

ADABAS D

LOAD



Manual Order Number: ESD611038WOU

This document is applicable to ADABAS D Version 6.1.1 PE and to all subsequent releases, unless otherwise indicated in new editions or technical newsletters.

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

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the address on the back cover.

© April 14, 2025, SOFTWARE AG, Germany & SOFTWARE AG of North America, Inc.

All rights reserved

Printed in the Federal Republic of Germany

The SOFTWARE AG documentation often refers to numerous hardware and software products by their trade names. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies.

1 Why Do You Need LOAD?

When developing new database applications or modifying existing ones, you usually need to load already existing data into the database.

The transfer of data from files may also be a recurring task if, for example, datasets are periodically transferred from one data processing system into another one.

Sometimes, it is only necessary to update columns of a database table. When expanding an application, new columns are defined that must be loaded separately.

A company's applications are often organized in subsystems that communicate with each other using file interfaces. Thus it must be possible to extract any desired datasets from the database and provide them in a file.

The normal administrative tasks required for a database application, such as defining tables, granting privileges, creating secondary indexes, are usually not ad hoc activities, but are automated using command files.

LOAD is a tool that simplifies these tasks considerably for system developers and system administrators.

To load external data, DATALOAD and FASTLOAD are the appropriate functions because they can read any possible data format (ASCII or EBCDIC plaintext, binary, fixed or variable field length).

The function DATAEXTRACT can be used to generate the same variety of data formats.

When migrating an ADABAS database from one platform to the other, the functions CATALOGEXTRACT/LOAD, TABLEEXTRACT/LOAD, and DBEXTRACT/LOAD are provided.

ORACLE data is transferred into ADABAS D using the function LOAD ORACLEDB. This may include migration from one hardware platform to another.

The syntax for LOAD is provided in Appendix 1.

2LOAD in Interactive Mode

LOAD activities are usually executed from command files and run in *background*; that is to say, without occupying a terminal during execution.

The background execution of command files is described in more detail in chapter 19. By embedding the necessary LOAD calls into procedures of the operating system (UNIX shell scripts), you can automate *DBA activities*.

LOAD functions may also be executed *interactively* on the screen. Even command file execution can be controlled and supervised from the screen. This simplifies the *creation and testing* of command files.

To start an interactive LOAD session, call

XLOAD

This call may vary depending on the particular operating system. For more information, see the platform-specific User Manual.

2.1 The Input Screen

After a successful connect, LOAD returns the following input screen:

```

LOAD ...  Input                Load/Update/Extract                nnn-mmm
-----  A D A B A S  -----
          Input Area for
          -  DATALOAD statements
          -  DATAUPDATE statements
          -  DATAEXTRACT statements
          -  other LOAD statements
          -  SQL statements

-----  <serverdb> : <user>  -----
system messages, key settings, LOAD commands

```

In the header line, LOAD displays the current command mode (Input) and the range of lines currently displayed on the input form (nnn-mmm). The version number of the executed program is also displayed (LOAD 6.1).

In the bottom boundary line, the SERVERDB name and the user name are displayed.

A section of the input form is displayed between the two horizontal boundary lines. This form is of variable length and 141 columns wide. Using the scroll keys **PageUp** and **PageDown** or **F7** and **F8** respectively, any section of the form can be displayed onto the screen.

Entering and modifying long statements are facilitated by user-friendly editor functions (see the platform-specific User Manual). If the left or right margin of the

input area is marked with '=== ' in each line, the prefix editor (according to XEDIT) is active; otherwise, the RAND-oriented key editor (UNIX) is.

2.2 Interactive Testing

LOAD supports the transfer of prepared statements from external files: the LOAD command NEXT calls the following statement of an open command file into the input area. The editor command GET copies the contents of any file into the input form.

Often the opposite applies: LOAD is invoked by a user who, using interactive testing and supported by the HELP function, wants to become acquainted with the possibilities offered by this component before constructing command files.

The statements in command files are separated from each other by comment lines so that the individual statements can be transferred to the screen one after the other. In interactive mode, however, execution will only be successful if a single statement is entered in the edit form.

Example of how to enter a DATALOAD statement:

```

LOAD ...   Input                Load/Update/Extract                001-018
-----
DATALOAD TABLE item          A D A B A S
                                IF POS 01-02 = 'is'
    itno      03-10  CHAR
    descr     11-41
    stock     42-45  INTEGER
    min_stock 46-47  INTEGER
    price     48-55  DECIMAL(2)
    weight    56-59  REAL
INFILE item.data
-----
                                <serverdb> : <user>
    1=Help 2=Reset 3=End 4=Print 5=Run 6=Next 7=Pick 8=Put 12=Mark
==>

```

The statements can be entered in *free format*; i.e., with any number of blanks, line feeds, and empty lines. Keywords, names, and comparison values, however, must not contain embedded blanks or line feeds.

Interactive testing is simplified by simultaneously constructing a LOAD command file. This is done with the editor command PUT. Add a comment line to the end of

a statement executed successfully and then store the contents of the edit form in a file using PUT (together with the option APPEND).

2.3LOAD Commands and Function Keys

Execution of LOAD functions is controlled by a series of LOAD commands, which are fully described in chapter 17.

The line at the bottom of the screen is used to enter commands and begins with '==>'.

LOAD commands consist of a command word and a series of - partly optional - parameters. Only the first four characters of a command word are significant. Lowercase characters in the command are converted into uppercase ones, apart from file names and other parameters enclosed in single quotation marks:

```
==> run item.inst k z      is converted into      RUN item.inst K Z
```

A series of LOAD commands can also be called by a *key*. The current key setting is displayed below the system line. Example:

```
1=Help 2=Reset 3=End 4=Print 5=Run 6=Next 7=Up 8=Down ...
```

This display depends on the chosen language. The command words themselves are always English.

As far as the labeled keys are concerned, the following may be of some importance to the LOAD user: the Help key for calling the HELP function and the PageUp and PageDown keys for scrolling the screen. The keys for the editor functions (insert, delete, copy line etc.) are described in the platform-specific User Manuals.

If LOAD is running at a terminal that does not have soft keys, both the soft keys and the Help, scroll, and editor keys are simulated by the keys F1,

, ..., . LOAD then displays the current settings of these keys below the input area.

Overview of the most important LOAD commands:

Command	Function
RUN	starts the statement in the input area
RUN 'fn'	starts the command file fn
PROT	displays the load log file on the screen
NEXT	calls in the next statement of the command file
SKIP	skips statements of the command file
SCAN	scans a command file
PRINT	prints out the contents of the input form
SET	sets user-specific parameters
HELP	gives information about LOAD
EXIT	terminates the current LOAD session

2.4 The HELP Function

The HELP function displays short descriptions of all available LOAD statements and commands on the screen. Information about editor functions and SQL statements can be requested.

The HELP function can be invoked in two ways:

- a) using the **Help** key or **F1**
- b) with the command **HELP**, optionally followed by a command name

In the case of call a), LOAD displays the HELP menu. It contains sections about the

- different LOAD commands,
- DATALOAD, DATAUPDATE, DATAEXTRACT statements,
- CATALOGLOAD and CATALOGEXTRACT statements,
- TABLEEXTRACT and TABLELOAD statements,
- DBEXTRACT and DBLOAD statements,
- the ORACLE crossloader,
- SQL statements,
- editor functions.

A particular chapter can be selected by placing the cursor on a highlighted word and pressing the **Help** key.

The description of a certain LOAD command (RUN, NEXT etc.) can be requested directly with b):

==> **HELP RUN**

If the HELP function is invoked after a syntax error message has been displayed, it implicitly branches to the corresponding HELP section. After returning, the cursor is still positioned on the error and the error message is displayed again.

The editor functions (e.g., PUT, GET) are described by the HELP function of the editor, selected from the HELP menu or invoked by entering a '?' in the command line.

2.5LOAD Restrictions

1.	DATALOAD/DATAUPDATE statements per load run	maximum	20
2.	DATAEXTRACT statements per extract run	exactly	1
3.	Tables per FASTLOAD run	exactly	1
4.	Columns per load or extract run	maximum	254
5.	IF and NULL conditions per load run	maximum	254
6.	NULL conditions in extract statements	maximum	254
7.	Length of a comparison value	maximum	141
8.	Length of an input value	maximum	254
9.	Length of a filename (in characters)	maximum	64
10.	Length of table names (without prefix)	maximum	18
11.	Length of table prefixes	maximum	18
12.	Length of column names	maximum	18
13.	Significant characters in command words like RUN, NEXT, PROMPT	maximum	4

3 Loading Database Tables with DATALOAD

3.1 The DATALOAD Statement

The load statement consists of (at least) two specifications, the first beginning with DATALOAD, the second with INFILE.

Example:

```
DATALOAD TABLE customer
  cno      1-4
  name     6-15
  city     17-36
  state    37-38
  zip      39-43
INFILE    cmaster.data
```

The DATALOAD specification describes the target table and the assignment of the input fields to the table columns.

The table name is specified in accordance with the SQL conventions. It can also contain a user name as a prefix (e.g., SALES.CUSTOMER).

Input fields can be assigned to table columns in any order. The order does not affect the speed of processing.

Optionally, the keywords KEY and SET can be placed before the column names. They are treated as comments in the DATALOAD statement but facilitate a later change into a DATAUPDATE statement.

The input fields are described by their starting and ending positions in the input record (beginning with 1). The specification of the end position is optional: if it is omitted, the field length defaults to 1.

If no input field has been assigned to a column in the target table, the entire column is set to default during loading. Without an explicit declaration, this is the NULL

value. Key and mandatory columns *must* be specified in the DATALOAD statement.

The name of the source file from which the records are to be loaded is entered after INFILE. Names of external files remain unchanged when being passed to the operating system. If the data is not placed in an extra file but is specified together with the statement, INFILE * is to be indicated. External data may also be read from tape. Further particulars can be found in the platform-specific User Manuals.

3.2 Data Types of File Fields

The external data format of the corresponding field value in the source file can be specified for each input field.

Example:

```
DATALOAD TABLE item
  itno      01-08  CHAR
  descr     09-39  CHAR
  stock     40-43  INTEGER
  min_stock 44-45  INTEGER
  orderdate 46-49  INTEGER
  delivdate 50-57  CHAR
  price     58-65  DECIMAL(2)
  weight    66-69  REAL
INFILE ...
```

The structure of the external data formats is described in chapter 12.

The data format specification is only needed when the field values are not stored in plaintext (CHAR).

LOAD can edit input data in any numerical data format (INTEGER, DECIMAL, ZONED, and REAL) generated by an application program for any numerical column type of the table (FIXED, FLOAT, SMALLINT, or INTEGER).

Plaintext values can be used as input for any column type. If the values are to be entered into CHAR or DBYTE columns, the input length must not exceed the column length. Shorter input values are allowed.

If the target column requires a numerical value, it must be possible to interpret the plaintext input value as a number. Fractional digits are implicitly truncated when the precision of the column is less than that of the input value. Integral digits, on the other hand, must completely fit into the target column.

To be able to edit the input file in every environment, each of the mentioned data formats can also be supplied in hexadecimal representation, where two hexadecimal digits represent one byte. The additional specification HEX must be added to the data type in this case, namely

```
[CHAR] HEX           or
INTEGER HEX          or
REAL HEX             or
DECIMAL [fraction] HEX or
ZONED [fraction] HEX.
```

A valid hex constant only consists of the characters '1' to '9' and 'A' to 'F' or 'a' to 'f'. In case of HEX data types, the length of an input field must be even.

Example:

```
DATALOAD TABLE item
  itno      01-08  CHAR
  descr     09-19  CHAR
  stock     20-27  INTEGER HEX
  min_stock 29-32  INTEGER HEX
  orderdate 34-41  INTEGER HEX
  delivdate 43-50  CHAR
  price     52-59  DECIMAL(2) HEX
  weight    61-68  REAL HEX NULL IF POS 61-62 = '00'
INFILE *
01785523hammer      00002710 03E8 00001388 19911130 0001195C 00
...
```

The hexadecimal representation does not solve the problem of differing representations of binary data formats, such as INTEGER and REAL, on different computers. This example requires a 2-byte integer to have the format high-byte, low-byte. The data record then contains the values 10,000 (for STOCK), 1,000 (for MIN_STOCK), 5,000 (for ORDERDATE), and 11.95 (for PRICE).

3.3 Dimension and Precision of Numerical Values

The functions SCALE, ROUND, and TRUNC edit numerical values before they are inserted into the database.

The SCALE option can be used to load fixed point numbers that have no explicit decimal point but implicitly are assumed to have one at a certain position.

The SCALE option is also used to load numerical values that are stored in the file in another decimal scale dimension than in the database. Example: kilogram values, possibly with a decimal point, are to be loaded as gram values.

The scaling factor, which is specified after the keyword SCALE, can be positive or negative. The value to which the function refers is multiplied by the corresponding decimal power.

The functions ROUND and TRUNC determine the *fractional digits* of a number. The number n of fractional digits must lie between 0 and 18. If the number has no fractional digits, the functions have no effect. Floating point numbers are internally converted into fixed point representation.

TRUNC n means that the n+1st and all the following fractional digits of the number are 0, while the first n fractional digits remain unchanged.

ROUND n means that the number is rounded starting with the n+1st fractional digit. If this digit is greater than or equal to 5, the nth digit will be incremented by 1. Also in this case, the result is a number with the n+1st and all the following fractional digits equal to 0; the first digits of the number, however, may be modified by rounding up.

Both ROUND and TRUNC can be applied in combination with the SCALE function. The functions must be specified in the following order: SCALE before ROUND or TRUNC. The order of processing corresponds to this order.

Example:

```
DATALOAD TABLE distance
...
cm  20-25  SCALE 2
km  20-25  SCALE -3 ROUND 0
...
INFILE meter.input
```

These three functions are allowed in the field descriptions of DATALOAD, FASTLOAD, and DATAUPDATE statements.

3.4 Format Specifications Related to a File and Other File Options

The Format Specifications DEC, DATE, TIME, and TIMESTAMP

The plaintext format of floating or fixed point numbers and of date and time specifications can be determined specifically for a file. Thus independence is obtained from the current SET default definition.

A format specification consists of one of the keywords DEC (decimal representation), DATE (date representation), TIME (time representation), or TIMESTAMP (time representation) followed by a mask enclosed in single quotation marks. The same syntax rules apply to this mask that are valid for the corresponding SET parameter.

Example:

```

DATALOAD TABLE time_entry
  day      1-8  DEFAULT NULL
  from     10-14 DEFAULT NULL
  to       16-20 DEFAULT NULL
  break    22-24 DEFAULT NULL
INFILE *    EBCDIC
  DATE 'dd.mm.yy'
  TIME 'hh:mm'
  DEC  '//.//'
  NULL ' - '
01.04.95 08:05 17:00 0.5
02.04.95 07:55 16:00 0.5
...

```

Numbers in default format (no thousand sign, decimal point) can always be loaded. If the used decimal representation is, e.g., `//.`, then it is also possible to load numbers such as 99 999.99 or 1,2345E+04.

Floating point numbers must not contain blanks between mantissa and exponent. Unsigned or one-digit exponents are allowed.

Date and time values are converted into the 8-digit SQL default representation. In case of DATE, a reasonableness check is made for day and month (Is there a February 29 in this year?). A TIME value has four digits for the hour; these must not coincide with a clock time (0.00 to 24.00 hours). Minutes and seconds must lie between 0 and 59. Timestamp values consist of a date and a time field. The time field has two digits for the hour, two digits for the minutes, two digits for the seconds, and six digits for the micro seconds. This means, it has a length of 20 digits in its default representation.

The Format Specifications ASCII and EBCDIC

If the input file was generated on another computer and the code is to be converted from ASCII into EBCDIC or vice versa, then a corresponding file option can be specified.

ASCII indicates that the current file is ASCII coded and is to be converted into EBCDIC when the current computer is using this code. The same applies when EBCDIC is specified. When file code and computer code coincide, the option is ignored.

The code is converted field by field when the input values are processed. A file containing CHAR and DECIMAL fields which was generated by a COBOL program on an EBCDIC computer, for example, can be loaded into an ASCII computer in *one single* run with CHAR fields (for example date specifications) being converted into ASCII, but DECIMAL fields remaining unchanged.

The Format Specification for the Representation of INTEGERS

If the input file was generated on another computer and a conversion from one INTEGER representation into another one has to take place, then a corresponding file option can be specified.

INTEGER HILO indicates that the current file contains INTEGER values in the following representation: HIGH byte on the left, LOW byte on the right, just as a number in plaintext has the tens on the left and the units to the right of them. Of course, the same is valid for numbers of greater lengths. INTEGER LOHI indicates that the INTEGER values were generated on a byte-swap machine and that therefore the byte with the lowest significance is on the left. If this representation does not coincide with that of the current computer, the order of the bytes will be reversed.

Representation of NULL Values in the File

If the load statement contains DEFAULT NULL conditions, the file option NULL can be used. This option determines which string is to represent the NULL value.

The NULL string must be specified after the keyword NULL and must be enclosed in single quotation marks. Its maximum length is 20 characters. When comparing the NULL string with the file fields of different lengths, the shorter value is padded with blanks.

If the NULL option is omitted, the corresponding value of the SET parameters is taken.

Representation of BOOLEAN Values in the File

The BOOLEAN representation is specified in single quotation marks after the keyword BOOLEAN. Within the quotation marks, the TRUE representation is separated from the FALSE representation by a slash; TRUE is on the left of the slash, FALSE on the right of it. Both values can have up to ten characters.

If the BOOLEAN option is not specified, the corresponding value of the SET parameters is taken.

COMPRESS for Non-tabular Data

An input file without tabular structure in which every line contains its data elements one after the other separated by a distinguishing character (e.g., a semicolon) must be loaded with the COMPRESS option. This option runs:

```
<compress option> ::= [ COMPRESS ]  
                      [ SEPARATOR '<character>' ]  
                      [ DELIMITER '['<character>']' ]
```


In COMPRESS mode, the position specifications after the column names denote the relative position of the values in the file. To find a value designated by <n>, <n> - 1 separators defined by the SEPARATOR specification are skipped when reading. Separators may also follow immediately one after the other (input length 0). If the relative position is greater than the number of values in the line, the value, too, has the input length 0. By definition, values of length 0 satisfy the NULL condition of the column.

The DELIMITER specification serves to delimit character strings, for example, when these contain the separator symbol. Exactly those characters will be inserted which are placed between the two delimiters. If the first non-empty character of a value is not the delimiter, then all characters up to the next separator will be inserted.

Specifying COMPRESS is equivalent to the specification of SEPARATOR ',' and DELIMITER ''.

CONCATENATE and CONTINUEIF for Multiple-line Data

The options CONTINUEIF and CONCATENATE can be used to combine several lines of a file to form one logical input line. The options run:

```
CONTINUEIF {<pos> | LAST} {= | <>} [x] '<character>'
```

and

```
CONCATENATE <number of lines>
```

CONCATENATE <number> always combines the predefined number of lines, while for CONTINUEIF, the concatenation depends on whether a certain character is located on a certain position of the line just read. The position can be fixed (number) or variable (LAST). Equal (=) and not equal (<>) are allowed as

comparison operators. The character must be enclosed in single quotation marks. 'x' precedes a hexadecimal constant. If the CONTINUEIF condition is met because the continuation character was found, this character will not be inserted into the logical input line.

Example:

```
DATALOAD TABLE time_entry
    day      1
    from     2
    to       3
    break    4
INFILE *
    COMPRESS SEPARATOR ','
    CONTINUEIF LAST = '&'
01.04.95,08:05,17:00&
,0.5
02.04.95,07:55,16:00,0.5
...
```

Multi-volume Processing with COUNT Option

The option COUNT must be used first for data extraction. It is therefore explained in detail in the corresponding section of this manual. Magnetic tapes written with the COUNT option always begin with a control line "VOLUME nnnnnnnnnn". This control line can either be skipped by a corresponding table condition or used to check the order of the tapes. In the second case, the number specified for data extraction must also be specified after COUNT.

