolation level* has been specified.

A *transaction* reads a *row* without locking it against the access of other transactions. A second transaction updates or deletes this row and commits it (COMMIT WORK, end transaction) while the first transaction is still open. If the first transaction reads the row again, it either finds the values to be changed or it fails.

non-significant attribute value

For *tables* of the CALL DML *table style*, an *attribute value* containing exclusively *default value characters*. This value is not stored in the database. This type of default value character must not be confused with the *SQL default value*.

normalize

Conversion of a Unicode string in which characters and strings which can be represented in various ways in Unicode are represented uniformly. In Normalization Form D, characters are dismantled as far as possible; in Normalization Form C, compound characters (e.g. “Ä”) are also used. In *SQL*, Unicode strings should be stored in Normalization Form C.Normalization is offered by SESAM/SQL in the SQL function NORMALIZE().

NOT NULL constraint

Integrity constraint that requires a *column* to contain a value other than the *null value*. If a NOT NULL constraint has been defined for a particular column, SESAM/SQL prevents the null value from being entered in this column of a particular *table* row.

NULL value

Special value that indicates that the contents of a *column* or the result of an *expression* is undefined or unknown.

occurrence (of a multiple column)

See *multiple column*

outer join

Join operation that results in the *derived table* containing not only the result of an *inner join* but also each row in the *join column* of one table for which no matching value could be found in the join column of the other table. The missing values are represented by *NULL values* in the derived table. The keywords LEFT, RIGHT and FULL can be used to define from which of the two tables the additional rows are to be fetched.

partial replication

A *replication* that does not contain all the *spaces* of the original database.

partition

Area (*rows*) of a *partitioned table* which is saved on a space. The partition boundaries are determined via intervals of the *primary key* value.

partitioned table

Base table with a special physical structure, the *partitions*. Up to 16 partitions can be defined for a partitioned table.

password (CALL DML; SEPA)

Only applies to *tables* of the CALL-DML *table style*. Three-character string which enables the CALL DML user to access protected CALL DML tables.

A password can allow access to individual records (rows) or *attributes* of a CALL DML table, or a particular access mode, i.e. read, write, read/write. The passwords and information on the access authorization are kept in the *password catalog*. The password catalog is created using the SEPA utility.

password catalog (CALL DML)

The password catalog only applies to *tables* of the CALL DML *table style* . It is only activated if the *CALL-DML* interface is used. The password catalog is created and maintained using the SEPA utility. It contains the list of passwords and information on access authorization required for access control.
A CALL DML table that is subject to SEPA access control is assigned its own password catalog on the same *space* .

PBI file

database-specific file . PBI files are required by SESAM/SQL to ensure the consistency of SESAM backup copies (as disk copies or with ARCHIVE) created online using the COPY utility statement. SESAM/SQL logs the *physical before-images* (PBIs) of all blocks which are updated by SQL or CALL DML statements in the course of a copy operation.

persistent data

Data is called persistent if it exists until it is deleted by the user. This is not true of data which is deleted automatically at the end of a *transaction* or *session*, or an ESQL-COBOL program.

phantoms

Phenomenon that can occur if a particular *isolation level* has been specified.With a query (e.g. *SELECT expression*, *query specification*), a *transaction* selects rows from a table that satisfy certain conditions. While this transaction is still active, another transaction adds rows to the table that also satisfy the condition. If the first transaction repeats the query, the result also contains the added rows.

physical after image

Block after an update. The physical after-image (PAI) is required to ensure physical consistency in the event of a restart.

physical before-image

Block before an update. Physical before-images (PBIs) are required by SESAM/SQL to ensure the consistency of backup copies of the *database*created online using the COPY *utility statement*. PBIs are also needed to restore physical consistency after a restart.

physical database name

Name of an existing SESAM/SQL database. A physical database name may differ from its corresponding *logical database name*, which is used to access a SESAM/SQL database.

plan buffer

Memory area maintained by the *DBH* to buffer *SQL access plans* .

pragma

Special SQL comment. Provides information for executing an SQL or *utility statement* . A pragma affects the entire statement, including the views used. The PREFETCH pragma even affects all operations with a *cursor* .

