Adabas SQL Server
Reference Manual

Manual Order Number: ESQ143-030ALL

This document applies to Adabas SQL Server Version 1.4 and to all subsequent releases.

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

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

Documentation@softwareag.com

TABLE OF CONTENTS

PREFACE

Using This Manual – Some Basic Information

This manual describes the language elements of Adabas SQL Server. Adabas SQL Server is Software AG's implementation of the ANSI/ISO Standard SQL with certain enhancements to accommodate additional functionality.

It is intended for application programmers with a basic knowledge of the concepts and facilities of Standard SQL and Adabas as well as end-users who need help in formulating adhoc queries.

Chapter 1	describes in detail the common elements in the SQL syntax and their usage and limitations.
Chapter 2	describes each SQL statement with syntax diagrams in detail following a format shown below in the section **Statement Page Layout.**
Appendix A	is a glossary of Adabas SQL Server terms.

Other Helpful Manuals

Other manuals you may need are:

- Adabas SQL Server Programmer's Guide
- Adabas SQL Server Installation and Operations Manual (separate for each platform: UNIX, OpenVMS, MVS)
- Adabas SQL Server Messages and Codes
- a set of platform-specific Adabas Manuals including Adabas Messages and Codes
- ANSI/ISO Standards SQL (X3.135-1989, ISO 9075).

Statement Page Layout

The chapter **SQL Statements** and, where feasible, some parts of the chapter **Common Elements** have a uniform page design to enable easy access to the required information. Under the following headings relevant information will be found.

Sample Statement

Function:

This section gives a brief overview of a statement's function to enable the reader to quickly determine if the following information is of interest.

Invocation:

This section shows the possible modes in which a statement can be invoked. The applicable alternatives for each statement are marked. For details refer to the *Adabas SQL Server Programmer's Guide*, chapter **Introduction to Adabas SQL Server**, sections **Interactive/Embedded/Dynamic SQL.**

Syntax:

This section shows the syntax definitions for the SQL statement.

Syntax definitions are depicted graphically. Valid syntax constructions follow a path through the syntax graph. Alternation and iteration are indicated by branching and looping. Roman type (enclosed in oval boxes) indicates items that are to be entered literally and italic type (enclosed in rectangle boxes) indicates items to be supplied by the user. Delimiters are enclosed in circles.

Description:

This section gives a detailed description of the statement's purpose, functionality and application.

Limitations:

This section covers points which require special attention.

ANSI Specifics:

This section covers points of special interest not covered above and occurring particularly when operating in ANSI mode.

Adabas SQL Server Specifics:

This section covers points of special interest not covered above and occurring particularly when operating in Adabas SQL Server (default) mode.

Example:

This section is reserved for examples, whenever they are feasible and enhance the above information.

1

COMMON ELEMENTS

Character Set

As the most simple language elements, characters are used to construct higher-level language elements. Depending on the specific environment, Adabas SQL Server supports the EBCDIC/ASCII character set:

Letters	upper- and lowercase A – Z
Digits	the digits 0 – 9
Special characters	characters other than the above mentioned

Data Types

With Adabas SQL Server, data can be manipulated in many ways; the smallest unit of data is a value. A value can result from several different origins:

a column, a constant, a function, an expression, a host variable, a subquery

For data type definitions refer to the section **Table Column Element** later in this chapter. The interpretation of a value depends upon its data type. The following data types exist:

Adabas SQL Server Data Types	**General Data Types**
CHARACTER/CHAR	Fixed Length Character
INTEGER/INT	Integer
SMALLINT	Small Integer
REAL/FLOAT	Single Precision Floating Point
DOUBLE PRECISION	Double Precision Floating Point
DECIMAL	Packed Decimal
NUMERIC	Unpacked Decimal
BINARY/BIN	Adabas Specific Unsigned Integer/Adabas Binary

All data types include the NULL value, which should not be confused with a string of zero length or a numeric 0.

Note:
The data types NATURAL DATE and NATURAL TIME are for use in conjunction with Adabas ODBC Client, only.

Character-String Data Type

A value of data type character-string is a sequence of characters. The length of the character-string is derived from the definition of the data type or from the value itself.

Fixed-Length Character String

The length of a value with the data type fixed length character-string is determined by the definition of the origin of the value, e.g. if the value originates from a column which has been defined as a fixed length character-string with length 15, a value originating from this column will always have a length of 15. The maximum length also depends on the origin of the value:

Origin	Maximum Length
column	16381
constant	the number of characters specified
expression	see **Expressions**
function	see **Functions**
host variable	host-language-dependent
subquery	see **Query Specification**

Numeric Data Types

Numeric data types are used to specify the representation form of numeric values. Adabas SQL Server supports 4 different representation forms of numeric values:

Each numeric value's form of representation has a precision and a sign. The precision for each of the forms of representation is specified as follows:

Representation Form	Unit of Precision
Unpacked	digit
Packed	digit
Floating Point	double or single

In addition, the decimal forms of representation have a scale. The scale of a numeric value is defined as the number of digits in the fractional part of the number. The scale can not be larger than the precision nor can it be negative.

Small Integer specifies a binary representation of a numeric value with a precision of 15 bits. The value range of a Small Integer number is –32768 to +32767.

Integer specifies a binary representation of a numeric value with a precision of 31 bits. The value range of an Integer number is –2147483648 to +2147483647.

Single Precision Floating Point specifies a floating point representation with single precision. The value range of a single precision floating point number depends on the hardware platform.

Double Precision Floating Point specifies a floating point representation with double precision. The value range of a double precision floating point number depends on the hardware platform.

Numeric	specifies an unpacked decimal representation with a user-specified scale and precision. The range of the precision is between 1 – 27. The range of the scale is 0 to the value of the precision. The value range of an unpacked decimal number is –999999999999999 to +999999999999999.
Decimal	specifies a packed decimal representation with a user-specified scale and precision. The range of the precision is between 1 – 27. The range of the scale is 0 to the value of the precision. The value range of an unpacked decimal number is –999999999999999 to +999999999999999.

Binary Data Type

Adabas SQL Server supports the Adabas binary data type. This binary data type is not to be confused with the Standard ANSI SQL2 type of BIT, although the two types do display a similarity. Whereas the bit data type is a bit pattern, the binary data type is effectively an unsigned integer and is subject to byte swapping where appropriate. This binary data type can be interpreted in the application program as desired, in most cases this will be as a bit pattern.

Binary values have a maximum length of 126 bytes. The maximum possible number of bits is 1008.

Data-Type Conversion

Adabas SQL Server is capable of converting a value of a certain data type to another data type.

The convertible data types and the conversion rules are as follows:

- Converting **small integer** to **packed decimal**

  A value of the data type small integer is converted to a value of the data type packed decimal with a precision of 5 and a scale of 0.

- Converting **integer** to **packed decimal**

  A value of the data type integer is converted to a value of the data type packed decimal with a precision of 11 and a scale of 0.

- Converting **small integer** to **unpacked decimal**

  A value of the data type small integer is converted to a value of the data type decimal with a precision of 5 and a scale of 0.

- Converting **integer** to **unpacked decimal**

  A value of the data type integer is converted to a value of the data type decimal with a precision of 11 and a scale of 0.

- Converting **unpacked decimal** to **packed decimal**

  A value of the data type unpacked decimal is converted to a value of the data type packed decimal with the same precision and scale.

- Converting **packed decimal** to **unpacked decimal**

  A value of the data type packed decimal is converted to a value of the data type unpacked decimal with the same precision and scale.

- Converting **binary** to **binary of longer length**

  A value of the data type binary cannot be converted to either a value of the data type numeric or a value of the data type character. However, a value of the data type binary can be converted to another value of the data type binary which is of a longer length. In such a case, appropriate padding of the more significant bits will be performed automatically.

Approximate Numeric Data Type Conversion to and from Exact Numeric Data Types

Decimal values are exact numeric values as opposed to single and double precision floating point values which are approximate numeric values. Converting an exact numeric value to an approximate numeric value might lead to a loss of accuracy. Vice versa, when converting an approximate numeric value to an exact numeric value, there might be a loss of precision.

Language Elements

Like any other language, SQL consists of lexical units called tokens. Tokens are:

- constants,
- identifiers,
- keywords,
- delimiters.

Delimiters are used to separate tokens. See section **Delimiters** in this chapter for further details.

Constant Specification

A constant is one origin for a value. For each data type, constants can be specified. Each data type has its own rules on how a constant of that type is to be specified.

Character-String Constants

A character-string constant represents a fixed length character-string value. A character-string constant is a sequence of characters which begins and ends with the character-string delimiter apostrophe (') or double quotation marks.

Note:
In future versions, the alternative use of the character-string delimiter quotation mark (") will no longer be permitted.

The length of a character-string constant is determined by the number of characters between the beginning and ending character-string delimiters. Should a character-string have to contain a quotation mark or apostrophe, this character is to be repeated:

Example:

The constant:	`'this isn''t a string with 4 "''" characters'`
represents the character-string:	this isn't a string with 4 " ' " characters

Numeric Constants

A numeric constant represents a numeric value. The sign, precision and scale are derived from the constant itself.

Integer constant

An integer constant represents a numeric value of data type integer. It is a sequence of digits optionally preceded by a plus (+) or minus (–) character. The precision of an integer constant is 31 bits. If the first digit is not preceded by a plus or minus character the sign of the value is assumed to be positive.

Examples: +234 –43 4323

Floating Point constant

A floating point constant represents a numeric value of data type double precision floating point. It consists of two numbers separated by the character 'E'.

The first number is a sequence of digits which may contain a decimal point and may be preceded by a plus (+) or minus (–) character.

The second number is a sequence of digits optionally preceded by a plus or minus character. The value of a floating point constant is the result of the multiplication of the first number and the power of 10 specified by the second number. The precision is double precision. If the first number is not preceded by a plus or minus character, the sign of the value is assumed to be positive.

Examples: +2.05E2 –0.345
35E–3 5.E+4

Decimal constant

A decimal constant represents a numeric value of data type Decimal. It is a sequence of digits containing a decimal point and optionally preceded by a plus (+) or minus (–) character.

The precision is determined by the total number of digits specified. The scale of the constant is determined by the number of digits after the decimal point. If the first digit is not preceded by a minus character, the sign of the value is assumed to be positive.

Examples: +234.0 –0.34535
.554 1.

Binary Constants

A binary constant can be specified either as a bit constant, i.e. using the binary counting system or as a hex constant, i.e. using the hexadecimal counting system. The two are freely interchangeable and are equivalent.

Binary Literal

A binary literal is signified by the prefix letter ‘y’, either upper or lower case. The ’Y’ stands for binarY. It is then followed by zero or more binary digits (either ’1’ or ’0’), enclosed in single quotes up to the permitted maximum of 1008 digits. For example, an 8 digit binary literal could be expressed as follows:

```
Y’11100100’
```

No space is permitted between the ‘Y’ prefix and the leading quote.

In order to improve legibility, it is permitted to split the literal up into smaller groups of digits. Any white space character is permitted between each group of digits. Alternatively it is not mandatory to have any character separating a group of digits. For example :

```
Y’11’ ’10’ ’01’ ’00’
```

Such a representation, regardless of the configuration of the groups of digits is entirely equivalent to the concatenated form.

Hexadecimal Literal

A hexadecimal literal is signified by the prefix letter ’H’, either upper or lower case. It is then followed by zero or more hex digits (’0’ to ’9’ and ’A’ to ’F’), enclosed in single quotes up to the permitted maximum of 126 digits. For example, an 8-digit hex literal could be expressed as follows:

```
H’A2BFC78D’
```

No space is permitted between the ’H’ prefix and the leading quote.

In order to improve legibility, it is permitted to split the literal up into smaller groups of digits. Any white space character is permitted between each group of digits. Alternatively, it is not mandatory to have any character separating a group of digits. For example:

```
H’A2’ ’BF’ ’C7’ ’8D’
```

Such a representation, regardless of the configuration of the groups of digits, is entirely equivalent to the concatenated form.

Identifiers

Identifiers are used to identify or name objects. An identifier is a character string consisting of uppercase or lowercase letters, digits and the underscore character.

Two basic rules apply:

- the first character must always be a letter,
- an identifier must not be identical to an SQL keyword.

In general, Adabas SQL Server can handle identifiers of up to 32 characters. Additional limitations for certain type of identifiers are described below. The following types of identifiers exist:

Column Identifier	identifies a column of a base table or viewed table. Adabas SQL Server supports column identifiers of up to 32 characters.
Connection Identifier	identifies the name of a connection specified in a CONNECT statement. Adabas SQL Server supports connection identifiers of up to 32 characters.
Constraint Identifier	is used to identify a constraint in a base table. Adabas SQL Server supports constraint identifiers of up to 32 characters.
Correlation Identifier	is used to temporarily identify a table within an SQL statement. Adabas SQL Server supports correlation identifiers of up to 32 characters.
Cursor Identifier	identifies a cursor. Adabas SQL Server supports cursor identifiers of up to 18 characters.
Database Identifier	is a logical name for a database. Adabas SQL Server supports database identifiers of up to 32 characters.
Host Variable Identifier	identifies a host variable in a host program. The naming conventions for a host variable identifier must conform to the naming conventions for the host language in question. Depending upon the context the host variable identifier may be a single host variable, a single field within a host variable structure or a host variable structure reference. Host variable identifiers of up to 32 characters are supported.
Index Identifier	is used to identify an index in a base table. Adabas SQL Server supports index identifiers of up to 32 characters.

Password Identifier	identifies the password associated to a user-ID. Adabas SQL Server supports password identifiers of up to 20 characters. The password identifier may consist of any alpha characters or numbers and the following special characters: ! # $ % * ( ) _ – + = / ? : : . ,, < >
Schema Identifier	identifies a schema. Adabas SQL Server supports schema identifiers of up to 32 characters.
Server Identifier	identifies a server as specified in the routing file/table. Adabas SQL Server supports server identifiers of up to 8 characters.
Shortname Identifier	is used to assign a short name to an Adabas field. This identifier must consist of two characters, the first of which may be any uppercase character between A – Z excluding E and the second one may be any uppercase character between A – Z or a digit 0 – 9.
Statement Identifier	identifies a prepared statement. It is only used in the context of dynamic SQL. Adabas SQL Server supports statement identifiers of up to 32 characters.
Table Identifier	identifies a base table or a viewed table. Adabas SQL Server supports table identifiers of up to 32 characters.
User Identifier	identifies a host variable or a constant in a host program which contains the user-ID. Adabas SQL Server supports user identifiers of up to 32 characters.

Keywords

The following names (keywords) have a prescribed meaning in SQL and can not be used for any other purposes. The following is a list of SQL keywords:

A ABS ABSOLUTE ACCOUNTING ACRABN ACTION ACTIVATE ADABAS ADD ADDDATE ADDTIME ADD_MONTHS AFTER ALIAS ALL ALLOCATE ALPHA ALTER ANALYZE AND ANSI ANY ARE AS ASC ASCENDING ASCII ASSERTION ASSO ASSOPFAC AT AUDIT AUTHORIZATION AVG

B BAD BEGIN BEGINLOAD BETWEEN BIN BINARY BIT BIT_LENGTH BLOCK BLOCKSIZE BOTH BY BUFFER BUFFERPOOL BYTE

C CACHELIMIT CACHES CASCADE CASCADED CASE CAST CATALOG CEIL CHAR CHARACTER CHARACTER_LENGTH CHAR_LENGTH CHECK CHR CLEAR CLOSE CLUSTER CLUSTERED COALESCE COLD COLLATE COLLATION COLUMN COMMENT COMMIT CONCAT CONFIG CONNECT CONNECTED CONNECTION CONSISTENCY CONSOLE CONSTRAINT CONSTRAINTS CONTIGUOUS_AC CONTIGUOUS_DS CONTIGUOUS_NI CONTIGUOUS_UI CONTINUE CONVERT COPY CORRESPONDING COSTLIMIT COSTWARNING COUNT CREATE CREATETAB CROSS CURRENT CURRENT_DATE CURRENT_TIME CURRENT_TIMESTAMP CURRENT_USER CURSOR

D DATA DATABASE DATAPFAC DATE DATEDIFF DAY DAYOFWEEK DAYOFYEAR DAYS DB2 DBA DBPROC DBPROCEDURE DBYTE DEALLOCATE DEC DECIMAL DECLARE DECODE DEFAULT DEFERRABLE DEFERRED DELETE DERIVED DESC DESCENDING DESCRIBE DESCRIPTION DESCRIPTOR DESTPOS DEVICE DEVSPACE DIAGNOSE DIAGNOSTICS DIGITS DIRECT DISCONNECT DISTINCT DIV DOMAIN DOMAINDEF DOUBLE DROP DS DSDEV DSETPASS DSRABN DSREUSE DSSIZE DUPLICATES

E EBCDIC EDITPROC ELSE END ENDLOAD ENDPOS EQ ESCAPE EUR EXCEPT EXCEPTION EXCLUSIVE EXEC EXECUTE EXISTS EXIT EXPAND EXPLAIN EXPLICIT EXTERNAL EXTRACT

F FALSE FETCH FILE FILENAME FIRST FIRSTPOS FIXED FLOAT FLOOR FNULL FOR FORCE FOREIGN FORMAT FOUND FREAD FREEPAGE FROM FULL FWRITE

G GE GET GLOBAL GO GOTO GRANT GRANTED GRAPHIC GREATEST GROUP GT

H HAVING HEX HEXTORAW HOLD HOUR HOURS

I IDENTIFIED IDENTITY IGNORE IMMEDIATE IMPLICIT IN INCLUDE INDEX INDEXNAME INDICATOR INIT INITCAP INITIALLY INNER INPUT INSENSITIVE INSERT INSTR INT INTEGER INTERNAL INTERSECT INTERVAL INTO IS ISN ISNREUSE ISNSIZE ISO ISOLATION

J JIS JOIN

K KEEP KEEPING KEY

L LABEL LABELS LANGUAGE LAST LASTPSO LAST_DAY LE LEADING LEAST LEFT LENGTH LEVEL LFILL LIKE LINK LOAD LOCAL LOCALSYSDBA LOCATIONS LOCK LOG LONG LOWER LPAD LT LTRIM

M MAKEDATE MAKETIME MAPCHAR MATCH MAX MAXDS MAXISN MAXNI MAXRECL MAXUI MDELETE MEGABYTE MFETCH MICROSEC MICROSECOND MICROSECONDS MICROSECS MIN MINSERT MINUS MINUTE MINUTES MIXDSDEV MOD MODE MODULE MONITOR MONTH MONTHS MONTHS_BETWEEN MSELECT MULTIPLE MUPDATE

N NAMES NATIONAL NATURAL NCHAR NE NEXT NEXTVAL NIRABN NISIZE NO NOFORMAT NOLOG NORMAL NOROUND NOT NOWAIT NULL NULLIF NUM NUMBER NUMERIC NVL

O OCTET_LENGTH OF OFF ON ONLY OPEN OPTIMISTIC OPTIMIZE OPTION OR ORACLE ORDER OUT OUTPUT OUTER OVERLAPS OVERWRITE

P PACKED PAD PAGES PARAM PARSE PARSEID PARTIAL PARTICIPANTS PASSWORD PATTERN PCTFREE PERMLIMIT PGMREFRESH PHONETIC PLAN POINTER POS POSITION POWER PRECISION PREPARE PRESERVE PREV PRIMARY PRIOR PRIV PRIVILEGES PROC PROCEDURE PROCPARAM PSM PUBLIC

Q QUALIFIER QUERYNO QUICK

R RANGE RAW RAWTOHEX READ READONLY READWRITE REAL RECONNECT REFERENCED REFERENCES REJECT RELATIVE RELEASE RENAME REPLACE REPLICATION RESET RESOURCE REST RESTART RESTORE RESTRICT REUSE REVOKE RFETCH RFILL RIGHT ROLLBACK ROUND ROW ROWID ROWNO ROWNUM ROWS RPAD RTRIM

S SAME SAVE SAVEPOINT SCHEMA SCROLL SEARCH SECOND SECONDS SECTION SEGMENT SELECT SELECTIVITY SELUPD SEQNO SEQUENCE SERVERDB SESSION SESSION_USER SET SHARE SHORTNAME SHOW SHUTDOWN SIGN SIZE SMALLINT SOME SOUNDEX SOUNDS SOURCEPOS SPACE SQL SQLCODE SQLERROR SQLID SQLMODE SQLSTATE SQLWARNING SQL_DB SQRT STAMP STANDARD STARTPOS STAT STATE STATEMENT STATISTICS STDDEV STOGROUP STORE SUBDATE SUBPAGES SUBSTR SUBSTRING SUBTIME SUBTRANS SUM SUPPRESSION SYNONYM SYSDATE SYSDBA SYSTEM_USER

T TABID TABLE TABLEDEF TABLESPACE TEMP TEMPLIMIT TEMPORARY TERMCHAR THEN TIME TIMEDIFF TIMEOUT TIMESTAMP TIMEZONE TIMEZONE_HOUR TIMEZONE_MINUTE TO TO_CHAR TO_NUMBER TRAILING TRANSACTION TRANSFILE TRANSLATE TRANSLATION TRIGGER TRIGGERDEF TRIM TRUE TRUNC

U UID UIRABN UISIZE UNION UNIQUE UNKNOWN UNLOAD UNLOCK UNPACKED UNTIL UPDATE UPPER UQINDEX USA USAGE USER USERGROUP USERID USING

V VALIDPROC VALUE VALUES VARCHAR VARGRAPHIC VARIANCE VARYING VERIFY VERSION VIEW VIRTUAL VSAM VTRACE

W WAIT WEEKOFYEAR WHEN WHENEVER WHERE WITH WORK WRITE

Y YEAR YEARS YES

Z ZONE ZONED

Delimiters

A delimiter is used to separate the lexical units in the language for compiler processing. Delimiters are either spaces, control characters, comments or special tokens. A comment must be preceded by double hyphens (--).

Special tokens in SQL are all of the following symbols:

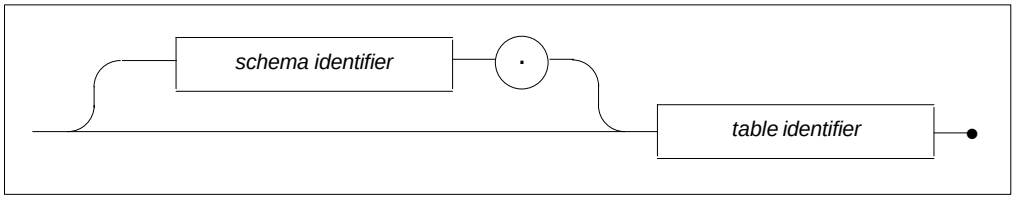
,	( )	<	>	.
:	=	+	-	*
<>	<=	>=	/	;
?	~>	~<	~=	

Table Specification

A table specification is used to identify a base table or viewed table (view) in an SQL statement. A table specification has one of the following purposes:

- to define a table (in a CREATE TABLE, CREATE TABLE DESCRIPTION or CREATE VIEW statement).
- to identify a table (in all other applicable statements).

A table specification can be either a qualified table name or synonym or an unqualified table name or synonym.



Qualified Table Specification

A qualified table specification consists of a schema identifier followed by a table identifier separated by a period. A qualified table specification must uniquely identify a table within a catalog, whereas a table identifier alone must be unique within a schema. Adabas SQL Server supports qualified specifications of up to 65 characters including the period.