3.5 Selecting Records from the Source File

Records of a source file can be loaded into a table according to their contents:

Example 1:

```

DATALOAD TABLE reservation IF POS 13-22 >= '29.04.1995'
  rno          1-4
  cno          5-8
  hno          9-12
  arrival      13-22
  departure    23-32
  price        33-40
INFILE  rmaster.data
DATE  'dd.mm.yyyy'

```

Example 2:

```

DATALOAD TABLE hotel
  IF (POS 48-52 REAL < '1 000.00')
  AND
  (POS 48-52 REAL >= '100.00')
  hno          01-04  INTEGER
  name         09-18
  city         20-39
  state        40-41
  zip          42-46  DECIMAL
  price        48-52  REAL
INFILE  hotel.data

```

The *selection criterion* begins with the keyword IF. One or more conditions follow. These can be negated (NOT), linked by AND and OR, and bracketed in any form. If no parentheses are used, the operators have the following precedence: NOT is stronger linking than AND and OR, AND is stronger linking than OR. Operators with the same precedence are evaluated from left to right.

Only those records that match the simple or composed condition are loaded.

Every simple condition begins with the keyword POS. This is followed by the position and, if necessary, by the data type of a value in the input record, the comparison operator, and a constant.

A comparison value in the input record is described, like the actual input fields, by its position and an optional format specification. The format of the file value must only be specified when it is not the default CHAR.

The constant must be enclosed in single quotation marks and be specified as a plaintext value. It will be converted in such a manner that it will have the same type as the file value.

If the defined condition compares two numerical values with each other, the constant must have a valid number format; that is, it must either be a floating point number in mantissa/exponent representation or a fixed point number in the currently determined or default decimal representation.

If the defined condition compares two character strings with each other, then constant and input field must have the same length.

3.6 Inserting NULL Values

A NULL condition can be specified for any target column that may contain NULL values. It describes the condition for loading the NULL value as a column value.

Example:

```
DATALOAD TABLE item
  itno      01-08
  descr     09-39      NULL IF POS 09-11 = '    '
  stock    40-43 INTEGER NULL IF POS 40-43 INTEGER < '0'
  min_stock 44-45 INTEGER
  price     46-53 DECIMAL(2) NULL IF POS 1 <> 'X'
                                OR POS 46-53 DECIMAL < '0'
  weight    54-57 REAL
INFILE ...
```

The NULL condition begins with NULL IF. The syntax is similar to the selection criterion for input records described in the preceding section.

The system tests whether the condition is true for each input record; if it is true, the NULL value is inserted, if it is not true, the value of the assigned record field is inserted.

Columns of the target table which are *not* specified in the DATALOAD statement are set to the default value in all inserted rows.

Note that columns cannot be set to the NULL value when they are defined in the database catalog with the KEY or NOT NULL option. LOAD rejects the definition of a NULL condition for such columns.

Columns with a default value definition other than NULL cannot contain a NULL value either, but in this case LOAD inserts the explicit default value instead of the NULL value; i.e., it evaluates the statement as "DEFAULT IF POS".

The DEFAULT NULL condition can be utilized when the representation of the NULL values is the same for all columns of the file. In this case, the character string, which is otherwise specified after every NULL IF POS, is defined only once as file option NULL '...!'.
Any number of DEFAULT NULL conditions is allowed in the LOAD statement. During evaluation, the shorter value is padded with blanks.

Example:

```
DATALOAD TABLE address
      identification 1
      telephone      5 - 15 DEFAULT NULL
      city            19 - 58 DEFAULT NULL
INFILE customer.data
NULL '-              ';
```

The LOAD statements generated by DATAEXTRACT FOR DATALOAD look like the statements shown in this example.

3.7 Loading Any Constants

A DATALOAD statement usually describes the fields of the input file and the columns of the table which are to be used for loading. In some cases, however, the table contains an additional attribute (e.g., origin of the data) which is not included in the input file because it is given the same value in each row generated from the file.

For this case, a constant instead of a position specification may be inserted into the DATALOAD statement:

Example:

```
DATALOAD TABLE item
  itno      01-08
  descr     09-39          NULL IF POS 09-11 = '    '
  stock     40-43 INTEGER NULL IF POS 40-43 INTEGER < '0'
  min_stock '50'
INFILE ...
```

The constant must always be enclosed in single quotation marks to distinguish it from position specifications.

During loading, the constant is treated as a plaintext input value of the file and is converted into the format of the target column.

If the constant is to be loaded into a numerical column, it must have a valid numerical format, which means it must either be a floating point number in mantissa/exponent representation or a fixed point number in the currently determined or the default decimal representation.

If the input file is empty, constants that may have been specified are not loaded either.

3.8 Loading Special Constants

Apart from loading constants instead of file values as described in the preceding section, it is possible to load the special values STAMP, USER, USERGROUP, DATE, TIME, TIMESTAMP, TRUE, and FALSE.

For this purpose, one of the above mentioned keywords must be provided instead of a position indication.

Example:

```
DATALOAD TABLE item
  modified_by USER
  modified_on DATE
  modified_at TIME
  itno        01-08
  descr       09-39          NULL IF POS 09-11 = '  '
  stock      40-43 INTEGER NULL IF POS 40-43 INTEGER < '0'
INFILE ...
```

The keyword USER designates the current user name. If the current user belongs to a usergroup, then USERGROUP designates the name of this group. Otherwise, USERGROUP is equivalent to USER. The column to be loaded with one of these values must have the type CHAR (n) with $n \geq 8$.

The keyword DATE designates the current date and the keyword TIME, the current time and the keyword TIMESTAMP, the current timestamp. The column to be loaded with one of these values must have the corresponding type.

The keyword STAMP designates a unique value generated by ADABAS. This value can only be loaded into a column having the type CHAR (n) BYTE with $n \geq 8$.

TRUE and FALSE can be specified to set a BOOLEAN column to a constant value.

It is also possible to load user-generated sequence numbers with optional initial values and distances.

Example:

```
DATALOAD TABLE item
  itno          SEQNO 10000 5
  descr         09-39      NULL IF POS 09-11 = '  '
  stock        40-43 INTEGER NULL IF POS 40-43 INTEGER < '0'
INFILE ...
```

The keyword SEQNO indicates that sequence numbers are to be generated. The first number after SEQNO is the initial value (and the first value to be loaded), the second number indicates the distance between the values to be loaded. Both numbers may be negative.

If only one number is specified after SEQNO, it is increased by 1 for every step. If no number is specified after SEQNO, the sequence of numbers begins with 0 and is increased by 1 for every step.

The computable range of numbers is regarded as cyclical; i.e., the value succeeding the greatest representable value is the smallest representable value. In this way, as many sequence numbers can be generated as needed, and these numbers may be repeated several times, if need be. One column with the type FIXED (10) is sufficient to store these numbers.

If the input file is empty, special constants are also not loaded.

Note that in SQLMODE ANSI, only the function USER is allowed.

In SQLMODE DB2, USER and CURRENT SQLID are mapped to USER, CURRENT DATE is mapped to DATE, CURRENT TIME is mapped to TIME, and CURRENT TIMESTAMP generates a timestamp value.

In SQLMODE ORACLE, the functions USER, SYSDATE (generating a timestamp value), and UID (generating a unique user identification) are available.

3.9 Loading Several Tables in a Single Run

Several tables can be loaded from one source file in a single run. In this case, every target table must be described by a DATALOAD specification of its own.

Example:

```

DATALOAD TABLE customer  IF POS 1 = 'c'
  cno      02-05
  name     07-16
  city     18-37
  state    38-39
  zip      40-44
DATALOAD TABLE item      IF POS 1 = 'i'
  itno     02-09
  descr    10-40
  stock    41-44  INTEGER
  min_stock 45-46  INTEGER
  price    47-54  DECIMAL(2)
  weight   55-58  REAL
INFILE ...

```

The source file records can be distributed among the target tables using the selection criterion. In the example above, the first column of the source file is used for distribution (punch card method).

If the criterion is missing for one of the target tables, each input record is selected for this table.

If several tables are loaded at the same time, then an individual INSERT statement is executed for each row. Otherwise, as many records are loaded at the same time as their lengths allow. Of course, single inserts take more time, but the error position can be determined for rejected records. If there is no other way to determine the cause of the error, two DATALOAD statements for the same table will force LOAD to execute single inserts reporting the error position.

3.10 Selection with OTHERWISE

The condition OTHERWISE specifies a selection criterion which is fulfilled when the record has not been selected for any other target table.

With this condition, the "remainder" of the input file records for which no coherent selection criterion can be defined may be loaded into a specific table.

Example:

```
DATALOAD TABLE part1    IF POS 1 = 'a'
  number      02-10      CHAR
  descr       11-30
DATALOAD TABLE part2    IF POS 1 = 'b'
  number      02-10      CHAR
  descr       11-30
DATALOAD TABLE part3    OTHERWISE
  number      02-10      CHAR
  descr       11-30
INFILE ...
```

OTHERWISE may only be specified for the last table in a sequence of DATALOAD statements.

3.11 Loading with DUPLICATES Clause

Often, a table needs to be refreshed by the contents of a file, in which many rows already exist and only a few are new.

A normal DATALOAD statement would reject all rows having a key value that already exists in the table; a normal DATAUPDATE statement would miss the rows which, up to this moment, are only available in the file.

This situation can be resolved by using the mixed form DATALOAD with DUPLICATES clause.

This statement consists of the following parts:

1. A DATALOAD statement, with table condition, if necessary
2. One of three possible DUPLICATES clauses (also see the Reference manual, INSERT)
3. The body of the DATALOAD statement specifying the columns
4. The INFILE specification

Example 1:

```
DATALOAD TABLE customer
  UPDATE DUPLICATES
    KEY    cno      1-4
          city     6-25
          street   27-46
          telephone 48-60
  INFILE address.list
```

Example 2:

```
DATALOAD TABLE birthdaycalendar
  IGNORE DUPLICATES
    KEY    name      1-30
          birthday   31-40
          day_in_year 41-43
  INFILE staff.catalog DATE 'EUR'
```

In the first example, the addresses of available customers are updated and data related to new customers is entered into the table.

In the second example, the birthdays of the colleagues already inserted cannot change. Therefore, it is sufficient to add the birthdays of new colleagues.

The clause REJECT DUPLICATES can also be used. The effect of such a statement is identical to that of a simple DATALOAD statement.

3.12 Input of Test Data on the Screen

If * is specified after INFILE, the input data can be entered using the input form displayed on the screen. The file options DATE, TIME, etc. can also be used after INFILE *.

Example:

```

DATALOAD TABLE customer
  cno          1-5
  firstname    7-16
  name         18-27
  city         29-48
  state        50-51
  zip          52-56
INFILE *
DEC '/ / /'
1 001 JULIE      ANDREWS    SANTA CLARA      CA95054
1 002 WARREN     BEATTY      DALLAS          TX75243
1 003 BURT       LANCASTER  HOLLYWOOD       CA90029

```

3.13 DATALOAD with Input Made in Default Format

A DATALOAD statement without column descriptions can be used when the input file was created from *one* table with the statement DATAEXTRACT * FROM. In this case, the input file contains the necessary column descriptions.

Example:

```

DATALOAD TABLE customer
INFILE customer.load
NULL '-@-'

```

LOAD generates the required column descriptions from the INFILE and then waits for data for all the columns of the table at those positions which follow from the SET parameters or file options.

Any file options for overriding the SET parameters are possible. NULL values are entered if the NULL string is found at a computed position. This format of the DATALOAD statement can also be used together with the file option COMPRESS.

4 Loading Database Tables with FASTLOAD

4.1 The FASTLOAD Statement

LOAD usually creates a series of INSERT statements from a DATALOAD statement and executes them in groups in a sequence of transactions. The inserted table rows are additionally written into the database log whereas rejected records are written into the LOAD protocol file. The affected tables are available for other users even during the load run and can be read and modified afterwards.

The FASTLOAD statement achieves faster loading times, but dispenses with much of the comfort provided by the normal DATALOAD function.

The FASTLOAD statement is meant for cases where the data to be loaded is assumed to be correct and a backup copy of the database is created after loading (see ADABAS CONTROL manual: SAVE PAGES or SAVE DATA).

As no database log is written with FASTLOAD, this statement is particularly suited for cases in which otherwise, loading a very large data file would require a log DEVSPACE beyond the requirements of normal database operation.

Loading with FASTLOAD instead of DATALOAD does not restrict normal database operation; every user can load his tables independently of other users without logging.

There is, however, a considerable difference with regard to the input data: if the table contains KEY columns, only those rows may be inserted which, in ascending order, can be included *after* the already existing rows.

The table is write-protected after having been accessed with FASTLOAD. Nevertheless it may be loaded with data from other files in further FASTLOAD runs.

The FASTLOAD statement has the same structure as a DATALOAD statement and provides the same options:

Example:

```
FASTLOAD
  TABLE PUBLIC.germ_eng
    german      01-20
    english_1   21-40
    english_2   41-60
    english_3   61-80
INFILE germeng.data
```

FASTLOAD, however, imposes the following restrictions:

- FASTLOAD may only be used by the owner of the table.
- Only one FASTLOAD statement is allowed per INFILE, which means that it is not possible to load several tables simultaneously in one run.
- If the table contains KEY columns, the input data must be sorted in ascending order according to its key values.
- No index may exist for the table.
- The table must not contain LONG columns.
- The table must be assigned to the current SERVERDB.
- While loading the table, it is continuously locked against reading and writing by other users.
- The inserted table rows are not written into the database log.
- The table is locked against all write operations, except FASTLOAD until the next backup of the database (SAVE PAGES or SAVE DATA); however, it may be read.
- If the table cannot be loaded successfully, all entries made *during this load run* are deleted again. The data previously inserted is preserved.

- If the database must be RESTARTed before the table has been backed up, the rows inserted with FASTLOAD will not be there any more.

In contrast to a DATALOAD run, where counters are displayed every time a transaction has been closed, FASTLOAD uses a clock to determine the display rate.

If a user cancels a load run (see chapter 17.3 "RUN with STOP Option"), the LOAD protocol file shows the record of the file after which the run was terminated.

4.2 Loading with the USAGE Option

During loading, the space that a table occupies on a storage device is usually used to about 85% only to allow for subsequent INSERT and UPDATE operations.

If the table will be modified only slightly or not at all, it is convenient to use the storage device to a higher degree. On the other hand, a usage of less than 85% may be preferable if a considerable dynamic expansion of the table is expected.

The USAGE option predefines the desired storage usage. This is specified by a percentage number between 50 and 100 (inclusive) referring to the following load run.

Example:

```
FASTLOAD WITH 100% USAGE
  TABLE PUBLIC.germ_eng
    german      01-20
    english_1   21-40
    english_2   41-60
    english_3   61-80
INFILE germeng.data
```

The USAGE option can only be specified within FASTLOAD statements. Note that it does not guarantee that the specified level of storage usage is actually reached. The current storage usage can be checked for the corresponding table by using CONTROL.

5 Updating Table Columns with DATAUPDATE

If new columns are defined for a database table, they are first initialized with the default value (usually NULL). The values for the new column must be loaded separately as a rule.

In other cases, values within specific table fields must be updated because of an activity file.

LOAD provides the DATAUPDATE statement for these purposes. Two simple examples of its notation:

Example 1:

```
DATAUPDATE TABLE customer
  KEY   cno      1-4      (access via
  SET   city     6-26     table key)
INFILE  removal.change
```

Example 2:

```
DATAUPDATE TABLE addresses
           city      1-20      (access via a
  SET      area_code 21-25      non-key field)
INFILE    post.upd
```

Target table and source file are specified in the same way as for DATALOAD.

With regard to the column assignment, DATAUPDATE differentiates between qualification columns which must be entered first and the actual target columns which must be preceded by SET.

For each input record, LOAD accesses the table via the qualification columns using the assigned values from the source file as search argument.

Usually the key columns are used for qualification. The keyword *KEY* must then precede all qualification columns. For each input record, the table row with the specified key is updated in the target columns.

If no key is used for access, *all* table rows to which the qualification applies are updated in the target columns.

Constant input values can be specified for the qualification columns as well as for the target columns (see chapter 3.7 "Loading Any Constants", 3.8 "Loading Special Constants", the DATALOAD description).

In case of DATAUPDATE, the displayed counter reading gives the number of updated table rows, not the number of used file records.

DATAUPDATE offers the same options as DATALOAD:

- * Various kinds of data formats in the source file:

```
CHAR, INTEGER, DECIMAL, ZONED, REAL, addition HEX
```

- * Loading with the functions SCALE, ROUND, and TRUNC:

```
SET km 20-25 SCALE(-3)
```

- * Selecting records from the source file:

```
DATAUPDATE TABLE customer IF POS 1-3 <> '***'
```

- * Selective setting of a column to NULL:

```
SET city 6-26 NULL IF POS 6 = '-'
```

- * Updating columns in several tables:

```
DATAUPDATE TABLE customer IF POS 1='C'
...
DATAUPDATE TABLE item IF POS 1='I'
...
INFILE ...
```

- * Input of test data on the screen with INFILE *

6 Creating Dataextracts with DATAEXTRACT

With ADABAS LOAD, extracts from the database can be provided in external files.

In contrast to the functions TABLEEXTRACT and TABLEUNLOAD, the user can determine the format of the target file and further process the created file.

Files created with the DATAEXTRACT statement can easily be reloaded with a corresponding DATALOAD statement.

6.1 The DATAEXTRACT Statement

The DATAEXTRACT statement consists of a database query and the OUTFILE description.

Example 1:

```
DATAEXTRACT
  firstname, name, city FROM customer;
OUTFILE cmaster.data
```

Example 2:

```
DATAEXTRACT WITH LOCK
  name, city FROM hotel
  WHERE zip LIKE '9*'
  ORDER BY name;
OUTFILE hotel.list
```

Example 3:

```
DATAEXTRACT
  customer.cno, name, reservation.arrival, price
  FROM customer, reservation
  WHERE customer.cno = reservation.cno;
OUTFILE cres.data
```

The *database query* is formulated in the same way as a SELECT statement in SQL, except that the keyword DATAEXTRACT or DATAEXTRACT WITH LOCK is used instead of SELECT. The query must produce an unnamed result table. All options of the SELECT statement are allowed here:

- selecting the result columns and determining their sequence in the result table,
- joining several tables,
- selecting result rows by using qualifications,
- defining a particular sort sequence.

The query must always end with a semi-colon (;).

If the option WITH LOCK is specified, all tables from which rows are to be selected are read-locked so that other users cannot modify these tables during the extract run.

The name of the target file is specified after OUTFILE. Usually, the target file is a disk file. The DATAEXTRACT statement has the effect that the result table is written to the target file. If the target file already exists, it is completely overwritten; otherwise a new one is created.

The data of the target file can be sent directly to the tape device or printer. For testing purposes, some rows of the result table can be displayed on the screen. The filename specific to the operating system (see the platform-specific User Manual) can be used for selection.

If two OUTFILE descriptions are specified, LOAD generates a DATALOAD statement for the extracted data. The statement will, however, only be executable if only one table was used for data selection and all mandatory columns were included in the SELECT list. As usual, the first filename designates the statement file, the second the data file.

6.2 Format Specifications for the Output File

The format specifications related to the file described in chapter 3.4 DEC (decimal representation), DATE (date representation), TIME (time representation), ASCII or EBCDIC (code conversion), etc., are also allowed in DATAEXTRACT statements. These options may be specified in any order.