predicates

Operation contained in a *search condition*. Predicates can return: true, false, or unknown.

prefetch buffer

Area of memory in which the rows read in by a FETCH NEXT statement are buffered in *block mode*.

prefetch cursor

Cursor for which *block mode*was activated by the PREFETCH *pragma*.

preparable SQL statement

SQL statement that is not compiled until the program in the host language (e.g. COBOL) is executed. Thus, database queries which were not known when the program was generated can be formulated dynamically.

primary index

See *index* Synonym: primary key index

primary key

One or more *columns* that uniquely identify a *row* in a *table*. A primary key is defined using a *primary key constraint*. Only one primary key may be defined for each table.

primary key constraint

Integrity constraint for a *column* or combination of columns of a *table*. The primary key constraint requires that the *UNIQUE constraint* be fulfilled and that the associated column (combination) not contain the *null value*. Only one primary key constraint may be defined for each table.

primary key index

See *index*

privileges

Privileges define the *SQL* statements the user is permitted to execute on a *table* or *column*. SESAM/SQL distinguishes between SELECT, INSERT, UPDATE, DELETE, EXECUTE and REFERENCES privileges. In addition to these, there are a number of *special privileges*.

Synonym: access authorization

procedure

A procedure is used to store sequences of *SQL* statements in the database which can be executed later with a single call. A procedure is comparable to a subroutine which runs entirely in the *DBH*, in other words without exchanging data with the application program.Procedures are called using the *SQL* statement CALL. They have input and output parameters.

Synonym: Stored Procedure

procedure parameter

Call parameter of a *procedure*. A name, a data type, and the type of procedure parameter (IN, OUT or INOUT) must be specified for each procedure parameter.

procedure statement

A procedure statement is an *SQL* statement which may be used in a procedure. Every *procedure* contains precisely one procedure statement. SESAM/SQL recognizes the following procedure statements:

- *SQL* statements without *cursor*: SELECT, INSERT, UPDATE, DELETE, MERGE, CALL
- *SQL* statements with *cursor*: OPEN, FETCH, UPDATE, DELETE, CLOSE
- *SQL*statements for procedure control: COMPOUND, IF, LOOP, LEAVE, SET

projection

Selected set of *columns* from one or more *tables* that is included in a *derived table*.

pubset

Pubsets are sets of shared (public) disks which provide a file storage location in BS2000. One of the most attractive features of public volume sets is the fact that, as well as the files themselves, they also contain all the metadata required for file management purposes (file catalog, user catalog, etc.). As well as public volume sets, BS2000 also supports private volumes.

qualified name

A qualified name is used to identify an *SQL*object by adding a higher-level SQL object. A qualified name is followed by a dot in front of the lower-level SQL object. By adding the schema name, it is thus possible to distinguish *tables*with identical names in different *schemas*.

query expression

Element of an *SQL* statement that is used to define a new *table* (called a *derived table*) on the basis of *base tables* and *views*. A query expression can be a *SELECT expression*, a *join expression*, or a combination of SELECT expressions and join expressions linked by the keyword UNION.

record lock

A record lock is applied by a *transaction* to prevent other transactions from accessing this record (or row) (see *transaction concurrency*).

recovery

Generic term for all backup and recovery methods supported by *database systems*. On the one hand, a recovery process allows you to quickly restart the database system when inconsistencies are found. On the other hand, it protects against loss of data due to hardware errors or system failures.
There are three principal concepts:

- *transaction recovery*
- *restart* (external/internal) of the DB or DB/DC systems (also called system recovery)
- *media recovery*

REF value

Each *BLOB* is assigned a unique REF value which references the associated object. SESAM/SQL uses this REF value to access the object in question.

referential constraint

Integrity constraint defined by the *foreign key*.

In the case of single-column foreign keys, the referential constraint requires that each non- *null value* of the foreign key of a *table* (called the referencing table) which satisfies the *UNIQUE constraint* be contained in a particular column of another table (called the referenced table).