All table specifications are effectively qualified, even ones which are not explicitly qualified, as a default schema identifier will be used internally for the implicit qualification. This default is specified prior to the compilation process generally using a precompiler parameter.

An explicit schema identifier must be used in those cases where the default schema identifier does not correspond to the schema identifier of the table to be specified. Should the combination of default schema identifier and the table name not correspond to an actual table, then the compilation will fail as the table reference can not be resolved. In such a case, the table must be explicitly qualified with the correct schema identifier. Another consequence is that if two tables have the same table identifier, but of course different schema identifiers, then an adequate qualifier must be supplied, either explicitly or implicitly. The following example references two different tables:

Example:

```
FROM schema1.cruise, schema2.cruise
```

A table reference is in scope not only within the actual query specification in which it is declared but also for all subqueries that occur as part of this query specification. When the table is declared again in a lower subquery the columns associated with this table refer to the local declaration and not the outer one. Columns which refer to tables declared in an outer query specification are called outer references.

Unqualified Table Specification

An unqualified table specification can be used in those cases where the default schema identifier coincides with the schema identifier of the table to be specified.

Example:

```
FROM cruise, sailor;
```

Correlation Identifiers

Correlation identifiers can only be defined in the FROM clauses of either a query specification or a DELETE statement or in an UPDATE statement.

A correlation identifier assigns a new identifier to a table, which can only be used locally within the statement where it has been defined. The scope of a correlation identifier consists of the query specification or statement where it has been defined and all the subqueries present within that query specification or statement.

If a correlation identifier has been defined for a table and a column of the table needs to be qualified, ***only*** the correlation name can be used to do so. The original table name or synonym ***can not*** be used.

A correlation identifier is mandatory whenever two separate occurrences of the same table need to be distinguished, for example:

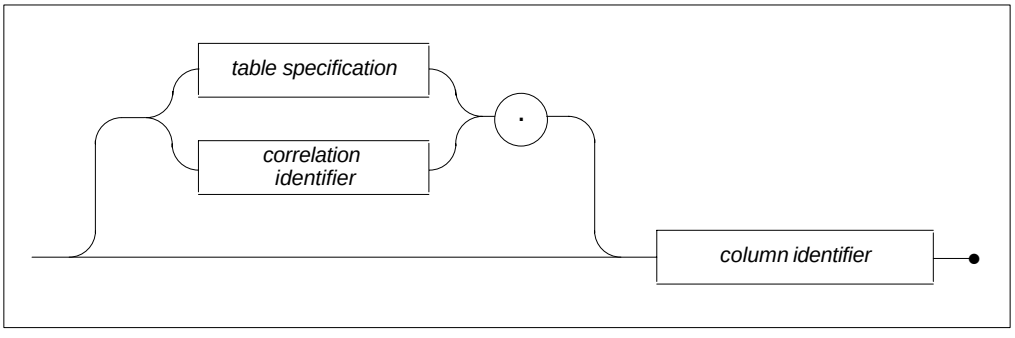
```
SELECT * FROM cruise a, cruise b;
```

The table CRUISE now logically exists twice and can be referenced as either A or B. This method can also be used merely to have a shorter qualifier available for use within the statement.

Column Specification

A column specification is used to identify a column in an SQL statement. A column is specified by a column identifier. A column specification is used for one of the following purposes:

- to define a column of a table (in a CREATE TABLE, CREATE TABLE DESCRIPTION or CREATE VIEW statement).
- to identify a column (in a CREATE INDEX statement).
- to represent the value of a column (in an expression in the SELECT clause, or in a search condition).
- to represent all the values of the resultant rows to which the clause in question is applied (in WHERE or HAVING clauses as well as in the GROUP BY or ORDER BY clauses).
- to represent all the values of the rows resulting from the grouping operation (as an argument in a function).



A column specification can be either a qualified column specification or an unqualified column identifier. A qualified column specification explicitly specifies the table to which the column relates, an unqualified column identifier does not make this relation explicit.

The general method applied by Adabas SQL Server to relate a column specification to one and only one table specification conforms to the following rules:

- the current query specification is defined as the one in which the column specification occurs.
- successive query specifications are analyzed from the current query onwards.
- the first table specification that contains the definition of the column specification in question is taken. This is the candidate table specification.
- within that query specification, no other candidate table specifications may occur.

Note:
If the candidate table is contained in a higher query specification than the current one, the column is an outer reference.

Unqualified Column Specification

An unqualified column specification can be used in those cases where it is possible to relate the column unambiguously to one table. This is the case when only one table within the same query specification contains the column identifier in question.

Example:

```
SELECT cruise_id,contract_id
     FROM cruise,contract;
```

Qualified Column Specification

A qualified column specification consists of a table specification followed by a column identifier separated by a period. Qualified column specifications must be used in those cases where it is otherwise impossible to relate a column unambiguously to a table. This is the case when more than one table in the same query specification contains columns with the same column identifier.

Example:

The example below shows how the column specification distinguishes between two columns of the same name in different tables. Note, that both tables have to be specified in the FROM clause.

```
SELECT contract.id_cruise, sailor.id_cruise
     FROM contract,sailor;
```

Another reason can be that the same table needs to be referenced more than once in the same query specification. In this situation simply qualifying the column identifier with the table specification will not suffice. Instead a correlation identifier is required to distinguish between the different references of the same table (refer to the section **Table Specification** in this chapter for more information).

Example:

To find the least expensive cruise for each destination, the following syntax applies, where the first instance of the table cruise has to be correlated with the letter X, as the subquery needs to distinguish between two identical column references on two different ’instances’ of the same table.

```
SELECT cruise_id,start_harbor,cruise_price
     FROM cruise X
     WHERE cruise_price = ( SELECT MIN(cruise_price)
          FROM cruise
          WHERE destination_harbor = X.destination_harbor );
```

Outer references are a special type of qualified column specification. Strictly speaking, they are only required if a column identifier can not unambiguously be related to a single table. The use of qualified column specifications for outer references increases the readability of an SQL statement. An outer reference is a reference to a column of a table specified in a higher-level query specification.

Example:

To identify all contracts that cost more than double the cruise price of the cruises that the contracts identify , the following syntax applies. This example shows how the id_cruise column is an outer reference as the table it references is contained in the higher query specification.

```
SELECT contract_id FROM contract
    WHERE (price*2) > (SELECT cruise_price FROM cruise
        WHERE cruise_id = contract.id_cruise );
```

Host Variable Specification

Host variables serve as a data exchange medium between Adabas SQL Server and the application program written in a host language. When used in an SQL statement, a host variable specification has one of the following purposes:

- to identify a variable in the host language program which is to receive a value(s) from Adabas SQL Server.
- to identify a variable in the host language program which is to pass a value(s) to Adabas SQL Server.

A host variable is a single variable or structure declared in the host program.

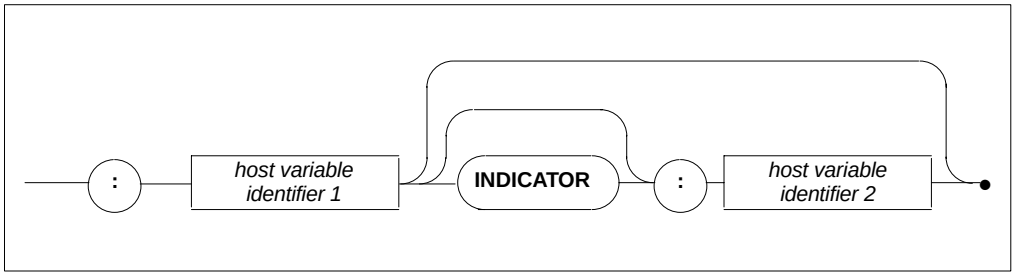
A host variable identifier is used to identify a single host variable or structure from within an SQL statement.

A host variable specification consists of a host variable identifier and an associated optional INDICATOR variable and defines either a single variable, a structure, or an element in a structure.

Single Variables

The identified single host variable may actually be a single element within a host variable structure. Such a reference is not permitted in ANSI compatibility mode.

A single host variable is identified by a host variable identifier which has the following syntax:



host variable identifier 1	identifies a single variable which is assigned any value but the NULL value.
host variable identifier 2	identifies an INDICATOR variable, see section **INDICATOR Variables** below.

Example:

To select the price of the cruise with a cruise ID of 5064 into a host variable the following syntax applies.

```
SELECT cruise_price
     INTO :host_variable1
          FROM cruise
          WHERE cruise_id=5064;
```

INDICATOR Variables

An INDICATOR variable can serve as one of two purposes:

- Signifies the presence of a NULL value in a host variable assignment.
  If the NULL value is to be assigned to a target host variable specification then an accompanying INDICATOR variable must be present and is assigned a negative value to signify the NULL value. If the NULL value is to be assigned and the INDICATOR variable is missing, then a runtime error will occur.

  The INDICATOR variable must be of a numeric data type with the exception of double precision, real and floating point data types. It must be of the appropriate data type for the host language.

  Example:

  To select the cancellation date of Contract 2025 into a host variable, the following syntax applies. (The column 'date_cancellation' could contain NULL values)

```
SELECT date_cancellation
    INTO :host_variable1 INDICATOR :host_variable2
        FROM contract
            WHERE contract_id=2025 ;
```

- Signifies that truncation has occurred in a host variable assignment.
  If truncation occurred during the assignment of a character string to a host variable, then the INDICATOR variable will show the total number of characters in the originating source prior to truncation.

  SUMMARY:

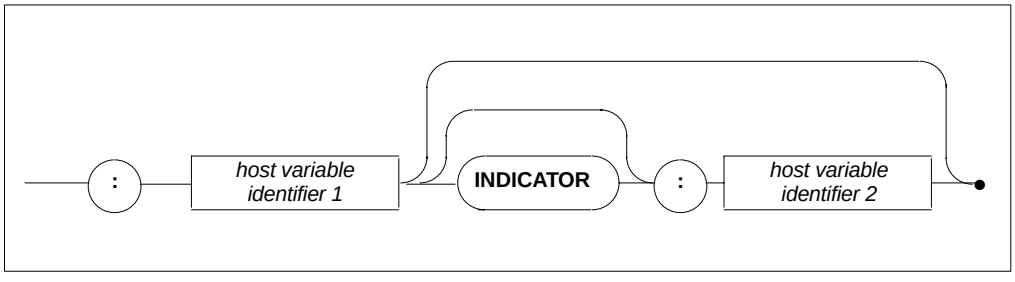
INDICATOR Value	Meaning	Host Variable Value
<0	signifies NULL value	undefined
=0	signifies non-NULL value	actual value
>0	number of characters	actual value in originating source

Host Variable Markers

A dynamic SQL statement can not contain host variables directly. It is, however, possible to provide a dynamic SQL statement after it has been prepared with value parameters at execution time. The dynamic statement must then contain a host variable marker for every host variable specification. A host variable marker is represented by a question mark (?) in the statement's source text. For details refer to *Adabas SQL Server Programmer's Guide*, chapter: **Dynamic SQL.**

Host Structures

A host structure is a C or a PL/I structure or a COBOL group that is referenced in an SQL statement. The exact rules to which a host structure must conform are described in the host language chapters of the *Adabas SQL Server Programmer's Guide.*



host variable identifier 1 identifies a host structure. It can only be specified in the INTO clause of a single row SELECT or FETCH statement. A reference to a host structure is equivalent to a reference to each element in that structure.

Each element of the host structure identified by host variable identifier 1 is a host variable which is assigned a value, if that value is not the NULL value.

host variable identifier 2 is an INDICATOR structure. An INDICATOR structure is a host structure consisting of elements each identifying an INDICATOR variable.

Each element of the INDICATOR structure identified by host variable identifier 2 identifies an INDICATOR variable, see also section **INDICATOR Variables** in this chapter.

The *i*th element in the host structure indicated by host variable identifier 2 is the INDICATOR variable for the *i*th element in the host structure indicated by host variable identifier 1.

Note:
Pointer expressions will be supported in the next release version.

Assume the number of elements in the host structure identified by host variable identifier 1 is m and the number of elements in the host structure identified by host variable identifier 2 is n:

- If m > n, then the last m-n elements in the host structure identified by host variable identifier 1 do not have an INDICATOR variable.
- If m < n, then the last n-m elements in the host structure identified by host variable identifier 2 are ignored.

Examples:

If two host structures have been declared, one for actual returned values and one for indicator values, and the variables 'struct1' and 'indicator1' identify these structures respectively, then the following syntax shows how values from a derived column list are entered into host variables (assuming that the host structures match the derived columns).

```
SELECT cruise_id,start_date,cruise_price
     INTO :struct1 INDICATOR :indicator1
          FROM cruise;
```

To insert a resulting value from a query into a particular 'Element' of a defined structure, the following syntax applies. Where 'struct1' is a structure identifier that contains an element identified by 'price_element' and 'indicator1' is a structure identifier that contains the element identified by 'price_ind'.

```
SELECT cruise_price
     INTO :struct1.price_element INDICATOR :indicator1.price_ind
          FROM cruise;
```

Query Specification

Function:

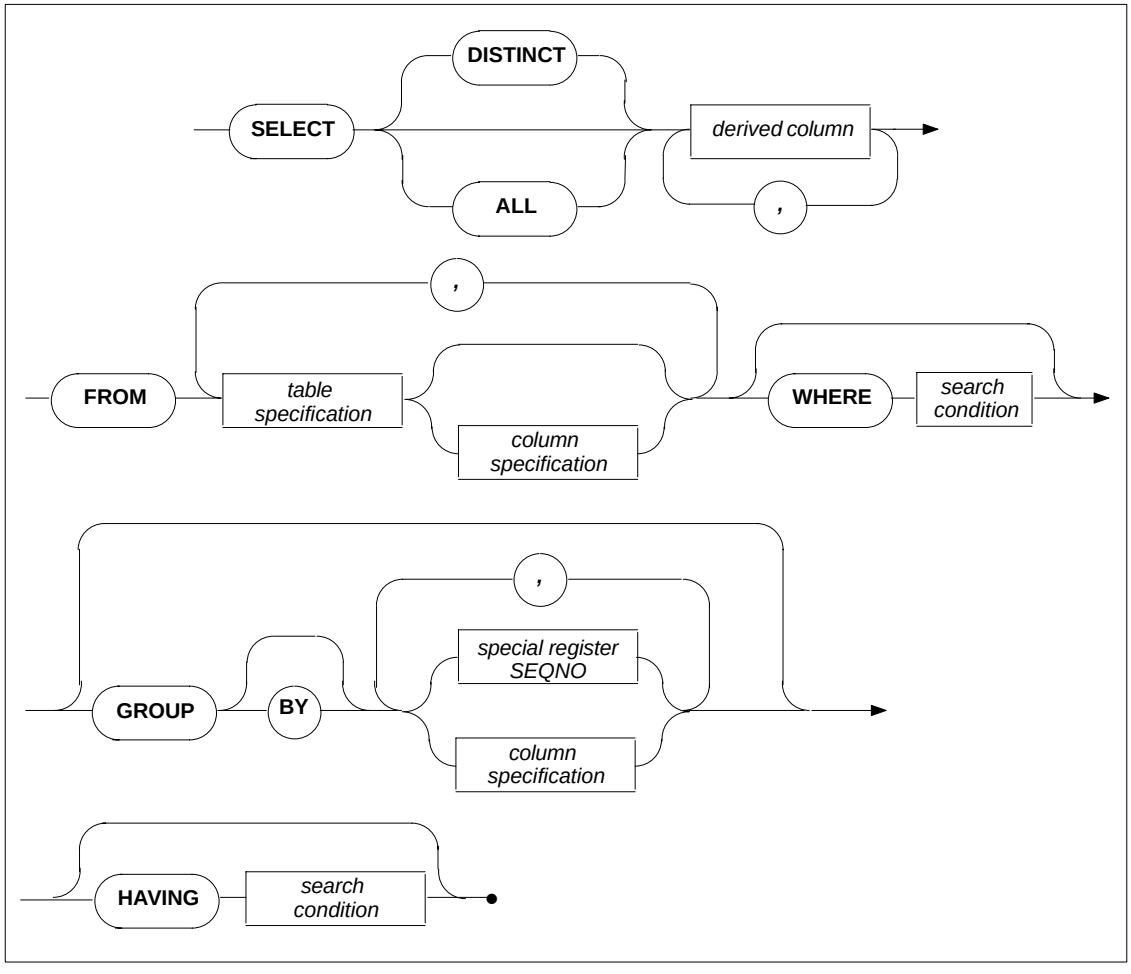
A query specification is used to define a resultant table.

Invocation:

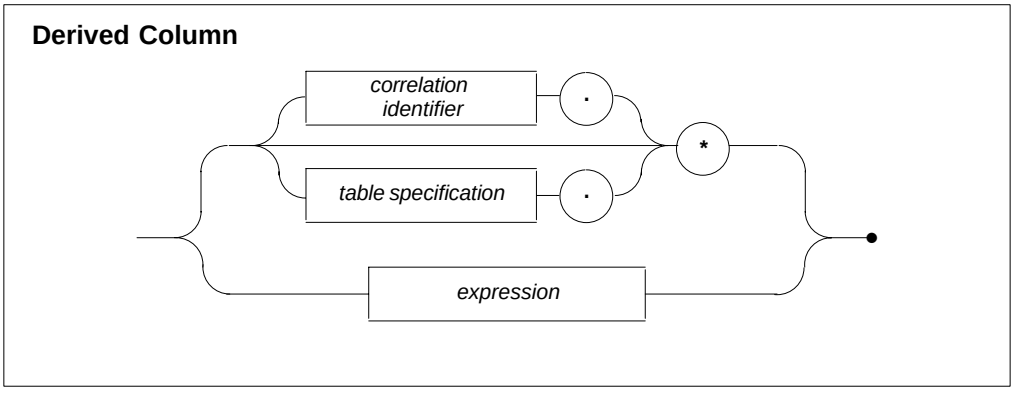
A query specification can appear in one of the following contexts:

- as the operand of a query expression (in a DECLARE CURSOR statement),
- in a subquery within, for instance, a COMPARISON predicate,

Syntax:


DISTINCT
SELECT
derived column
ALL
,
,
FROM
table specification
column specification
WHERE
search condition
,
GROUP
BY
special register SEQNO
column specification
HAVING
search condition

DISTINCT	is an optional directive which forces all rows of the resultant table to be unique, i.e duplicate rows will not be returned.
ALL	is the default setting and means that duplicate rows will be returned. The use of this directive is superfluous.
derived column	is the specification of the corresponding columns in the final resultant table derived by the query. Derived columns are separated by commas and all of them together are referred to as the derived column list (see separate diagram below).
table specification	is the specification of tables or views from which the resultant table is to be defined. Table and view names are separated by commas and all of them together are referred to as the table list.
correlation identifier	is a means of giving an alternative name to a particular table for use within the query and subqueries which are in scope.
WHERE clause	is the specification of a search condition which candidate rows must fulfil in order to become part of the resultant table.
GROUP BY clause	is the specification of the desired grouping columns. A grouping column is the column by which the resultant table will be grouped.
Special register SEQNO	enables access to Adabas information such as ISN or occurrence numbers.
HAVING clause	specifies a search condition which candidate groups must fulfil in order to become part of the resultant table.



correlation identifier — is a means of giving an alternative name to a particular table for use within the query and subqueries which are in scope.

table specification — is the specification of a table or view. The correlation identifier and table specification must be specified in the table list of the FROM clause.

* — is an abbreviated form of listing all columns of the table identified by the correlation identifier or the table specification. If this is specified, all columns of all tables specified in the table list of the FROM clause are selected. SEQNOs are not selected unless they are included in the table description as a named column. In ANSI compatibility mode, the qualification of the asterisk in the form of the correlation identifier or the table specification is not permitted.

expression — is a valid expression as described in the section **Expressions** in this chapter.

Description:

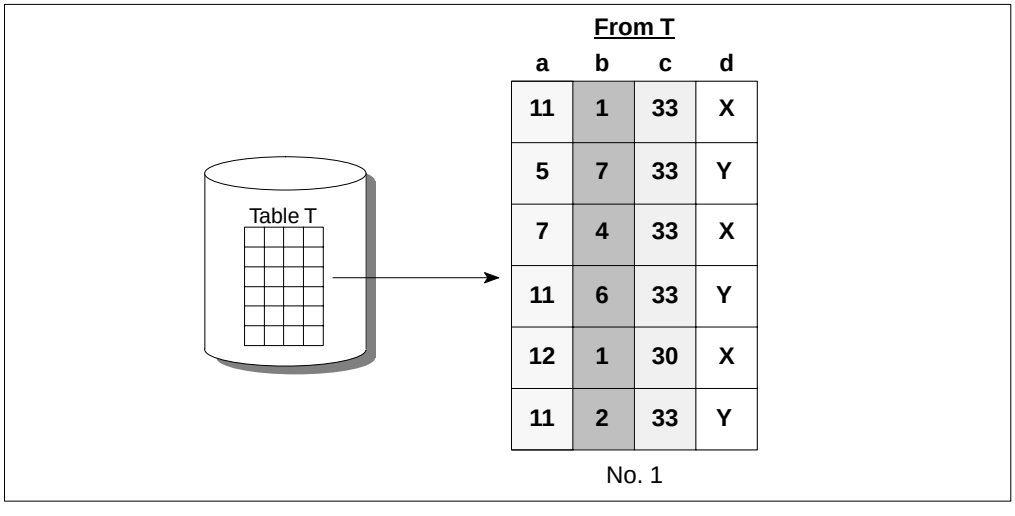
A query specification defines the resultant table specified in the derived column list derived from the tables or views given in the table list, subject to the conditions imposed by the optional WHERE and/or HAVING clause and optionally grouped according to the GROUP BY clause.

Example:

The following describes the step-by-step processing of a query with the respective intermediate resultant tables. The abstract example uses a base table named T and columns named a, b, c and d. The apparent ordering of the intermediate resultant tables is due to ease of representation rather than of any predetermined ordering of the resultant tables.

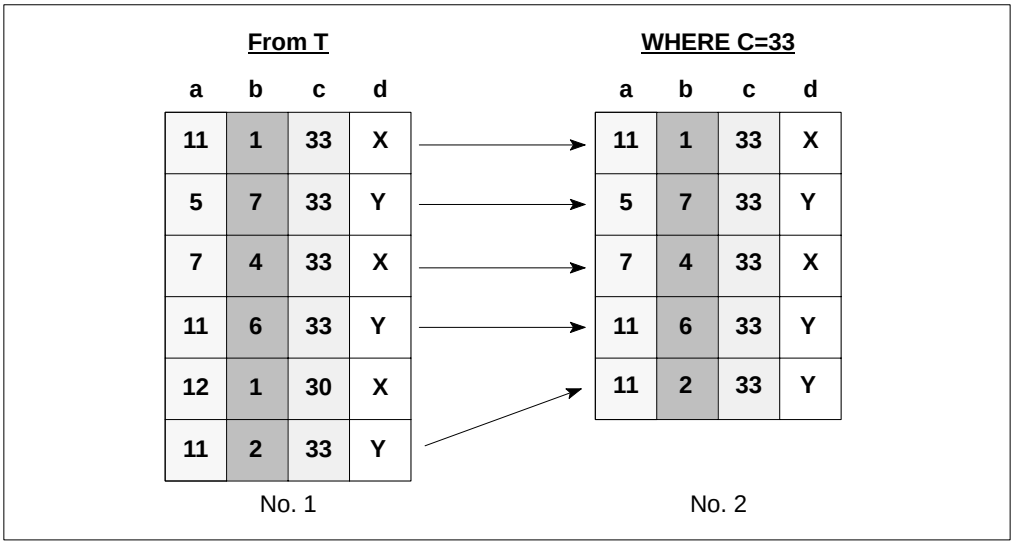
```
SELECT a + 10, d, MAX(b) + 2
     FROM T
     WHERE c = 33
          GROUP BY a, d
          HAVING MIN(b) > 3;
```

1 The table list in the **FROM clause** actually defines all the candidate rows which may become part of the result. Conceptually, the first processing step of a query specification is to establish an intermediate resultant table containing all columns and all rows as defined in the table list. If only one table is involved, then the resultant table will be equivalent to the base table. However, should more than one table be listed, then all the tables in the list must be conceptually joined.