For output files, the option APPEND which determines that an existing file with the same name will not be overwritten. The extracted data is written successively to the end of this file instead.

The options COMPRESS, SEPARATOR '<character>' and/ or DELIMITER '<character>' can be used to produce a compressed output file. The data is written without leading or closing blanks, with each column value separated by a separating character. The default SEPARATOR is the comma. The option DELIMITER causes character strings (not numbers) to be enclosed in single quotation marks.

If the output file becomes so large that it must be distributed over several magnetic tapes, then the option COUNT must be used. The number of records that fit on the tape must be specified after COUNT. In this case, LOAD writes a line with the sequence number of the partial file to each tape. If the specified number of lines was written, the user is requested to mount the next tape.

6.3 Structure of the Target File

If no specifications for the structure of the target file have been made, the following rules apply:

- Each result row produces an output record.
- The sequence of the output records is determined by the result table.
- The sequence of columns in the output record is determined by the result table.
- The first column starts at position 1 of the output file.
- All output values are output in plaintext.
- The column values are separated from each other by a character string (may be empty) currently set as separator (see chapter 18 "User-specific SET Parameters").

Diverging from these default rules, it is possible to specify exactly which columns are to be written to which positions of the output record.

Example 1:

```
DATAEXTRACT title, name, account FROM customer;
  title      01-05
  name       10-19
  account    20-29
OUTFILE custextract
```

Example 2:

```
DATAEXTRACT customer.cno, name, reservation.rno, hno
FROM      customer, reservation
WHERE     customer.cno = reservation.cno;
          1      01-05
          2      07-13
          3      15-19
          4      21-25
OUTFILE custextract
```


A *column description* assigns the values of a result column to a specific output field in the target file.

If column descriptions are used, the query should only select columns that are output or used for sorting in an ORDER BY condition.

The sequence of column descriptions is arbitrary.

The columns can be specified with their *names* or their *sequence numbers* in the result table.

The *output fields* are described by their starting and ending positions, just like the input fields for DATALOAD. Gaps between fields are of no consequence because LOAD fills these gaps with blanks, if necessary.

If an output field is longer than required, numerical values are justified to the right and character strings to the left. The remaining gaps are padded with blanks.

If an output field is shorter than required, character strings are truncated on the right. The execution of the DATAEXTRACT statement will be rejected for numerical values, if the loss of significant digits is to be expected.

6.4 Data Formats in the Target File

On request, LOAD converts the output values into the specified data format (see chapter 12 "External Data Formats").

Example:

```
DATAEXTRACT * FROM item;
  itno      01-08  CHAR
  descr     09-39
  stock     40-43  INTEGER
  min_stock 44-45  INTEGER
  price     47-53  DECIMAL(2)
  weight    54-57  REAL
OUTFILE ...
```

The external data format need only be specified in an extract statement if the relevant column is to be output in another format than CHAR. Output in CHAR format is possible for all column types.

LOAD can convert columns of the type FIXED, SMALLINT, and INTEGER into any of the numerical data formats INTEGER, DECIMAL, ZONED, and REAL; if, in doing so, any significant digits could be lost, a syntax error is reported. Refer to the annotations for the external data format INTEGER in chapter 12.

FLOAT-type columns can only be converted into REAL (or CHAR) ones.

In order to edit the output file in every environment, each of the mentioned data formats can also be supplied in hexadecimal representation with two hexadecimal digits representing one byte. In this case, the data type must be provided with the additional specification HEX, namely

```
[CHAR] HEX                                or
[CHAR] FLOAT HEX                          or
INTEGER HEX                               or
REAL HEX                                  or
DECIMAL [fraction] HEX                    or
ZONED [fraction] HEX.
```

Each HEX-formatted data field needs exactly double the space it would need for the same format without the HEX option specification.

6.5 Options for the Output of Numerical Columns

All numerical columns of the result table can be edited by using the functions SCALE, ROUND, and TRUNC. The data can have any external format because the functions are applicable to columns that are to be output in CHAR format as well as to columns in REAL, ZONED, or other formats.

The scaling factor specified after the keyword SCALE can be positive or negative. The value to which the function is referring will be multiplied by the corresponding decimal power.

The ROUND and TRUNC functions determine the *fractional digits* of a number. The number n of fractional digits must lie between 0 and 18. If the number has no fractional digits, the functions have no effect.

TRUNC n means that the n+1st and all the following fractional digits of the number are set to 0, while the first n fractional digits remain unchanged.

ROUND n means that the number is to be rounded from right to left starting with the n+1st fractional digit. If this digit is greater than or equal to 5, the nth digit will be incremented by 1. In this case, too, the result is a number with the n+1st and all the following fractional digits equal to 0; but the first digits of the number may be modified by rounding up.

Both ROUND and TRUNC can be applied in combination with the SCALE function. The functions must be specified in the following order: SCALE before ROUND or TRUNC. The order of processing corresponds to this order.

Example:

```
DATAEXTRACT * FROM distance
...
cm      10-13 INTEGER  SCALE 2
meter   14-17 INTEGER
km      18-21 INTEGER  SCALE -3 ROUND 0
...
OUTFILE dimensions.bin
```

If numbers are output in plaintext, the CHAR FLOAT option can be used to obtain a floating point representation of these numbers, regardless of their size. If the SCALE function is applied and numbers to be output become so large or so small that they cannot be represented any more as fixed point numbers, then they are automatically output in floating-point format.

6.6 Output of NULL Values

A character string that can be defined using the SET command is used to represent NULL values in the target file.

For each output column, a particular constant can be declared that will be written into the output record when a NULL value occurs:

Example 1:

```
DATAEXTRACT hno, arrival, departure FROM reservation;
hno      01-05
arrival   07-13
departure 15-21 IF NULL SET POS 15-29 = 'permanent_guest'
OUTFILE  ...
```

The keywords IF and SET can be omitted, whereas the keywords NULL and POS are mandatory. The default operator is '='. Further operators are not allowed.

The position specification of the output field for the constant is made in the same way as all other position specifications. LOAD does not check whether this output field overlays other fields.

It is only necessary to specify the data format of the constant in the output file when it is not the default CHAR. In any case, the constant must be specified within the statement as plaintext value enclosed in single quotation marks.

If the constant is to be output in one of the numerical data formats INTEGER, DECIMAL, or ZONED, it must have a valid number format. This means that it must either be a floating point number in mantissa/exponent representation, or a fixed point number in the currently determined or default decimal representation.

Example 2:

```
DATAEXTRACT itno,name,price,expiration_date FROM item;
  itno          01-03
  name          07-26
  price         30-42 ZONED NULL POS 30-42 ZONED '-1,00'
  expiration_date 46-51
OUTFILE  item.extract
  DATE  'yymmdd'
  DEC  '/' /./'
```

Specifying a NULL condition for NOT NULL columns does not produce a syntax error as it does for DATALOAD and DATAUPDATE, because no subsequent errors can result from it.

6.7 Text Constants in the Target File

Additional fields with text constants can be placed into the target file among those output fields which are filled from the result table:

Example:

```
DATAEXTRACT cno, firstname, name FROM customer;

'Customer Number:'  1 - 15
'Name :'            22 - 28

cno                16 - 21
firstname          29 - 38
name              39 - 48
OUTFILE  ...
```

The text constant enclosed in single quotation marks is specified in this case instead of the column name or column number.

The text constant is output in CHAR format and truncated or padded with blanks on the right, if necessary.

There is no sequence for the description of output fields; output columns and output constants can be mixed.

If the statement contains the constant but no column descriptions, all result columns are entered into the output record in accordance with the default conventions. Constants entered previously may be overwritten in this case.

6.8 Generating Command Files with DATAEXTRACT

A special format of the DATAEXTRACT statement can be used if you want to write the data of a table into an external file and make this file be usable as a command file for reloading the table as well.

DATAEXTRACT FOR DATALOAD

Syntax:

```
DATAEXTRACT [WITH LOCK]
  FOR DATALOAD | FASTLOAD
  TABLE <table name>
  [ <order clause> ] ;
<external outfile spec>
[ ; <external outfile spec> ]
```

Example:

```
DATAEXTRACT FOR DATALOAD
  TABLE customer ;
OUTFILE customer.load ;
OUTFILE customer.data ;
```

This statement generates a command file that allows a table to be completely restored. In contrast to the simple DATAEXTRACT statement, it is therefore not possible to exclude columns or rows from the table.

The command file contains a CREATE TABLE statement only when the table is a base table and belongs to the executing user. The command file always contains a DATALOAD statement and the complete table contents.

When the option WITH LOCK is specified, the table will be read locked during execution so that no simultaneous modifications can be made to it.

When FASTLOAD is specified, a FASTLOAD statement is generated instead of the usual DATALOAD statement. The FASTLOAD statement contains a USAGE clause. 80 is the default percentage. According to the user's requirements, this percentage can be changed to any value between 50 and 100 by editing the generated command file.

If the order of the table rows in the file is important, e.g., for FASTLOAD, you need to formulate an ORDER BY statement.

As a precaution, table and column names are treated as <special identifier>s in the output file and are enclosed in double quotation marks. This notation is mandatory if a name contains special characters or is identical to an SQL keyword, or if upper- and lowercase characters are to be distinguished.

When two OUTFILE specifications are made, the first file contains the statements, the second one the data. This allows statements to be edited when the table contains BYTE columns or is very large.

File options like ASCII/EBCDIC, DATE format, etc. refer only to the file for which they have been specified. The SET values will be inserted for missing options.

When the COMPRESS option is specified for the generated file of statements, it has the effect that the column names are only output in their actual lengths. When the COMPRESS option is specified for the data, it has the same effect as for a normal DATAEXTRACT statement.

The generated DATALOAD statement contains all the file options so that the user is independent of the current SET statement values for loading.

As the same character string is always used to represent NULL values when extracting data, the generated DATALOAD statement contains DEFAULT NULL conditions for all optional columns. The NULL representation used is recorded as file option NULL '<string>'.

The selected data is written into the file in accordance with the default conventions. An explicit description of the output format (e.g., INTEGER, SCALE specification) is not possible.

If the table is empty, a special DATALOAD statement is generated that contains a selection condition in the format IF POS n-m <> '<literal>'.

A line containing this literal at the specified position is written to the desired data file.

Thus a file having the specified name is created with a separate command and data files, even if a table contains no data. This ensures that a load run is not interrupted because LOAD does not find the specified file.

If the data is written to the file containing the statements, then there is at least one input line included for a DATALOAD statement. This guarantees that the next statement of the command file is not misunderstood as data input.

No data is entered into the table formerly empty or its counterpart, because the input line generated by LOAD does not meet the selection criterion.

DATAEXTRACT FOR DATAUPDATE

This statement generates a command file which contains a DATAUPDATE statement and the extracted data. This command file can be used for restoring defined contents after modifying the table.

Syntax:

```
DATAEXTRACT [ WITH LOCK ]  
  FOR DATAUPDATE TABLE <table name>  
  [ <order clause> ] ;  
<external outfile spec>  
[ ; <external outfile spec> ]
```

Example:

```
DATAEXTRACT WITH LOCK
  FOR DATAUPDATE TABLE customer ;
OUTFILE customer.upd
```

The table <table name> must have at least one key column. Otherwise, it would be impossible to unambiguously restore the backed up state.

An SQL SELECT statement is executed. It selects all the columns and all the rows of the table. When the option WITH LOCK is specified, the table is read locked during execution so that no simultaneous modifications can be made to it. When an ORDER BY clause is specified, it is added to the SELECT statement.

A DATAUPDATE statement is generated for the table and written into the specified file. With this statement, modifications made to the table can be rolled back to the state represented by the selected data.

As a precaution, the table and column names are treated as <special identifier>s and are enclosed in double quotation marks. This notation is mandatory if a name contains special characters or is identical with an SQL keyword or if upper- and lowercase characters are to be distinguished.

If two OUTFILE specifications are made, the first file contains the statements, the second one the data.

The DATAUPDATE statement contains DEFAULT NULL conditions for the optional columns of the table. The NULL representation used is recorded as file option NULL '<string>'.

The DATAUPDATE statement contains complete specifications of all input formats and can therefore be executed independently of the current SET specification (see "DATAEXTRACT FOR DATALOAD").

The selected data is written into the file in accordance with the default conventions. An explicit description of the output format (e.g., DECIMAL, IF-NULL-SET-POS) is not possible.

Empty tables are handled in the same way as in case of DATAEXTRACT FOR DATALOAD (see chapter "DATAEXTRACT FOR DATALOAD").

6.9 Test Output on Screen

Entering OUTFILE * displays a section of the result table on the screen.

If the default representation is chosen, this section shows the left upper part of the result table. Beginning with the first row, only as many rows are displayed as fit a screen page, and rows longer than the screen width are truncated on the right.

Explicit column descriptions must be formulated so that the output rows fit the screen; otherwise, an error is reported.

The displayed line area can be shifted using a RUN option (see chapter 17.2 "RUN with Range Option") to determine a starting row that is other than the first row of the result table.

LOAD displays the number of output rows after the return to the input screen.

Example:

LOAD ... Output		Load/Update/Extract		001-018	
		A D A B A S			
1001	JULIE	ANDREWS	Santa Clara	CA95054	
1002	WARREN	BEATTY	Dallas	TX75243	
1003	BURT	LANCASTER	Hollywood	CA90029	
----- <serverdb> : <user> -----					
3=Return					

7 Loading and Extracting LONG Columns

LONG columns are loaded, updated, and unloaded by DATALOAD, DATA-UPDATE, and DATAEXTRACT.

7.1 Loading LONG Columns with DATALOAD

The DATALOAD statement can also insert data into LONG columns. The LONG value to be inserted, however, is not stored in the input file specified after INFILE. At the place equivalent to the position of the LONG column, the line of the input file contains a reference to the LONG value in the form of a filename.

In the following example, files with the model name <hotel name>.LNG contain the input data for the LONG column HOTEL.INF. The filenames contain no blanks and can therefore also be written without single quotation marks.

Example:

```

DATALOAD TABLE hotel
  hno      1-2
  name     6-14
  zip      18-22
  address  26-44
  info     48-62
INFILE *
/ *
10 | Congress | 20005 | 155 Beechwood Str. | 'CONGRESS.LNG'
30 | Regency  | 20037 | 477 17th Avenue   | 'REGENCY.LNG'
60 | Airport  | 60018 | 650 C Parkway     | 'AIRPORT.LNG'
90 | Sunshine | 33575 | 200 Yellowstone Dr. | 'SUNSHINE.LNG'

```

As each LONG value must be stored in a separate file, this procedure can only be used if not too many LONG values are to be loaded. Therefore, another way to load LONG values has been provided which allows LONG values to be defined as sections of a file by appending a starting and end position to the filename.

Example:

```
DATALOAD TABLE hotel
  hno      1-2
  name     4-11
  zip      13-17
  address  19-37
  info     39-63
INFILE *
/ *
10|Congress|20005|155 Beechwood Str. |'HOTEL.LNG' 1-866
30|Regency |20037|477 17th Avenue   |'HOTEL.LNG' 867-1026
60|Airport |60018|650 C Parkway     |'HOTEL.LNG' 1027-1313
90|Sunshine|33575|200 Yellowstone Dr.|'HOTEL.LNG' 1314-1888
```

This method allows you to load several LONG values from one file or, for DATAEXTRACT, to extract all LONG values belonging to one column to a single file.

The used position specifications need not be continuous, but each starting position must be greater than the previous end position.

The following restrictions apply to LONG columns:

1. The DATALOAD statement must only refer to one table.
2. FASTLOAD is not supported for LONG columns.

Any other load and file options are available.

7.2 Updating LONG Columns with DATAUPDATE

LONG columns can be specified in a DATAUPDATE statement like any other column. For the input file, the same conventions apply that are valid for DATALOAD.

Example:

```
DATAUPDATE TABLE hotel
  KEY hno      1
  SET info     2
INFILE * COMPRESS
/ *
10;"'CONGRESS.LNG'"
30;"'REGENCY.LNG'"
60;"'AIRPORT.LNG'"
90;"'SUNSHINE.LNG'"
```

In this example, the option COMPRESS is used. This option has the effect that the values are separated from each other by a semicolon and character strings are enclosed in quotation marks. The single quotation marks enclosing the filename can be omitted.

7.3 Extracting LONG Columns with DATAEXTRACT

As for DATALOAD, two methods are supported for DATAEXTRACT. In the simplest case, each extracted LONG value is written to a separate file.

Each line of the usual OUTFILE contains a reference to the corresponding LONG value in the form of a filename. The LONGFILE specification predefines the filename used. ###... denote a sufficient number of positions for the numbering of the LONG values. DATAEXTRACT implicitly names the LONG value file belonging to line 1 ...001, the LONG value file belonging to line 2 ...002, etc.

Example:

```

DATAEXTRACT * FROM hotel WHERE hno < 100 ;
    hno      1-2      ' | ' 3-5
    name     6-18     ' | ' 19-20
    zip      21-25     ' | ' 26-28
    address  29-47     ' | ' 48-50
    info     51-63
OUTFILE * NULL '-?-'
LONGFILE info HOTEL##0.INF

```

Result:

10		Congress		20005		155 Beechwood Str.		'HOTEL010.INF'
20		Long Island		11788		1499 Grove Street		-?-
30		Regency		20037		477 17th Avenue		'HOTEL030.INF'
40		Eight Avenue		10019		112 8th Avenue		-?-
50		Lake Michigan		60601		Oak Terrace		-?-
60		Airport		60018		650 C Parkway		'HOTEL060.INF'
70		Empire State		12203		65 Yellowstone Dr.		-?-
80		Midtown		10019		12 Barnard Str.		-?-
90		Sunshine		33575		200 Yellowstone Dr.		'HOTEL090.INF'

ADABAS supports more than one LONG column per table. For DATAEXTRACT, a separate LONGFILE specification with the column name is therefore required for each selected LONG column.

The procedure to extract each LONG value into a separate file can only be used for a table with a relatively small number of entries. For this reason, an option has been provided for DATAEXTRACT which allows all values belonging to one LONG column to be extracted to a single file.

The corresponding LONGFILE specification only contains the filename for the LONG column without the ###... marks denoting the number positions. In addition to the filename, the OUTFILE contains the starting and end positions for each LONG value at the corresponding place.

Example:

```
DATAEXTRACT * FROM hotel WHERE info IS NOT NULL ;
  hno      1-2      '|' 3
  name     4-11     '|' 12
  zip      13-17    '|' 18
  address  19-37    '|' 38
  info     39-60
OUTFILE *
LONGFILE info HOTEL.LNG
```

Result:

```
10|Congress|20005|155 Beechwood Str. |'HOTEL.LNG' 1-866
30|Regency |20037|477 17th Avenue   |'HOTEL.LNG' 867-1026
60|Airport |60018|650 C Parkway      |'HOTEL.LNG' 1027-1313
90|Sunshine|33575|200 Yellowstone Dr.|'HOTEL.LNG' 1314-1888
```

LONG columns with the value NULL are treated according to the usual conventions. The OUTFILE does not contain a filename but the defined representation of the NULL value.

Empty LONG columns are represented by an empty file or by a position specification with an end position less than the starting position.

8 Database Catalog Migration

7

The CATALOGEXTRACT statement is used to generate a LOAD command file that allows the database catalog to be restored. The statements of this file can be executed on any computer by using the corresponding CATALOGLOAD statement. If code conversion is required, it can be achieved with the file options ASCII or EBCDIC.

With CATALOGEXTRACT TABLE, a user can generate a command file containing catalog information about a base table for which the user has the OWNER privilege.

With CATALOGEXTRACT USER, a user can generate a command file containing the definitions of all his private objects.