In the case of a multiple-column foreign key, the referential constraint ensures that each non-null value combination is contained in a particular column combination of the referencing table. This combination of columns must fulfil a UNIQUE constraint.

regular expression

Regular expressions are precisely defined search patterns. They are a powerful means of searching large data sets for complex *search conditions* . They have long been used, for example, in the Perl programming language. In SESAM/ SQL they can be used in the *predicate* LIKE_REGEX.

relation

See *table*

remote access

With *distributed processing* with SESAM/SQL-DCN: access of an application program to a database in another *configuration*.

reorganization

The purpose of reorganization is to physically cluster logically associated *blocks*. It takes into account *free space reservation*, however. Reorganization is also applied to *spaces* that are physically relocated when they have been assigned to a new *storage group* by the *database administrator*. Both reorganization and relocation are started using the REORG *utility statement*.

replication

Copy of a database which can only be used for recovery in DML operation. This is a defined version of an original database which can be updated and restored to a defined status by means of the DA-LOG file. Replications can also be used to repair or reset an original database or to create a new original database.

requesting user

In *timesharing mode*, a requesting user is an interactive or batch program. In *transaction mode*, it is a terminal (UTM terminal in the case of UTM) or a combination of a (UTM) terminal and a (UTM) user ID.

restart

Also called system recovery. *Recovery* method for restarting operation following an error.
Transactions which are open at the time of the failure are rolled back to the most recent valid *consistency point* by the *DBH* , i.e. all the participating databases are in a consistent state. Depending on whether the current *DBH session* is interrupted or not, the DBH executes an *external* or *internal* restart. Restart is only possible if the DBH is running under *transaction management* .

retrieval

Reading the data in a *database* .

routine

Generic term for *procedure* and *User Defined Function (UDF)* in SESAM/SQL. Procedures and UDFs have an almost identical scope of functions, but they differ in how they are called and in their return information.

row

Ordered sequence of values arranged horizontally in an *SQL table* .
Synonym: *Tupel*

Scaliger's Julian day number

Integer value which specifies the number of days which have passed since a certain start date. For historical reasons, the starting date is 24th November 4712 B.C. (in accordance with the Gregorian calendar). This date has the Julian day number 0. The dates permitted by SESAM/SQL, 0001-01-01 to 9999-1231, correspond to Julian day numbers from 1721426 to 5373484.

schema

Schemas are held in the *catalog space* of a *database* .
They are subdivided into two categories: user-defined schemas and information schemas.
User-defined schemas contain the metadata for the *base tables* and *views* in the database. Moreover, user-defined schemas contain information on *privileges* . A user-defined schema is assigned a name and an owner who can access it using his/her *authorization identifier* . User-defined schemas are created using the *SQL* statement CREATE SCHEMA and can be modified using various other statements (e.g. CREATE TABLE).
There must be two information schemas for each database: INFORMA TION_SCHEMA and SYS_INFO_SCHEMA. They enable the user to access the *metadata* contained in the user-defined schemas, such as *base tables* , *views* , *integrity constraints* , *privileges* , etc.
The INFORMATION_SCHEMA can be accessed by every user using SQL. The SYS_INFO_SCHEMA contains system-specific data and can only be accessed by the *universal user* .

search (CALL-DML)

Central *CALL DML* statement for the *retrieval* of data.

search condition

SQL language resource which returns a truth value of true, false or unknown. It comprises *predicates* which can be linked by the logical operators AND, OR and NOT. The *search condition*s uses to restrict the quantity of rows in a derived table (e.g. if the WHERE clause of a *query expression* is used).

If the search condition is used in a *check constraint*, it restricts the values permitted for the *column(s)* concerned.

search with join (CALL-DML)

Feature that joins two *logical files*. The values of the join attribute (see *join column*) in one logical file are compared with the values of the join attributes in the other logical file two at a time. If the values match, the corresponding records (or rows) are joined and considered in the returned result.The two logical files can belong to the same *base table* or two different base tables.

secondary index

See *index*

SELECT expression

Query expression starting with the keyword SELECT. Its syntax is restricted.

select list

The select list defines the columns to be included in the *derived table*. It contains the names of *columns* from one or more *tables* as well as any other type of *value expression*.

selection

Set of *Rows* retrieved from one or more *tables* . The rows must satisfy defined conditions.

server

See *client/server architecture*

service task

One of various possible BS2000 tasks to which SESAM/SQL relocates CPU- intensive activities. For example, service tasks can be used for database administration functions and for sorting intermediate results.