Processing Step 1

2 The next processing step concerns the **WHERE clause**. Each row in the intermediate resultant table is conceptually subjected to the search condition specified in the WHERE clause. If the condition equates to true, then the candidate row proceeds to the next stage. Otherwise, it is eliminated from further consideration, thus reducing the size of the final resultant table. Should no WHERE clause have been specified or the condition equate to true for all candidate rows then the subsequent resultant table will contain all rows as illustrated by the intermediate resultant table No.1.

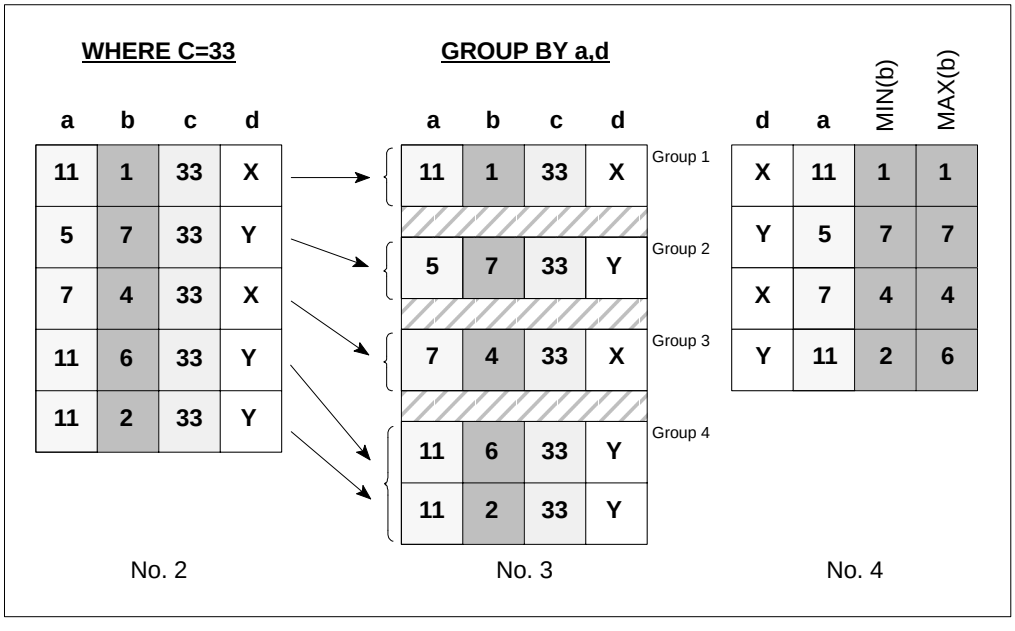


Processing Step 2

3 The next possible processing step concerns the **GROUP BY clause**. This step actually splits into two phases resulting in Tables No. 3 and No. 4. If built-in functions are used within a query, it is called a grouped query. The query is also grouped if a GROUP BY clause is specified, even if no functions are given. Built-in functions are aggregate operators which operate on a set of values in order to produce a single value as a result. These functions can be applied to the whole intermediate resultant table in order to produce a final resultant table of one row. In such a case, no GROUP BY clause is specified but the query is still grouped, as it uses built-in functions.

Any column referenced within a grouped query must be an operand of a function, a grouping column or appear anywhere in the WHERE clause. This is because outside of the WHERE clause, the query is concerned with groups instead of mere rows. The converse, however, is not true. A grouping column may appear in a function.

This is equally true for SEQNO special registers and named SEQNO columns. In the case of a special register, it must be specified exactly the same way as it appears in the SELECT list.



Processing Step 3

It is possible to divide the intermediate resultant table into groups. Groups are partitioned by specifying at least one grouping column in the GROUP BY list. A group is then established by extracting all candidate rows from the intermediate resultant table No. 2, where the value of the grouping column/s is/are equal. As many groups are established as there are differing values of the grouping column. There is no predetermined ordering of these groups.

Groups are established as follows:

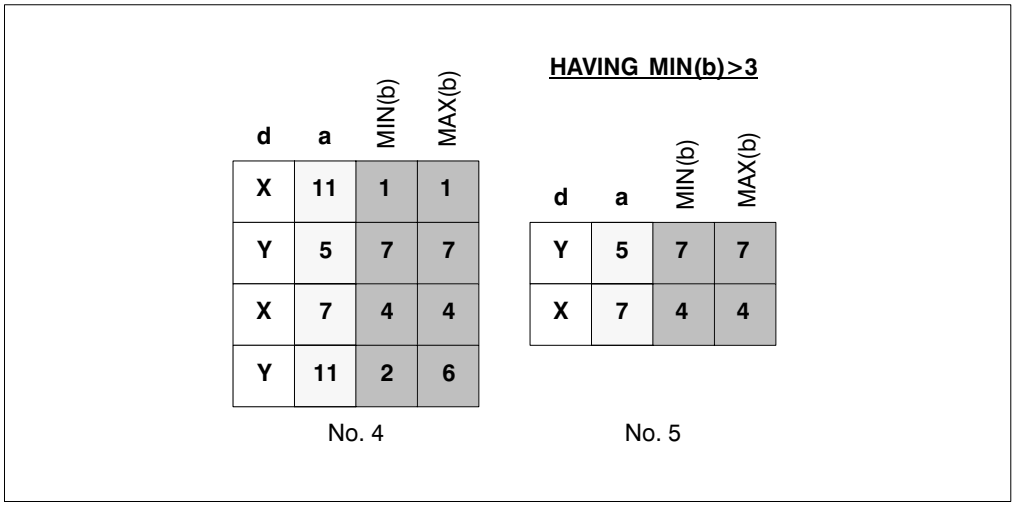
- identical values in the first grouping column are identified,
- if a match has been made, the values of the second grouping columns are compared (same procedure for all other grouping columns),
- if all values in the grouping columns are identical, a candidate row has been identified.

At this point the second phase is initiated. The query is examined in order to produce a list of the columns required for intermediate resultant table No. 4. These new columns are either grouping columns or columns derived from functions applied to columns in intermediate resultant table No. 3. In either case, only columns or functions appearing in the derived column list or the HAVING clause have to be considered. Thus, aggregate functions are applied to each group in turn resulting in one candidate row per group in intermediate resultant table No. 4.

The aggregate functions can now be applied to each group in turn resulting in one candidate row per group for the next conceptual intermediate table.

In conclusion, the GROUP BY clause establishes candidate groups which, when operated upon by the aggregate functions, are transformed into candidate rows, one per group, which form the next intermediate resultant table No. 4.

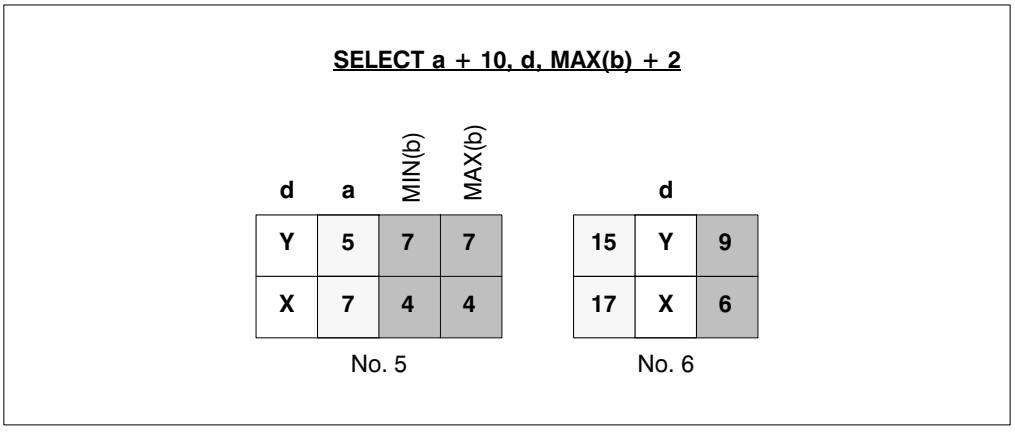
4 The next possible processing step concerns the **HAVING clause**. Each row in the intermediate resultant table is conceptually subjected to the search condition specified in the HAVING clause. If the condition equates to true, then the candidate row proceeds to the next stage, otherwise it is eliminated from further consideration. As such, it is analogous to the WHERE clause except it eliminates candidate groups rather than candidate rows. It is therefore permissible to use functions in the search conditions. In fact, columns which are not contained in a function must be specified in the GROUP BY list.



Processing Step 4

5 The final stage can now be executed, namely the production of the **final resultant table**. This is a derivation of the previous intermediate resultant table and is conceptually the same, regardless of whether it came from the HAVING, GROUP BY, WHERE or FROM clause. A resultant row is processed by evaluating each derived column in turn, based on the values contained in the corresponding row of the intermediate resultant table. This evaluation may be quite complex, depending on the nature of the expressions contained in the derived column's specification.

Step 5 finalizes the processing of this query by producing the final resultant table no. 6.



Processing Step 5

Derived Column List

A derived column list of at least one derived column must be specified. This may be done either as explicit expressions separated by commas or as an asterisk. The asterisk is an abbreviation representing all the columns as defined in the table list. The special register SEQNO is not selected as part of the asterisk abbreviation. An equivalent statement would simply list all columns explicitly, in the order in which they were defined in the original CREATE TABLE statement.

It is also possible to qualify the asterisk with a table specification which will result in all the columns belonging to the specified table only being derived.

Each derived column has an associated data type which is projected out of the subquery. The derived column may also have a derived column label by which the derived column can be identified externally to the query specification e.g. from within an ORDER BY clause. A derived column label is only present if the derived column is based exclusively on a column of a base table. In which case the derived column label is simply the fully qualified column specification. If one column of a resultant table does not have a derived column label then all other column labels can not be referenced.

The special register SEQNO also has a derived column label, when not specified within a numeric expression and only in connection with an ORDER BY clause. The derived column label is simply SEQNO. Should the special register SEQNO be attributed with a table level number, the derived column label will be attributed accordingly.

It should be noted that the use of an asterisk with a table list made up of more than one table can lead to extremely large derived column lists.

Tables

A query specification must have at least one table or view listed in the FROM clause. All column references must uniquely refer to one of these table references. If the same column name is present in more than one table in the FROM clause then it must be qualified by the appropriate table name, which itself may need to be explicitly qualified (refer to the section **Table Specification** in this chapter for details).

A table reference is in scope not only within the actual query specification in which it is declared but also for all subqueries that occur as part of this query specification. That is until the table is declared again in a lower subquery in which case columns referring to this table refer to the local declaration and not the outer one. Columns which refer to tables declared in an outer query specification are called outer references.

Should more than one table be declared in the FROM clause, then the query is said to be joined. It is possible for a table to be joined with itself but in such a case, in order to make the table references unique within the FROM clause, at least one correlation name must be given.

Query Specification/Subqueries

A subquery is a query specification which is subordinate to or nested in another query specification. In general, a subquery is also the origin of a value or a set of values. If this is the case, the number of derived columns in the derived column list of the query specification must be exactly one. The data type and length of such a value resulting from a subquery is the data type and length of that derived column.

A correlation name is a means of giving an alternative label to a table within the query specification. Hence, if a column reference is qualified with the table name and a correlation name has been specified, then the qualification must be the correlation name.

Limitations:

A subquery may only return a derived column list with a cardinality of one. Within an unquantified COMPARISON predicate only one value may be returned. Please refer to COMPARISON, IN and EXISTS predicates for more details.

A subquery which is specified as part of an unquantified COMPARISON predicate may not have a GROUP BY or a HAVING clause nor may the FROM clause reference a grouped view.

No outer reference columns may appear in the GROUP BY list.

Columns which are specified in grouped queries but are not themselves specified in functions are grouping columns and hence, must be listed in the GROUP BY list. This is only necessary for columns appearing either in the derived column list or in the HAVING clause, regardless of if they are referenced in a subquery of the grouped query or not. If there are no such columns then a GROUP BY list is not required, i.e the whole intermediate resultant table is considered to be a group. However one may be given if desired.

Outer reference columns may not appear in the derived column list of any subquery. They are therefore restricted to the WHERE clause or the HAVING clause of the subquery.

A grouped query which is derived from a view can not reference columns from that view in any kind of expression.

A grouped query can only reference one table. Similarly, it can not reference a joined view.

A DISTINCT directive may only appear once within the subquery. Hence, if the derived column list has been specified as DISTINCT then no functions may also be specified with DISTINCT, whether they are in the derived column list, in the HAVING clause or even in a contained subquery.

ANSI Specifics:

The keyword BY is mandatory in a GROUP BY clause.

Adabas SQL Server Specifics:

The keyword BY is optional in a GROUP BY clause.

Examples:

The following syntax applies when finding all the contract's and associated cruise identifiers for all cruises booked on August 12th, 1991.

```
SELECT contract_id,id_cruise
     FROM contract
     WHERE date_booking = 19910812;
```

The following syntax applies when requiring a list of the different start harbors available.

```
SELECT DISTINCT start_harbor
     FROM cruise ;
```

The following syntax applies when needing to identify all the contract IDs, customer IDs and cruise prices of all cruises that leave from Bahamas.

```
SELECT contract.contract_id, contract.id_customer, cruise.cruise_price
     FROM contract,cruise
     WHERE cruise.start_harbor = 'BAHAMAS'
          and contract.id_cruise = cruise.cruise_id;
```

To find the most expensive and least expensive cruise going to either Fethiye or Bodrum from Marmaris, the following syntax applies.

```
SELECT   start_harbor,
         destination_harbor,
         MAX(cruise_price),
         MIN(cruise_price)
         FROM cruise
         WHERE start_harbor = 'MARMARIS'
         GROUP BY start_harbor,destination_harbor
              HAVING destination_harbor = 'FETHIYE'
              OR destination_harbor = 'BODRUM' ;
```

Also see the detailed, illustrated examples earlier within this section.

Persistent Procedure Specification

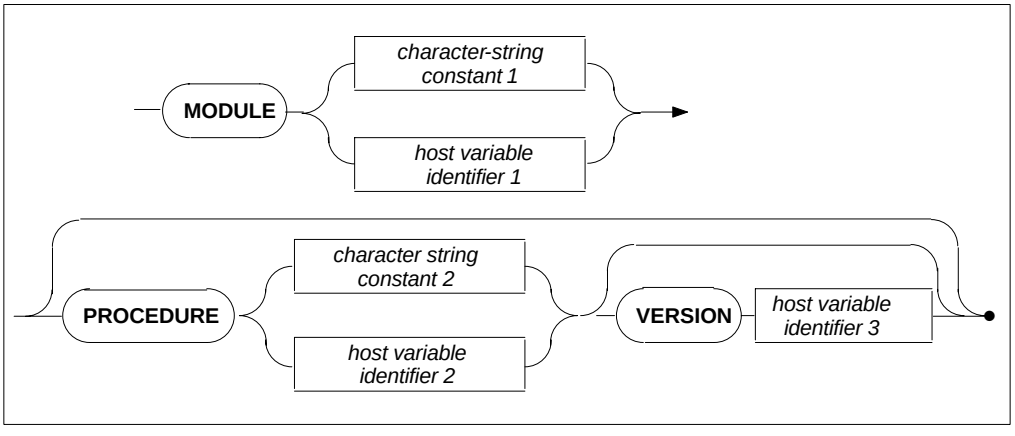
Function:

A persistent procedure specification defines the information that serves as a key when storing or retrieving dynamically prepared statements to or from the catalog.

Invocation:

Can appear in one of the following statements: PREPARE, DECLARE CURSOR, DESCRIBE, OPEN, EXECUTE and/or DEALLOCATE PREPARE

Syntax:



character string constant 1 specifies the name of the module. The length of the name is limited to 27 characters.

host variable identifier 1 is a valid single host variable which is used to contain the name of the module.

character string constant 2 specifies the name of the procedure. The length of the name is limited to 5 characters.

host variable identifier 2 is a valid single host variable which is used to contain the name of the procedure.

host variable identifier 3 is a valid single host variable which is used to contain the version code that will be stored or checked against. The version code consists of 8 bytes of binary data.

Description:

When preparing SQL statements dynamically, there is the option to produce a temporary or a persistent version of the executable form (meta program). Statements prepared to be persistent will be stored in the catalog and can be retrieved anytime by any session or process.

This is in contrast to the statement prepared for temporary purposes. The executable form of this preparation can not be shared by multiple sessions and does not exist beyond the end of session.

In the catalog, a persistent SQL statement is identified by three items: a module name, a procedure name and a version code. The module and procedure names combined serve as a key and this combination must uniquely identify each meta program in the catalog. The version code serves as a checkmark to verify the correct generation.

Static embedded SQL statements also result in meta programs stored in the catalog, once they have been compiled. This happens at the time of their first invocation. The identification of such a meta program is done by compilation unit identifiers, statement sequence numbers and precompilation timestamps. Compared to persistent dynamic meta program identification, the compilation unit identifiers correspond to the module names, the statement sequence numbers map into the procedure names and the precompilation timestamp is used as a version code.

Each meta program stored is equipped with the above-mentioned identification items during execution of a PREPARE statement. Each further reference of the meta program, for example an OPEN statement, must provide the same values. If the meta program is not found by the key value (module and procedure names) an error message will be generated. Another error message will be generated if a meta program with the key value was found but the version code differs from that provided in the statement.

With a DEALLOCATE PREPARE statement, meta programs can be removed from the catalog. In this case, the version code must not be specified. The DEALLOCATE PREPARE statement can either remove a single meta program or by omitting the procedure name, all meta programs of a specific module.

Limitations:

If a persistent procedure specification is used in a DEALLOCATE PREPARE statement, the VERSION clause must be omitted. If the PROCEDURE clause is also omitted, the whole module is specified.

If the persistent procedure specification is used in any other appropriate statement, the PROCEDURE and VERSION clauses must be specified.

ANSI Specifics:

The persistent procedure specification is not part of the Standard.

Adabas SQL Server Specifics:

None.

Example:

The following statement will remove a prepared statement that has been stored in the catalog. The host variables *mod* and *proc* must contain the module and procedure names.

```
DEALLOCATE PREPARE MODULE :mod PROCEDURE :proc;
```

An OPEN statement using a persistent procedure specification might look as follows:

```
OPEN :cursorname CURSOR FOR
     MODULE :mod PROCEDURE :proc VERSION :vers;
```

Privilege Specification

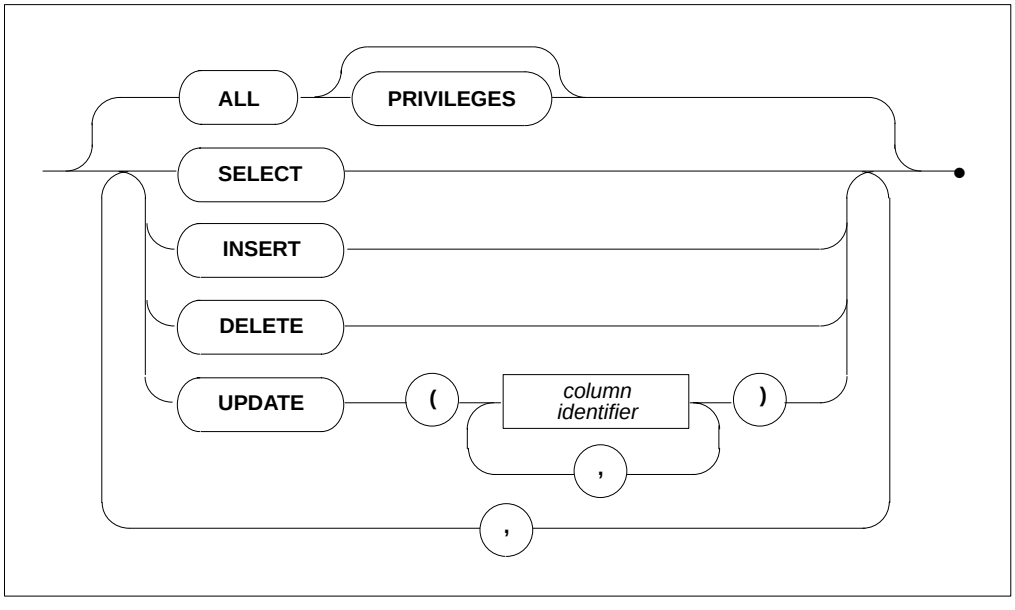
Function:

Defines the privileges which may be granted or revoked.

Invocation:

It can appear in GRANT and REVOKE statements.

Syntax:



column identifier identifies the column upon which the GRANT or REVOKE statements will be based. The column name must be a column defined on the specified table or view. If more than one table or view is specified, then the column identifier must be valid for all tables or views.

Description:

Defines the privilege or set of privileges to be granted or revoked. These privileges are defined for specified tables or views.

The following privilege specifications may be defined:

SELECT	enables the selection of data from the table(s) or view(s).
INSERT	enables the insertion of data in the table(s) or view(s).
DELETE	enables the deletion of rows from the specified table(s) or view(s).
UPDATE	enables the updating of data in the specified table(s) or view(s). The UPDATE privilege can be specified for a list of columns within the table(s) or view(s).

Limitations:

If a view is based on more than one base table (read-only view), then the SELECT privilege is the only one to be granted in this case.

ANSI Specifics:

The keyword “PRIVILEGES” is mandatory when specifying “ALL”.

Adabas SQL Server Specifics:

The keyword “PRIVILEGES” is optional when specifying “ALL”.

Example:

See the GRANT/REVOKE statements for examples.

Grantee Specification

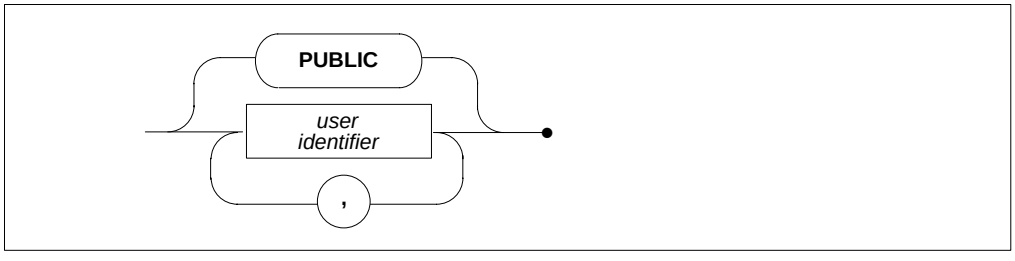
Function:

Identifies the individual(s) to whom privileges are to be granted or from whom privileges are to be revoked.

Invocation:

It can appear in GRANT and REVOKE statements.

Syntax:



user identifier identifies the user to be granted/revoked privileges

Description:

Defines whether the privilege or set of privileges is to be granted to or is to be revoked from a particular user, from a list of users, or from all users. If the option PUBLIC is specified, all present and future users will automatically be affected by the granting or revocation of the specified privilege.