With CATALOGEXTRACT ALL, a SYSDBA can generate a command file that defines the complete database catalog.

With CATALOGLOAD TABLE or CATALOGLOAD USER, a user can generate his private partial catalog. Only a user with SYSDBA status can define the complete catalog with CATALOGLOAD ALL.

As not only SQL statements but also the contents of system tables are needed to reload the procedural database objects trigger, DB procedure, and DB function, these are only exported by CATALOGEXTRACT if the statement is a part of DBEXTRACT.

The CATALOGEXTRACT and CATALOGLOAD statements can be part of a command file and may be executed as a batch job.

8.1 CATALOGEXTRACT TABLE

Syntax:

```
CATALOGEXTRACT TABLE <table name>;  
OUTFILE <external file name>
```

CREATE statements for all objects of the database catalog defined by the current user that are related to the specified table are written to the file specified after OUTFILE (* as the filename is not allowed).

GRANT statements are written to the command file that define the privileges for these objects that the current user has granted to other users.

The generated command file contains

- the CREATE TABLE statement,
- CREATE SYNONYM statements for table name synonyms the user has defined,
- GRANT statements for the privileges granted for this table to other users,
- CREATE INDEX statements for the indexes the user has defined for the table,
- ALTER TABLE ... FOREIGN KEY statements for the definition of link associations defined between this and other tables,
- CREATE VIEW statements for the views the user has defined on this table,
- COMMENT ON statement for the comments the user has defined for the table and its columns and indexes.

Within the generated command file, the statements are stored in the order they will be executed; i.e., a view definition is placed after the CREATE statement for the base tables and any further view tables it refers to.

The generated command file contains one statement that should be handled in a special way. The command file should therefore be executed with CATALOGLOAD TABLE.

8.2 CATALOGEXTRACT USER

Syntax:

```
CATALOGEXTRACT USER;  
OUTFILE <external file name>
```

CREATE statements for all objects of the database catalog that the current user has defined are written to the file specified after OUTFILE (* as the filename is not allowed).

GRANT statements are written into the command file for the privileges the current user has granted for these objects to other users.

The generated command file contains

- CREATE TABLE statements for the user's base tables,
- CREATE SYNONYM statements for the table name synonyms the user has defined,
- GRANT statements for all privileges the user has granted to other users,

- CREATE INDEX statements for the indexes defined for all the tables of the user,
- ALTER TABLE ... FOREIGN KEY statements for the link definitions referring to all tables of the current user,
- CREATE VIEW statements for the user's view tables,
- COMMENT ON statements for the comments the user has defined for the tables, columns, indexes and domains,
- CREATE DOMAIN statements for the user's domains.

The generated command file contains one statement that should be handled in a special way. The command file should therefore be executed with CATALOGLOAD USER.

8.3CATALOGEXTRACT ALL

Syntax:

```
CATALOGEXTRACT ALL;  
OUTFILE <external file name>
```

This statement can only be executed by a SYSDBA.

CREATE statements for all objects of the database catalog are written into the file specified after OUTFILE (* as the filename is not allowed).

GRANT statements are written into the command file for all the privileges granted to other users.

The generated command file contains

- CREATE USER statements for all users and user groups defined in the catalog. Passwords are encrypted.
- CREATE DOMAIN statements for all DOMAIN definitions,
- CREATE TABLE statements for all base tables,
- CREATE SYNONYM statements for all table name synonyms,
- GRANT statements for all privileges granted to other users,
- CREATE INDEX statements for all indexes,
- ALTER TABLE ... FOREIGN KEY statements for all link associations,
- CREATE VIEW statements for all view tables,
- statements for reloading triggers and DB procedures,
- COMMENT ON statements for all comments.

The statements are written to the command file in an executable order.

Where required, OWNER statements are written into the generated command file. These serve to associate the objects with their original owners when restoring the catalog.

These OWNER statements are executed and the passwords are decrypted successfully only when the command file is performed with CATALOGLOAD ALL.

8.4 CATALOGLOAD TABLE

Syntax:

```
CATALOGLOAD TABLE <table name>;  
INFILE <external file name>
```

The command file specified after INFILE is scanned and the SQL statements included there are executed.

8.5 CATALOGLOAD USER

Syntax:

```
CATALOGLOAD USER;  
INFILE <external file name>
```

The command file specified after INFILE is scanned and the SQL statements included there are executed.

8.6 CATALOGLOAD ALL

Syntax:

```
CATALOGLOAD ALL;  
INFILE <external file name>
```

LOAD checks whether the executing user is SYSDBA. If this is the case, the file specified after INFILE is scanned and the SQL statements included there are performed. The encrypted passwords and the OWNER statements are converted into an executable format.

Further LOAD Statements

LOAD

9 Migration of the Database Contents

LOAD provides the functions UNLOAD and RELOAD as well as SAVE and RESTORE for any number of users. Tables which temporarily are not needed can be unloaded from the database using UNLOAD to make space; they can be reloaded into the database using RELOAD. The functions SAVE and RESTORE generate backup versions of a table which can be loaded again; e.g., after an application program has run.

TABLEEXTRACT, TABLEUNLOAD, and TABLELOAD complete CATALOGEXTRACT and CATALOGLOAD and transfer the contents of database tables as well as LONG columns and indexes.

TABLEEXTRACT performs the function SAVE which backs up the database without modifying it. If temporary space is needed in the database, the statement TABLEUNLOAD can be used instead of TABLEEXTRACT. TABLEUNLOAD unloads tables; i.e., the contents of the tables are deleted and operations on these tables are not possible until they have been reloaded using TABLELOAD.

With TABLELOAD, the data can be loaded whether it is still available in the database (after TABLEEXTRACT) or not (after TABLEUNLOAD). This means, this command performs the functions RELOAD and RESTORE as well.

Only the data pages are saved. Information about the catalog is managed by LOAD. Thus the scope of table definitions regarded as compatible with this data is relatively large:

- It is not necessary to reload the data either into the same table or into a table having the same definition as the original table. Table and column names are not important for deciding whether the file and the target table are compatible with each other. Certain structural differences may be adapted by LOAD. In such a case, the target table will be modified so that its structure fits the file's data.
- Database contents can be transferred from one computer to another. The necessary conversions are made in the process.

- If the database contains unused pages, it can be reduced. To do so, all the data must be unloaded. Afterwards, the database must be reconfigured with smaller devspaces and the data must then be reloaded.

As for the CATALOG EXTRACT/LOAD statements, there are the three options ALL, USER, and TABLE.

The option ALL is only accepted for a user with SYSDBA status. Only the datasets residing on this location are extracted and only tables stored on this location can be reloaded.

Controlled by the options USER and TABLE, normal users can unload and reload all or particular base tables belonging to them.

The referenced statements may be part of a command file and may be executed in batch.

9.1TABLEEXTRACT TABLE

Syntax:

```
TABLEEXTRACT [TABLE] <table name>;  
OUTFILE <external file name>
```

The description of the extracted table, along with all the data pages pertinent to this table, is written to the file indicated after OUTFILE (* as the filename is not allowed). Different types of pages contain primary data, indexes, if any, and the contents of LONG fields. In a status line, LOAD displays the table name and the type of page currently being processed.

The generated port file is not readable and must not be modified.

9.2 TABLEEXTRACT USER

Syntax:

```
TABLEEXTRACT USER;  
OUTFILE <external file name>
```

An internal list is generated containing all the base tables of the current user (in analogy to CATALOGEXTRACT).

TABLEEXTRACT <table name> is executed for each table included in this list. The extracts are appended to the output file which, consequently, consists of a series of table descriptions and table contents. From the status messages, the user can see which table LOAD is extracting.

9.3 TABLEEXTRACT ALL

Syntax:

```
TABLEEXTRACT ALL;  
OUTFILE <external file name>
```

This statement can only be performed by a SYSDBA.

An internal list is generated containing all the users and their base tables (in analogy to CATALOGEXTRACT). All entries that refer to tables residing on other locations are deleted from this list.

TABLEEXTRACT <table name> is executed for each table included in this list. The extracts are appended to the output file which, consequently, consists of a series of table descriptions and table contents of different users.

9.4TABLEUNLOAD

TABLEUNLOAD has the same syntax as TABLEEXTRACT. This statement also allows you to extract one table, all the tables of one user, or all the tables of the database location.

In contrast to TABLEEXTRACT, all the data included in the tables addressed by TABLEUNLOAD are deleted from the database. Operations on the tables are not possible until the data has been reloaded using TABLELOAD.

9.5TABLELOAD TABLE

Syntax:

```
TABLELOAD [TABLE] <table name>;  
INFILE <external file name>
```

The input file specified after INFILE is scanned. The information about the source table stored there is compared to the definition of the target table specified in the statement.

As long as the only differences are range of value limitations, default values, and index definitions, the structure of the target table is adapted to the structure of the source table during the load prolog. Types and lengths of every column pair formed from source and target table must be identical.

Although source table and target table have the same external structure, they may differ in their internal structure. This is the case when columns have been dropped from one of the tables or columns have been added to an existing table. To be able to continue the load run, the complete column definitions must be loaded in this case.

If a table has indexes that differ from those of the other table, LOAD produces the state of the source file also for the target table; i.e., it drops all the indexes that only exist in the target table and creates the indexes that are available for the source table but are missing in the target table.

The same applies when both tables have different DEFAULT definitions. A column without explicit default, implicitly has the default NULL. The defaults of all columns are passed from the source table to the target table. This means, that explicit defaults of the target table may be deleted.

In interactive mode, the user is asked whether the required adaptations are to be made or cancelled. In case of BATCH runs, LOAD assumes that the user agrees to the adaptations.

The target table must not contain any entries before importing data. This is achieved by using DELETE without qualification.

TABLELOAD, like FASTLOAD, puts the table into READ ONLY mode.

9.6TABLELOAD USER

Syntax:

```
TABLELOAD USER;  
INFILE <external file name>;  
OUTFILE <external file name>
```

The file specified after INFILE contains a series of logical individual files. TABLELOAD TABLE is performed for each of these logical files. The respective table name is read from the file; i.e., the name of the source table is used. A (target) table with the same name and a suitable structure must therefore exist in the database.

If a logical file cannot be loaded, a corresponding message containing the name of the table is output to the protocol file.

The blocks of lines that belong to rejected tables are continuously written to the file specified after OUTFILE. This file may be used as input file for an update run after the corresponding corrections have been made to the database catalog.

Such problems do not occur when a record of a file generated by CATALOGEXTRACT USER and TABLEEXTRACT USER is reloaded with CATALOGLOAD USER and TABLELOAD USER.

9.7TABLELOAD ALL

Syntax:

```
TABLELOAD ALL;  
INFILE <external file name> ;  
OUTFILE <external file name>
```

This statement can only be performed by a SYSDBA.

The file specified after INFILE contains a series of logical individual files.

TABLELOAD <table name> is performed for each of these logical files, whereby the table name and user name are read from the file. Such a user must therefore exist in the database and be owner of a table with the same name and a suitable structure.

If a logical file cannot be loaded, a corresponding message containing the names of the table and user is output to the protocol file.

The blocks of lines that belong to rejected tables are continuously written to the file specified after OUTFILE. This file may be used as the input file for an update run after the corresponding corrections have been made to the database catalog.

No problems occur when CATALOGEXTRACT ALL and TABLEEXTRACT ALL are combined with CATALOGLOAD ALL and TABLELOAD ALL (statements DBEXTRACT and DBLOAD).

10 Migration of Catalog and Contents

10.1 DBEXTRACT

DBEXTRACT combines the two statements CATALOGEXTRACT ALL and TABLEEXTRACT ALL to form one statement. This statement along with DBLOAD is the only way to migrate the procedural database objects TRIGGER, DBPROCEDURE, and DBFUNCTION with LOAD means.

Syntax:

```
DBEXTRACT;  
OUTFILE <external file name> ;  
OUTFILE <external file name>
```

The first generated output file is a plaintext file. It contains the catalog definitions. The second output file is not readable. It contains the table contents.

DBEXTRACT can only be executed by a SYSDBA.

10.2DBLOAD

DBLOAD combines the two statements CATALOGLOAD ALL and TABLELOAD ALL to form one statement.

Syntax:

```
DBLOAD;  
INFILE <external file name> ;  
INFILE <external file name> ;  
OUTFILE <external file name>
```

CATALOGLOAD ALL and TABLELOAD ALL are implicitly executed one after the other. The first INFILE name must therefore indicate the file with the CREATE statements for the database catalog, the second INFILE name must indicate the file containing the pertinent table contents, and the file specified after OUTFILE will receive the tables that could not be loaded.

DBLOAD can only be executed by a SYSDBA.

11 The ORACLE Crossloader

11.1 Loading an ORACLE Database

A file generated with the ORACLE function EXP containing any section of an ORACLE database can be loaded into an ADABAS database by using LOAD.

Syntax:

```
FASTLOAD ORACLEDB | LOAD ORACLEDB  
INFILE <external file name> INTEGER LOHI
```

LOAD executes the definition (DDL) statements contained in the file. When doing so, the ORACLE statement GRANT CONNECT is converted into the ADABAS statement CREATE USER and the username is defined as password. As for CATALOGLOAD ALL, the ORACLE statement CONNECT <user> is used as LOAD statement OWNER <user> to change the current user. All ORACLE statements starting with one of the keywords ALTER, COMMENT, CREATE, or GRANT are executed in SQLMODE ORACLE.

INSERT statements are converted into a DATALOAD or FASTLOAD statement and executed immediately, like the DDL statements. The data to be loaded is in this way read from the following lines of the input file and converted from the available internal ORACLE format.

The statements are recorded in the protocol file in the way in which they are executed by LOAD.

Every user can perform LOAD ORACLEDB as long as database objects are created for this user. If the input file requires a change to another user, the run is aborted when the executing user is not the SYSDBA.

11.2 Loading the Contents of an ORACLE Table

An ORACLE table is loaded with a DATALOAD statement containing the keyword ORACLE after the INFILE specification. For security reasons, the integer representation used in the EXP file (INTEGER LOHI) should be specified as well. Descriptions of fields can be used if the definition of the ORACLE table is not identical to that of the table to be loaded. The position specification in the field descriptions must denote the sequence numbers of the columns in the original table. It is not necessary to load all columns into the new table. It is possible to load constants or values of functions.

Example:

```
DATALOAD TABLE reservation
  resnumber      SEQNO
  date           SYSDATE
  customername   2
  city           4
INFILE ora_res.exp
ORACLE          INTEGER LOHI
```

In this case, LOAD expects such a file generated using the ORACLE function EXP as for the LOAD ORACLEDB statement. This file, however, should not contain more than one table. If more tables are stored in the EXP file, LOAD attempts to use the first occurring data block for the loading.

12 External Data Formats

In the following, *external data format* identifies the coding of a field value in the source or target file, and *column type* identifies the type of a table column as it was declared when defining the table.

The following external data formats are processed by LOAD:

CHAR	according to the computer, ASCII or EBCDIC code; maximum length is 254 bytes
INTEGER	binary code (machine specific) 1, 2, or 4 bytes long, sign in bit 0, negative values in two's complement form
REAL	floating point representation with mantissa and exponent, 4 or 8 bytes long, in machine-specific representation
DECIMAL	packed decimal: one digit per half byte, sign coded in the rightmost half byte, 1 to 10 bytes long, at most 18 digits, the position of the virtual decimal point is derived from the type of the table column
DECIMAL (n)	n specifies the number of digits to the right of the virtual decimal point
ZONED	zoned decimal: allowed are the /370 Zoned Data Format, the COBOL Standard Zoned Data Representation, as well as COBOL Zoned Data With Leading Sign, and COBOL Zoned Data With Trailing Sign in separate position, 1 to 18 bytes long, the position of the virtual decimal point is derived from the type of the table column
ZONED (n)	n specifies the number of digits to the right of the virtual decimal point

Examples of type declarations in different programming languages:

External Format	C Type
INTEGER	INT
REAL	FLOAT

External Format	COBOL Type
INTEGER	COMP-1
REAL	COMP-2
DECIMAL	PIC S9 (n) V9 (n) COMP-3
ZONED	PIC S9 (n) V9 (n)

Information concerning DATALOAD and DATAUPDATE:

Any numerical data format (INTEGER, REAL, DECIMAL, ZONED) can be converted into any *numerical* column type (FIXED, SMALLINT, INTEGER, FLOAT).

ASCII/EBCDIC coded values can be converted into *any* column type.

REALs must be coded in the form that is specific to the computer. They cannot be transferred between computers of different types. INTEGER values are interpreted as signed values. Their representation can be adapted (e.g., for DATALOAD) using the file option INTEGER. ZONED and DECIMAL values have a defined coding; i.e., they are not a problem.

Information concerning DATAEXTRACT:

The numerical data types FIXED, SMALLINT, and INTEGER can be converted into all listed data formats.

If an overflow occurs when a FIXED, SMALLINT, or INTEGER column value is converted into an external INTEGER value, asterisks ("*") instead of the binary number are inserted into the data field, and the extract run is aborted.

Limiting values are for a

```
1-byte integer : -128 and +127
2-byte integer : -32768 and +32767
4-byte integer : -2147483647 and +2147483647
```

The numerical data type FLOAT can be output in any CHAR format or as REAL.

13 The Loading Process

A load run is started by entering the command RUN in the command line or by pressing the function key **Run**. LOAD thus changes from INPUT to EXECUTE mode which does not permit the user to make any input on the screen.

13.1 The Reasonableness Check

In the first phase, LOAD checks whether the syntax of the statement is correct and whether the table and column names are compatible with the database catalog.

For DATAEXTRACT with *explicit* column definitions, an additional test is made to determine whether the output fields for numerical columns are wide enough and the length of the output row complies with limitations such as the screen window size.

If an error is detected, LOAD displays and highlights the relevant line on the screen, positions the cursor on the error, and outputs a system message.

Example:

```

LOAD ...   Input           Load/Update/Extract           001-018
-----
                                A D A B A S
DATALOAD TABLE customer
  cno           01-05
  firstname     09-18  NULL IF POS 6 >> ' '
  name          20-29      - (cursor)
  city          32-51
  state         52-56
  zip           57-61
  price         48-55  DECIMAL(2)
  weight        56-59  REAL
INFILE cmaster.data
-----
                                <serverdb> : <user>
  1=Help 2=Reset 3=End 4=Print 5=Run 6=Next 7=Pick 8=Put 12=Mark
-12307 Incorrect comparison operator in condition
==>

```

LOAD returns here to the Input mode. The incorrect statement can be corrected by the user and restarted.

If the HELP function is called in this situation, the system branches at once to the description of the incorrect statement (see chapter 2.4 "The HELP Function").

13.2 Status Messages During Loading

The following information refers to the DATALOAD and DATAUPDATE statements.

LOAD reads the records of the source file sequentially. Each record is tested to determine which target file fulfills the selection criterion. Per target table, one table row is created for each selected record.

Rows being inserted into the table may be rejected by the database because they violate the *integrity of the database* (e.g., duplicate key value). Rejected rows are

recorded in the protocol file along with the error message (see chapter 14 "The LOAD Session Log").

During execution, two counters keep track of the number of rows inserted or rejected for each statement and target table.