SESAM backup copy

Copy created with the SESAM statement COPY.

SESAM CLI (Call Level Interface)

Procedural interface for complex operations of SESAM/SQL which are either cumbersome or impossible to execute with *SQL* statements.

SESAM/SQL DBH

See *database handler*

SESAM/SQL-DCN

Add-on product for processing *distributed databases* with SESAM/SQL.

SESAM/SQL-specific statements

Type of statements in *SQL* syntax that is not part of the *SQL standard*, e.g. *utility statements*.

SESDCN control statement

SESDCN control statements include the *DCN options* and the statements for defining the *distribution rule*.

The *system administrator* enters SESDCN control statements when starting the *SESDCN distribution component*.

SESDCN distribution component

Main component of the *SESAM/SQL-DCN* product. The principal tasks of SESDCN are the generation and maintenance of the *distribution rule* as well as the receiving and passing on of *remote accesses* . Furthermore, SESDCN has administrative and monitoring functions.

session

See *DBH session, SQL session*

set function

Function used to calculate a value from a set of *column values* (AVG, MAX, MIN, SUM, COUNT) or *rows* (COUNT*).

single primary key

Primary key that consists of only one *column*.

single system image

The SESAM/SQL DBH task family presents itself to the *system administrator* as a unit. The individual tasks are not relevant for administration purposes (see *multitasking*).

space

Named memory area which is held in a BS2000 file and assigned to a *storage group.* There is one *catalog space* and one or more *user spaces* for each *database* . Spaces are the organizational units for *reorganization* and *recovery* .

space list

Recovery unit required for *media recovery*, which consists of a number of *user spaces*. For the purposes of *recovery*, it is possible for users to create a space list themselves from several user spaces with an identical time stamp. Spaces with a common time stamp are created by a shared copy statement. The time stamp indicates the time the copy was taken and must be identical for all *user spaces*in the space list.

space set

Recovery unit required for *media recovery*, which consists of a number of *user spaces*. A space set is created by a shared COPY statement involving several user spaces, and is identified by its time stamp. The time stamp indicates the time the copy was taken and must be identical for all *user spaces* in the space set.

special privileges

Special privileges are used to describe which definition and administration statements an *SQL* user can execute. SESAM/SQL distinguishes between the following special privileges: CREATE USER, CREATE SCHEMA, CREATE STOGROUP, UTILITY and USAGE ON STOGROUP.

SQL

The language most commonly used for processing relational databases. In contrast to the procedural languages of non-relational database systems, SQL is descriptive, i.e. the user describes, in set-oriented form, the result of a database operation rather than the steps required to get there. SQL offers extensive language resources for data definition, data manipulation, transaction management, access control, and so on. Embedded SQL (ESQL) makes it possible to access a database using *embedded SQL* statements from host language programs (e.g. in COBOL). SQL was first standardized by the International Organization for Standard- ization in 1987. SESAM/SQL is based on the up-to-date standard, referred to as *SQL standard* . SESAM/SQL supports the language core defined in SQL (referred to as Core SQL), optional language constructs of SQL, and supple- mentary functions which are not provided in SQL (*utility statements*).

SQL access plan

Evaluation rule which the DBH creates internally for every *SQL* statement. An *SQL* access plan comprises at least one subarea, usually more, referred to as *SQL scans*. The optimized format of a scan forms the *internal statement format*.

SQL database catalog list

Contains an entry for every *database* to be processed in the course of a *DBH session*. The *DBH* uses the SQL database catalog list to obtain information on which database is assigned to which BS2000 user ID.

The *system administrator*creates the SQL catalog list using a *DBH start statement*.

The *system administrator* can add entries to, or delete entries from, the SQL database catalog list using *administration statements*.