Limitations:

Owners of tables hold all privileges for their tables by default and should, therefore, not additionally grant (or revoke from) themselves privileges on these tables.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

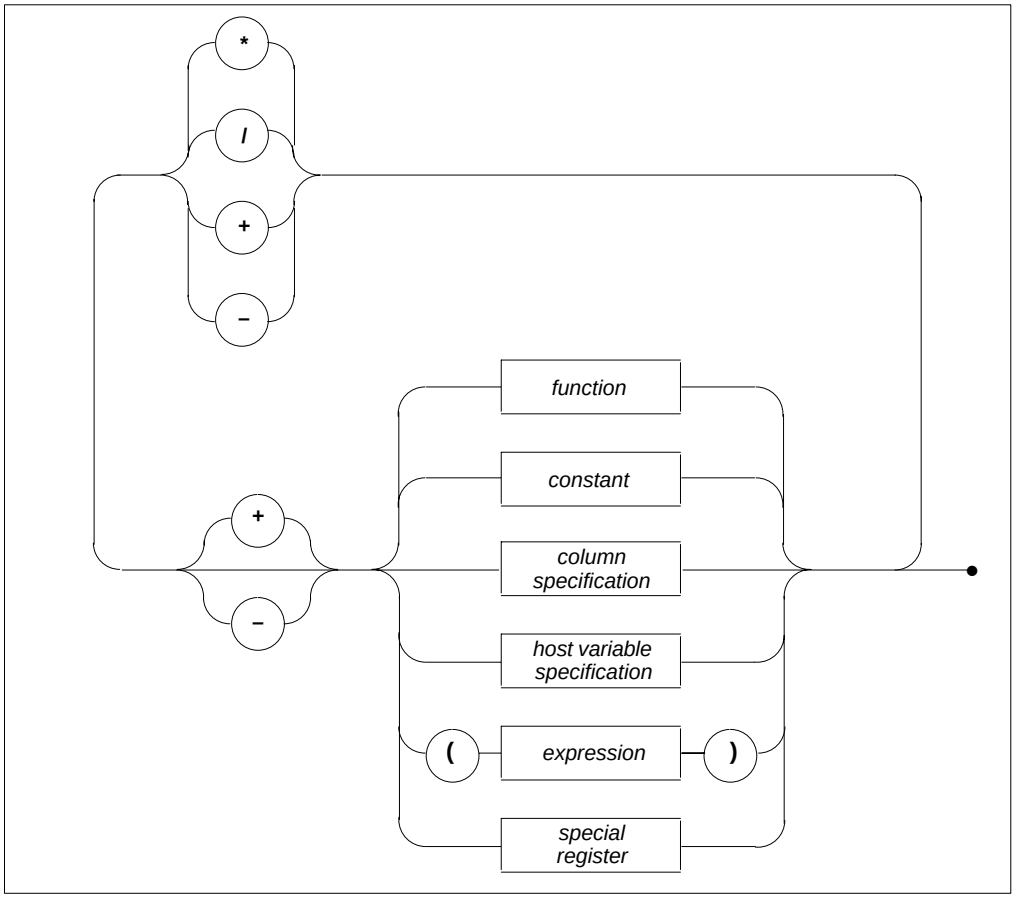
None.

Example:

See the GRANT/REVOKE statements for examples.

Expressions

In general, an expression is a combination of opeands separated by operators. An expression produces a result and is, therefore, an origin of a value. The following diagrams define the syntax of an expression:



Note:
In this context, the host variable specification can only reference a single host variable or a single field within a structure.

Expressions Without Operators

If an expression is used without operators, the result is the value represented by the object specified, e.g., the result of an expression consisting of a column specification is the value represented in that column specification.

Expressions With Operators

The operators which can be used in expressions can be divided into monadic and diadic operators.

Monadic Operators are prefix operators and have one operand. Monadic operators include the monadic plus (+) and the monadic minus (–) operators. The monadic plus operator does not change the value of its operand. The monadic minus operator changes the sign of the value of its operand. Monadic operators can only be used with one operand of data type numeric.

Diadic Operators are infix operators and have two operands. Diadic operators include the addition (+), subtraction (–), multiplication (*) and division (/) operators. Diadic operators can only be used with operands of numeric data type. The data type of the result of an expression with two operands and a diadic operator depends on the data types of the two operands and on the operator. The rules which apply to Adabas SQL Server are described as follows:

Precedence of Operators and Parentheses

The operators in an expression are processed in a certain order. This order of precedence can be influenced by the use of parentheses. Operators of equal precedence are applied from left to right.

The following table lists all operators and parentheses in the order of their precedence:

Operator	Function	Example
( )	Parentheses override precedence rules. Operations inside parentheses are applied first.	(x+y) * (x-y)
+ –	Monadic plus/monadic minus	–1
* /	Multiply, divide (diadic)	y/2
+ –	Add, subtract (diadic)	y-2

Note:
A diadic operator must not be immediately followed by a monadic operator. Otherwise, in the case of '—' (two minus signs) it will be assumed that this is an SQL comment.

Integer Operands

If a diadic operator has two operands of the data type Integer or Small Integer, the result is of data type Integer. In the case of a division operation, the possible remainder will be lost, as any result must lie in the range of the Integer data type.

Decimal Operands

If a diadic operator has two operands with data type Decimal, the result is also of data type Decimal. Operations are performed using the Packed Decimal instructions of the underlying hardware or software simulation. The precision and scale of the result depend on those of the two operands and on the operation applied. Let P_1 and S_1 denote the precision and scale of one operand and P_2 and S_2 the precision and scale of the other. Let M denote the maximum precision of 27.

Addition and Subtraction

The precision (P) and scale (S) of the result on mainframe systems are determined by the following formulae:

$$P = \min (M, \max (P_1 - S_1, P_2 - S_2) + \max (S_1, S_2) + 1)$$

$$S = \max (S_1, S_2)$$

If both operands do not have the same scale, the operand with the smaller scale is copied to a temporary variable with the same scale as the other operand. The value is extended with zeros.

Multiplication

The precision and scale of the result on mainframe systems are determined by the following formulae:

$$P = \min (M, P_1 + P_2)$$

$$S = \min (M, S_1, + S_2)$$

Division

The precision and scale of the result on mainframe systems are determined by the following formulae:

$$P = M$$

$$S = \max (0, M - P_1 + S_1 - S_2)$$

Unpacked Decimal Operands

Unpacked Decimal operands are converted to Decimal data type and processed accordingly (see the sections **Decimal Operands** above as well as the section **Data Type Conversion** earlier in this chapter).

Single Precision Floating Point Operands

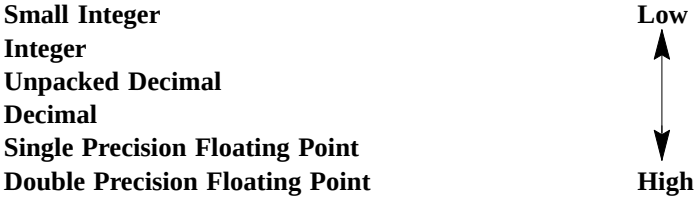
Single Precision Floating Point operands are converted to data type Double Precision Floating Point and processed accordingly. For details, see the section **Data Type Conversion** earlier in this chapter.

Double Precision Floating Point Operands

If a diadic operator has two operands of the data type Double Precision Floating Point, the result is also of this data type. Operations are performed using the floating point instructions of the underlying hardware.

Mixed Operands

If a diadic operation has two operands which are not of the same data type, one of the operands is converted to the data type of the other. Conversion is always done to a “higher” data type in the following order:



Assignments and Comparisons

All operations in SQL can be broken down to two basic operations:

assignment of values and
comparison of values.

Values are assigned during the processing of FETCH, UPDATE, INSERT and single-row SELECT statements. Comparison of values take place during the execution of statements that contain predicates. Both assignment and comparison operations have two operands. An assignment operation has a receiving operand and a sending operand. In an assignment, the value of the sending operand is given to the receiving operand. A comparison operation has two comparison operands whose values are compared with each other. For both assignment and comparison operations, both operands must have a comparable data type.

Assume operand 1 has data type x. Operand 2 has a comparable data type only if its data type is:

- x or
- a data type which can be converted to x or
- a data type to which x can be converted, unless operand 1 is the receiving operand of an assignment operation. In this case, the data type is fixed and can not be changed.

In general, this means that data types character-string, binary, and numeric are not comparable.

If both operands have different but yet comparable data types and a conversion has to be performed, this is always done from a ‘lower’ data type to a ‘higher’ data type (see **Mixed Operands** above). For detailed information on data type conversion rules refer to the section **Data Type Conversion** in this chapter.

Character-String Assignment

When a value of data type character-string is assigned to a value recipient (a value recipient is either a host variable, or a column), the length of the value and the length with which the value recipient has been defined are compared.

- If both lengths are the same, the value is simply assigned to the recipient and after the assignment, the value and the value of the recipient are identical.
- If the length of the value is smaller than the length of the recipient, the value is padded with blanks.
  - If the length of the value is greater than the length of the recipient, the value is truncated. If the INDICATOR variable was specified, it will show the number of truncated characters.

Numeric Assignment

When a value of data type numeric is assigned to a recipient, data type conversion is performed when the data types of the value and the recipient are not identical. The data-type conversion rules are described in the section **Data Type Conversion** in this chapter.

Binary Assignment

When a value of data type binary is assigned to a value recipient (a value recipient is either a host variable or a column), the length of the value and the length with which the value recipient has been defined are compared.

- If both lengths are the same, the value is simply assigned to the recipient and after assignment, the value and the value recipient are identical.
- If the length of the value is greater than the length of the recipient, then an error condition is raised.
- If the length of the value is smaller than the length of the recipient, the missing most significant digits of the value are appended with the value ’0’.

If the application program is a remote client and Adabas SQL Server resides on a server machine where normally during client/server communication, ASCII/EPCDIC and/or byte swapping conversions would be induced, for such host variables these conversions are suppressed. It is up to the host program to interpret the contents of such host variable. For further information refer to the *Adabas SQL Server Programmer’s Guide*, chapters: **Dynamic SQL** and **Embedding SQL Statements in Host Languages**.

Character-String Comparison

The comparison of two values of data type character-string, is performed by comparing each corresponding character in each string. If the two strings do not have the same length, the shorter one of the two is appended with as many blanks as necessary, so that both strings have the same length. Note that the padding is done with the appropriate environment-dependent hexadecimal representation for a blank (e.g. x'20' for an ASCII environment and x'40' for an EBCDIC environment) and that padding is either to the right or to the left, depending on the underlying hardware architecture.

- Two values of data type character-string are equal if and only if both strings are empty (i.e. have a length of zero), or every corresponding character is the same. The comparison is done either from left to right or from right to left depending on and according to the underlying hardware architecture.
- Two values of data type character-string are unequal if at least one corresponding character is found to be unequal. The order of two unequal character-string values is determined by the first unequal character found during the comparison process (either from the left or from the right depending on the underlying hardware architecture). The order is then determined by the EBCDIC or ASCII collating sequence.

Numeric Comparison

The comparison of two values of data type numeric is performed following the normal algebraic rules taking the sign into account.

Example: -5 is less than -3

Numeric comparison is always done between two values of the same data type. If two numeric values do not have the same data type, data type conversion is performed as described in section **Data Type Conversion** in this chapter.

Binary Comparison

The comparison of two values of data type binary is performed by comparing each corresponding bit digit in each value. The two values are equal if every corresponding digit is identical.

If the two values are of different lengths, then the most significant missing digits of the shorter value are appended with the value '0'.

The comments regarding host variables and binary assignment, as described above, also apply to comparison.

Query Expression

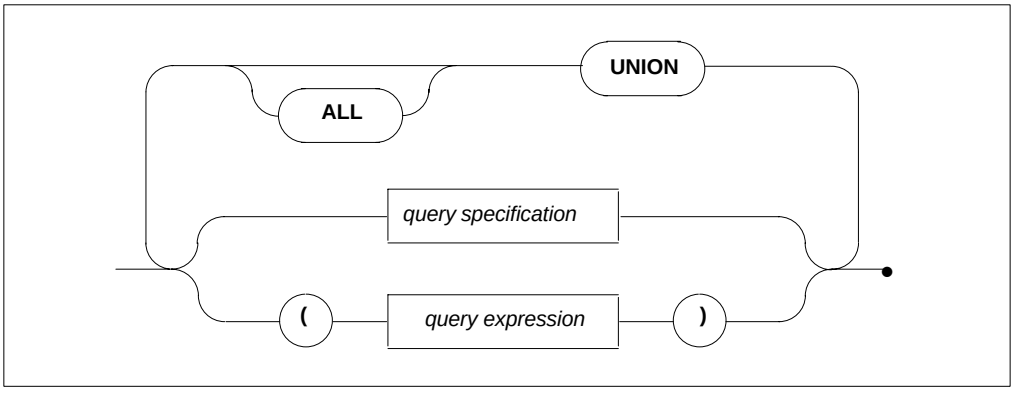
Function:

A query expression is an expression involving one or more query specifications connected using the UNION operator. It is used exclusively in a DECLARE CURSOR statement.

Invocation:

Common Element of a DECLARE CURSOR statement.

Syntax:



ALL	signifies that duplicate rows originating from different UNION operands are to be retained.
UNION	is a diadic operator which takes specifications of resultant tables as its operands, be they query specifications or deeper nested query expressions.
query specification	is the basic element of a query expression. It specifies a resultant table derived from a query.
query expression	another query expression may be specified.

Description:

A query expression specifies a resultant table made up of the possible UNION of several resultant tables as specified in corresponding query specifications. In its simplest form, a query expression can consist of just a single query specification. However, it is possible to add on subsequent resultant tables to this initial query specification with the aid of the UNION operator in order to produce a larger result.

The result of a UNION operation with two base tables is a resultant table which contains all rows belonging to either or both the operands.

Conceptually, the result of the UNION operation is formed by establishing a resultant table which contains all rows from both operands and then eliminating any duplicates. The specification of DISTINCT in any of the query specifications is irrelevant as duplicates are eliminated anyway.

By specifying ALL in the expression, rows duplicated by the two operands are retained. Specifying DISTINCT in the query specification is therefore significant.

As a consequence, the use of parentheses when ALL is either always specified with each UNION operator or never specified within the query expression is completely superfluous. However, if the ALL qualifier is only partially used, then the order of evaluation determines the final result and hence, the use of parentheses may be significant.

Query expressions specified within parentheses are evaluated first and thereafter the order of evaluation will be from left to right.

When a UNION operator is specified, then the columns of the resultant table do not have derived column labels.

Limitations:

The two operands must be UNION-compatible, i.e, the derived column lists of the two operands must be of the same format. Hence, each derived column list must have the same number of derived columns and each derived column must be of the same data type as its corresponding derived column in the derived column list of the other operand.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

None.

Example:

To list all cruise IDs for any contracts that require final payment or start before the 30th December 1991, the following syntax applies.

```
SELECT cruise_id
     FROM cruise
     WHERE start_date < = 19911230
UNION
SELECT id_cruise
     FROM contract
     WHERE date_payment < = 19911230;
```

Row Amendment Expression

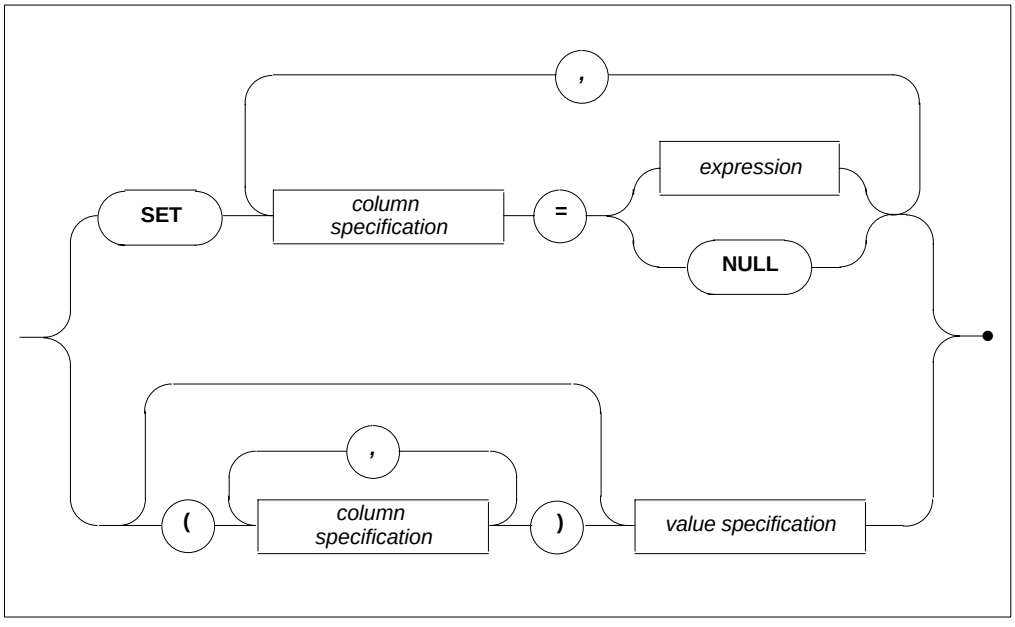
Function:

The row amendment expression specifies values which are to be assigned to columns of the table/view which is to be amended.

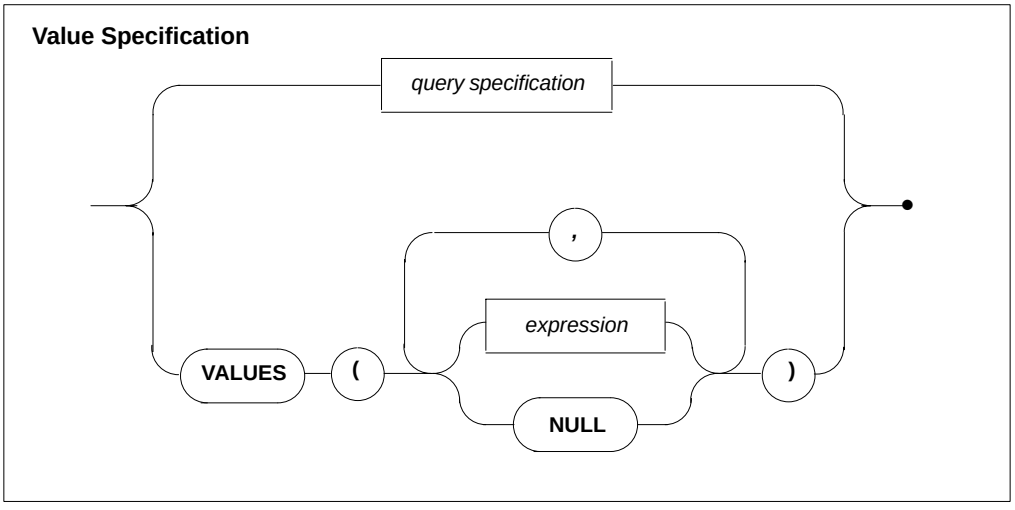
Invocation:

Common element of an UPDATE or INSERT statement.

Syntax:



column specification	refers to a column of the target table and thus signifies that this particular column is either to have an explicit value assigned to it (in the case of INSERT) or is to be updated.
expression	defines the value which is to be assigned to the specified column.
value specification	defines the values to be assigned to specified columns.



query specification	defines a resultant table which will be used as a source of values for assignment to the specified columns.
expression	defines the value which is to be assigned to the specified column.

Description:

The row amendment expression is part of either an UPDATE statement or an INSERT statement. Its function is to specify values which are to be assigned to columns of the table which is to be amended. This table is referred to as the target table.

SET/VALUES Formats

One of two formats may be used depending upon the options and the statement concerned. Both formats are equivalent.

- The SET format (upper branch of the main diagram) is normally part of the UPDATE statement. However, when compiled in Adabas SQL Server mode, it may be used within an INSERT statement. Only a single row may be specified with this format.
- The VALUES format (lower branch of the main diagram) itself can take one of two forms. Either explicit values can be specified in a similar manner to the set format or a query specification may be given. The VALUES format is normally part of an INSERT statement. However, when compiled in Adabas SQL Server mode it may be used within an UPDATE statement, in which case a query specification is not permitted.

Column Values

A column can be specified with a default value, upon table creation. Default values are irrelevant within an UPDATE statement. Therefore, columns of the target table which are not specified in an UPDATE statement remain unaltered. However, should a column not be explicitly specified as a target column within an INSERT statement, then any default value will be assigned. Should there not be a default value, then a compilation error will be generated. In the case of columns which support the NULL value and where an explicit default has not been specified, then the default is assumed to be NULL. Otherwise, an explicit default may have been specified as an appropriate literal or the underlying Adabas default for the column has been specified.

Should the target table in fact be a view, any columns of the underlying base table which are not enclosed within the view definition, will also be assigned the default value upon insertion into the view.

Columns may only be referred to in a numeric expression when the row amendment expression is contained in an UPDATE statement. The actual value of the column in the row currently under consideration is then taken. The data type of the numeric expression must be comparable to that of the specified column. If the target column supports NULL then it may be assigned the value NULL.

If the column specification list is omitted, then all columns of the target table require a corresponding value. Otherwise, the number and type of the values specified must correspond with the column specified in the list. Likewise, if a query specification is given, each column of its derived column list must match the corresponding column in the list.

If a query specification is given as part of an INSERT statement, then as many rows are inserted as are returned by the query specification. The query specification may not reference the target table.

INSERT/UPDATE – Subtables

INSERT or UPDATE statements may operate on Level 1 or Level 2 tables. In general this works like any other table with the following restrictions:

- Within an UPDATE statement, columns which are of a lower level than the target table can appear as target columns. However, at run time, the actual value which is to be assigned to the column must be equal to its current contents. It is, therefore, not physically possible to change the value of a lower level column from within the higher level target table. Such columns, therefore, serve the purpose of enforcing referential integrity.
- Updating of a column of the same level as the target table is also permitted. It should be noted, however, that should a lower level table be dependent upon that column, then changing its contents will also be reflected in the subtable. This enforces referential integrity and is equivalent to a cascaded update operation.
- Within an INSERT statement, columns which are of a lower level than the target table can appear as target columns. However, at runtime the actual value which is to be assigned to the column must be equal to its current contents. Such columns, therefore, serve the purpose of enforcing referential integrity.
- Within an UPDATE statement, during execution, should it be determined that the target column is a rotated column and that the target value is equivalent to the Adabas default value and the column is defined as having suppression equal to on, then such an update is rejected.

Use of Special Register SEQNO

Although the explicit use of the special register SEQNO is not permitted within the row amendment expression, column specifications which are based upon it are, under certain circumstances.

- Within an UPDATE statement, such columns can appear as target columns. However, at runtime the actual value which is to be assigned to the named SEQNO column must be equal to its current contents. It is, therefore, not possible to actually change the value of a named SEQNO column.
- Within an INSERT statement, a Level 0 named SEQNO column can be a target column and can be assigned a value. This value must lie within the permitted range of values for such columns. Such a column serves the purpose of representing the underlying Adabas ISN.

  The direct assignment of a value to a Level 1 or Level 2 named SEQNO column is not permitted. However, within an INSERT statement, targeted at a Level 1 or Level 2 table, lower level named SEQNO columns may be specified as target columns. In this case, the value which they are assigned must equal values which already exist.

  When the target table is a Level 1 table, such a column therefore serves the purpose of helping to identify the Adabas row into which a new Adabas MU or PE field is to be inserted.