The current counter readings for the relevant target table are displayed if one of the following conditions is true:

- a) The current transaction was terminated because the number of rows determined by the transaction size (SET parameter) was inserted.
- b) The current row was incorrect and could not be inserted.

```

LOAD ...   Execute           Load/Update/Extract           001-018
-----
                                A D A B A S
DATALOAD TABLE customer
  cno          01-05
  firstname    09-18  NULL IF POS 6 <> ' '
  name         20-29
  zip          57-61 city 32-51
INFILE cmaster.data
-----
                                <serverdb> : <user>
1=Help 2=Reset 3=End 4=Print 5=Run 6=Next 7=Pick 8=Put =Prot
customer                                : inserted 820, rejected 7 lines
==>

```

Once the end of the source file is reached, the counter readings are accumulated. In the case of a DATALOAD statement, the total number of processed and rejected rows is displayed on the screen. The sum of the two figures corresponds to the number of executed INSERT statements. The displayed message has, e.g., the following format:

```
Sum of inserted lines : 5241, sum of rejected lines : 9
```

In the case of DATAUPDATE statements, it is always the number of *modified* rows which is displayed. The accumulated message has the format:

```
Sum of updates : 9822, sum of invalid lines : 3
```

The respective message is also written into the protocol file.

13.3 Status Messages During Data Extraction

The DATAEXTRACT statement is not executed when LOAD detects that significant digits could be lost during the output of numerical values because the output fields are not wide enough.

Nevertheless, should an overflow occur (e.g., in case of output made in INTEGER format), then asterisks (*) are output instead and the extract run is aborted. During the extraction, a counter keeps track of the number of rows written.

After a certain number of rows has been output (depending on the lengths of the result rows), LOAD displays the current counter reading:

```
Rows written :      40
```

Once the result table has been completely processed, the final counter reading is written into the protocol file and displayed:

```
Rows written :   5241
```

13.4 Aborting a Run

If a serious database error occurs, such as LOG FULL, DATABASE FULL, the load run is implicitly aborted and those counter readings are recorded which represent the state of the transaction most recently committed.

If the run has been aborted implicitly, LOAD informs the user on the screen about the cause of the error and then ends the current LOAD session.

The recorded counter readings allow an aborted run to be continued (after having removed the cause of the error, if any) with

```
==> RUN FROM n
```

where n is the number of the first row that has not yet been processed. Therefore, interrupted load runs need not be completely repeated.

With various options of the RUN command, the user himself can determine when the run is to be terminated. If a load run is started, e.g., as follows

```
===> RUN FROM 1 FOR 100
```

the first 100 rows of the input file are loaded and then the run is terminated, independent of the number of inserted and rejected rows.

Interactive load runs are interrupted when a certain number of rows have been rejected. A special screen is displayed with several options, one to terminate the run. The number of inserted rows is not taken into consideration when deciding to interrupt or not.

13.5 Transactions

Every LOAD session is operated with the lock mode NORMAL (see the Reference manual); i.e., any locks that may be required are set implicitly.

Load Statements:

In the case of DATALOAD and DATAUPDATE, LOAD concludes the current transaction after a certain number of rows has been inserted or updated; then it opens a new transaction.

The transaction size currently set (default is 100 rows per transaction) can be displayed and modified using the SET command.

Extract Statements:

For a syntax check, a SELECT statement is created from the DATAEXTRACT statement and executed. If result sets were generated, the current transaction is concluded and a new one started. If DATAEXTRACT WITH LOCK has been specified, all tables from which selections are to be made are read-locked.

During extraction, the table rows are read from the unnamed result table which was created by the SELECT statement. If the locks are no longer required (at the latest, at the end of the extract run), they are released by concluding the transaction.

In AUTOCOMMIT mode, each SQL statement is implicitly concluded with COMMIT. With the AUTOCOMMIT mode, an interactive user is prevented from unvoluntarily holding locks, thus hindering other users. In command files, this mode can be disabled by using AUTOCOMMIT OFF if a group of SQL statements is to be executed within a transaction and rolled back, if necessary.

14The LOAD Session Log

At the beginning of a LOAD session, a file is created or updated into which the following is logged:

- * database and user names
- * points in time of starting and ending the file
- * executed command file
- * all executed SQL statements
- * all executed load statements
- * all input records rejected for each load statement, with information about the cause of the error
- * the accumulated counter readings for each load statement (inserted/updated and rejected rows)
- * all executed extract statements, (with a description of the output file format, if this is not stored in the target file)
- * a message, if the run was aborted.

The name of the session log file (PROT) can be set with the SET command. For the default setting, refer to the platform-specific User Manual. The log file is used with APPEND; i.e., it is not deleted with each LOAD call.

The log file is structured in such a way that parts of it can easily be used to generate a command file. In a new record, all comments begin with '/' *' or '/' <letter>'. Thus they will be ignored by a RUN <file> command. All message or error text lines in the log file are indicated by a letter at the beginning of the line.

- '/ M' informative system messages, such as the number of inserted and rejected rows, last committed transaction etc.
- '/ E' lines with error messages. It indicates, e.g., the two lines which, in case of rejected rows, contain a table and position specification, the error number, or an error text.
- '/ R' rows with error messages resulting from an IF \$SRC (SQL statement) check. These error messages do not occur in the summary at the end of the log file.
- '/ X' rejected rows with binary data. These rows are recorded in hexadecimal representation in blocks of 20 bytes each (i.e., of 40 hexadecimal digits).
- '/ C' comment lines of the executed command file which were inserted into the log file. These are all lines beginning with '/' or '*' that are longer than three characters.
- '/ S' the summary of all errors that occurred at the end of the load run.

At the point in time of error detection, data records rejected in block statements (e.g., DATALOAD, FASTLOAD) are no longer available in the memory of LOAD. To be able to record more than the line number, LOAD converts the data from the available internal format into a compressed plaintext representation. In this case, the error position indicates the number of the column the input value of which is invalid. To reload such data records from the log file, the load statement must exclude the comment lines by a table condition IF POS 1 <> '/', and it must contain the file options NULL 'NULL' and SEPARATOR '|'.

Example of the log file contents:

```

/ M TESTUSER          on XDBDIST1
/ *
/ M START 28.02.1995          17.44.19
/ *
run item.load
/ *
IF $RC (DELETE FROM ITEM) = -4004
/ *
/ R -4004 UNKNOWN TABLE NAME:ITEM
/ *
CREATE TABLE item
( itno          char (8) key,
  descr         char (20),
  stock         fixed (10),
  min_stock     fixed (5),
  orderdate     fixed (10),
  delivdate     date,
  price         fixed (6,2),
  weight        float (5) )
/ *
DATALOAD TABLE item
  itno          01-08  CHAR
  descr         09-28  CHAR          DEFAULT NULL
  stock         29-36  INTEGER HEX  DEFAULT NULL
  min_stock     37-40  INTEGER HEX  DEFAULT NULL
  orderdate     41-48  INTEGER HEX  DEFAULT NULL
  delivdate     49-58  CHAR          DEFAULT NULL
  price         59-67  CHAR          DEFAULT NULL
  weight        68-78  CHAR          DEFAULT NULL
INFILE load.in
DATE 'EUR'
DEC '/ / ./'
NULL '?'
/ *
"01785523"|"hammer"|10 000|1 000|5 000|30.11.1989|11.95|5.0000E-01
/ E Table      "ITEM"          "
/ E      200 DUPLICATE KEY
"01785524"|"chisel"|8 100| 1 000|5 000|30.11.1989|19.90|1.5000E+00
/ E Table      "ITEM"          "
/ E      200 DUPLICATE KEY
"01785225"|"screw driver"|795|300|500|07.08.1990|2.95|2.0000E-02
/ E Table      "ITEM"          "
/ E      200 DUPLICATE KEY
"01785526"|"drilling machine"|600|10|NULL|NULL|198.95|3.2700E+00

```

```
/ E Table      "ITEM      "  
/ E      200 DUPLICATE KEY  
"01785527"|"screw"|10 000|5 000|3 000|05.02.1992|0.99|5.0000E-03
```

```

/ E Table      "ITEM      "
/ E      200 DUPLICATE KEY
"01785828"|"dowel"|NULL|NULL|NULL|05.02.1992|0.99|NULL
/ E Table      "ITEM      "
/ E      200 DUPLICATE KEY
/ M Last transaction committed at input line 16
/ *
/ M ITEM                      : inserted 10, rejected 6 lines
/ *
DATAUPDATE TABLE item
      itno      01-08
      descr     10-24
SET stock      25-31
SET orderdate  32-38
SET delivdate  40-49
SET price      51-56
INFILE *
DEC '/././'
/ *
01785523 hammer          10.000  5.000 15.01.1991  13,95
/ E Table      "ITEM      " INFILE column      40
/ E -17411 Input string is too long
01785525 screw driver    795      500 15.01.1991   2,95
/ E Table      "ITEM      " INFILE column      40
/ E -17411 Input string is too long
/ M ITEM                      : 0 lines updated, 2 lines invalid
/ *
/ M Last transaction committed at input line 2
/ *
/ M ITEM                      : updated 0, rejected 2 lines
/ *
DATAEXTRACT * FROM item ; OUTFILE item.new
/ *
/ M RECORD-Size :      112
/ M OUTFILE-Format :
ITNO              1-8          CHAR
DESCR             12-31       CHAR
STOCK             35-48       CHAR
MIN_STOCK         52-58       CHAR
ORDERDATE         62-75       CHAR
DELIVDATE         79-86       CHAR
PRICE             90-98       CHAR
WEIGHT           102-112      CHAR
/ *
/ M Rows written : 257
/ *
/ S   6 x DBKERN          200 DUPLICATE KEY

```

```
/ S 2 x LOAD      17411 Input string is too long
/ *
/ M STOP 28.02.1995      17.46.02
```


15 Statements in a Command File

LOAD can start runs that process a series of statements from an external file.

A typical field of application comprises command files that install a database application.

The name of the command file currently open is displayed in the heading of the edit form.

15.1 Structure of a Command File

A command file can contain SQL statements and LOAD statements as well as input data for load statements. The statements must be separated from each other by separator lines.

The following general structure of a command file is recommended for performance reasons:

1. CREATE TABLE statements
2. DATALOAD statements
3. Other DDL statements such as CREATE INDEX

Separator lines begin with '/' or '*' in the first position.

After these symbols, the lines may contain comments which LOAD will ignore.

Example of a command file:

```

* *****
* Defines the two tables CUSTOMER and RESERVATION
* Loading tables from the file VYECRES.DATA
* Reading in updates from the file VUPCRES.DATA
*
CREATE TABLE reservation
( rno          FIXED (4)  KEY RANGE BETWEEN 1 AND 9999,
  cno          FIXED (4)  NOT NULL RANGE BETWEEN 1 AND 9999,
  hno          FIXED (4)  NOT NULL RANGE BETWEEN 1 AND 9999,
  arrival      DATE,
  departure    DATE,
  price        FIXED (6,2) RANGE BETWEEN 0 AND 1000)
*
CREATE TABLE customer
( cno          FIXED (4)  KEY RANGE BETWEEN 1 AND 9999,
  title        CHAR  (2)  RANGE IN ('Mr','Ms'),
  firstname    CHAR  (10),
  name         CHAR  (10) NOT NULL,
  city         CHAR  (20) NOT NULL,
  state        CHAR  (2)  NOT NULL,
  zip          FIXED (5)  NOT NULL RANGE BETWEEN 1 AND
99999,
  account      FIXED (7,2) RANGE BETWEEN -10000 AND 10000)
*
DATALOAD TABLE customer  IF POS 1-2 = 'cs'
  cno          4-8
  title        10-12  NULL IF POS 16-18 = '---'
  name         29-40
  city         42-61
  state        62-63
  zip          64-68
  account      70-78
DATALOAD TABLE reservation IF POS 1-2 = 'rs'
  rno          4-8
  cno          10-14
  hno          16-19
  arrival      42-49
  departure    51-58  NULL IF POS 51-53 = '---'
  price        60-68
INFILE vyecures.data
*
DATAUPDATE TABLE reservation IF POS 1-2 = 'rs'
  KEY    rno          4-8
  SET    hno          25-28

```

```
      SET      departure 51-58  NULL IF POS 51-53 = '---'  
INFILE vupcures.data  
*****
```

15.2 SQL Statements in a Command File

Although LOAD permits (nearly) all SQL statements in a command file, it is best suited for the execution of commands concerning the database administration (e.g., CREATE, DROP, ALTER).

Queries to the database or to the database catalog (SELECT, FETCH, and SHOW) usually make no sense in a command file, because LOAD does not display any results apart from a status message.

COMMIT and ROLLBACK statements may be placed within a command file to explicitly conclude or reset transactions. These statements, however, only have an effect if they are executed in AUTOCOMMIT OFF mode.

Usually, LOAD terminates the current transaction implicitly (AUTOCOMMIT) at the end of command execution. Otherwise, there could be the risk that an interactive user unintentionally holds locks, thus hindering other users.

15.3 Input Data in the Command File

If a load statement in a command file specifies INFILE *, data is also read from the command file. This data can be put directly after the load statement or be separated from it by a separator line (/ or * in the first position).

During interactive execution, the data, together with the statement, is copied to the screen if the input data in the command file is placed directly after the load statement *without* being separated from it by a *separator line*. In this case, the size of the edit form must be taken into consideration because it possibly does not allow for a complete representation of the data.

```
* *****  
* Example of unseparated input data  
* *****
```

```
DATALOAD TABLE customer
```

```

      cno      1-4
      name     6-15
      city     17-36
      state    37-39
      zip      40-44
INFILE *
1001 JULIE  ANDREWS      SANTA CLARA  CA95094
1002 WARREN BEATTY      DALLAS    TX75243
1003 BURT   LANCASTER   HOLLYWOOD CA90029

```

If the input data in the command file is *separated* from the load statement by a separator line, the data is not transferred to the screen during execution. This form should be chosen if the data contains non-representable characters or cannot be represented completely:

```

* *****
* Example of separated input data
* *****

DATALOAD TABLE regular_customer
      cno      1-4
      firstname 6-15
      name     19-28
      requests 34-1000
INFILE *
*****
1001 JULIE  ANDREWS      ...
1002 WARREN BEATTY      ...
1003 BURT   LANCASTER   ...

```

A DATALOAD or DATAUPDATE statement referring to separated input data from the command file *cannot* be executed more than once with RUN, even if it is still displayed in the input area.

If a load statement is aborted before its normal completion, all records in the command file with input data for this statement are skipped.

15.4 Control Statements in the Command File

The execution of statements in a command file can be controlled with conditions. A range of control statements serves this purpose. The control statements described below are written into the command file together with the executable statements.

To ensure a correct processing, the control statements must be clearly separated from the executable statements (DATALOAD etc.); a separator line (/** or something else) has to be included as soon as the type of statement changes. In principle, every executable statement must be fetched to the screen by itself; this aim can be achieved by comment lines included before and after the statement concerned.

IF-THEN-ELSE

Conditions can check the result of a statement or define a constant branching.

In the first case, the condition is introduced by IF \$SRC followed by one of the comparison operators =, <, > etc. and an integral number. The value inserted for \$SRC is the return code of an SQL statement or the number of rows rejected during a load run.

A variant of this request form prevents a load run from being interrupted, because an SQL statement produced a negative return code. The statement to be executed is specified after \$SRC enclosed in parentheses; then follow the comparison operator and operand. In this case, \$SRC is a function; the condition evaluates the return code of the SQL statement enclosed in parentheses. The statement and the error message, if any, are recorded. These errors, however, are not listed in the summary at the end of the command file.

In case of constant branching, the condition is simply IF TRUE or IF FALSE. Such conditions can be used to exclude statements which are usually executed from particular applications.

It is mandatory that a condition preceded by IF be followed by a THEN branch, which can be followed by an ELSE branch. Each of these branches can contain a

single statement or a block of statements. A block of statements is a sequence of statements bracketed by BEGIN and END. Even a complete IF THEN ELSE structure will be accepted as a single statement.

Example:

```
SHOW TABLE customermaster
*
IF  $RC = 0
THEN BEGIN
*
    DATAUPDATE TABLE customermaster
        KEY cust_no    1-4 INTEGER
        SET STATUS     55
    INFILE customer.updates
*
    DATAEXTRACT * FROM customermaster;
    OUTFILE customer.list
    END
*
ELSE
    BEGIN
*
        CREATE TABLE customermaster
            ( cust_no FIXED (10),
              ...
              STATUS CHAR (1) BYTE)
*
        DATALOAD TABLE customermaster
            cust_no    1-4 INTEGER
            ...
        INFILE customer.list
*
        END
```

Command files containing control statements can be executed like any other command file. They are usually started with BATCH, but they can also be executed interactively.

The commands NEXT and SKIP take the control flow into account; i.e., only those statements are displayed or skipped which would have been executed with the option NOPROMPT.

If an error is detected in any nesting level during the execution of a statement, LOAD branches to the INPUT mode. The error can then be corrected, but it is also possible to execute any arbitrary command or statement. Commands that continue processing a command file, restart at the corresponding line in the control flow.

The command SCAN (abbreviated sc), on the other hand, fetches the next block of statements to the screen without distinguishing between control and other LOAD statements. None of these statements is automatically executed. The command SCAN can be used to manually skip statements that will not or cannot be executed.

In contrast to SKIP and NEXT, all statements can be reached using SCAN both the THEN part and the ELSE part will be displayed), and there are no side effects (IF \$RC (DELETE FROM ..) = 0). NEXT NOPROMPT can be used to execute the rest of the file automatically at any time. When doing so, a restart point appropriate for the control statements should be chosen.

As with SKIP, it is possible to use SCAN to pass the beginning of a word as parameter. In this case, the command file is scanned until the specified (sub) string is recognized in a part of the statement.

INCLUDE

A command file (primary file) may contain INCLUDE statements for further command files (secondary files). This helps to modularize blocks of statements and to clearly structure the command file.

The statement runs

```
INCLUDE filename [options]
```

The filename is passed to the operating system in exactly the same way it was entered by the user.

Up to five command files can be open at the same time; i.e., four nested INCLUDE statements are possible.

An INCLUDE statement opens and scans the specified command file before the next statement of the command file previously opened is executed.

For LOAD, an INCLUDE file is not a new command file in which statements such as RETURN and (SET) RETURN CODE have local effects. The statements are executed as if they were stored in the main command file. Therefore, INCLUDE is not a single statement and should be placed within a block enclosed by BEGIN and END to protect the control structure of the primary file. The optional specifications for code type and SQL mode, however, only refer to the INCLUDE file.

SAY

The SAY statement displays user-defined comments on the screen during a LOAD BATCH run. Thus the author of a command file can determine the places at which LOAD shall give a "sign of life" and the user need not consider and comprehend each single statement.

SAY can be used in interactive mode as well. In this case, the comment is output in one of the two system message lines.

The position indicators &U, &1, ..., &9 within the comments are replaced by the current parameters, if any. The variable \$RC can also be used to generate the return code of the last statement into the comment.

Example:

The statement

```
SAY The table &1 cannot be accessed (error $RC)
```

could display the message on the screen

```
The table CUSTOMER cannot be accessed (error -4004)
```

RETURN and STOP

Two special keywords can be used to terminate the execution of a command file at once: RETURN causes a command file to be closed and branches to the INPUT mode, STOP (with an optional return code specification for LOAD) causes the component to be quit. Thus RETURN in the command file is equivalent to END RUN in the command line, and STOP in the command file is equivalent to EXIT in the command line.

Setting the RETURNCODE

The return code for LOAD can also be set, e.g., according to \$RC, during command file execution. The statement

```
RETURNCODE <integer>
```

serves this purpose.

The return code can only assume values between 0 and 127.

If the command file does not contain any RETURNCODE statement, then LOAD is terminated with one of the following default codes:

```
7  <=> SQL error
8  <=> LOAD error
9  <=> Lines rejected by DATALOAD or DATAUPDATE
10 <=> File error in a statement
```

The value of the return code with which a LOAD session terminates depends on the error encountered last. Subsequent statements that execute successfully do not reset this code to 0.

The actual error message or the number of rejected rows, which can be checked as \$RC, appears in the log file of LOAD.

15.5 Parameters in the Command File

Position indicators &1 to &9 contained in command files can be replaced by any character string, if the corresponding parameters are specified with a preceding -p in the RUN or BATCH command.