SQL default value

Value specified as the default value for a *column* in the DEFAULT clause of the CREATE TABLE statement. It is automatically assigned to this column. The SQL default value must not be confused with the *non-significant attribute value*.

SQL mode

Operating mode of an application program that starts with the execution of an *SQL* statement within a transaction and terminates when the transaction is terminated.

SQL return code

See *SQL status code*

SQL scan

Part of an *SQL access plan* referring to exactly one base table. The SESAM/SQL DBH generates an SQL access plan for each *SQL* statement.

Synonym: scan

SQL session

Sequence of *SQL* statements. A session starts when SESAM/SQL is initiated and ends when the connection is cleared.

SQL standard

Short name used in the SESAM/SQL manuals for the up-to-date international SQL standard.

SQL status code

The SQL status code (SQLSTATE) contains information on whether an *SQL* statement has executed with or without error.

Execution without error results in one of the following status codes: “successful completion”, “warning” or “no-data”.

If the execution terminates with error, the *SQL* status code contains information on the type of the error.

The SQL return codes (SQLCODEs) are no longer contained in the SQL standard and are only supported for reasons of compatibility to earlier SESAM/SQL versions. They have the same function as SQL status codes, but are not as detailed.

SQL table

See *table style*

SQLCODE

See *SQL status code*

SQLSTATE

See *SQL status code*

storage group

A named set of disks under the same *catalog ID*. All the disks must be of the same device type. Storage groups allow the user to control where the *spaces* (BS2000 files) are created. Every space is assigned to a storage group. The space is then created on the disks of this storage group.

storage structure language (SSL)

Statements used to administer the storage structure; SSL is used to create and process *storage groups*, *spaces* and *indexes*. Storage Structure Language is not a component part of the *SQL-Norm*.

suborders

When *SQL statements* are processed, suborders are identical with *SQL scans*, i.e. subareas of an *SQL access plan*. In the case of CALL DML, suborders are *logical files* for CALL DML requests.

subquery

Query expression enclosed in parentheses returning a simple column value, a row comprising a number of column values, a *column* or a *table*.

surrogate pairs

In *UTF-16*, a sequence of two code units which represent a Unicode character with a code point greater than U&'\FFFF'. NX'D800' through NX'DBFF': Leading Surrogate; NX'DC00' through NX'DFFF': Trailing Surrogate.

A leading and a trailing surrogate pair map the code points from U&'\+010000' through U&'\+10FFFF'.

symbolic attribute name (CALL DML)

Three-character name of an attribute, which identifies that attribute in *CALL DML* statements.

synonym

See *correlation name*

SYS_INFO_SCHEMA

See *schema*

system administration

Area of responsibility covering the start and termination of the *DBH* and administration of the current *DBH session*.

system administrator

Person or group of people responsible for *system administration*.

Synonym: SESAM/SQL system administrator

system data buffer

Area in main memory which the *DBH* reserves and manages for *system data*.

system entry

A pair, comprising the *system user identification* and the *authorization identifier* (for an SQL user). The system entry permits a user to access a SESAM/SQL database. A system entry is created using the *SQL* statement CREATE SYSTEM_USER. This does not apply to the *universal user*.

system user identification

Identifies a TIAM or openUTM user in BS2000.
A TIAM user is identified by the (symbolic) computer name and the BS2000 ID.
A UTM user is identified by the (symbolic) computer name, the name of the UTM application and the KDCSIGN or LSES name.
To work with a SESAM/SQL database, a TIAM or UTM user must be assigned an *authorization identifier* (see *system entry*).

table

A table is a two-dimensional arrangement of data elements comprising *rows* (horizontal) and *columns* (vertical).