  When the target table is a Level 2 table, such columns serve the purpose of helping to identify the Adabas row and the subsequent Adabas PE into which a new Adabas MU field is to be inserted.

Limitations:

Within an UPDATE statement, a query specification must not be used.

Within an INSERT statement, expressions may not reference any columns. Additionally, numeric expressions can not contain any operators.

The query specification may not reference the target table or any of its columns, whether directly or when hidden by a view reference.

A column may only be assigned the NULL value if it supports it.

ANSI Specifics:

Within an UPDATE statement, the VALUES format is not permitted. Within an INSERT statement, the SET format is not permitted.

Adabas SQL Server Specifics:

Within an UPDATE statement, the VALUES format is permitted. Within an INSERT statement, the SET format is permitted.

Example:

To insert a new record into the cruise table using the SET format of the row amendment expression, the syntax is as follows.

```
INSERT INTO cruise
SET cruise_id = 1234,
     start_date = 19920925,
     start_time = 12,
     end_date = 19921206,
     end_time = 14,
     start_harbor = 'ACAPULCO',
     destination_harbor = 'LIVERPOOL',
     cruise_price = 2050,
     bunk_number = 7
     bunks_free = 10
     id_yacht = 146,
     id_skipper = 244,
     id_predecessor = 5037,
     id_successor = 5039;
```

To increase all cruise prices by 100 and delay the start times of all cruises by 2 hours using the VALUES format of the row amendment expression, the following syntax applies.

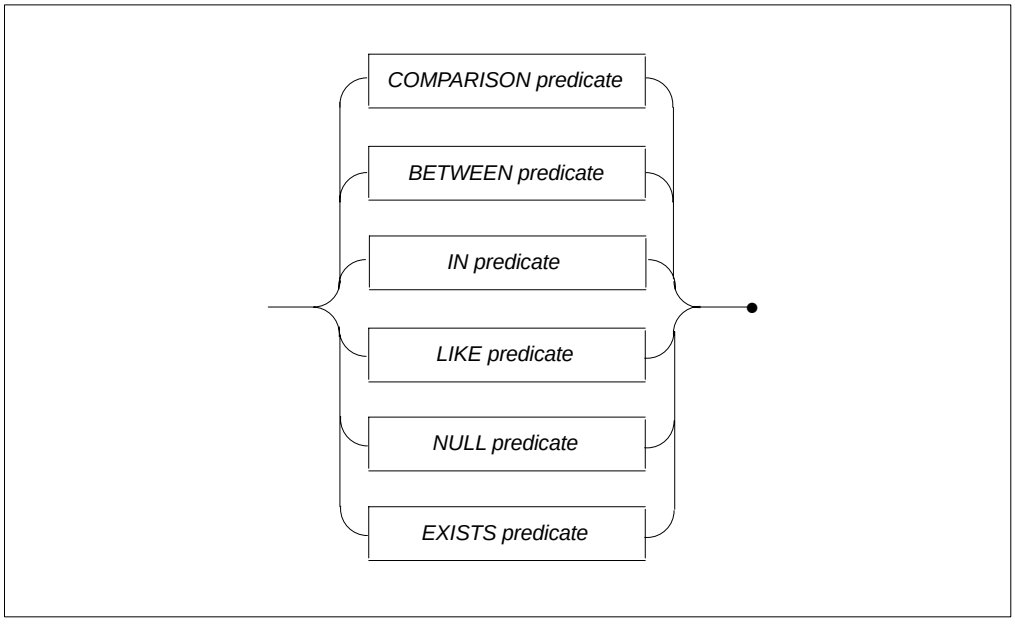
```
UPDATE cruise
( cruise_price, start_time )
     VALUES ( cruise_price+100, start_time+2 ) ;
```

To insert a new row into the table cruise by supplying the data for the rows cruise_id and start_date from the table contract, the following syntax applies for using a query specification in the row amendment expression.

```
INSERT INTO cruise ( cruise_id,start_date)
     SELECT id_cruise,date_reservation
          FROM contract
          WHERE contract_id=2007 ;
```

Predicates

A predicate is a tri-state (true, false, unknown) boolean expression which constitutes a search term contained within a search expression. A predicate can take one of six forms as described below. Predicates which necessitate the use of comparison operations obey the rules as defined in the section **Expressions**, **Assignment and Comparisons** earlier in this chapter.



BETWEEN Predicate

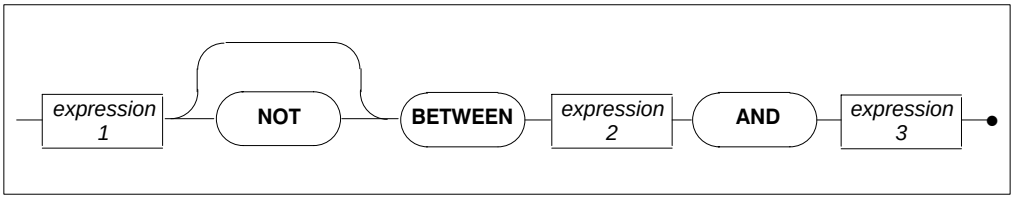
Function:

This predicate checks to see if a specified value lies within the range defined and returns a tri-state boolean result.

Invocation:

One of the six predicates which constitute a search term.

Syntax:



expression 1	is a valid expression as described in the section **Expressions** in this chapter.
NOT	is an operator which negates the result of the predicate.
expression 2 & 3	each is a valid expression as described in the section **Expressions** in this chapter.
AND	is not to be confused with its use as a boolean operator. AND simply separates expressions 2 and 3.

Description:

The BETWEEN predicate checks if the value specified by expression 1 lies within the range specified by the values derived from expression 2 and expression 3 respectively. As such, it is entirely equivalent to the following pair of COMPARISON predicates:

```
expression1 BETWEEN expression2 AND expression3

(expression1 >= expression2) AND (expression1 <= expression3);
```

In fact, Adabas SQL Server processes the BETWEEN predicate as if it was expressed in this form.

The use of the NOT operator would simply negate the result of the boolean expression.

All expressions must have comparable data types. Should either of the expressions evaluate to NULL, then the predicate returns the tri-state value of unknown.

Limitations:

None.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

None.

Example:

The following syntax applies if we want to find all the cruise IDs that have a cruise price between and including 800 and 2000.

```
SELECT cruise_id
     FROM cruise
     WHERE cruise_price BETWEEN 800 and 2000 ;
```

COMPARISON Predicate

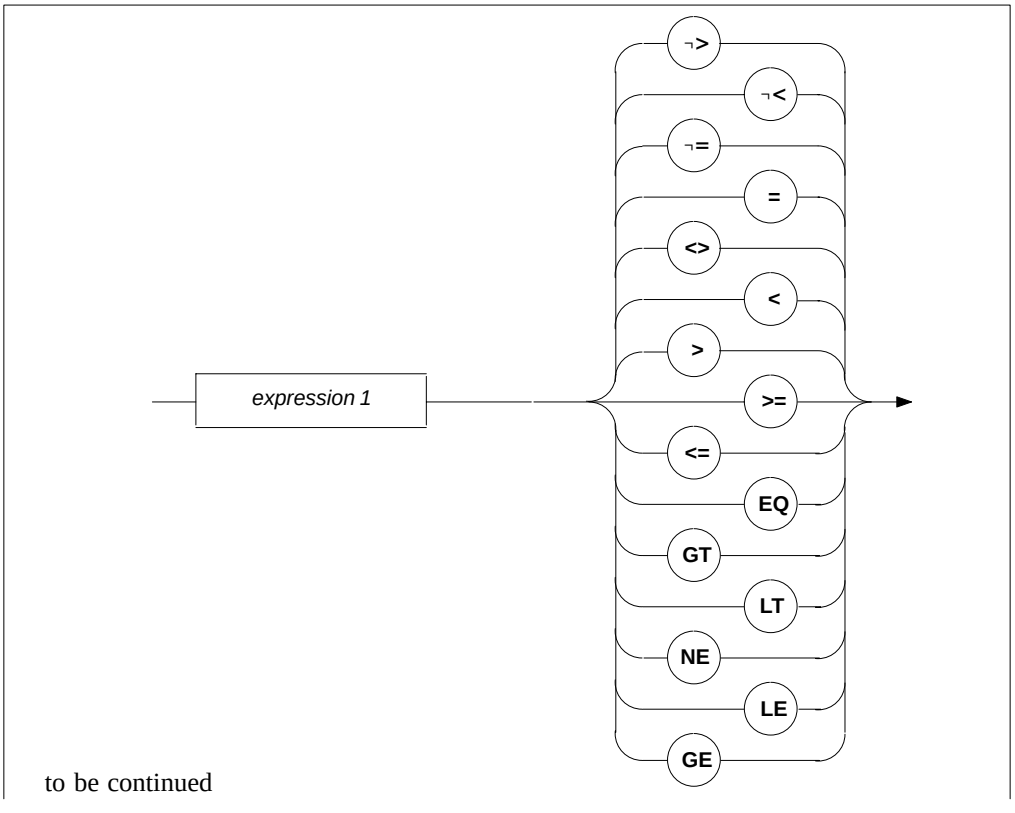
Function:

This predicate performs a comparison between two expressions and returns a tri-state boolean result.

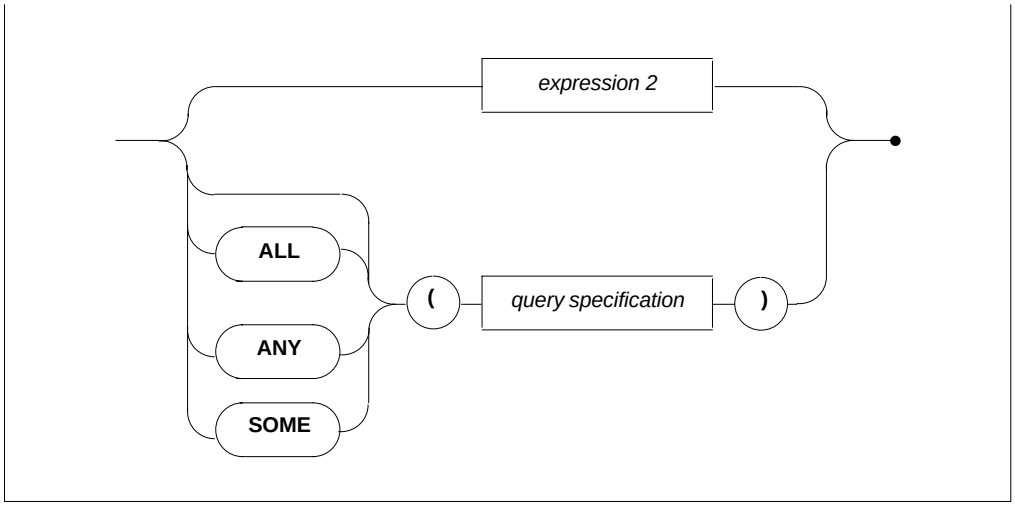
Invocation:

One of the six predicates which constitute a search term.

Syntax:



to be continued



expression 1	is a valid expression as described in the section **Expressions** in this chapter.
operator	is one of the possible operators which must be chosen in order to perform the desired comparison.
expression 2	is a valid expression as described in the section **Expressions** in this chapter.
query specification	is contained in parentheses and may be given instead of the *expression 2*.
ALL, ANY, SOME	are three keywords. One of these may optionally be specified in order to transform the comparison expression from being unquantified to being quantified.

Description:

As already stated both operands, expressions and query specifications, must have a comparable data type. Should either of the expressions evaluate to NULL, then the predicate returns the tri-state value of UNKNOWN.

A query specification may be given instead of the expression 2. Such a query is often referred to as a subquery or a subselect. Due to the comparable data type requirement, the subquery may only specify one resultant column in its derived column list. It is said to have a cardinality of one. When used within an unquantified COMPARISON predicate, the resultant table may only return one value, thus 'mimicking' a normal expression. A run time error will be returned should the subquery produce more than one result or no result at all. Naturally this is something which can not be checked at compilation time. Should the query return a value of NULL, then the predicate equates to unknown.

The operator specifies the actual comparison operation to be performed. There are various alternative representations for the operators, depending upon which mode is current, as shown below.

The use of one of the keywords ALL, ANY or SOME changes the nature of the subquery and makes the predicate quantified. The subquery may now return more values; it is no longer restricted to zero or just one. When you use ALL, the predicate equates to true if the comparison with expression 1 is true for all values returned by the subquery. When you use ANY, only one of the comparisons need be true for the predicate to be true. The keyword SOME is entirely equivalent to ANY.

Should any particular value equate to NULL, then the predicate returns the value UNKNOWN.

Strings can also be deemed to be greater or less than other strings. For example 'Swindon' < 'Swinton' would equate to true.

Limitations:

When a subquery is used in an unquantified comparison predicate, then that subquery can not contain a GROUP BY clause or a HAVING clause, as this would violate (in general) the requirement to return just one value. Likewise the subquery may not reference a grouped view as its source table.

ANSI Specifics:

ANSI only allows the following operator representations:

= > < <> <= >=

Adabas SQL Server Specifics:

In addition to the ANSI representations, Adabas SQL Server also allows the following operator representations:

EQ	GT	LT	NE	LE	GE

Example:

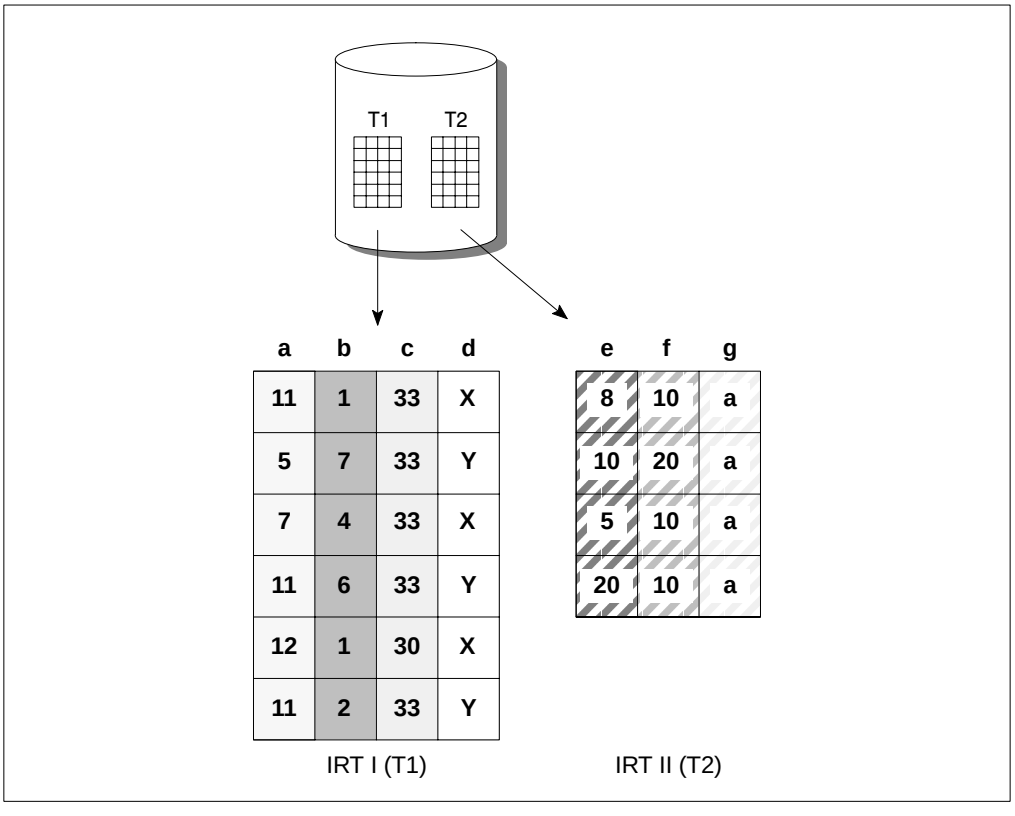
If used within an unquantified COMPARISON predicate, the subquery must only return one result. The following syntax applies when trying to identify cruises which are less expensive than the price for a cruise with Yacht ID no. 145.

```
SELECT cruise_id, destination_harbor, cruise_price
     FROM cruise
     WHERE cruise_price <
     ( SELECT MIN (cruise_price)
          FROM cruise
          WHERE id_yacht=145);
```

If used within a quantified COMPARISON predicate, the subquery may return more than one result. The following describes the step-by-step processing of a query with the respective intermediate resultant tables. The abstract example uses a base tables named T1 with columns named a, b, c and d and T2 with columns named e, f and g. The apparent ordering of the intermediate resultant tables is due to ease of representation rather than of any predetermined ordering of the resultant tables.

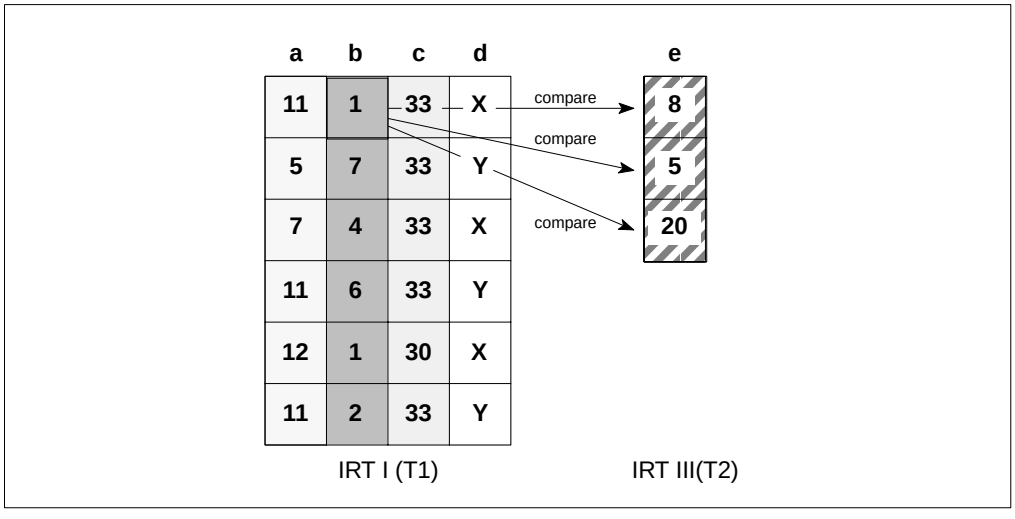
```
SELECT a,d
     FROM T1
     WHERE b < ALL
          (SELECT e
               FROM T2
                WHERE f = 10);
```

1 The first processing step of a query specification establishes an intermediate resultant table containing all columns and all rows as defined in the table list for T1, i.e.,IRT I. Thereafter, conceptually for each row of IRT I, the subquery is evaluated and as described in the section **Query Specification** i.e. IRT III is established from IRT II. This step needs to be performed for each occurrence of a row in IRT 1, as the result of the subquery may depend on values contained in IRT I. This occurs when the subquery contains an outer reference in its search condition.



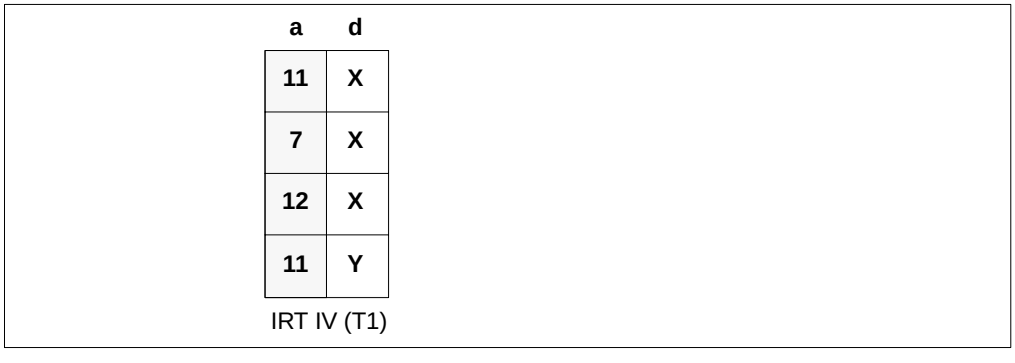
Processing Step 1

2 During the second processing step, the subquery has been established as intermediate resultant table (IRT) III. The comparison can now take place.



Processing Step 2

3 During the third processing step, all rows of T1 containing a value in column b which is smaller than ALL values in column e of T2 qualify for the intermediate resultant table IV which is the final result of the query.



Processing Step 3

EXISTS Predicate

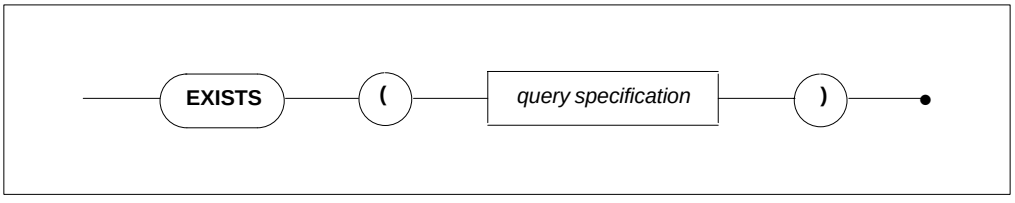
Function:

This predicate tests to see if a particular resultant table, as specified by the given subquery, actually exists, i.e., if any resultant rows were identified.

Invocation:

One of the six predicates which constitute a search term.

Syntax:



query specification is the subquery whose resultant table is to be tested for existence.

Description:

The subquery, in this instance, may specify a derived column list of any desired cardinality and of any number of resultant rows. This is because the nature of the resultant table is unimportant. What counts is whether the resultant table exists or not. Adabas SQL Server does not evaluate the derived column list.

If the resultant table does exist, then the predicate equates to true otherwise it is false. The predicate never equates to unknown.

In fact, all COMPARISON or IN predicates involving a subquery are internally transformed to an EXISTS predicate as shown above.

WHERE ‘op’ is any valid COMPARISON predicate operator:

- ```
  WHERE x op (SELECT y FROM t)
  WHERE EXISTS (SELECT * from t WHERE x op y)
  ```

Note:
The limitation that the subquery must result in only one value is lost in the transformation.
  ```

- ```
  WHERE x op ANY (SELECT y FROM t)
  ```
  ```
  WHERE EXISTS (SELECT *  FROM t WHERE x op y);
  ```
- ```
  WHERE NOT x op ALL (SELECT y FROM t)
  ```
  ```
  WHERE NOT EXISTS (SELECT * FROM t WHERE x NOT op y);
  ```
- ```
  WHERE NOT a op ALL (SELECT y FROM t)
  ```
  ```
  WHERE EXISTS (SELECT * FROM t WHERE x NOT op y)
  ```

Note:
The transformation for the second, third and fourth examples is only allowed if x and y can not result in the NULL value.

Limitations:

None.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

None.

Example:

To identify all cruises where the destination harbor is NOT a starting point for any other cruise, the syntax below applies.