The individual parameters are separated from each other by blanks. If a character string consisting of several words is inserted for &n, then the nth parameter must be enclosed in single quotation marks.

If a statement contains a character string &n and less than n parameters are specified with the call, then &n is replaced by an empty character string. This is not true if no parameters were specified at all.

16 Further LOAD Statements

The statements described in this chapter can either be used in a command file or be entered interactively in the edit form and executed.

16.1 AUTOCOMMIT OFF/ON

LOAD usually works in AUTOCOMMIT mode; i.e., each SQL statement entered by the user will be concluded with COMMIT, and statements such as DATALOAD always issue a COMMIT after a certain number of INSERT statements.

The control statement AUTOCOMMIT OFF prevents LOAD from concluding transactions. In this mode, SQL statements can be combined to form units that are committed or rolled back as a total. This mode is not only valid for SQL statements, but also for all LOAD statements except CATALOGEXTRACT/LOAD and TABLEEXTRACT/LOAD. The size of the log, however, determines the size of the datasets that may be loaded without a COMMIT.

The statement AUTOCOMMIT ON can be used to reestablish the normal state.

16.2 USE SERVERDB

The USE SERVERDB statement can be used to change to another database without leaving LOAD. Prerequisite is that a user with the same name and password exists on the new database. The database connection is not established in addition to, but instead of the current connection.

Syntax:

```
USE SERVERDB database name [ON servernode name]
```

Database name and servernode name are not automatically converted into uppercase characters.

If the ON option is omitted the servernode on which the user is working is assumed.

16.3 USE USER

The USE USER statement can be used to terminate the current database connection and to establish a new one with the specified parameters.

Syntax:

```
[USE] USER user password  
      [SERVERDB database name [ON servernode name]] [NOLOG]
```

The keyword USER must be followed by the name and password of the new user. These are converted into uppercase characters by the database system, unless they are enclosed in double quotation marks. Database name and servernode name are not automatically converted into uppercase characters. If the SERVERDB option is omitted, an attempt is made to connect to the database on which the user is working. If NOLOG is specified, then a NOLOG database session is opened.

USER &U establishes a database connection with the parameters that were used for the call of LOAD, thus renewing the initial session.

16.4 USE USERKEY

A variant of the USER statement is the USERKEY statement for which an XUSER entry is used to provide any required information about the new user.

Syntax:

```
USE USERKEY key name
```

The name is not automatically converted into uppercase characters.

16.5 Setting the SQLMODE

The SQLMODE determines which SQL dialect will be understood by the database system. The mode name is specified with the CONNECT. Usually, this is ADABAS. LOAD, however, also considers deviating entries in the XUSER file and executes all commands entered by the user as well as all INSERTs and UPDATES generated from DATALOAD or DATAUPDATE statements in the defined mode. Command files are interpreted in the way determined by the mode.

The LOAD statement SQLMODE <mode name> can be used to change temporarily to another mode. <mode name> can assume one of the values ANSI, DB2, ORACLE, or ADABAS. The new mode is valid until it is changed by another SQLMODE statement.

Syntax:

```
SQLMODE { ADABAS | ANSI | DB2 | ORACLE }
```

Mode names other than the valid ones set the SQLMODE to the default ADABAS.

In ANSI mode, LOAD does not automatically conclude statements with COMMIT. If two consecutive dashes are found in a statement, the rest of the line is taken as a comment. Blank lines separate statements from each other.

In ORACLE mode, /* introduces a comment which is only terminated by */. A semicolon separates statements from each other.

16.6 USE TERMCHARSET

The USE TERMCHARSET statement can be used to terminate the current database connection and to establish a new one with the same user name, password, database name, and servernode name used so far, but with another of the TERMCHAR SETs defined using XCONTROL.

Syntax:

```
USE TERMCHARSET charsetname
```

The name is automatically converted into uppercase characters by the database system, unless it is enclosed in double quotation marks.

16.7 IGNORE TERMCHARSET

The IGNORE TERMCHARSET statement can be used to terminate the current database connection and to establish a new one with the same user name, password, database name, and servernode name used so far, but without a TERMCHAR SET.

Syntax:

```
IGNORE TERMCHARSET
```

16.8 MESSAGE ON

The control statement MESSAGE ON informs LOAD that the table with the system messages is available and the needed messages have been loaded. This statement is used in the installation command files and has the effect that, already during the current session, the detailed messages are displayed instead of the default messages.

17LOAD Commands

LOAD commands are entered on the screen in the command line marked with '==>'.

Keywords can be abbreviated arbitrarily as long as they remain unique; four characters will be checked at the most.

Without options, most commands can also be called using *keys*. LOAD displays the current settings of these function keys at the bottom of the screen.

17.1RUN Without Command File

RUN without argument starts the statement currently displayed in the input area.

Call:

A screenshot of a command line interface showing the word "RUN" in blue text on a light gray background.

RUN

Annotations:

During execution, LOAD is in EXECUTE mode in which no input is accepted.

If LOAD detects an error when checking a statement or command, the position in which the error occurred is indicated on the screen. LOAD produces an error message and returns to the INPUT mode. Otherwise, a start message appears.

At the beginning of a load run, a source or target file is opened. In the case of source files, an error is displayed if the file does not exist or is already open. In the case of target files, an error is displayed if a file has already been opened or does not exist and cannot be created. An existing file with the name of the target file is

overwritten unless the option APPEND has been specified. APPEND determines that the data is to be written continuously to the end of the file.

If a great number of records is rejected within a transaction, the time in which tables may be locked without being accessed may be exceeded. As a result, the transaction is implicitly rolled back and the load run aborted. The number of the last inserted line of file is recorded in the protocol file.

At the end of the run, the source file or target file is implicitly closed.

If LOAD reacts to the RUN command with *'Re-connected after timeout, ...'*, the current session has been implicitly terminated because of prolonged inactivity, then reopened by the RUN command.

The RUN command can then be called again. However, the last transaction was rolled back and all result tables were closed at the implicit session end.

17.2 RUN with Range Option

With the following *RUN command options*, the user can specify that the run should begin with the nth input/result record and should be interrupted after m processed records:

Call:

```
RUN FROM n [FOR m]
```

Examples:

```
run from 100  
run from 100 for 200
```

Annotations:

DATALOAD or DATAUPDATE only begins with the nth record of the source file; DATAEXTRACT begins with the nth result row.

With m, you can specify the number of *read input row* after which LOAD is to interrupt the load run. (Default: all rows up to the end of the input file can also be specified explicitly with *). Counting starts with 1 if the record denoted by n has been read.

The source file and the command file, if any, are closed when a run is aborted.

If 0 is specified for one or both of the parameters, only the syntax of the statement is checked for errors.

17.3 RUN with STOP Option

The STOP option of the RUN statement can be used to interrupt the execution of a DATALOAD or DATAUPDATE statement after a given number of *rejected records*.

Call:

```
RUN [FROM n [FOR m]] STOP k
```

When a run is interrupted, a special screen is displayed in which three or four function keys are set by LOAD according to the phase of execution:

Quit (**F3**)

cancels the run

Go On (**F5**)

concludes the run without interruption

Test (**F6**)

interrupts the run once more when the number of rows rejected corresponds again to the 'stop option' specification

Prot (**F9**)

displays the online protocol file. This key is only set when the protocol file contains an entry.

Example 1:

```
run stop 10
```

In this example, the run is interrupted after 10 records have been rejected.

Example 2:

```
run from 100 for 300 stop 1
```

The second example shows how the STOP option can be combined with the range option: the load run only starts with the 100th record of the file and is interrupted after 300 records have been processed. If one record is rejected, the run is interrupted.

A load run is interrupted, even without an explicit specification of the STOP option, when the number of records indicated by the current transaction size has been rejected. This is not valid for BATCH runs.

17.4 Displaying the LOAD Log File (PROT)

PROT can be used to display the current section of the session log file.

Call:

```
PROT
```

Annotations:

PROT can be called in INPUT mode or when a DATALOAD or DATAUPDATE statement has been interrupted.

Only those records are displayed which have been *rejected* during the execution of the DATALOAD or DATAUPDATE statement.

The section only comprises as much as can be displayed on the screen.

Any new DATALOAD, DATAUPDATE, or DATAEXTRACT statement deletes this online protocol file.

17.5 Starting a Command File with RUN

The RUN command also starts command files in interactive mode. LOAD opens the specified command file, transfers the statements sequentially from file to screen, and executes them. The name of the current command file is displayed in the heading of the input screen.

Call:

```
RUN filename [code_option] [interrupt_option] [ [-p] parameters]

code_option:      ASCII  or EBCDIC
interrupt_option: PROMPT or NOPROMPT
parameters:      up to nine arbitrary character strings
```

Annotations:

Chapter 15 describes how the command file must be structured so that it can be processed by LOAD.

Before being executed, each statement of the command file is copied to the screen. Input data in the command file is also copied as long as it is not separated from the relevant statement by a separator line (* or /).

When copying, each row is searched for the character strings &1 to &9, which will be replaced with the first to ninth parameters. The default is the empty character string.

If no current parameters have been specified, the execution of the command file is cancelled as soon as a position indicator is found.

If the file coding differs from the machine-specific code, the available code must be specified after the file name so that the lines of the file can be converted before being transferred to the screen.

If NOPROMPT is specified, the statements of the command file are executed in the specified order without any interaction with the user. NOPROMPT is the default setting.

LOAD implicitly switches from NOPROMPT to PROMPT if an error must be reported.

PROMPT puts LOAD in SCAN mode; control statements are not interpreted. LOAD copies one statement (LOAD, SQL, or control statement; or data) to the screen and waits for the user to

- explicitly start the statement with RUN;
- skip the statement with NEXT or SCAN; or
- enter any other command.

If a statement has been started explicitly, it remains on the screen after execution. LOAD then waits for the user to

- fetch the next statement;
- start the same statement again; or
- enter any other command.

All SQL statements executed successfully are stored in the protocol file. Load statements are written to the protocol file as usual (see chapter 14 "The LOAD Session Log").

The command file currently open is only closed when

- it has been completely processed,
- END RUN (command line) or RETURN (control statement) is executed,
- a new command file is started,
- the current session is ended.

17.6 Fetching the Next Statement (NEXT)

NEXT can be used to copy the next statement from the command file currently open to the screen.

Call:

```
NEXT [run_option PROMPT or NOPROMPT]
```

Annotations:

Before copying the statement, the input form is cleared.

With PROMPT (default setting), LOAD returns the INPUT screen after having copied the statement and waits for the user to

- explicitly start the statement with RUN;
- skip the statement with NEXT or SCAN; or
- enter any other command.

Before starting a statement explicitly, any modifications can be made to it. After execution, it remains on the screen. LOAD then waits for the user to

- fetch the next statement;
- start the same statement again; or
- enter any other command.

NOPROMPT executes the remaining command file statements without any further interaction with the user. NOPROMPT must be specified explicitly.

17.7 Displaying All Statements (SCAN)

SCAN can be used to display a statement of the currently open command file without interpreting it. Thus, the control statements can also be displayed.

Call:

```
SCAN <literal>
```

Annotations:

If SCAN is issued without parameters, LOAD displays the next statement of the command file. Everything up to the next separator line is taken as statement.

If a character string is specified as a parameter, then the command file is searched for the next statement that contains this character string. During this operation, lowercase letters are converted into uppercase. The word must be specified without single quotation marks.

In contrast to NEXT and SKIP, values missing for &1 to &9 will be ignored when running through a file with SCAN.

17.8 Skipping Statements (SKIP)

SKIP ignores the next n statements of the command file currently open and then fetches the next command to the screen.

There are two forms of the SKIP command: one specifies how many statements are to be ignored; the other searches a particular keyword.

Call:

```
SKIP [ n ]      or      SKIP <literal>
```

Annotations:

If SKIP is specified without parameters, LOAD ignores the next command file statement and then acts like NEXT PROMPT.

If a number n is specified, LOAD ignores the next n statements of the command file and then acts like NEXT PROMPT.

If a LOAD or SQL keyword is specified as a parameter, the command file is searched for the next statement containing this keyword. During this operation, lowercase letters are converted into uppercase. The keyword must be specified without single quotation marks. It is sufficient for the search to specify the start of a word.

17.9 Closing the Command File

END RUN can be used to close a command file before it has been completely scanned.

Call:

```
END RUN
```

Annotations:

END RUN has the same effect as a RETURN statement specified in the command file itself.

17.10 Leaving LOAD (EXIT)

Entering EXIT on the command line terminates LOAD immediately and returns control to the operating system.

Call:

```
EXIT
```

Annotations:

In addition to the EXIT command, the command END, which is usually called with the corresponding function key, returns control to the operating system only if the command is executed twice in succession. After the first execution, a corresponding message is displayed.

18 User-specific SET Parameters

The SET command provides the user with a form that contains a series of control parameters. LOAD produces a default setting for each of these parameters. Every user can modify these settings according to his own requirements. The new values remain valid beyond a session's end.

After issuing the command SET, the following form is displayed containing the default settings of the SET parameters:

```

LOAD ... Set
----- A D A B A S -----
Language          ENG
Null String       ?
Decimal           //./
Boolean           TRUE/FALSE
Date              INTERNAL
Time              INTERNAL
Timestamp         INTERNAL
Separator         STANDARD
Print Format       DEFAULT
Number of Copies  1
System Editor     vi
LOAD Presentation  DEFAULT
LOAD Protocol File load.prot
Transaction Size  100
----- <serverdb> : <user> -----
          3=Quit 4=Default 5=Save 10=Printer 11=Presen
Overwrite for new values and press function key

```

The displayed values of the SET parameters can be modified by overwriting them. Outside the input fields, the display form is write-protected.

The individual SET parameters have the following meanings:

1. *Language* defines the language for the output of the database and LOAD messages: ENG stands for English, DEU for German. A language can only be specified if messages are actually available for it.
2. *Null String* defines the character string for the representation of NULL values from the database. This string may have a maximum length of 20 characters.
3. *Boolean* defines the character strings for the representation of BOOLEAN values from the database. The character strings may have a maximum length of 10 characters. In case of <true>/<false>, <true> defines the character string for values that are true, and <false> defines the character string for values that are false.
4. *Decimal* defines the characters to be used for decimal numbers: in case of /<t>/<d>/, <t> defines the character for the thousands separator and <d> the character for the decimal sign; <t> may be omitted.
5. *Date* defines the format in which DATE column values are to be input and output. This format is valid for both LOAD commands and SQL statements.

The name of a standard format or a user-defined format can be specified. If a standard representation is chosen, it is automatically applied to *the three* date parameters. In SQL statements, user-defined formats are treated as INTERNAL.

Standard formats are

ISO	which corresponds to	YYYY-MM-DD,
USA	which corresponds to	MM/DD/YYYY,
EUR	which corresponds to	DD.MM.YYYY,
JIS	which corresponds to	YYYY-MM-DD,
INTERNAL	which corresponds to	YYYYMMDD.

where D stands for D(ay), M for M(onth), and Y for Y(ear).

If three positions are specified for the month, the name of the month will be output in its common abbreviation (Oct for October). User-defined formats need not contain each of the three symbols for the date portions.

6. *Time* defines the format in which TIME column values are to be input and output. This format is valid for both LOAD commands and SQL statements.

Standard formats are:

ISO	which corresponds to	HH.MM.SS,
USA	which corresponds to	HH:MM AM (PM) ,
EUR	which corresponds to	HH.MM.SS,
JIS	which corresponds to	HH:MM:SS,
INTERNAL	which corresponds to	HHHHMMSS.

where H stands for H(our), M for M(inute), and S for S(econd).

7. *Timestamp* defines the format in which TIMESTAMP column values are to be input and output. This format is valid for both LOAD commands and SQL statements.

Standard formats are:

ISO	which corresponds to	YYYY-MM-DD-HH.MM.SS.NNNNNN,
USA	which corresponds to	ISO,
EUR	which corresponds to	ISO,
JIS	which corresponds to	ISO,
INTERNAL	which corresponds to	YYYYMMDDHHMMSSNNNNNN.

where N stands for milliseconds and microseconds; the other letters have the same meaning as explained for *date* and *time*.

8. *Separator* defines the character string which is used to separate result table columns from each other. If this string is to contain blanks at its end, it must be enclosed in single quotation marks. The string may have a maximum length of 20 characters. The default value 'STANDARD' corresponds to the string '|'. STANDARD column separators are displayed on the screen as drawn vertical lines if the monitor is capable of representing semigraphics.
9. *Print Format* defines the format of the printout. Here the user can specify either a print format provided with the installation or a user-defined print

format. Up to eight print formats can be defined - see the description of the Printer Key at the end of this chapter.

10. *Number of Copies* specifies how many copies are to be printed.
11. For *System Editor*, the user can define an editor of his choice. This editor will be called with the command SYSED.
12. *LOAD Presentation* allows a user to specify a presentation for his personal use in LOAD. The presentation name denotes a certain setting of screen colors and attributes. This setting can be modified, enabling the user to adapt LOAD to his own liking.

With the installation, various presentations are provided which are immediately available to every user. These presentations can be paged through or redefined. Up to eight presentations can be defined - see the description of the Presen Key at the end of this chapter.

13. The structure of the name of the *LOAD Protocol File* depends on the operating system. If the name is changed, LOAD closes the old file and opens a protocol file with the new name for the execution of the next statement. The character combinations &U, &D, &P, &T, and &N are position indicators for user name, SERVERDB name, process-id, terminal-id, and terminal number. Within the names, they can be specified at any place. They are meant to separate the protocol files of LOAD applications that are performed simultaneously.
14. *Transaction Size* indicates the number of rows after the processing of which the current transaction is concluded and the current counter readings are displayed.

The key accepts the newly entered values and leaves the SET mode.

The key leaves the SET mode without the modifications becoming effective.

The key sets all displayed parameters to predefined default values. These must not be identical with the values displayed after the first call of LOAD, because the system administrator is allowed to choose other default settings, and

these will be displayed for any users who have not yet defined a parameter set of their own.

The keys **Printer** and **Presen** branch to further forms and are described in the following.

18.1 The Printer Key

The **Printer** key switches from SET mode to a submenu where you can define the print formats.

```

LOAD ... Set
----- A D A B A S -----
Printformat Name          DEFAULT
Printer                   1p
Page Width                80
Page Length               68
Left Margin               10
Right Margin              5
Top Margin                5
Bottom Margin             5
New Page                  OFF
----- <serverdb> : <user> -----
3=Quit  4=Default  5=Save  6=Delete  9=Copy
More entries via up/down

```

Initially, the currently set print format is displayed. If more formats are defined, a message tells you about it. You can switch from one format to the other using the scroll keys.

The settings can be modified by overwriting the entries.

The following settings can be defined in such a format:

1. *Printformat Name* displays name given to the defined format.
2. *Printer* specifies the desired printer. The printer depends on the installation.
3. *Page Width* defines the width of a print page. The value may be a maximum of 254.
4. *Page Length* defines the complete length of a print page in number of lines.
5. *Left* and *Right Margin* define the number of blanks to be output to the left and to the right of the text.
6. *Top* and *Bottom Margin* define the number of blank lines to be output above and below the text.
7. *New Page* defines whether (ON) or not (OFF) a form feed is to be performed for each separate print job.

The keys , , and have the same meanings as in the superior SET form. If the user returns to the first form by using , the last displayed format becomes the current format, i.e., its name is displayed for *Print Format*.

Defined formats can be deleted with the key.

The key generates a new entry without a format name. The other parameters are set to the values of the previously displayed setting and can be modified at will.

18.2 The Presen Key

The Presen(tation) key switches from SET mode to a menu where you can define the presentations.