A distinction is made between *base tables* , *views* , *derived tables* , and *abstract tables* .
Synonym: relation

table constraints

Integrity constraint, defined as a property of a *base table* with CREATE TABLE or ALTER TABLE. It can be a *UNIQUE constraint*, a *referential constraint* or a *check constraint*.

table space

User space in which at least one *base table* is stored. The table space of a specific *base table* is the *space* where that *base table* is stored.

table style

This specifies whether a *base tablecan be processed via the SQL interface and/or the CALL DML interface*. The table style is of particular importance in the context of *migration*. The following different table styles are distinguished:

- tables which can be processed with *CALL DML* only (CALL DML only tables)
- tables which can be processed with CALL DML and (with certain restric- tions) with *SQL* (CALL DML/SQL tables)
- tables which can be processed with *SQL* only (SQL tables).
- tables which are used exclusively for storing *BLOB*s (*BLOB tables*)

CALL DML only tables and CALL DML/SQL tables are subsumed in the CALL DML table style. A *non-significant attribute value* must be declared for every *attribute* in a CALL DML table.

TA-LOG buffer

Area in main memory in which the *DBH* buffers the backup information for transaction management. If the buffer overflows or a consistency point is inserted, the DBH transfers the contents of the TA-LOG buffer to the *TA-LOG files*.

TA-LOG files

Backup files containing the information for *transaction management* , *transaction recovery* and for internal and external *restarts* . The contents of the TA-LOG files include *before-images* and *after-images* .

Synonym: transaction log files

thread

See *multithreading*

timesharing mode

In timesharing mode, a number of users work independently in *batch* or *interactive mode*, each with a separate application program.

transaction

Sequence of related statements which move a *database* from one consistent state to another consistent state. A transaction is performed either in its entirety or not at all.

Special statements are available for transaction processing:

CALL DML has one statement each for the following operations: opening, closing (committing) and rolling back a CALL DML transaction.

SQL has special statements for ending and rolling back a transaction only. In openUTM applications, the corresponding UTM calls must be used for this purpose. There is no special *SQL* statement for defining the start of a transaction: the first transaction-initiating statement after the previous transaction has been committed or rolled back or after the program has been started is taken by SESAM/SQL to be the start of the transaction.

transaction concurrency

Simultaneous access by different *transactions* to the same data resources.The locking concept in SESAM /SQL means that the *consistency* of the data processed is ensured, even during concurrent access. Normally, every trans- action locks the rows it is accessing against access by other transactions. The lock is only lifted when the transaction has been completed. It is, however, possible for the user to change the locking behavior to suit requirements by means of the isolation level and thus to increase the level of transaction concurrency and consequently the throughput. In SQL applications the *isolation level* is used for this purpose.

transaction log file

See *TA-LOG file*

transaction management

Transaction management provides mechanisms for handling error situations automatically and, in the event of an error, uses rollback mechanisms to restore the *consistency* of the data resources. This means that transaction management forms the basis for the *recovery* mechanisms *transaction recovery* and *restart* (internal/external).

transaction mode

In transaction mode, several users use the same application program simulta- neously to access one or more databases. The openUTM transaction monitor or the DCAM data communication system is responsible for coordinating the application users.

transaction processing

This ensures that a database can only be accessed within a *transaction*. This means that the consistency of a *database* is ensured at all times.

transaction recovery

Recovery mechanism for resetting an individual *transaction*after a user error.

transaction status

See *access mode of transaction*

transaction-initiating SQL statement

Transaction-initiating *SQL* statements include all statements other than DECLARE CURSOR, SET TRANSACTION, SET CATALOG, SET SCHEMA, SET SESSION AUTHORIZATION, PERMIT, WHENEVER, ALTER TABLE with pragma UTILITY MODE=ON and the *utility statements*.

transcoding

Conversion of a national string from the UTFE character set to the UTF-16 character set. In *SQL* a transcoding is performed explicitly with the SQL function TRANSLATE(). See also: *transliteration*.

transfer container

Area of main memory which the DBH reserves and manages for *SQL scans* and for the query and response areas of *logical files*.

transliteration

Conversion of an alphanumeric string into a national string and vice versa. In *SQL* a transliteration is performed explicitly with the SQL function TRANSLATE(). If required, implicit transliterations are executed with the *SQL* statement ALTER TABLE ALTER COLUMN and with the *utility statements* LOAD and UNLOAD. See also *transcoding* .

tuning

Optimization of the performance (response behavior, use of resources, throughput) of the *database system* .