```
SELECT cruise_id FROM cruise x
     WHERE NOT EXISTS (SELECT * FROM cruise
                       WHERE x.destination_harbor = start_harbor) ;
```
```

IN Predicate

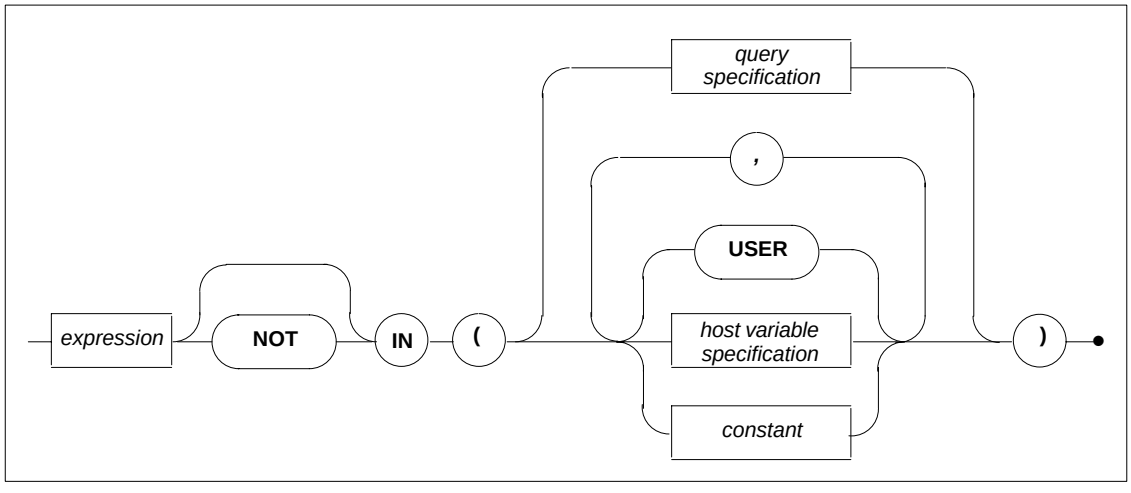
Function:

This predicate tests whether a given value is contained within a specified set of values and returns a tri-state boolean result.

Invocation:

One of the six predicates which constitute a search term.

Syntax:



expression	is a valid expression as described in the section **Expressions** in this chapter.
NOT	is an operator which negates the effect of the membership test.
host variable specification	is a valid single host variable specification and its value specifies a set member.
USER	is in the special register USER (see the section **Special Registers** later in this chapter).
query specification	is contained in parentheses and may be given instead of an explicit list separated by commas.

Description:

The IN predicate may be expressed as a search expression containing comparison predicates linked by the OR operator. In fact, Adabas SQL Server processes the IN predicate in this way.

- ```
  x IN ( 1,2,3 )
  x = 1 OR x = 2 OR x = 3
  ```

- ```
  x IN ( subquery )
  x = ANY ( subquery )
  ```

The *expression* and all members of the set, be they explicitly given or returned as the result of the subquery, must be of a comparable data type. Should either the *expression* or any of the set members evaluate to the NULL value, then the predicate returns the tri-state value of unknown.

The query specification follows the rules as given for subqueries within a quantified COMPARISON predicate. Hence, the subquery may only specify one resultant column in its derived column list, although many different values/rows may be returned.

String comparison follows the same rules as specified for the COMPARISON predicate.

Limitations:

None.

ANSI Specifics:

In ANSI compatibility mode, the use of the special register USER is not supported.

Adabas SQL Server Specifics:

None.

Examples:

To identify all skippers who are on cruises starting from BAHAMAS, PANAMA or TRINIDAD the following syntax applies.

```
SELECT id_skipper
    FROM cruise
    WHERE start_harbor IN ( 'BAHAMAS', 'PANAMA', 'TRINIDAD' );
```

To identify all customers who will be starting a cruise from MIAMI the following syntax applies.

```
SELECT id_customer
FROM contract
WHERE id_cruise IN ( SELECT cruise_id
    FROM cruise
    WHERE start_harbor = 'MIAMI' );
```

LIKE Predicate

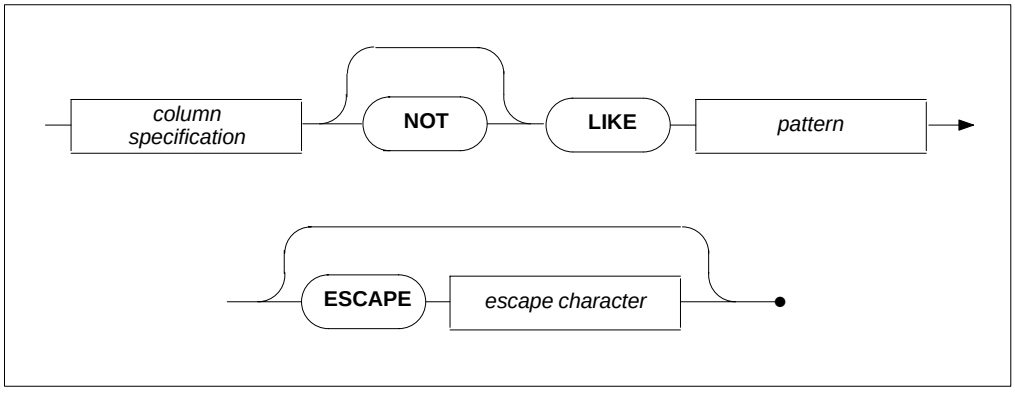
Function:

The LIKE predicate compares a column of a base table or view with a pattern. Wildcard characters may optionally be specified

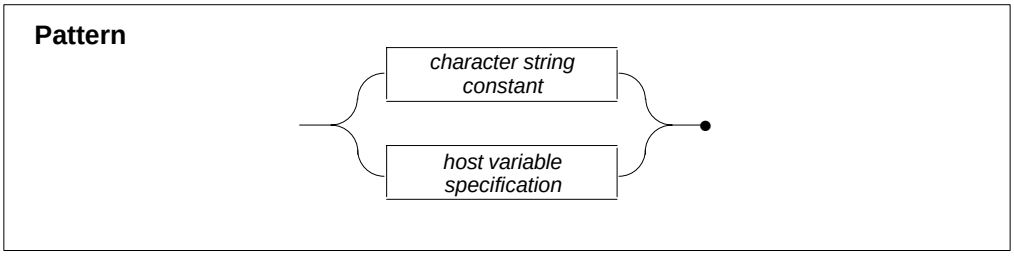
Invocation:

One of the six predicates which constitute a search term.

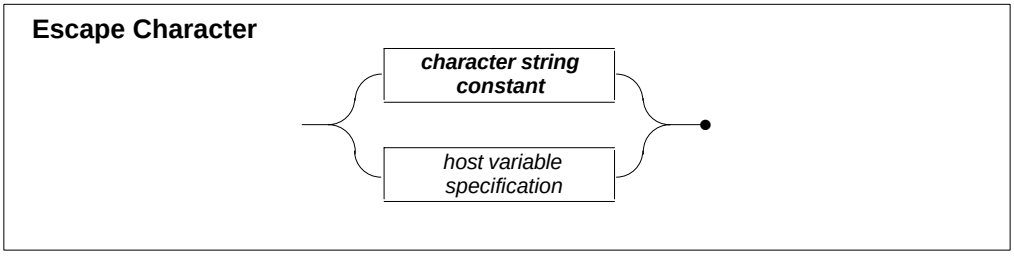
Syntax:



column specification	is a column of a base table or view which is to provide the value against which the comparison is to be made. The column must be of data type character-string.
NOT	is an operator which negates the result of the LIKE predicate.



pattern is the form to which the column must conform. It can be expressed as either a hard coded constant or a single host variable specification of the data type character string. The use of wildcard characters is supported.



escape character is a single escape character. The wildcard characters themselves can be considered in any pattern matching by preceding them with an escape character.

Description:

The LIKE predicate performs a comparison between the specified column value and a given pattern. Should a match be found, then the predicate equates to true otherwise false. Should the column or the pattern equate to a NULL value, then the predicate's result is unknown.

In general, for the predicate to equate to true, there needs to be a one-to-one match between the two strings. However, wildcard characters can be used in order to make the comparison more flexible.

The wildcard character '_'

It takes the place of any single character in the pattern. Should a particular position in the string.irish–times.i be of no significance, then it can be masked out by the use of an underscore character in the pattern.

For example, with a pattern of 'ABCDE', only 'ABCDE' will result in 'true'. However, a pattern of 'AB_DE' will not only give a true result for 'ABCDE' as before but also for 'ABZDE' or, in fact, for any string that is five characters long and starts with 'AB' and ends with 'DE'. Note the comparison of 'ABZZDE' would fail for this pattern as an extra character has been introduced.

The wildcard character '%'

It takes the place of zero or more characters in the pattern.

If the pattern were specified as 'AB%DE', then a column value of 'ABZZDE' would indeed give a true result as would a string of any length that started with 'AB' and finished with 'DE'.

If the pattern is not of an identical size to the column, no space padding takes place and so, no match will be found. This is opposite to a normal COMPARISON predicate.

For example, if the column first_name has provision for 10 characters and contains the value 'TIMOTHY' then the following COMPARISON predicate will evaluate to true:

```
WHERE first_name = 'TIMOTHY'
```

However, the following LIKE predicate will evaluate to false:

```
first_name LIKE 'TIMOTHY'
```

This is because no space padding takes place. The following two LIKE predicates would evaluate to true:

```
first_name LIKE 'TIMOTHY   '

first_name LIKE 'TIM%'
```

Note:
In the above case, the wildcard character % would also result in a row containing the value 'TIMMY ', for example, being found.

In theory, the pattern can be made as complex as required, with no limitations being placed on the mixing of wildcard characters.

Escape Character

If either or both of the wildcard characters were required to stand for their actual meaning, then an escape character must be specified. This is any single character which must precede either the ‘%’or the ‘_’ thus signifying that the following wildcard character is to be taken literally.

For example, if an exact match for the string ‘AB_DE’ was required and the escape character had been defined as ‘?’, then the pattern would have to be specified as ‘AB?_DE’.

Limitations:

Should the column reference a view, then this viewed column must be derived exclusively from a column of a base table. This applies to all three modes.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

None.

Example:

To find out if a person whose name ends with the characters ’ann’ is registered, the following syntax applies:

```
SELECT person_id
    FROM person
    WHERE first_name_1 LIKE ’%ann’;
```

NULL Predicate

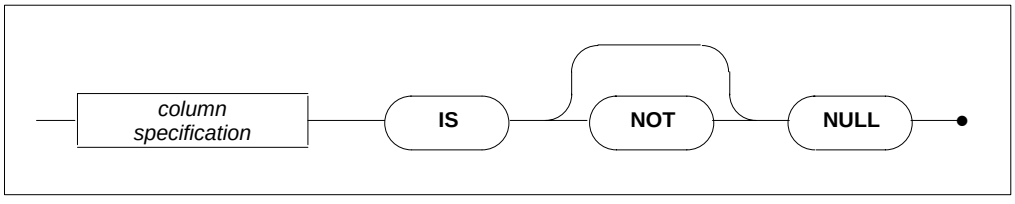
Function:

The NULL predicate tests a particular column to see if it contains the NULL value.

Invocation:

One of the six predicates which constitute a search term.

Syntax:



column specification	may reference any valid column even if it does not support NULL values.
NOT	is an operator which negates the result of the predicate.

Description:

This predicate tests to see if a given column holds the NULL value. As such, this predicate can only return either true (column IS NULL) or false (column holds a definite value). The result can never be unknown.

Limitations:

None.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

None.

Example:

To find out if any cruises were offered, for which no reservations have been made yet, the following syntax applies:

```
SELECT ID_CRUISE
     FROM CRUISE, CONTRACT
     WHERE CRUISE_ID = ID_CRUISE AND DATE_RESERVATION IS NULL;
```

Search Condition

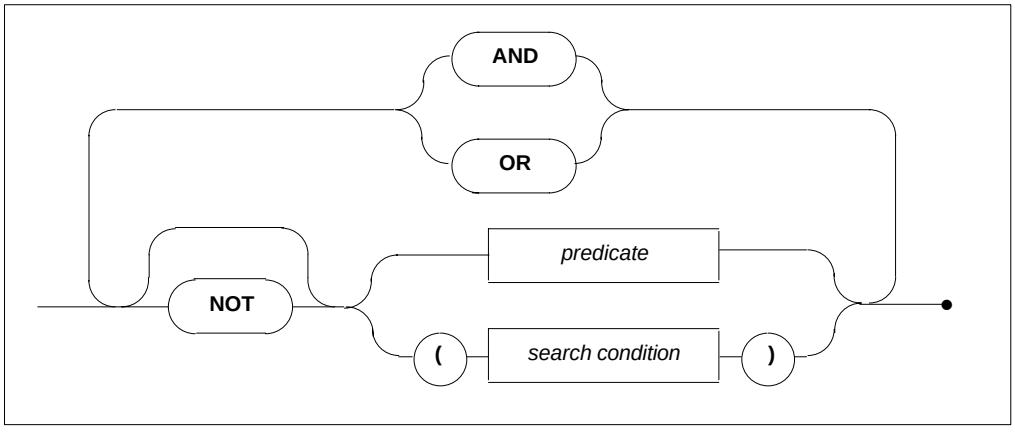
Function:

A search condition is a boolean expression of one or more predicates which defines whether a candidate row or group is to be included in the resultant table of the query, depending upon whether the condition equates to true.

Invocation:

A search condition may appear as the body of a WHERE clause in either a query specification or a searched DELETE or UPDATE statement and as the body of a HAVING clause.

Syntax:



predicate is the basic building block of a search condition and constitutes one of the possible ‘search terms’. All predicates equate to true, false or unknown.

search condition is a recursive construction enabling, in theory, search conditions of unlimited complexity. Such recursive constructions must be enclosed in brackets. As they are built up of predicates, search conditions also equate to true, false or unknown and constitute the other possible search term.

NOT	is an operator which negates the result of either the predicate or the included search condition.
AND/OR	are boolean operators which combine predicates and parenthesized search conditions to form a final search condition.

Description:

Should a search condition, constituting the body of a WHERE clause, equate to true, then the candidate row which is currently under consideration is deemed to be a member of the resultant table. Otherwise it is rejected.

Should the search condition actually constitute the body of a HAVING clause, then the candidate group is included if the search condition equates to true.

Individual search terms of the search condition can be combined using the boolean operators AND or OR. The order of precedence of the operators is NOT followed by AND followed by OR. Operators of the same precedence are evaluated from left to right. Search terms which are search conditions are evaluated first.

Because predicates can result in the state unknown, the operators are able to evaluate ‘tri-state logic’. The truth tables are as follows:

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN
AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN
OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Limitations:

None.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

None.

Example:

To search for the person IDs of all people who's surname starts with the letter 'W' and are not from the city of DERBY, the following syntax applies.

```
SELECT person_id
     FROM person
     WHERE surname LIKE 'W%' AND NOT city = 'DERBY';
```

To delete all contract data about people who made a reservation on the 4th of September 1991, where the cruise does not cost more than 2000 or the amount deposited is not more than 700, the following syntax applies.

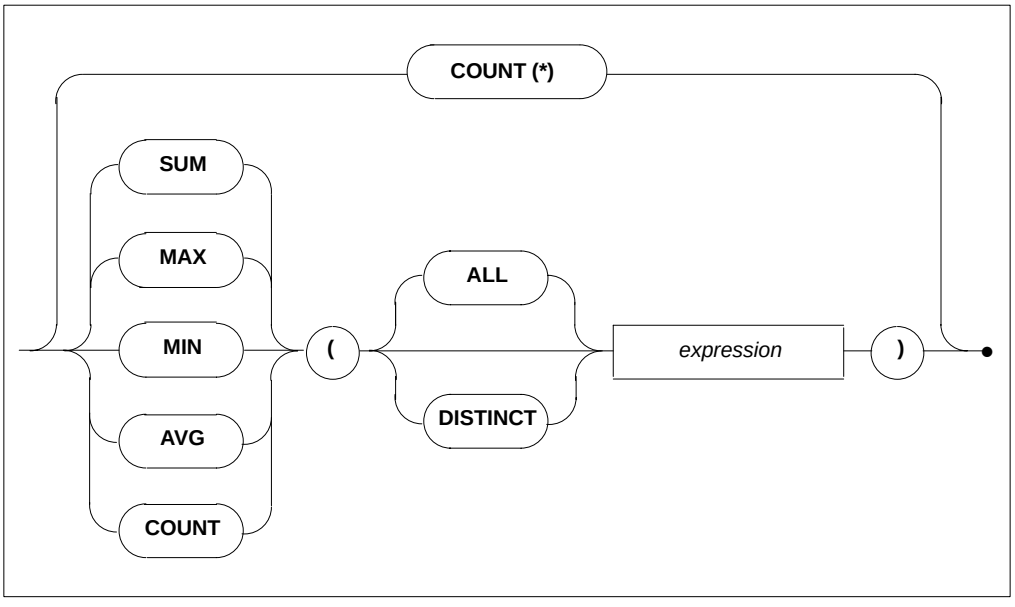
```
DELETE FROM contract
     WHERE date_reservation = 19910904
          AND ( NOT price > 2000 OR NOT amount_deposit > 700 );
```

To find the average price of all cruises that go to MARMARIS and that starts from RHODOS or FETHIYE and have a starting time of 16.00 or 17.00, the following syntax applies.

```
SELECT start_harbor, destination_harbor, start_time, AVG(cruise_price)
     FROM cruise
     WHERE destination_harbor = 'MARMARIS'
          GROUP BY start_time,start_harbor
          HAVING (start_harbor = 'RHODOS' OR start_harbor = 'FETHIYE')
          AND (start_time = 16 OR start_time = 17);
```

Functions

A function is an origin of a value; it optionally takes one or more arguments and calculates a result. The data types of the argument(s) and of the result depend upon each other and the particular function. Please refer to the section **The COUNT Function** for the limitations of usage.



Functions can not be nested. For example: `MAX (MIN(cruise_id))` is not valid. Under certain circumstances, an expression may only consist of a column specification. The combination of the function COUNT with ALL is not permitted.

In ANSI mode, a function which contains the keyword DISTINCT may not be placed in an expression which contains any diadic operators.

Invalid: `SELECT hv - MAX(DISTINCT cruise_id);`

Valid: `SELECT - MAX(DISTINCT cruise_id);`

Furthermore, in ANSI mode, a function whose arguments contain an outer reference may not contain any operators and may not be placed in an expression which contains any diadic operators.

The following example is correct:

```
SELECT COUNT (*) FROM cruise GROUP BY cruise_id
     HAVING cruise_id = ANY (SELECT id_cruise FROM CONTRACT
                        WHERE MAX(cruise.cruise_price) > price);
```

The above example would be incorrect if the last line looked as follows:

```
WHERE MAX(cruise.cruise_price * 90/100) > price
WHERE MAX(cruise.cruise_price) * 90/100 > price;
```

The SUM Function

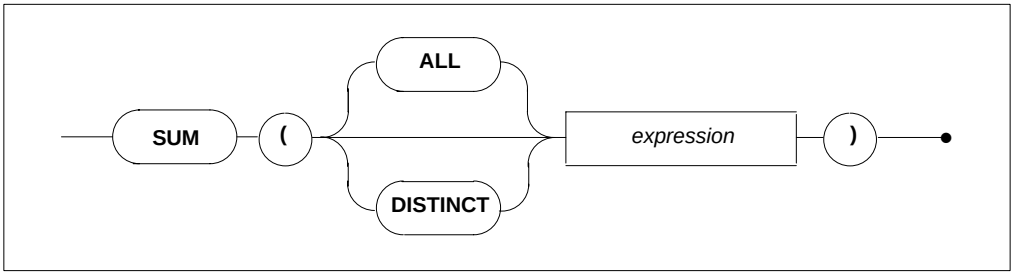
Function:

The SUM function returns the sum value of the set of values identified by the argument.

Invocation:

In the derived column list or in the HAVING clause of a grouped query specification.

Syntax:



expression is a valid expression as described in the section **Expressions** in this chapter.

Description:

In general, the argument of the function is an expression. The expression must not contain another function.

The data type of the argument is restricted to numeric data types. The length and data type of the result depend on the length and data type of the argument:

Data Type of Argument	Data Type of Result
Small Integer	Integer
Integer	Integer
Unpacked Decimal(P, S)	Unpacked Decimal(M*, S)
Packed Decimal(P, S)	Packed Decimal(M*, S)
Single Precision Floating Point	Double Precision Floating Point
Double Precision Floating Point	Double Precision Floating Point

* M denotes the maximum precision value of 27.

The result is the sum value based on the set of values specified by the argument. The set of values is derived from the rows of the intermediate resultant table as it has been established during the processing of a query and after applying the GROUP BY clause. NULL values are not included in the set of values. If the keyword DISTINCT is specified, all duplicate values are also eliminated from the set of values. If the set of values is an empty set, the result of the function is the NULL value. The keyword ALL has no influence on the way the function is evaluated.

Limitation:

At least one column must be specified.

ANSI Specifics:

If the keyword DISTINCT is used, the argument must be a column specification.

Adabas SQL Server Specifics:

If the keyword DISTINCT is used, the argument can be an arbitrary expression but with only one column referenced.

Example:

The following syntax applies when trying to identify the total turnover of all contracts serviced:

```
SELECT SUM(price)
     FROM CONTRACT;
```

The MAX Function

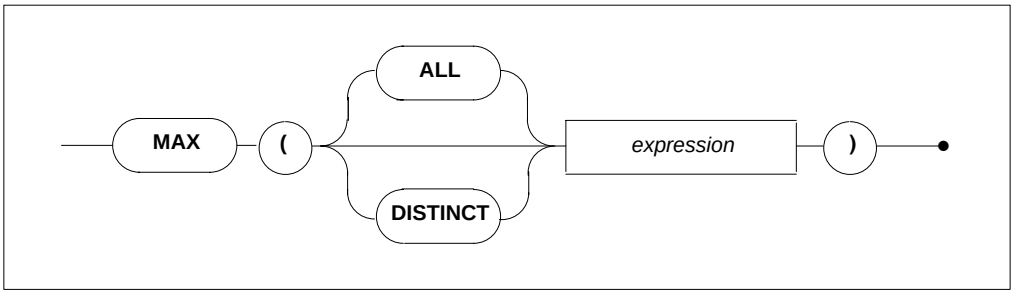
Function:

The MAX function returns the maximum value in the set of values identified by the argument. This usually applies to numeric values but can also apply to characters, in which case their ASCII values are evaluated.

Invocation:

In the derived column list or in the HAVING clause of a grouped query specification.

Syntax:



expression is a valid expression as described in the section **Expressions** in this chapter.

Description:

In general, the argument of the function is an expression. The expression must not contain another function.

The data type and length of the result are the same as the data type and length of the argument.

The result is the maximum value in the set of values indicated by the argument. The set of values is derived from the rows of the intermediate resultant table as established during the processing of a query and after applying the GROUP BY clause. NULL values are not included in the set of values. If the set of values is an empty set, the result of the function is the NULL value. The keywords ALL and DISTINCT have no influence on the way the function is evaluated.

Limitation:

At least one column must be specified.

ANSI Specifics:

If the keyword DISTINCT is used, the argument must be a column specification.

Adabas SQL Server Specifics:

If the keyword DISTINCT is used, the argument can be an arbitrary expression but with only one column referenced.