LOAD ... Set		
	A D A B A S	
Presentation Name	DEFAULT	
Text normal	ATTR1	()
Text enhanced	ATTR2	()
Title	ATTR3	()
State	ATTR4	()
Info Message	ATTR5	()
Error Message	ATTR6	()
Graphic	ATTR7	()
----- <serverdb> : <user> -----		
2=Mark 3=Quit 4=Default ... 10=Backgr 11=Foregr 12=Attribute		
More entries via up/down		

Initially, the currently set presentation is displayed. A message tells you if more presentations are defined. You can switch from one presentation to the other using the scroll keys.

In such a presentation, the different physical properties are assigned to the sixteen logical attribute names. LOAD only uses the first seven logical attribute names. Each logical attribute name (ATTR1 to ATTR16) is displayed in the menu, together with the attributes and colors assigned to it.

The kinds of representation and color available depend on the installation and the system used. If colors cannot be set, the keys **Backgr** and **Foregr** are not displayed.

To change such an assignment, mark one or more attributes with an "x" and press the keys **Attribute**, **Foregr** or **Backgr**. Popup menus appear where the desired settings for the color and kind of representation can be chosen by checking them with an "x".

The toggle switch is used to mark all attributes with an "x". If all attributes are already marked with an "x", this key removes them instead.

The keys , , , , and have the same functions as in the other SET forms.

19LOAD in BATCH Mode

For a BATCH call of LOAD, a command file with LOAD statements and SQL statements is processed without any user interaction on the screen.

A BATCH call makes it possible to call LOAD from a SHELL, which is processed in the background.

If a BATCH call is not used, the terminal is not available during execution, even if no screen is displayed.

The BATCH command

```
BATCH command file name [ code_option ] [ [-p] parameters ]  
code_option: ASCII or EBCDIC
```

is specified with the LOAD call in a format that is specific to the operating system.

Example:

```
load -b customer.inst      (UNIX)
```

Of course, a connection to the database must be set up in BATCH mode as well. The ways in which the required parameters can be provided are described in the platform-specific User Manuals.

If the coding of the command file differs from the machine-specific coding, the existing code must be specified after the filename so that the lines of file can be converted before being processed.

Using the option -p, parameters can be passed to LOAD which textually replace the &1 to &9 occurring in the command file. Blanks separate the individual

parameters. If a character string itself is to contain blanks, it must be enclosed in single quotation marks; these do not belong to the text.

In BATCH mode, the individual command file statements are executed sequentially without interruption. Each executed statement is recorded in the protocol file, together with the counter readings, if applicable.

For the execution of command files, there is a basic difference between interactive runs started with NOPROMPT and BATCH runs. In interactive mode, command file processing is interrupted as soon as an error has to be reported, whereas a BATCH job will be processed completely. A BATCH job will only be aborted before normal completion if a fatal error occurs that, in interactive operation, would force a session to be ended.

Incorrect statements are written to the protocol file together with the error messages. If the statement contains a syntax error, it is only copied up to the incorrect line, and the position of the error is marked with a \$ sign:

```
        DATALOAD TABLE customer
            name  001-012 CHAR
                $
/**
/** Invalid position specification
/**
```


20Appendix 1: Syntax of the Load Statements

The syntax notation used in this document is BNF (Backus-Naur-Form), with the following conventions:

```
clause ::= rule
```

The statements are made up of clauses. The rules describe how simple clauses are assembled into more complex ones.

```
clause1 clause2
```

The two clauses are written one after the other, separated by at least one blank.

```
[clause]
```

Optional clause: may be omitted without substitution.

```
| clause1 |  
    ...           or   clause1 | ... | clausen  
| clausen |
```

Alternative clauses: just one must be used.

```
clause  
    ...
```

The clause can be repeated as often as is desired.

For those clauses that are not further resolved, the SQL syntax is valid.

Statements to be entered in edit mode:

<code><fastload spec></code>	<code>::= FASTLOAD [<usage spec> TABLE <target spec> <load column spec>... <infile spec></code>
<code><usage spec></code>	<code>::= WITH <unsigned integer> % USAGE</code>
<code><dataload spec></code>	<code>::= DATALOAD TABLE <target spec> [<duplicates clause>] <load column spec>... <infile spec></code>
<code><target spec></code>	<code>::= <table name> [<if condition>]</code>
<code><if condition></code>	<code>::= IF <condition> OTHERWISE</code>
<code><duplicates clause></code>	<code>::= IGNORE DUPLICATES REJECT DUPLICATES UPDATE DUPLICATES</code>

<code><load column spec></code>	<code>::= <column descriptor> [<null condition>] <column assignment></code>
<code><column descriptor></code>	<code>::= <column name> <field spec> <format spec></code>
<code><null condition></code>	<code>::= NULL [IF] <condition> DEFAULT NULL</code>
<code><column assignment></code>	<code>::= <column name> '<literal>' <column name> <generate spec></code>
<code><generate spec></code>	<code>::= <ADABAS value> <ansi value> <db2 value> <oracle value> <sequence number></code>

<adabas value>	::= USER USERGROUP STAMP DATE TIME
<db2 value>	::= USER CURRENT SQLID CURRENT DATE CURRENT TIME CURRENT TIMESTAMP
<oracle value>	::= USER SYSDATE UID
<sequence number>	::= SEQNO [<start> [<increment>]]
<start>	::= [signed] integer
<increment>	::= [signed] integer

<dataupdate spec>	::= DATAUPDATE TABLE <target spec> <access column spec>... <set column spec>... <infile spec>
<access column spec>	::= <key column spec> <simple column spec>
<key column spec>	::= KEY <simple column spec>
<simple column spec>	::= <column descriptor> <column assignment>
<set column spec>	::= SET <load column spec>
<load column spec>	::= <column descriptor> [<null condition>] <column assignment>

<extract statement>	::= DATAEXTRACT [[WITH] LOCK]
---------------------	---------------------------------

	<pre> <select expression> ; [<output field spec>...] [<outfile spec>] <outfile spec> [<longfile spec> ...] </pre>
<select expression>	<pre> ::= SELECT statement of SQL without the keyword SELECT </pre>
<output field spec>	<pre> ::= <column id spec> <literal field spec> </pre>
<column id spec>	<pre> ::= <column id> <field spec> <format spec> [<null presentation>] </pre>
<column id>	<pre> ::= <column name> <column number> </pre>
<null presentation>	<pre> ::= [IF] NULL [SET] POS <field spec> = '<literal>' </pre>
<literal field spec>	<pre> ::= '<literal>' <field spec> </pre>
<longfile spec>	<pre> ::= LONGFILE <column id> <external file name> </pre>

<restore statement>	<pre> ::= DATAEXTRACT [[WITH] LOCK] <restore spec> TABLE <table name> [<order clause>] ; <external outfile spec> [; <external outfile spec>] </pre>
<restore spec>	<pre> ::= FOR DATALOAD FOR FASTLOAD FOR DATAUPDATE </pre>

<format spec>	<pre> ::= <format of numeric columns> [HEX] <options> [CHAR] [HEX] <format of oracle exp columns> </pre>
<format of numeric columns>	<pre> ::= [CHAR] [FLOAT] </pre>

```

| DECIMAL [ <fraction> ]
| ZONED   [ <fraction> ]
| INTEGER
| REAL

<options> ::= [ <scale spec> ]
              [ <round or trunc spec> ]
```

```

<scale spec>          ::= SCALE <scale factor>

<round or trunc spec> ::= ROUND <fraction>
                       | TRUNC <fraction>

<scale factor>        ::= <integer>
                       | ( <integer> )

<integer>             ::= <sign> <unsigned integer>
                       | <unsigned integer>

<fraction>           ::= <unsigned integer>
                       | ( <unsigned integer> )

```

```

<format of oracle exp columns> ::= DATE | NUMBER

```

```

<condition>          ::= <simple condition>
                       | ( <condition> )
                       | <condition> AND <condition>
                       | <condition> OR <condition>
                       | NOT <condition>

<simple condition>    ::= POS <field spec> [ <field format> ]
                       [HEX] <comp op> '<literal>'

<field spec>         ::= <start pos>
                       | <start pos> - <end pos>

<field format>       ::= [CHAR]
                       | DECIMAL [ <fraction> ]
                       | ZONED [ <fraction> ]
                       | INTEGER
                       | REAL

<comp op>            ::= < | <= | = | >= | > | <>

<start pos>          ::= <unsigned integer>

<end pos>            ::= <unsigned integer>

```

```

<infile spec>        ::= INFILE * <file format>

```

```
                | <external infile spec>  
<external infile spec> ::= INFILE <external file name>  
                           <file format>
```

```
<outfile spec>          ::=  OUTFILE * <file format>
                             | <external outfile spec>

<external outfile spec> ::=  OUTFILE <external file name>
                             <file format> [ APPEND ]
```

```
<file format>           ::=  [<code spec>]
                             [<number layout spec>]
                             [<date layout spec>]
                             [<time layout spec>]
                             [<timestamp layout spec>]
                             [<null representation spec>]
                             [<bool representation spec>]
                             [<intpres spec>]
                             [<compress spec>]
                             [<count spec>]
                             [<concatenate spec>]

<code spec>             ::=  ASCII
                             | EBCDIC

<number layout spec>    ::=  DEC '<number layout>'

<number layout>         ::=  / [<thousand char> ] / <decimal point> /

<date layout spec>      ::=  DATE '<date mask>'

<date mask>             ::=  <free format mask>
                             | <standard mask>

<standard mask>         ::=  INTERNAL
                             | EUR
                             | ISO
                             | JIS
                             | USA

<time layout spec>      ::=  TIME '<time mask>'

<time mask>             ::=  <free format mask>
```

| <standard mask>

<timestamp layout spec> ::= TIMESTAMP '<timestamp mask>'


```

<part spec> ::= TABLE <table name>
              | USER
              | ALL

```

```

<tableextract statement> ::= TABLEEXTRACT <part spec> [;]
                          <external outfile spec>

```

```

<tableunload statement> ::= TABLEUNLOAD <part spec> [;]
                          <external outfile spec>

```

```

<tableload statement> ::= TABLELOAD <part spec> [;]
                       <external infile spec> ;
                       <external outfile spec>

```

```

<part spec> ::= [TABLE] <table name>
              | USER
              | ALL

```

```

<dbextract statement> ::= DBEXTRACT
                       ; <external outfile spec>
                       ; <external outfile spec>

```

```

<dbload statement> ::= DBLOAD
                    ; <external infile spec>
                    ; <external infile spec>
                    ; <external outfile spec>

```

```

<load oracledb statement> ::= LOAD ORACLEDB
                             INFILE <oracle exp file>
                             INTEGER LOHI

```

```

<load oracle table> ::= DATALOAD TABLE <target spec>
                     <load column spec>...
                     INFILE <oracle exp file>
                     ORACLE
                     INTEGER LOHI

```

<other statement>	::= <change autocommit stmt> <change sqlmode stmt> <change user stmt> <change serverdb stmt> <change termcharset stmt> <long message stmt>
<change autocommit stmt>	::= AUTOCOMMIT OFF AUTOCOMMIT ON
<change sqlmode stmt>	::= SQLMODE ANSI SQLMODE DB2 SQLMODE ORACLE SQLMODE ADABAS
<change user stmt>	::= [USE] USER <username> <password> [SERVERDB <dbname> [ON <node>]] [USE] USER &U USE USERKEY <xuserkey>
<change serverdb stmt>	::= USE [SERVERDB] <dbname> [ON <node>]
<change termcharset stmt>	::= USE TERMCHARSET <charset name> IGNORE TERMCHARSET
<long message stmt>	::= MESSAGE ON

Statements which determine the control flow of a command file:

<control stmt>	::= <if stmt> <return stmt> <stop stmt> <set returncode stmt> <include controlfile stmt> <say stmt>
----------------	--

```
<if stmt> ::= <if part> <then-else part>
```

<if part>	::= IF TRUE IF FALSE IF \$RC [(<sql stmt>)] <comp op> <integer>
<comp op>	::= < <= = >= > <>
<then-else-part>	::= THEN <compound> [ELSE <compound>]
<compound>	::= BEGIN <stmt list> END <single stmt>
<stmt list>	::= <single stmt> [<stmt list>]
<single stmt>	::= <separator line> <statement> <separator line>
<statement>	::= <control stmt> <load stmt> <sql stmt>

<return stmt>	::= RETURN
<stop stmt>	::= STOP [<integer>]
<set returncode stmt>	::= RETURNCODE <unsigned integer>
<include controlfile stmt>	::= INCLUDE <external filename>
<say stmt>	::= SAY <comment>

Alternative Keywords:

DATALOAD	oder	LOAD DATA
DATAUPDATE	oder	UPDATE DATA

DATAEXTRACT

oder

EXTRACT DATA

TABLELOAD

oder

LOAD TABLE

TABLEUPDATE	oder	UPDATE TABLE
TABLEUNLOAD	oder	EXTRACT TABLE
DBLOAD	oder	LOAD DB
DBEXTRACT	oder	EXTRACT DB
CATALOGLOAD	oder	LOAD CATALOG
CATALOGEXTRACT	oder	EXTRACT CATALOG

Commands which may be specified in the command line or with the LOAD call:

<batch command>	::= BATCH <file spec> [<parameter>]
<parameter>	::= <token> <parameter> '<string>' <parameter>

<close controlfile command>	::= END RUN
-----------------------------	-------------

<end command>	::= END
---------------	---------

<exec command>	::= EXEC [ASYNC] <command>
<command>	::= operating system command

<exit command>	::= EXIT
----------------	----------

<help command>	::=	HELP
		HELP <helpid>

<helpid>	::=	BATCH
		END
		EXIT
		HELP
		NEXT
		PROT
		RUN
		RUN FILE
		SCAN
		SET
		SKIP

<next command>	::=	<next prompt>
		<next noprompt>

<next prompt>	::=	NEXT
		NEXT PROMPT

<next noprompt>	::=	NEXT NOPROMPT
-----------------	-----	---------------

<prot command>	::=	PROT
----------------	-----	------

<run command>	::=	<run command file>
		<start command>

<run command file>	::=	RUN <external file name>
		[<code spec>]
		[PROMPT NOPROMPT]
		[<parameter>]

<parameter>	::=	<token> <parameter>
		'<string>' <parameter>

```
<start command> ::= RUN <line ranges>
```

```
<line ranges> ::= FROM <count>  
                [FOR <lines>]  
                [STOP <count>]
```

```
<count> ::= <unsigned integer>
```

```
<lines> ::= <count> | *
```

```
<scan command> ::= SCAN [ <literal> ]
```

```
<set command> ::= SET
```

```
<skip command> ::= SKIP <number of statements>  
                | SKIP <keyword>
```

```
<number of statements> ::= <unsigned integer>  
                        | [ 1 ]
```


21Appendix 2: LOAD Error Messages

If LOAD is used to load the system messages, the text of an error message may not be available yet. The message and the explanatory text for it are contained in the manual ADABAS Messages and Codes. If the table with the messages does not exist when LOAD is called, default messages of the format

```
message number: <error number>
```

or

```
message number: <error number>    parameter: <message parameter>
```

are output. The first two digits of the error number allow the range to be determined which the message is associated with. LOAD messages lie in the range of 12000 and 12999. Within this range, the messages are ordered according to their contents. The range from 12900 to 12999 comprises status messages which usually do not require any user action.

The default text

```
LOAD < error number> message not available <parameter1> <parameter2>
```

is output when the messages are available in principle, i.e., the message table exists, but the particular message is not stored there, e.g., in the language set.

LOAD can end with the following return codes:

- 1 <=> Start of database machine required
- 2 <=> Restart of database system required
- 3 <=> Too many database users active
- 4 <=> username/password illegal

- 5 <=> Invalid call option
- 6 <=> No write access to PROT file

Default Return Codes

- 7 <=> Some SQL error has occurred
- 8 <=> Some LOAD error has occurred
- 9 <=> During a LOAD run lines have been rejected
- 10 <=> Some file error has occurred

The error messages 1 - 5 implicitly mean for BATCH runs that the run has not been started.

22Index

- A -

aborting a batch run.....	19-2
aborting a load run.....	13-5
AND, in condition, DATALOAD.....	3-11
APPEND, file option.....	17-1
ASCII, file option, DATAEXTRACT.....	6-3
ASCII, file option, DATALOAD.....	3-6
ASCII, file option, RUN.....	17-7
attribute.....	18-7
AUTOCOMMIT.....	16-1
AUTOCOMMIT OFF/ON.....	16-1

- B -

backing up the database, FASTLOAD.....	4-1
BATCH, description.....	19-1
BEGIN.....	15-6
BOOLEAN.....	18-2
BOOLEAN value, notation, DATALOAD.....	3-8
BOOLEAN, DATALOAD.....	3-15

- C -

call of LOAD, batch.....	19-1
call, HELP function, LOAD.....	2-6
CATALOGEXTRACT ALL, statement.....	8-4
CATALOGEXTRACT TABLE, statement.....	8-2
CATALOGEXTRACT USER, statement.....	8-3
CATALOGEXTRACT, syntax.....	20-7
CATALOGLOAD ALL, statement.....	8-7
CATALOGLOAD TABLE, statement.....	8-6
CATALOGLOAD USER, statement.....	8-6

CATALOGLOAD, syntax.....	20-7
checking the statement.....	13-1
code conversion, DATAEXTRACT.....	6-3
code conversion, DATALOAD.....	3-6
color setting.....	18-7
column assignment, DATAUPDATE.....	5-1
column description, DATAEXTRACT.....	6-5
column name, DATAEXTRACT.....	6-5
column name, DATAEXTRACT/DATALOAD.....	6-12
column name, DATAEXTRACT/DATAUPDATE.....	6-14
column type.....	12-1
column type, DATAEXTRACT.....	6-6
column type, DATALOAD.....	3-2
command file, batch.....	19-1
command file, close.....	17-12
command file, conditions.....	15-5
command file, DATAEXTRACT.....	6-11
command file, DATALOAD.....	15-4
command file, DATAUPDATE.....	15-4
command file, end at once.....	15-9
command file, example.....	15-2; 15-6
command file, generate.....	2-3
command file, input data.....	15-3
command file, interactive.....	17-7
command file, parameters.....	15-10
command file, SQL statements.....	15-3
command file, start.....	17-7
command file, structure.....	15-1
command timeout.....	17-3
comments, command file.....	15-1
COMMIT.....	15-3
comparison value, DATALOAD.....	3-12
COMPRESS, DATAEXTRACT.....	6-3
COMPRESS, DATAEXTRACT/DATALOAD.....	6-12
COMPRESS, DATALOAD.....	3-8; 3-21
conditions, command file.....	15-5
conditions, DATAEXTRACT.....	6-8
conditions, DATAEXTRACT/DATALOAD.....	6-13

conditions, DATALOAD.....	3-11
conditions, DATAUPDATE.....	5-2
conditions, keywords.....	15-5
constant, any, DATALOAD.....	3-14
constant, DATAEXTRACT.....	6-8
constant, DATALOAD.....	3-12
constant, DATAUPDATE.....	5-2
constant, format, DATAEXTRACT.....	6-9
constant, position, DATAEXTRACT.....	6-8
constant, special, DATALOAD.....	3-15
contents of database.....	9-1
continuing a load run.....	13-5
control statement, command file.....	15-5
control statements, syntax.....	20-9