tuple

See *row*

two-phase commit

Terminates a *transaction* in two phases during *distributed processing*:

The “provisional end of the transaction” request (PREPARE TO COMMIT) first initiates the end of the transaction. All the components involved in the transaction now try to achieve a state in which the transaction can be both rolled back and committed. If one or more of the *DBH*s involved in the distributed processing cannot achieve this state, the entire transaction is rolled back. Otherwise, a second phase issues the final request “end of transaction”.

uncorrelated function call

Function call of a *UDF* with constant input values. Constant input values do not refer to the SQL statement which contains the function call.

union join

Join operation whose *derived table* contains rows from the table on the left of the UNION operator and from the table on the right of the UNION operator, each supplemented by the columns in the other table which are set to *NULL values*.

UNIQUE constraint

Integrity constraint for a *column* or combination of columns of a table. Integrity constraint applied to a column or column combination to ensure that no two rows of a table have the same value or value combination for the specified column (combination). If the UNIQUE constraint has been defined, SESAM/SQL prevents this from happening.

UNIQUE index

See *index*

Universal Character Set 2 (UCS-2)

In *UTF-16* all Unicode characters from NX'0000' through NX'FFFF' are encoded in 2 bytes. The characters in this range are also referred to as Universal Character Set 2 (UCS-2).

universal user

SQL user with comprehensive *privileges* defined for every *database* when the database is generated with the *utility statement* CREATE CATALOG. The universal user is described by the first *system entry*.

The *SQL* statement CREATE SYSTEM USER allows the universal user to create additional system entries with the privileges of the universal user. This is done by assigning the universal user system entry to other *system user identifications*.

updatable

A *View* or *cursor* is called “updatable” if it can be used to update the underlying *base table(s)*. The corresponding *privileges* for the base table(s) are required regardless of this capability.

user data

Data in the *user spaces* of the *database*. The user data is distinguished from the *metadata* in the database's *catalog space*.

user data buffer

Area in main memory which the *DBH* reserves and manages for *user data*.

User Defined Function (UDF)

A UDF is used to store sequences of *SQL* statements in the database which can be executed later with a single call. A UDF is comparable to a subroutine which runs entirely in the *DBH*, in other words without exchanging data with the application program.

UDFs can be used in almost all expressions by means of their function call. They have input parameters and supply a return value.

user space

Space containing the user's *tables* and *indexes* as opposed to the *catalog space*.

user view

see *View*

UTF-16

Unicode encoding defined by the Unicode Consortium. In the case of UTF-16, all Unicode characters from NX'0000' through NX'FFFF' are encoded in 2 bytes.
All characters above NX'FFFF' are represented by 4 bytes, so-called surrogate pairs. SESAM/SQL stores values internally in UTF-16 format in columns of the data type N[VAR]CHAR and also uses UTF16 for national values in host variables.

UTF-8

Unicode encoding defined by the Unicode Consortium. UTF-8 uses a variable number of bytes (one to four bytes) for encoding the Unicode characters. The byte represenatation of the ASCII characters remains unchanged. In the case of a character which is encoded in several bytes, none of the individual bytes represents a valid character.

UTF-EBCDIC (UTFE)

Unicode encoding for machines which use the EBCDIC character set.Here the Unicode characters from NX'0000' through NX'009F' are represented by the corresponding EBCDIC characters, and all other Unicode characters by strings of 2 to 5 bytes.

In SESAM/SQL, Unicode with UTF-EBCDIC coding can be used in the case of the UNLOAD and LOAD *utility functions* with DELIMITER_FORMAT.

utility mode

Mode which the user can set when adding, changing, or deleting one or more *columns* of a *base table* using the *pragma* UTILITY MODE. When the utility mode is set, the associated *SQL* statement cannot initiate a transaction and *transaction logging*is switched off.

utility monitor

Database administration tool with the following functions:

- execution of *SQL* and *utility statements*
- evaluation of *INFORMATION_SCHEMA* and *SYS_INFO_SCHEMA* for the catalog space
- reading and updating of the *metadata* for the *catalog recovery file* and for the *spaces*

The utility monitor also allows you to call the administration program SESADM as a subroutine.