Example:

The following syntax applies when trying to identify the most expensive journey.

```
SELECT MAX(cruise_price)
     FROM CRUISE;
```

The following syntax applies when trying to identify the biggest difference between the cost of a cruise and the amount paid for a deposit.

```
SELECT MAX( price – amount_deposit )
     FROM contract ;
```

The MIN Function

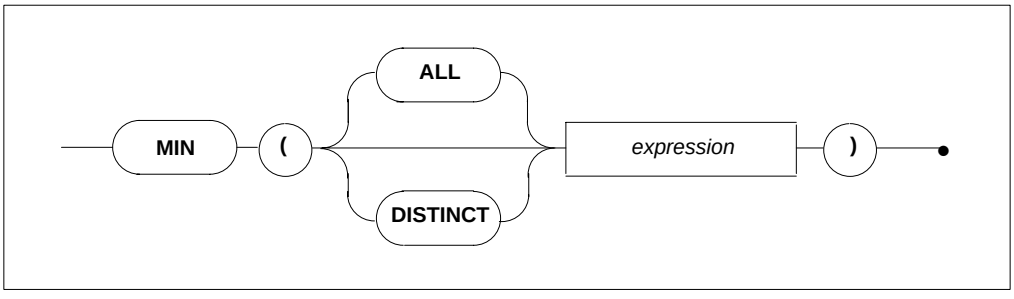
Function:

The MIN function returns the minimum value in the set of values identified by the argument. This usually applies to numeric values but can also apply to characters, in which case their ASCII values are evaluated.

Invocation:

In the derived column list or in the HAVING clause of a grouped query specification.

Syntax:



expression is a valid expression as described in the section **Expressions** in this chapter.

Description:

In general, the argument of the function is an expression. The expression must not contain another function.

The data type of the result is the same as the data type and length of the argument.

The result is the minimum value in the set of values indicated by the argument. The set of values is derived from the rows of the intermediate resultant table as it has been established during the processing of a query and after applying the GROUP BY clause. NULL values are not included in the set of values. If the set of values is an empty set, the result of the function is the NULL value. The keywords ALL and DISTINCT have no influence on the way the function is evaluated.

Limitation:

At least one column must be specified.

ANSI Specifics:

If the keyword DISTINCT is used, the argument must be a column specification.

Adabas SQL Server Specifics:

If the keyword DISTINCT is used, the argument can be an arbitrary expression but with only one column referenced.

Example:

The following syntax applies when trying to identify the least expensive journey.

```
SELECT MIN(cruise_price)
     FROM cruise
```

The following syntax applies when trying to identify the smallest difference between the cost of a cruise and the amount paid for a deposit.

```
SELECT MIN( price – amount_deposit )
     FROM contract ;
```

The AVG Function

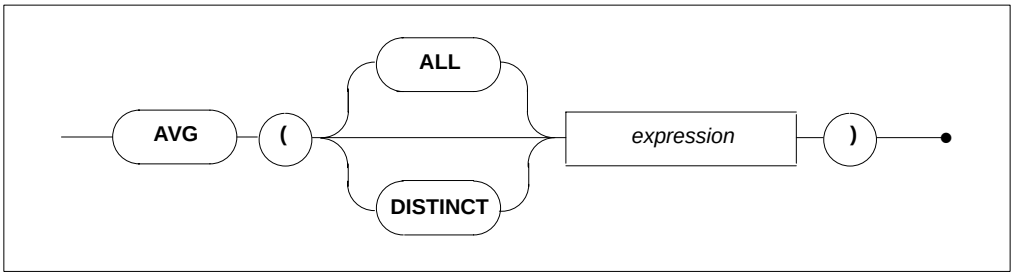
Function:

The AVG function returns the average value in the set of values identified by the argument.

Invocation:

In the derived column list or in the HAVING clause of a grouped query specification.

Syntax:



expression is a valid expression as described in the section **Expressions** in this chapter.

Description:

In general, the argument of the function is an expression. The expression must not contain another function.

The data type of the argument is restricted to numeric data types. The length and data type of the result depend on the length and data type of the argument:

Data Type of Argument	Data Type of Result
Small Integer	Integer
Integer	Integer
Unpacked Decimal(P, S)	Unpacked Decimal(M, M-P=S)
Packed Decimal(P, S)	Packed Decimal(M, M-P=S)
Single Precision Floating Point	Double Precision Floating Point
Double Precision Floating Point	Double Precision Floating Point

M denotes the maximum precision value of 27.

The result is the average value based on the set of values specified by the argument. The set of values is derived from the rows of the intermediate resultant table as established during the processing of a query and after applying the GROUP BY clause. NULL values are not included in the set of values. If the keyword DISTINCT is specified, all duplicate values are also eliminated from the set of values. If the set of values is an empty set, the result of the function is the NULL value. The keyword ALL has no influence on the way the function is evaluated.

Limitation:

At least one column must be specified.

ANSI Specifics:

If the keyword DISTINCT is used, the argument must be a column specification.

Adabas SQL Server Specifics:

If the keyword DISTINCT is used, the argument can be an arbitrary expression but with only one column referenced.

Example:

The following syntax applies when trying to identify the average cost of a cruise:

```
SELECT AVG(cruise_price)
     FROM CRUISE;
```

To select the average amount put down for a deposit when the TOTAL number of deposits placed is not to be taken into account, ONLY each individual price, the following syntax applies.

```
SELECT AVG(DISTINCT amount_deposit)
    FROM contract ;
```

The COUNT Function

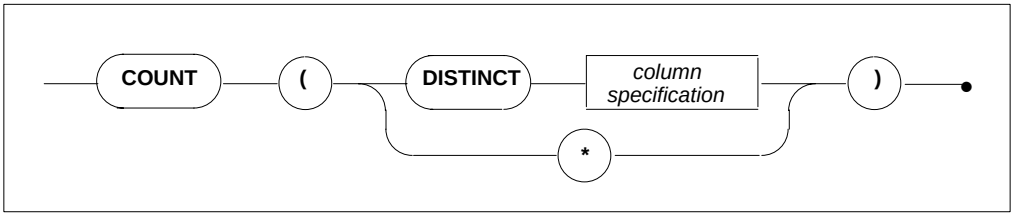
Function:

The COUNT function returns the number of rows or values in the set identified by the argument

Invocation:

In the derived column list or in the HAVING clause of a grouped query specification.

Syntax:



column specification is a column of a base table or view.

The argument of the function is either an asterisk '*' or the keyword DISTINCT followed by an expression.

Description:

The data type of the result is Integer.

If the argument is an asterisk, the result of this function is simply the number of rows contained in the query specification's resultant table.

If the argument of the function is an expression, then the expression must not contain another function. The result is the number of values in the set of values indicated by the argument. The set of values is derived from the rows of the intermediate resultant table as established during the processing of a query and after applying the GROUP BY clause. NULL values are not included in the set of values. If the set of values is an empty set, the result of the function is ZERO. If the keyword DISTINCT is specified, all duplicate values are eliminated from the set of values.

Limitations:

None.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

None.

Example:

To find out how many contracts have been signed as of today, the following syntax applies:

```
SELECT COUNT(*)
      FROM CONTRACT;
```

To find out how many DIFFERENT destination harbors are available, the following syntax applies.

```
SELECT COUNT (DISTINCT destination_harbor)
      FROM cruise ;
```

Special Registers

A special register is an origin of a value which is derived by Adabas SQL Server itself. This value does not depend on any data contained in the database. Currently, only the special registers USER and SEQNO are supported.

USER

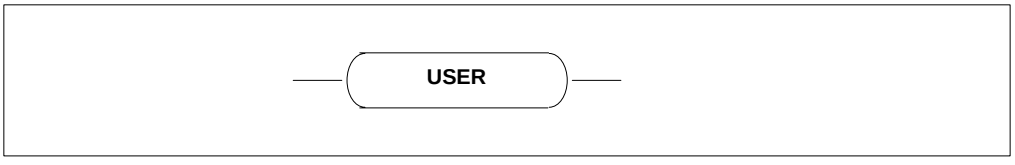
Function:

The special register USER contains the user ID as specified by the last implicit or explicit CONNECT statement.

Invocation:

The special register USER can be used in an expression as the origin of a value.

Syntax:



Description:

The data type of the special register USER is of type character. The register delivers a fixed length string of 32 characters. The string may be padded with spaces to the right.

Limitations:

None.

ANSI Specifics:

Currently a CONNECT statement is required to assign a value to the special register USER.

Adabas SQL Server Specifics:

The value can be explicitly set in a CONNECT statement.

Example:

To insert the values ‘5’ and ‘Harris’ into the table PERSONS, the following statement applies, provided that a prior CONNECT statement set the special register USER to ‘Harris’*:*

```
INSERT INTO persons (person_id, surname)
       VALUES (5, USER);
```

SEQNO

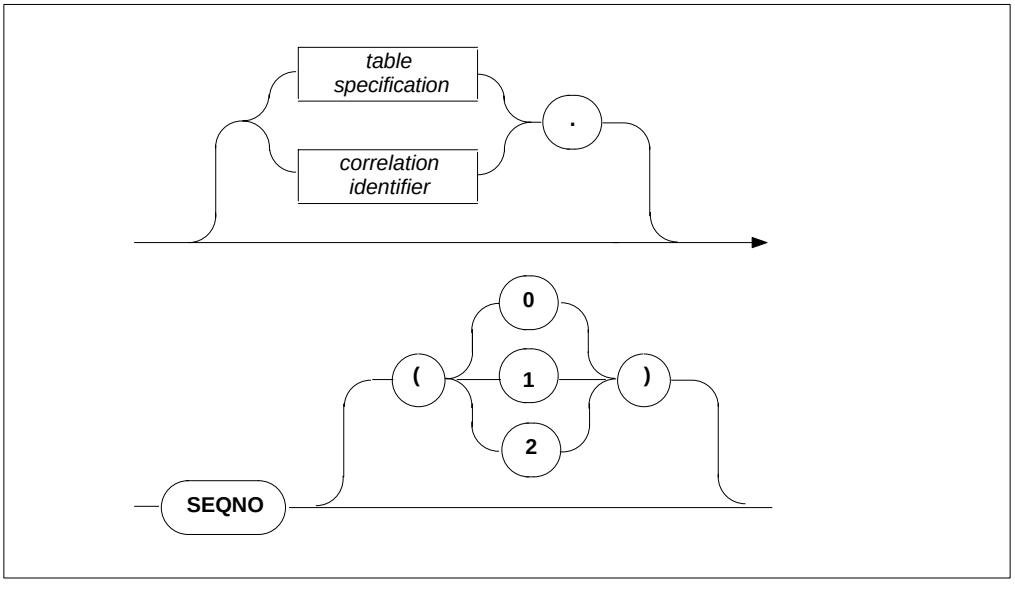
Function:

The SEQNO concept is Adabas SQL Server's way to reflect the Adabas record/field addressing technique. The special register SEQNO contains either the underlying Adabas table's ISN or, if it is not a Level 0 table, occurrence numbers of multiple-value fields or periodic groups. By specifying a level number, the SEQNO of a lower level can be accessed from within a higher level table.

Invocation:

An explicit SEQNO can be specified as a value source in an expression, in an ORDER BY clause or in a GROUP BY clause.

Syntax:



correlation identifier	is a valid correlation identifier which is in scope.
table specification	is a valid table specification which is in scope.

Description:

In the Adabas SQL Server environment, the special register SEQNO is considered to be a read-only column and can be accessed as such. A SEQNO returns an integer value and can be embedded in numeric expressions like any other integer data source.

Even if a SEQNO is not explicitly included within the SELECT list of a CREATE VIEW statement, it can still be accessed when accessing the view as long as the view is neither joined nor grouped.

The special register SEQNO may or may not be qualified with either:

- a correlation identifier or
- a table specification in an analogous fashion to a simple column specification.

The unqualified SEQNO can be used in those cases where it is possible to relate the SEQNO unambiguously to one table. This is the case when there is only one table in the FROM clause.

The qualification serves the purpose of identifying the particular table, from which the SEQNO is to derive its value.

If the SEQNO is not attributed with a table level number, then the level is zero by default.

Level 1 or level 2 tables have SEQNOs derived from the lower levels. These can be specified by supplying the appropriate level number.

Limitations:

The data type of the SEQNO is integer. However, some restrictions apply:

- a SEQNO can never be NULL
- a SEQNO can never be less than or equal to zero

A SEQNO is always unique.

A special register SEQNO must not be referenced as a target column in an UPDATE or INSERT statement. The named column SEQNO may be referenced as a target in an UPDATE or INSERT statement.

An asterisk in a SELECT list will return all columns of a table but not the special register SEQNO.

When included in the SELECT list of a CREATE VIEW statement, a special register SEQNO does not provide a derived column label. When used within an ORDER BY clause, a special register SEQNO does provide a derived column label.

ANSI Specifics:

The special register SEQNO is not part of the Standard.

Adabas SQL Server Specifics:

This is an Adabas SQL Server extension

Example:

To return the Adabas ISN the following syntax applies:

An unqualified special register SEQNO reference can be used if there is only one table in the FROM clause:

```
SELECT cruise_id, SEQNO from cruise ;
```

Qualification is mandatory when there is more than one table in the FROM clause:

```
SELECT cruise_id, contract_id, cruise.SEQNO, contract.SEQNO from cruise,
contract ;
```

Table Element

Function:

Table element defines the columns and table attributes of a table. All columns must be unique within a table. There may be two table attributes the same when one of those attributes is a table constraint element of type PRIMARY KEY or UNIQUE and the other is a table index element of type INDEX.

Invocation:

The Table Element specification is used as parts of the following statements:

Create Table/Create Table Description
Create Cluster/Create Cluster Description

Syntax:

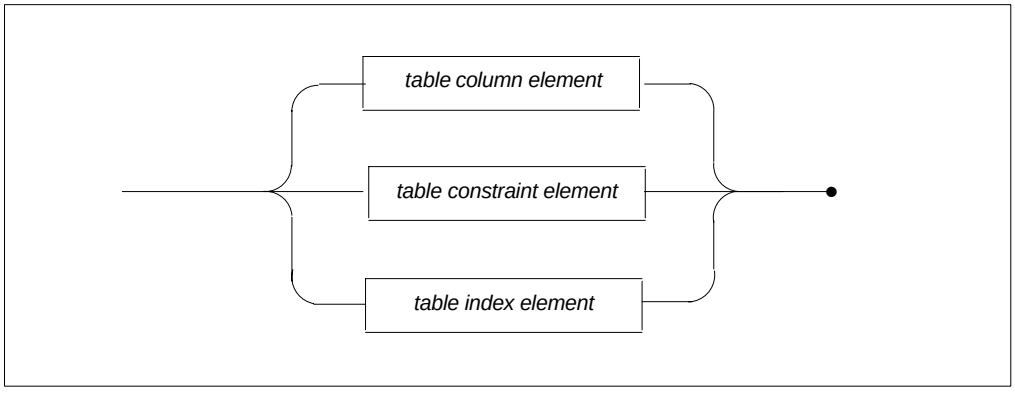


table column element	defines a column of a base table. A valid SQL table definition must contain at least one table column definition that is not of type SEQNO.
table constraint element	specifies a UNIQUE, PRIMARY KEY or FOREIGN KEY constraint.
table index element	Specifies an Index for the table. Table index element is not part of the ANSI SQL Standard.

Description:

The definition of foreign keys (part of Table Constraint Element) in the ANSI SQL standard, differs from that of Adabas SQL Server.

A table constraint element of type FOREIGN KEY may only be specified in the CREATE CLUSTER/CREATE CLUSTER DESCRIPTION statements.

Table Column Element

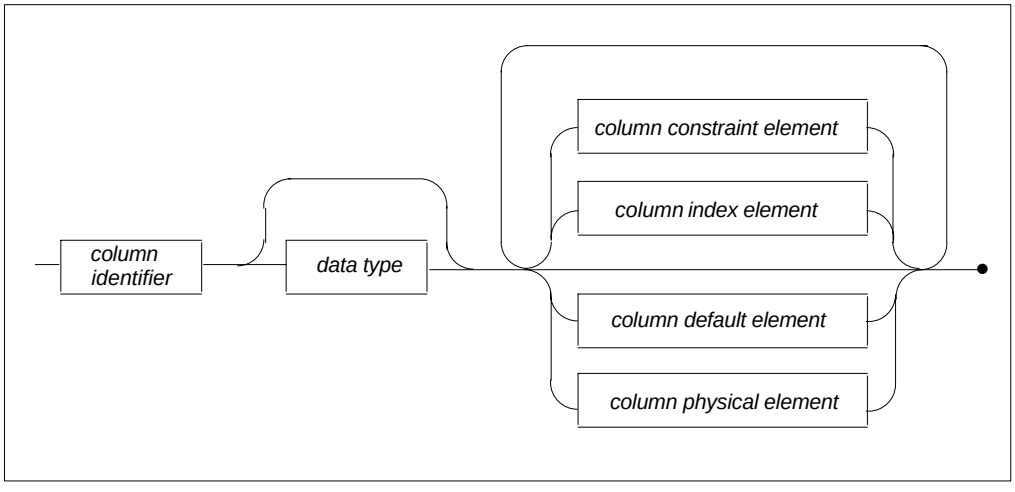
Function:

A table column element completely defines a column of a base table.

Invocation:

This element is part of the table element and of the alter add element (ALTER TABLE Statement).

Syntax:



column identifier	is a valid identifier for a column and must conform to the rules specified earlier in this chapter in sections **Identifiers** and **Column Specification**.
data type	specifies the data type of the column according to the rules specified below in the section **Data Type Definition**.
column constraint element	is optional and specifies constraints such as UNIQUE, NOT NULL, PRIMARY KEY, etc. For details see section **Column Constraint Element** below.
column index element	is optional and specifies an index for a column. For details see section **Column Index Element** below.

column default element	is optional and specifies what the default value for a column will be. There can be only one default value per column. For details see section **Column Default Element** below.
column physical element	is optional and describes the Adabas-specific information for each column, such as the short name, suppression, etc. For details see section **Column Physical Element** below.

Description:

The table column element specifies one column of a table with the attributes of this column (the attributes are constraints, indexes, default value, etc..).

As a minimum requirement for the CREATE TABLE, CREATE CLUSTER or ALTER TABLE statements, a column must be specified with the column identifier and the data type definition.

The minimum requirement of the CREATE TABLE DESCRIPTION or CREATE CLUSTER DESCRIPTION statement is the column identifier. Which must then be a valid Adabas short name, else it is required to specify the Adabas short name (part of column physical element).

By default, all columns that do not have the explicit attribute NOT NULL, have implicitly the attribute NULL.

In CREATE TABLE DESCRIPTION and CREATE CLUSTER DESCRIPTION statements, any unspecified attributes that belong to the underlying Adabas field are automatically generated. An exception to this is when dealing with the following attribute combinations:

- NOT NULL DEFAULT ADABAS
- NULL DEFAULT ADABAS
- NOT NULL SUPPRESSION
- NULL SUPPRESSION

Limitations:

The column identifier must be unique within a table.

The following must be unique within a schema:

- Index identifier (if specified), one will be generated when not specified.
- Constraint identifier (if specified), one will be generated when not specified.

If a statement type of CREATE TABLE or CREATE CLUSTER is specified then only 926 columns may be specified within one table. For CREATE TABLE DESCRIPTION and CREATE CLUSTER DESCRIPTION statements this limitation is lifted, as you may specify elements of a PE or MU in a rotated format.

If a column is of data type Character and the precision is greater than 253 characters, then the following must hold true:

- The column attribute NOT NULL is mandatory.
- The column may not have attributes from Column Constraint Element (other than the above) or Column Index Element.
- The column may not have the attribute SUPPRESSION.

The following attributes are not allowed to be combined :

- SUPPRESSION and FIXED
- NULL and NOT NULL
- NOT NULL and DEFAULT NULL

The table below shows which parts of table column element are optional for which statements.

STATEMENT	DATA TYPE Definition	Column Constraint Element	Column Index Element	Column Default Element	Column Physical Element
Create Table	Mandatory	Optional	Optional	Optional	Disallowed
Create Table Description	Optional	Optional	Optional	Optional	Optional(1)
Create Cluster	Mandatory	Optional	Optional	Optional	Optional(2)
Create Cluster Description	Optional	Optional	Optional	Optional	Optional(3)
Alter Table	Mandatory	Optional(4)	Optional	Optional	Optional(5)

(1) The SHORTNAME specification is mandatory for this statement.
(2) The SHORTNAME specification is not allowed in this statement.
(3) The SHORTNAME specification is mandatory for this statement.
(4) The NOT NULL attribute is allowed when combined with either DEFAULT ADABAS or SUPPRESSION.
(5) The only attributes allowed in this statement are NULL and SUPPRESSION.

ANSI Specifics:

The following elements are not part of the standard:

- Column Index Element
- Column Physical Element
- In Column Default Element the keyword ADABAS
- In Data Type Definition the keyword SEQNO

Adabas SQL Server Specifics:

None.

Example:

To create one column of our sample base tables (cruise) the following syntax is used:

```
CREATE TABLE cruise
  (cruise_id NUMERIC(8) INDEX cruise1 NOT NULL UNIQUE);
```

To create the same table, but with a named SEQNO column, which enables access to the underlying Adabas ISN, the following syntax is used:

```
CREATE TABLE cruise
  (cruise_id NUMERIC(8) INDEX cruise1 NOT NULL UNIQUE, sequence_no SEQNO);
```

Data Type Definition

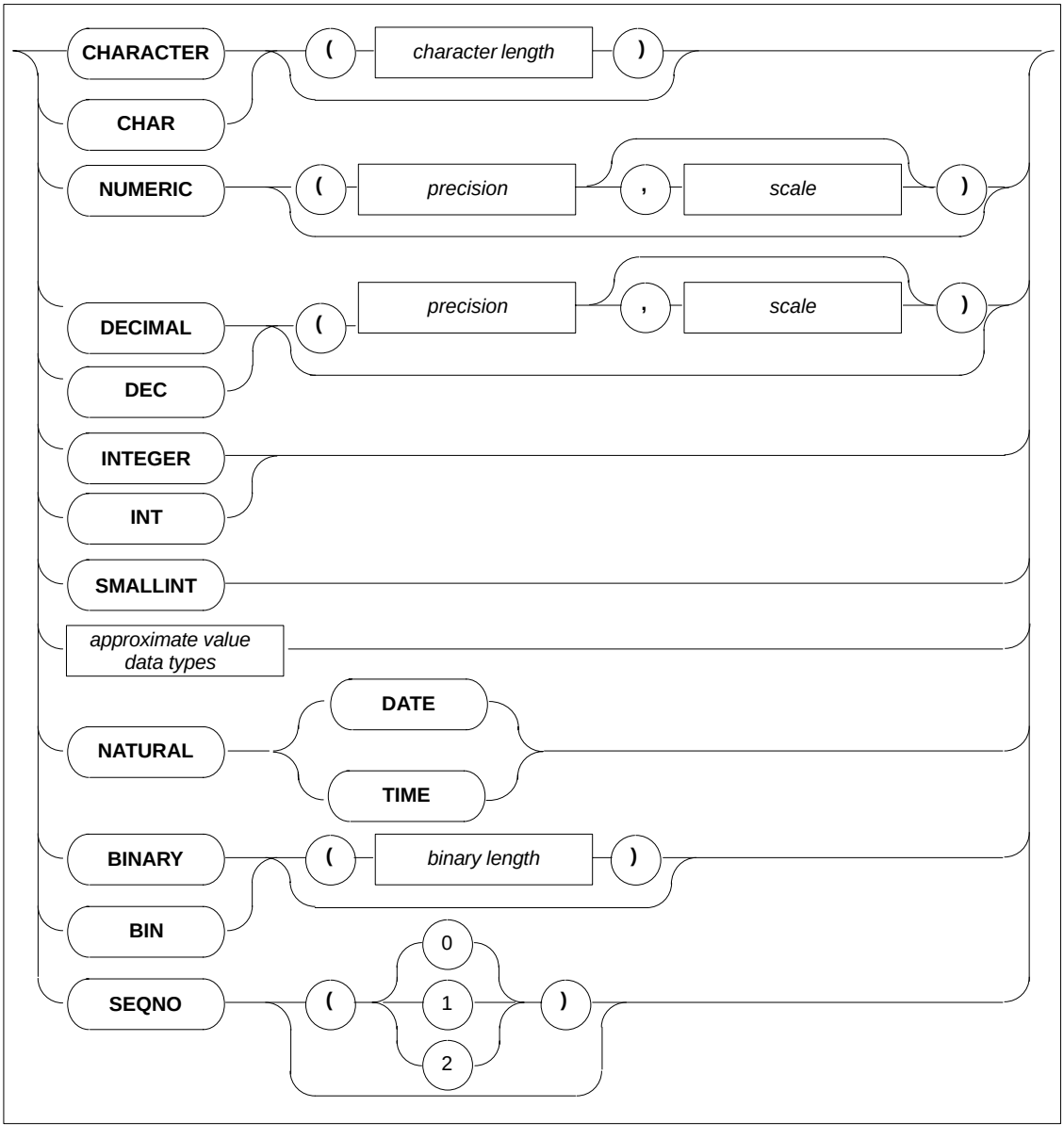
Function:

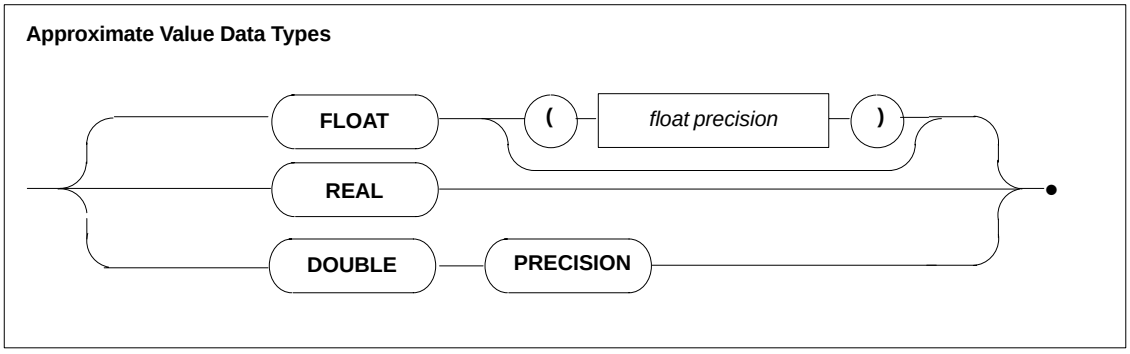
The data type definition specifies the SQL data type for a column.