- D -

data format, constant, DATAEXTRACT.....	6-9
data format, DATAEXTRACT.....	6-3; 6-6
data format, DATALOAD.....	3-2; 3-5
<i>data format, DATALOAD, example</i>	3-3
data format, DATAUPDATE.....	5-2
data format, numerical values.....	3-2; 5-2; 6-6
data formats, overview.....	12-1
database catalog.....	8-1
DATABASE FULL.....	13-5
database log.....	4-1
database query, DATAEXTRACT.....	6-2
DATAEXTRACT FOR DATALOAD.....	6-11
DATAEXTRACT FOR DATAUPDATE.....	6-13
DATAEXTRACT WITH LOCK.....	6-2; 6-11; 6-14
DATAEXTRACT, statement.....	6-1
DATAEXTRACT, syntax.....	20-3
DATAEXTRACT/DATALOAD, empty table.....	6-13
DATAEXTRACT/DATAUPDATE, empty table.....	6-15
DATALOAD, also DATAUPDATE.....	3-18
DATALOAD, default input.....	3-20

DATALOAD, several tables.....	3-17
DATALOAD, statement.....	3-1
DATALOAD, syntax.....	20-2
DATAUPDATE TABLE.....	5-1
DATAUPDATE, several tables.....	5-2
DATAUPDATE, statement.....	5-1
DATAUPDATE, syntax.....	20-3
DATE, DATAEXTRACT.....	6-3
DATE, DATALOAD.....	3-5; 3-15
DATE, format, SET.....	18-2
DATE, SET.....	18-2
DB function, migrate.....	8-1; 10-1
DB procedure, migrate.....	10-1
DB proeedure, migrate.....	8-1
DBEXTRACT, statement.....	10-1
DBLOAD, statement.....	10-2
DEC, DATAEXTRACT.....	6-3
DEC, DATALOAD.....	3-5
decimal representation.....	18-2
decimal representation, DATAEXTRACT.....	6-3
decimal representation, DATALOAD.....	3-5
DEFAULT NULL condition, DATAEXTRACT.....	6-12; 6-14
DEFAULT NULL condition, DATALOAD.....	3-13
DEFAULT NULL condition., DATALOAD.....	3-8
default return codes.....	21-2
default value, DATALOAD.....	3-1; 3-13
default value, DATAUPDATE.....	5-1
disk file, DATAEXTRACT.....	6-3
display, statement.....	17-10
DUPLICATES clause.....	3-18

- E -

EBCDIC, file option, DATAEXTRACT.....	6-3
EBCDIC, file option, DATALOAD.....	3-6
EBCDIC, file option, RUN.....	17-7
ELSE branch.....	15-5

END.....	15-6
END RUN.....	17-12
END RUN <-> RETURN.....	17-12
END, command.....	17-12
error message, batch run.....	19-2
error message, DATAEXTRACT.....	6-6; 6-15; 13-5
error message, DATALOAD.....	13-2
error message, DATAUPDATE.....	13-2
error messages, overview.....	21-1
EXIT, description.....	17-12
external file.....	15-1
external file, DATAEXTRACT.....	6-3; 6-4
external file, DATALOAD.....	3-2
extract.....	6-1
extract, contents of database.....	9-1
extract, control.....	17-3
extract, database catalog.....	8-1
extract, LONG columns.....	7-1

- F -

FASTLOAD.....	6-12
FASTLOAD, backing up the database.....	4-1
FASTLOAD, restrictions.....	4-1
FASTLOAD, statement.....	4-1
FASTLOAD, USAGE.....	4-3; 6-12
fetch, statement.....	17-9
field length, DATAEXTRACT.....	6-5
field length, DATALOAD.....	3-1
file option APPEND.....	6-3; 17-1
file option ASCII.....	3-6; 6-3
file option COMPRESS.....	3-8; 6-3
file option CONCATENATE.....	3-9
file option CONTINUEIF.....	3-9
file option COUNT.....	3-10
file option DATE.....	3-5; 6-3
file option DEC.....	3-5; 6-3

file option EBCDIC.....	3-6; 6-3
file option NULL.....	3-8; 6-12; 6-14
file option ORACLE.....	11-2
file option TIME.....	3-5; 6-3
file structure, DATAEXTRACT.....	6-4
file, external, DATAEXTRACT.....	6-3; 6-4
file, external, DATALOAD.....	3-2
format name.....	18-6
function keys.....	2-4

- G -

Go On key.....	17-4
----------------	------

- H -

HELP function, call, LOAD.....	2-6
hexadecimal representation, DATAEXTRACT.....	6-6
hexadecimal representation, DATALOAD.....	3-3
hexadecimal representation, DATAUPDATE.....	5-2

- I -

IF \$SRC.....	15-5
IF FALSE.....	15-5
IF THEN ELSE structure.....	15-5
IF TRUE.....	15-5
IF, DATAEXTRACT.....	6-8
IF, DATALOAD.....	3-11
IF, DATAUPDATE.....	5-2
INFILE *, command file.....	15-3
INFILE *, DATALOAD.....	3-2
INFILE *, file options.....	3-20
INFILE *, test input.....	3-20; 5-2
INFILE, CATALOGLOAD ALL.....	8-7

INFILE, CATALOGLOAD TABLE.....	8-6
INFILE, CATALOGLOAD USER.....	8-6
INFILE, DATALOAD.....	3-1
INFILE, DATAUPDATE.....	5-1
INFILE, DBLOAD.....	10-2
INFILE, fastload.....	4-1
INFILE, TABLELOAD ALL.....	9-7
INFILE, TABLELOAD TABLE.....	9-4
INFILE, TABLELOAD USER.....	9-6
input data, command file.....	15-3
input data, DATALOAD.....	15-4
input data, DATAUPDATE.....	15-4
input data, example.....	15-3
input field, DATALOAD.....	3-1
input file, DATALOAD.....	3-2
input screen.....	2-2
input screen, length.....	2-2
input screen, width.....	2-2
integer conversion, DATALOAD.....	3-7; 12-2
INTEGER, file option.....	12-2
INTEGER, file option, DATALOAD.....	3-7
interrupting a load run.....	13-6; 17-4

- K -

key column, DATALOAD.....	3-2
key column, DATAUPDATE.....	5-1
KEY, DATAEXTRACT/DATAUPDATE.....	6-13
KEY, DATALOAD.....	3-1; 3-13
keyword, notation.....	17-1

- L -

language.....	18-2
leaving LOAD.....	17-12
LOAD BATCH.....	19-1

LOAD commands, description.....	17-1
LOAD commands, notation.....	2-4
LOAD commands, overview.....	2-5
LOAD commands, syntax.....	20-11
load run.....	13-1
load, any constant.....	3-14
load, contents of database.....	9-1
load, control.....	17-3
load, database catalog.....	8-1
LOAD, in batch mode.....	19-1
LOAD, interactive.....	2-1
load, LONG columns.....	7-1
load, NULL value.....	3-12
load, several tables.....	3-17
load, special constant.....	3-15
loading.....	3-1
loading without logging.....	4-1
lock, DATAEXTRACT.....	6-2
lock, DATAEXTRACT WITH LOCK.....	13-7
lock, DATAEXTRACT/DATALOAD.....	6-11
lock, DATAEXTRACT/DATAUPDATE.....	6-14
lock, transaction concept.....	13-6
log file, description.....	14-1
log file, display.....	17-6
log file, example.....	14-3
LOG FULL.....	13-5
LONG columns.....	7-1

- M -

mandatory column, DATALOAD.....	3-2
margin, right, left, top, bottom.....	18-6
MESSAGE ON.....	16-6
messages, DATAEXTRACT.....	13-5
messages, DATALOAD.....	13-2
messages, DATAUPDATE.....	13-2

- N -

NEXT.....	15-6
NEXT, description.....	17-9
NOPROMPT.....	15-6
NOPROMPT, RUN.....	17-7
NORMAL, lock mode.....	13-6
NOT NULL, DATAEXTRACT.....	6-9
NOT NULL, DATALOAD.....	3-13
NOT, in condition, DATALOAD.....	3-11
notation, BOOLEAN value, DATALOAD.....	3-8
notation, column name.....	6-12; 6-14
notation, keyword.....	17-1
notation, LOAD commands.....	2-4
notation, NULL value, DATALOAD.....	3-8
notation, table name.....	6-12; 6-14
NULL condition, DATAEXTRACT.....	6-9
NULL condition, DATALOAD.....	3-12
NULL condition, DATAUPDATE.....	5-2
NULL IF, DATALOAD.....	3-12
NULL IF, DATAUPDATE.....	5-2
NULL value, DATAEXTRACT.....	6-8
NULL value, DATALOAD.....	3-12
NULL value, DATAUPDATE.....	5-2
NULL value, notation, DATALOAD.....	3-8
NULL value, representation.....	18-2
NULL, DATAEXTRACT.....	6-8
numbers, DATAEXTRACT.....	6-8
numbers, DATALOAD.....	3-6
numerical format, DATAEXTRACT.....	6-6; 6-9; 12-3
numerical format, DATALOAD.....	3-2; 12-2
numerical format, DATAUPDATE.....	5-2; 12-2
numerical formats, overview.....	12-1
numerical values, DATAEXTRACT.....	6-7
numerical values, DATALOAD.....	3-4

- O -

OR, in condition, DATALOAD.....	3-11
ORACLE, loading the database.....	11-1
OTHERWISE, DATALOAD.....	3-18
OUTFILE *, test output.....	6-15
OUTFILE, CATALOGEXTRACT ALL.....	8-4
OUTFILE, CATALOGEXTRACT TABLE.....	8-2
OUTFILE, CATALOGEXTRACT USER.....	8-3
OUTFILE, DATAEXTRACT.....	6-1
OUTFILE, DBEXTRACT.....	10-1
OUTFILE, TABLEEXTRACT.....	9-2
OUTFILE, TABLEEXTRACT ALL.....	9-3
OUTFILE, TABLEEXTRACT USER.....	9-3
OUTFILE, TABLELOAD ALL.....	9-7
OUTFILE, TABLELOAD USER.....	9-6
OUTFILE, two output files.....	6-12; 6-14
output field, DATAEXTRACT.....	6-5
output file, DATAEXTRACT.....	6-2

- P -

page length.....	18-6
page width.....	18-6
parameters, command file.....	15-10
plaintext representation, DATALOAD.....	3-2
plaintext representation, numbers.....	3-6; 3-14; 6-8
POS, DATAEXTRACT.....	6-8
POS, DATALOAD.....	3-11
POS, DATAUPDATE.....	5-2
position, DATAEXTRACT.....	6-5
position, DATALOAD.....	3-1
presentation.....	18-4
print format.....	18-4
print output.....	18-6
printer.....	18-6
PROMPT, RUN.....	17-7

PROT.....	17-6
Prot key.....	17-4
protocol file, SET.....	18-4
protocol, description.....	14-1
protocol, display.....	17-6

- Q -

Quit key.....	17-4
---------------	------

- R -

restrictions, LOAD.....	2-7
RETURN.....	15-9
RETURN <-> END RUN.....	15-9
return code.....	15-5
return codes, overview.....	21-2
RETURNCODE.....	15-9
ROLLBACK.....	15-3
ROUND, DATAUPDATE.....	5-2
RUN, description.....	17-1
RUN, file option ASCII.....	17-7
RUN, file option EBCDIC.....	17-7
RUN, NOPROMPT.....	17-7
RUN, PROMPT.....	17-7
RUN, STOP option.....	17-4
RUN, with range option.....	17-3

- S -

SAY.....	15-8
SCALE, DATAEXTRACT.....	6-7
SCALE, DATALOAD.....	3-4
SCALE, DATAUPDATE.....	5-2
SCAN.....	15-7

SCAN, description.....	17-10
SCAN, implicitly.....	17-8
screen, input.....	2-2
selection, DATALOAD.....	3-11
selection, DATAUPDATE.....	5-2
separator.....	18-3
separator lines, command file.....	15-1
SEQNO, DATALOAD.....	3-15
sequence number, DATAEXTRACT.....	6-5
sequence number, DATALOAD.....	3-15
SET.....	18-1
SET parameters, defaults.....	18-1
SET, DATAEXTRACT.....	6-8
SET, DATALOAD.....	3-1
SET, DATAUPDATE.....	5-1; 5-2
SKIP.....	15-6
SKIP, description.....	17-11
space, usage, DATAEXTRACT.....	6-12
space, usage, FASTLOAD.....	4-3
SQLMODE.....	16-4
STAMP, DATALOAD.....	3-15
STANDARD.....	18-3
start, command file.....	17-7
start, statement.....	17-1
STOP.....	15-9
STOP <-> EXIT.....	15-9
STOP, RUN.....	17-4
Syntax.....	20-1
SYSDBA.....	8-4; 8-7; 9-3; 9-7; 10-1; 10-2; 11-1
system editor, SYSED.....	18-4

- T -

table condition, DATALOAD.....	3-18
table name, DATAEXTRACT/DATALOAD.....	6-12
table name, DATAEXTRACT/DATAUPDATE.....	6-14
table, empty, DATAEXTRACT/DATALOAD.....	6-13

table, empty, DATAEXTRACT/DATAUPDATE.....	6-15
TABLEEXTRACT ALL, statement.....	9-3
TABLEEXTRACT USER, statement.....	9-3
TABLEEXTRACT, TABLE, statement.....	9-2
TABLELOAD ALL, statement.....	9-7
TABLELOAD TABLE, statement.....	9-4
TABLELOAD USER, statement.....	9-6
tables, loading.....	3-17
tables, updating.....	5-2
TABLEUNLOAD, statement.....	9-4
tape file, DATALOAD.....	3-2
TERMCHAR SET.....	16-5
TERMCHAR SET, select.....	16-6
terminating a load run.....	13-6
test input, on screen.....	3-20; 5-2
Test key.....	17-4
test output, on screen.....	6-15
testing, interactive.....	2-3
text constant, DATAEXTRACT.....	6-10
THEN branch.....	15-5
TIME, DATAEXTRACT.....	6-3
TIME, DATALOAD.....	3-5; 3-15
TIME, format, SET.....	18-3
TIME, SET.....	18-3
timeout.....	17-3
TIMESTAMP, DATALOAD.....	3-5; 3-15
TIMESTAMP, format, SET.....	18-3
TIMESTAMP, SET.....	18-3
transaction.....	13-6
transaction, DATALOAD.....	13-6
transaction, DATAUPDATE.....	13-6
transaction, explicit close.....	15-3
transaction, implicit close.....	13-6; 15-3
trigger, migrate.....	8-1; 10-1
TRUNC, DATAUPDATE.....	5-2

- U -

update, several tables.....	5-2
updating columns.....	5-1
updating rows.....	3-18
USAGE, FASTLOAD.....	4-3; 6-12
USER, change.....	16-2; 16-3; 16-4
USER, DATALOAD.....	3-15
USERGROUP, DATALOAD.....	3-15

- V -

value assignment, DATAEXTRACT.....	6-5
------------------------------------	-----

Table of Contents

1	Why Do You Need LOAD?.....
2	LOAD in Interactive Mode.....
2.1	The Input Screen.....
2.2	Interactive Testing.....
2.3	LOAD Commands and Function Keys.....
2.4	The HELP Function.....
2.5	LOAD Restrictions.....
3	Loading Database Tables with DATALOAD.....
3.1	The DATALOAD Statement.....
3.2	Data Types of File Fields.....
3.3	Dimension and Precision of Numerical Values.....
3.4	Format Specifications Related to a File and Other File Options.....
3.4.1	The Format Specifications DEC, DATE, TIME, and TIMESTAMP.....
3.4.2	The Format Specifications ASCII and EBCDIC.....
3.4.3	The Format Specification for the Representation of INTEGERS.....
3.4.4	Representation of NULL Values in the File.....
3.4.5	Representation of BOOLEAN Values in the File.....
3.4.6	COMPRESS for Non-tabular Data.....
3.4.7	CONCATENATE and CONTINUEIF for Multiple-line Data.....
3.4.8	Multi-volume Processing with COUNT Option.....
3.5	Selecting Records from the Source File.....
3.6	Inserting NULL Values.....
3.7	Loading Any Constants.....
3.8	Loading Special Constants.....
3.9	Loading Several Tables in a Single Run.....
3.10	Selection with OTHERWISE.....
3.11	Loading with DUPLICATES Clause.....
3.12	Input of Test Data on the Screen.....
3.13	DATALOAD with Input Made in Default Format.....
4	Loading Database Tables with FASTLOAD.....

4.1	The FASTLOAD Statement.....
4.2	Loading with the USAGE Option.....
5	Updating Table Columns with DATAUPDATE.....
6	Creating Dataextracts with DATAEXTRACT.....
6.1	The DATAEXTRACT Statement.....
6.2	Format Specifications for the Output File.....
6.3	Structure of the Target File.....
6.4	Data Formats in the Target File.....
6.5	Options for the Output of Numerical Columns.....
6.6	Output of NULL Values.....
6.7	Text Constants in the Target File.....
6.8	Generating Command Files with DATAEXTRACT.....
6.8.1	DATAEXTRACT FOR DATALOAD.....
6.8.2	DATAEXTRACT FOR DATAUPDATE.....
6.9	Test Output on Screen.....
7	Loading and Extracting LONG Columns.....
7.1	Loading LONG Columns with DATALOAD.....
7.2	Updating LONG Columns with DATAUPDATE.....
7.3	Extracting LONG Columns with DATAEXTRACT.....
8	Database Catalog Migration.....
8.1	CATALOGEXTRACT TABLE.....
8.2	CATALOGEXTRACT USER.....
8.3	CATALOGEXTRACT ALL.....
8.4	CATALOGLOAD TABLE.....
8.5	CATALOGLOAD USER.....
8.6	CATALOGLOAD ALL.....
9	Migration of the Database Contents.....
9.1	TABLEEXTRACT TABLE.....
9.2	TABLEEXTRACT USER.....
9.3	TABLEEXTRACT ALL.....
9.4	TABLEUNLOAD.....
9.5	TABLELOAD TABLE.....
9.6	TABLELOAD USER.....

9.7	TABLELOAD ALL.....
10	Migration of Catalog and Contents.....
10.1	DBEXTRACT.....
10.2	DBLOAD.....
11	The ORACLE Crossloader.....
11.1	Loading an ORACLE Database.....
11.2	Loading the Contents of an ORACLE Table.....
12	External Data Formats.....
13	The Loading Process.....
13.1	The Reasonableness Check.....
13.2	Status Messages During Loading.....
13.3	Status Messages During Data Extraction.....
13.4	Aborting a Run.....
13.5	Transactions.....
14	The LOAD Session Log.....
15	Statements in a Command File.....
15.1	Structure of a Command File.....
15.2	SQL Statements in a Command File.....
15.3	Input Data in the Command File.....
15.4	Control Statements in the Command File.....
15.4.1	IF-THEN-ELSE.....
15.4.2	INCLUDE.....
15.4.3	SAY.....
15.4.4	RETURN and STOP.....
15.4.5	Setting the RETURNCODE.....
15.5	Parameters in the Command File.....
16	Further LOAD Statements.....
16.1	AUTOCOMMIT OFF/ON.....
16.2	USE SERVERDB.....
16.3	USE USER.....
16.4	USE USERKEY.....

16.5	Setting the SQLMODE.....
16.6	USE TERMCHARSET.....
16.7	IGNORE TERMCHARSET.....
16.8	MESSAGE ON.....
17	LOAD Commands.....
17.1	RUN Without Command File.....
17.2	RUN with Range Option.....
17.3	RUN with STOP Option.....
17.4	Displaying the LOAD Log File (PROT).....
17.5	Starting a Command File with RUN.....
17.6	Fetching the Next Statement (NEXT).....
17.7	Displaying All Statements (SCAN).....
17.8	Skipping Statements (SKIP).....
17.9	Closing the Command File.....
17.10	Leaving LOAD (EXIT).....
18	User-specific SET Parameters.....
18.1	The Printer Key.....
18.2	The Presen Key.....
19	LOAD in BATCH Mode.....
20	Appendix 1: Syntax of the Load Statements.....
21	Appendix 2: LOAD Error Messages.....
22	Index.....