You can choose to use the utility monitor in interactive mode using a menu- driven interface or to have it read its input from a statement file.

utility statement

Utility statements are statements which use *SQL* syntax and which make available utility functions for *database administration*. The utility statements include statements for generating, loading, unloading, copying, reorganizing and migrating databases.

The utility statements cannot start transactions and do not form part of the *SQL standard*.

value expression

The result of a value expression can be a numeric, character-string, or date/ time value. It stands for a *column* of a table, a *constant* , a *set function* , a *host variable* , or a *subquery* . Any of these elements combined by operators can also be a value expression.

vector

Data object comprising a certain number of values of the same *data type*.

vector variable

Host variable for a *vector* used for storing the different occurrences of a *multiple column* .

view

Named virtual table which is derived from one or more *tables* . A view is defined in a *query expression* using the *SQL* statement CREATE VIEW. The *derived table* produced by the query expression is generated anew every time the view is referenced in an SQL statement. The derived table thus always contains the most current data from the database.

Synonym: user view

WA-LOG file

Information on control and backup of the DBH session and on the DBH restart is stored in the WA-LOG file. Each SESAM/SQL DBH has its own WA-LOG file. Among other things, the WA-LOG file contains the following:

- the DBH options which the DBH uses for the restart
- the physical before images
- information that describes the progress of the restart process
- information on synchronization with openUTM.

work container

Memory area maintained by the *DBH* for buffering *internal statement formats*.

Related publications

You will find the manuals on the internet at *http://bs2manuals.ts.fujitsu.com*. You can order printed versions of manuals which are displayed with the order number.

SESAM/SQL-Server (BS2000)
SQL Reference Manual Part 1: SQL Statements

User Guide

SESAM/SQL-Server (BS2000)
SQL Reference Manual Part 2: Utilities

User Guide

SESAM/SQL-Server (BS2000)
CALL-DM Applications

User Guide

SESAM/SQL-Server (BS2000)
Database Operation

User Guide

SESAM/SQL-Server (BS2000)
Utility Monitor

User Guide

SESAM/SQL-Server (BS2000)
Messages

User Guide

SESAM/SQL-Server (BS2000)
Performance
User Guide

WebTA access for SESAM/SQL

(Product document, also available on the manual server)

ESQL-COBOL (BS2000)
ESQL-COBOL for SESAM/SQL-Server

User Guide

SESAM-DBAccess

Server-Installation, Administration (available on the manual server only)

SESAM-KLDS (BS2000)

User Guide

DRIVE/WINDOWS (BS2000)
Programming System

User Guide

DRIVE/WINDOWS (BS2000)
Programming Language

Reference Guide

DRIVE/WINDOWS (BS2000)
Directory of DRIVE SQL Statements for SESAM/SQL

Reference Manual

DRIVE/WINDOWS (UNIX systems)
System Directory of DRIVE Statements

Reference Manual

DRIVE (BS2000)
Supplement

User Guide

openUTM Concepts and Functions

User Guide

openUTM (BS2000, UNIX Systems, Windows NT)
Programming Applications with KDCS for COBOL, C and C++

Core manual

openUTM (BS2000)
Using openUTM Applications under BS2000

User Guide

openUTM (BS2000)
Generating and Handling Applications

User Guide

ARCHIVE (BS2000)

User Guide

EDT (BS2000)
Statements

User Guide

FHS (BS2000)

User Guide

HSMS (BS2000)
Hierarchical Storage Management System

User Guide

SECOS (BS2000)
Security Control System - Access Control

User Guide

SECOS (BS2000)
Security Control System - Audit

User Guide

SDF (BS2000)
SDF Dialog Interface

User Guide

SHC-OSD (BS2000)
Storage Management for BS2000

User Guide

SNMP Management
SNMP Management (NET-SNMP) for BS2000

User Guide

SPACEOPT (BS2000)
Disk Optimization and Reorganization

User Guide

Unicode in BS2000

Core Manual

XHCS (BS2000)
8-Bit Code and Unicode Processing in BS2000

User Guide