Invocation:

The data type definition is part of the table column element.

Syntax:


CHARACTER
CHAR
(
character length
)
NUMERIC
DECIMAL
DEC
(
precision
,
scale
)
INTEGER
INT
SMALLINT
approximate value data types
NATURAL
DATE
TIME
BINARY
BIN
(
binary length
)
SEQNO
(
0
1
2
)



The precision of numeric and decimal data types and the length of the character and binary data types must, if specified at all, be in the range from 1 to the maximum length allowed for this data type (see table below for details).

The scale of numeric and decimal data types must, if specified at all, be in the range from zero to the maximum precision of this data type.

Description:

- **CHAR**
  CHARACTER

  determines the maximum length of a character column. If the length of this column is less than 253, then it will be mapped to a standard Adabas field of type alpha.

  If the length is greater than this limit, then a different mapping technique is required. Such a column is called a longalpha column. It‘s mapping platform-dependent:

  – UNIX/OpenVMS platforms: to an Adabas MU field
  – Mainframe environments: to either an Adabas LA field (if the Adabas version supports this feature), or like the UNIX and OpenVMS versions to a MU.

  This mapping is performed automatically by Adabas SQL Server. The maximum length of such a column is 16K (the same as the maximum possible for an LA). Nevertheless, the size of a longalpha column dependents on the maximum compressed record size, which again depends on the data storage blocksize.

For a longalpha column the following must hold true:

- The column attribute NOT NULL is mandatory.
- The column may not have attributes from the column constraint element (other than the above) or from the column index element.
- The column may not have the attribute SUPPRESSION.

- **INT or INTEGER**

  determines an integer.

- **SMALLINT**

  determines an small integer.

- **NATURAL DATE**

  defines a data type that is compatible to Natural's DATE format. This data type is an extension for use by Adabas ODBC Client only.

- **NATURAL TIME**

  defines a data type that is compatible to the normal NATURAL TIME format only, and not to the extended format which also contains the date. This data type is an extension for use by Adabas ODBC Client only.

- **REAL**

  determines a single precision floating point number.

- **DOUBLE PRECISION**

  determines a double precision floating point number.

- **FLOAT**

  determines a floating point number. If the float precision is less than 22, the data type is single precision floating point, otherwise double precision floating point.

- **BIN or BINARY**

  determines a binary column with the length indicated by the number of binary digits.

- **SEQNO**

  rather than being specified as a normal Adabas field with an associated data type, a column can be defined as a named SEQNO column. The SEQNO concept is the Adabas SQL Server's way to reflect the Adabas record/field addressing technique. The special register SEQNO contains either the underlying Adabas table's ISN or, if it is not a level 0 table, occurrence numbers of multiple-value fields or periodic groups. By specifying a level number, the SEQNO of a lower level can be accesses from within a higher level table. These fields can be accessed by assigning them a column name. Thereafter, they are accessed like any other column. There are some restrictions in their use, see section Row Amendment Expression earlier in this chapter.

The data type of such a named SEQNO column is integer. However, the value is always greater than zero.

The data type SEQNO does not generate a field in Adabas.

Note:
In the case of a PE data structure containing MU fields only, it is necessary to use an Adabas short name on the SEQNO(1) of the PE-subtable.

For details refer to sections **CREATE CLUSTER DESCRIPTION** and **SEQNO** for more details.

- **NUMERIC**
  DEC
  DECIMAL

  determines a packed or unpacked number. Precision constitutes the total number of digits and the default precision is 27 for all platforms. The scale is the number of digits to the right of the decimal point which must lie in the range of 0 to the precision.

  The following table shows:
  - how an SQL data type is translated into an Adabas format and length definition.
  - the default precision/length of each data type.

Adabas SQL Server Data Type	Precision		Adabas Format	Length in Bytes	Maximum Length in Bytes
	Max.	Default			
CHARACTER/CHAR	16381	1	A	N (1)	16381
INTEGER/INT	./.	./.	F	4	./.
SMALLINT	./.	./.	F	4	./.
REAL	./.	./.	G	4	./.
DOUBLE PRECISION	./.	./.	G	8	./.
FLOAT(1...21)	21	53	G	4	./.
FLOAT(22...53)	53	53	G	8	./.
BINARY/BIN	1008	1008	B	p/8 (2)(4)	126
DECIMAL/DEC	27	27,0 (3)	P	p/2+1 (4)	14
NUMERIC	27	27,0 (3)	U	p (4)	27

(1) When precision is less than 254, else 253.
(2) Rounded up to the nearest byte.
(3) These figures represent precision, scale.
(4) Formula in which *p* represents the actual precision

Limitations:

Refer to the individual data type descriptions above.

ANSI Specifics:

The keywords SEQNO, NATURAL DATE and NATURAL TIME are not part of the Standard.

Adabas SQL Server Specifics:

None.

Example:

To create a table with a numeric column and a named SEQNO column, which enables access to the underlying Adabas ISN, the following syntax is used:

```
CREATE TABLE cruise
   (cruise_id NUMERIC(8), sequence_no SEQNO);
```

Column Constraint Element

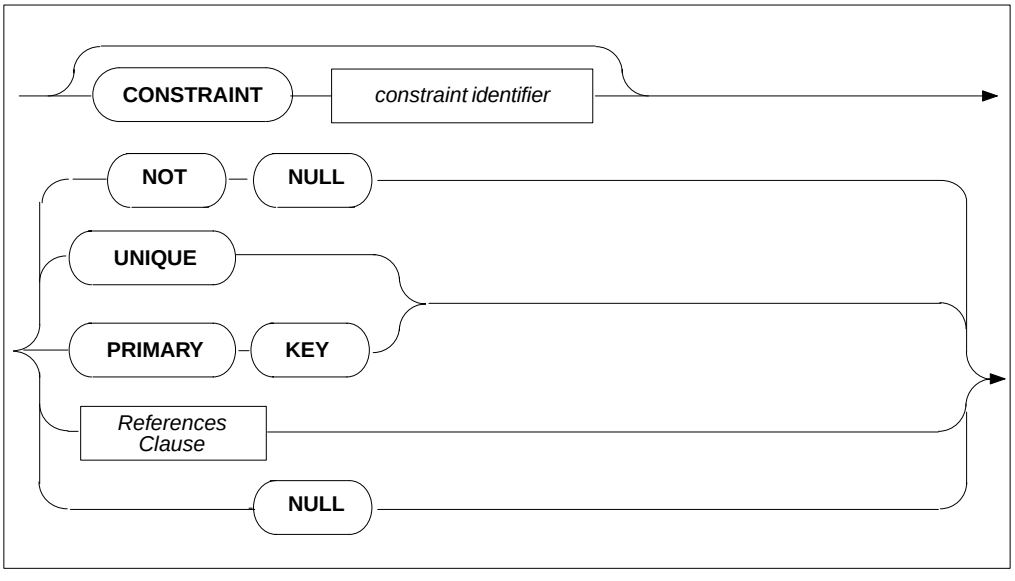
Function:

A column constraint element specifies the conditions which apply to each column.

Invocation:

This element is part of the table column element.

Syntax:



constraint identifier	is a valid identifier for a constraint and must be unique within a schema.
UNIQUE	only one UNIQUE constraint is allowed.
PRIMARY KEY	only one PRIMARY KEY is allowed in a table.
NULL/NOT NULL	indicates whether NULL values are permissible for this column.

Reference Clause	only allowed for subtables. The number of columns allowed in this particular case is one. For syntax regulations refer to section: **Table Constraint Element**, later in this chapter. For further restrictions refer to sections **CREATE CLUSTER DESCRIPTION/CREATE CLUSTER** in chapter **SQL STATEMENTS**.

Description

A constraint is a subobject of a base table which is defined to ensure the compliance of the actual data with the specified conditions.

Adabas SQL Server knows four different types of constraints: NOT NULL, UNIQUE, PRIMARY KEY and FOREIGN KEY. Syntactically, a constraint referring to a simple column can be defined within a table column element. Constraints referring to more than one column have to be defined by a table constraint element. The name (constraint identifier) of a constraint must be unique within the schema. It will be generated automatically, if not specified.

UNIQUE and PRIMARY KEY constraints are called ‘unique constraints’. A REFERENCES constraint is called ‘referential constraint’.

The following conventions hold true for the following explanations:

- Let C be the column for which this constraint is specified.
- Let T be the table where column C resides.

- UNIQUE:

  The UNIQUE constraint ensures, that no two rows of T carries the same value in column C. Rows with NULL values in column C don’t affect this constraint. A UNIQUE constraint implies an index definition, that, if not specified, will be generated.

- PRIMARY KEY:

  The PRIMARY KEY constraint ensures, that no two rows of T carry the same value in column C. When specifying a PRIMARY KEY constraint, it is also mandatory to specify the column attribute NOT NULL. A PRIMARY KEY constraint implies an index definition, that, if not specified, will be generated.

- NULL:

  The NULL constraint indicates, that null values are permissible in any row of the table for the column C.

- NOT NULL:

  The NOT NULL constraint indicates, that null values are not permissible in any row of the table for the column C.

- REFERENCES:

  For details on how to define the reference clause, refer to the section **Table Constraint Element,** particularly the FOREIGN KEY subclause, later in this chapter. Note that the number of columns allowed in this particular case is one.

  For details see section **Describing Adabas Nested Data Structures** in the *Adabas SQL Server Programmer's Guide*.

Limitations:

The CREATE CLUSTER and CREATE CLUSTER DESCRIPTION statements have the following restrictions:

- A Column level REFERENCES constraint may only be used to build the referential constraint between tables of level 0 (base tables) and tables of level 1 (subtables).
- The usage of the REFERENCES clause is only allowed in the CREATE CLUSTER/ CREATE CLUSTER DESCRIPTION statements

There may be a maximum of one PRIMARY KEY for a bases table (this includes a table constraint of type PRIMARY KEY).

When using a PRIMARY KEY constraint the attribute SUPPRESSION is not permitted.

When using a UNIQUE constraint in conjunction with a SUPPRESSION attribute the attribute NOT NULL is not permitted. For details refer to the *Adabas SQL Server Programmer's Guide*, chapter **Introduction**, section **Conversion of Adabas Field Attributes to Adabas SQL Server Column Attributes**.

When using a UNIQUE or PRIMARY KEY constraint in conjunction with a DEFAULT Adabas attribute, the attribute NULL is not permitted.

ANSI Specifics:

The default referential triggered action differs from the ANSI standard. The default is CASCADE and not NO ACTION.

The NULL constraint is not part of the Standard.

Adabas SQL Server Specifics:

No other option than CASCADE is supported.

Example:

The example below shows how to define a column constraint which disallows NULL values and values which are not unique:

```
CREATE TABLE contract (
     contract_id integer NOT NULL UNIQUE );
```

Column Index Element

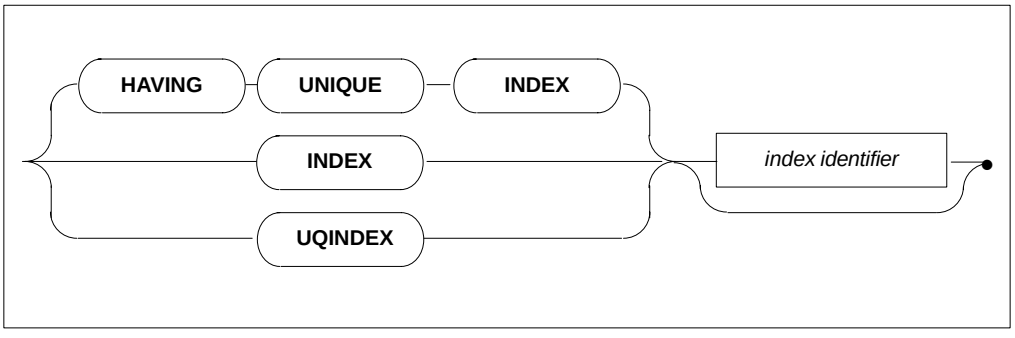
Function:

Specifies a column as an index.

Invocation:

This element is part of the table column element.

Syntax:



index identifier must be a valid identifier for an index and must conform to the rules specified earlier in this chapter in section **Identifiers**. The specification is optional and has to be unique within a schema.

Description:

The following conventions hold true for the following explanations:

- Let C be the column for which this column index element is specified.
- Let T be the table where column C resides.

- INDEX:

  An INDEX specification in the case that there is no range specification results in an Adabas descriptor being added. In the case of a range specification across only one column, an Adabas Subdescriptor will be generated.

- HAVING UNIQUE INDEX:

  A HAVING UNIQUE INDEX ensures that there are no two rows of T having identical values in the column C. Rows with NULL values column C do not effect this index. A HAVING UNIQUE INDEX implies a UNIQUE constraint, and is stored in the Adabas SQL Server catalog as a UNIQUE constraint.

- UQINDEX:

  A UQINDEX is an index that will generate an Adabas unique sub- or superdescriptor on a column that is part of a subtable. This descriptor is not considered to be unique in SQL terms and can, therefore, not be represented by a normal ”unique constraint”.

  For more details see the section **Indexes and Constraints**, chapter **Adabas SQL Server Data Structures** in the *Adabas SQL Server Programmer's Guide*.

Limitations:

A specification of HAVING UNIQUE INDEX is not allowed in subtables. A specification of UQINDEX is only allowed in subtables.

You are not allowed to specify a HAVING UNIQUE INDEX together with a UNIQUE constraint or a PRIMARY KEY.

When using a HAVING UNIQUE INDEX in conjunction with a SUPPRESSION attribute the attribute NOT NULL is not permitted. For details refer to the *Adabas SQL Server Programmer's Guide*, chapter **Introduction**, section **Conversion of Adabas Field Attributes to Adabas SQL Server Column Attributes**.

When using a HAVING UNIQUE INDEX in conjunction with a DEFAULT Adabas attribute, the attribute NULL is not permitted.

ANSI Specifics:

The Column Index Element is not part of the Standard.

Adabas SQL Server Specifics:

None.

Column Default Element

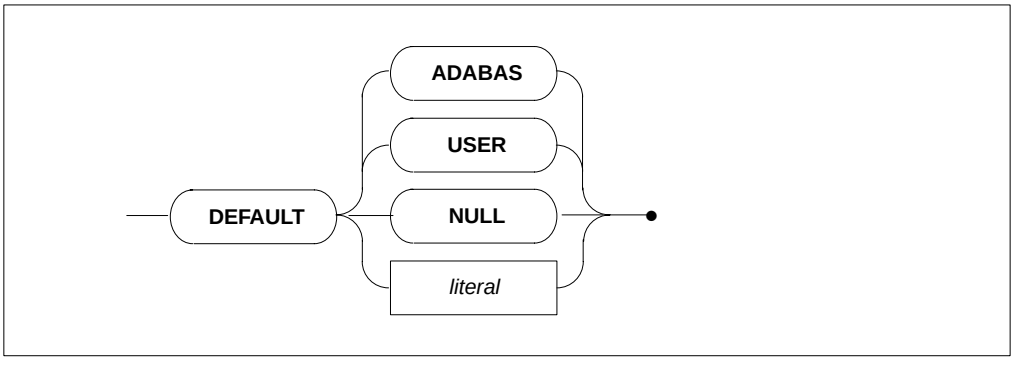
Function:

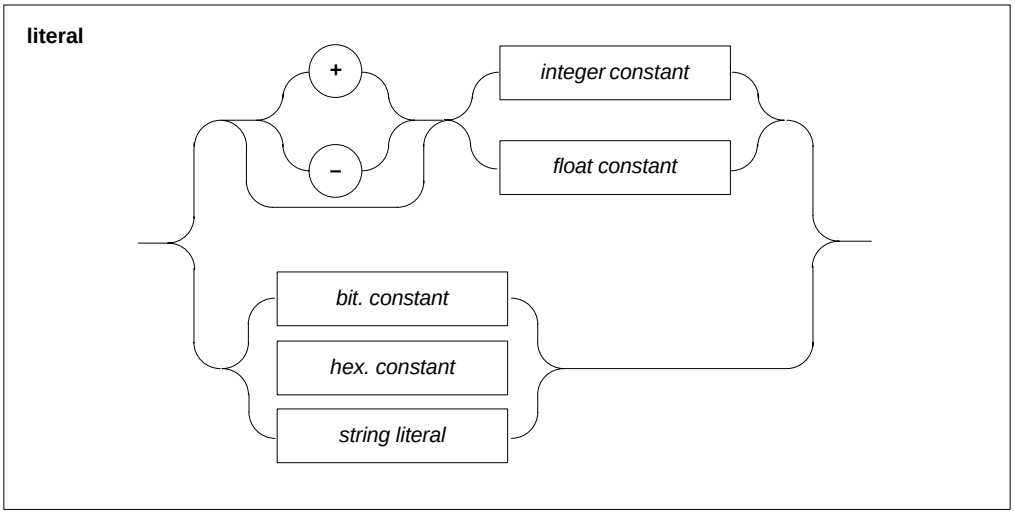
A column default element specifies a default value for a column.

Invocation:

This element is part of the table column element.

Syntax:





Description:

This element is optional. A default element specifies what the default value for the column is if no real value is given for the column in question in an insert statement. The following defaults are possible:

- ADABAS:

  With this the ADABAS defaults are defined. In this case for the column the corresponding Adabas field gets no Adabas options. That is no NC option.

- USER:

  A USER default defines that the special register USER will be used to insert a value into the specified column with the default value of DEFAULT USER. The column must be of type character and be sufficient in its definition to hold the returned value.

- NULL:

  If NULL is specified, the default is the NULL value. The column must of course be able to support the NULL value.

- Literal:

  In this case the default is a literal value of the appropriate data type.

Limitations:

Only one default specification is allowed per column.

The default value of ADABAS when used in conjunction with a UNIQUE, PRIMARY KEY constraint or HAVING UNIQUE INDEX is only allowed with the column attribute NOT NULL.

ANSI Specifics:

The DEFAULT ADABAS clause is not part of the Standard.

Adabas SQL Server Specifics:

None.

Example:

To give the column amount_deposit the default value of 10:

```
CREATE TABLE cruise
     (amount_deposit NUMERIC (13,3) DEFAULT 10.0)
```

Column Physical Element

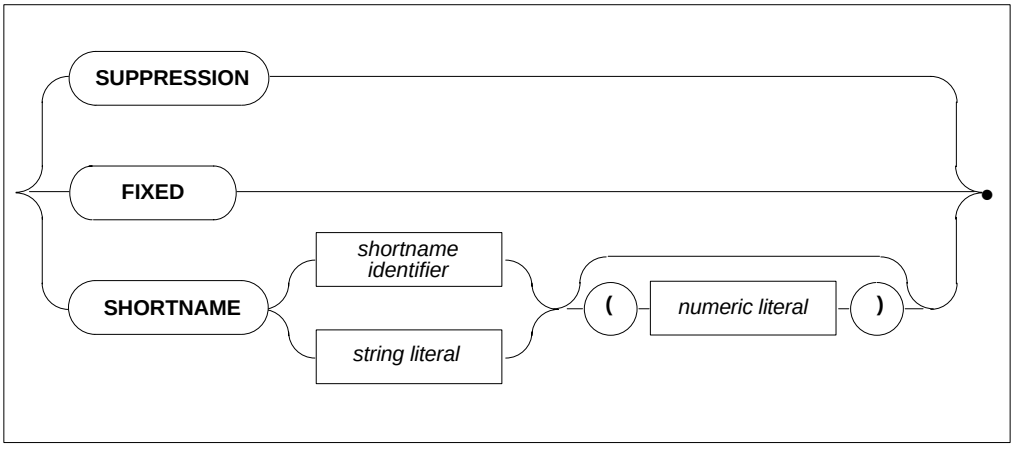
Function:

A column physical element is used to add Adabas-specific attributes to a column.

Invocation:

This element is part of the column element. This clause is only allowed in CREATE TABLE/ CREATE TABLE DESCRIPTION or a CREATE CLUSTER/CREATE CLUSTER DESCRIPTION statements.

Syntax:



shortname identifier	specifies an Adabas short name for a column.
string literal	represents the Adabas short name for a column.
numeric literal	is optional and specifies a rotated field. Its value must be less or equal to 191.

Description:

The column physical element can be specified in Adabas SQL Server mode exclusively. It is used to add Adabas-specific attributes to a column.

The following table shows which Adabas options will be set for the column, given the various Adabas SQL Server options:

Adabas SQL Server	Adabas
FIXED	FI
SUPPRESSION	NU

A shortname identifier specifies the Adabas short name of the corresponding field in the Adabas file to be described.

If numeric literal is specified, a so-called rotated field is specified, with the following meaning:

– If a particular MU or PE field has (semantically) a non varying number of occurrences, then the field can be 'rotated'. This means that each occurrence is mapped to an individual column. For example, should it be known that an MU will only ever have 12 occurrences, each representing a month, then each occurrence could be mapped to the columns January through to December.

A similar technique can be used for PE's, although here, each field within each occurrence must be individually mapped to a column.

Limitations:

An Adabas short name must consist of exactly two characters, the first of which must be between A and Z and the second can be between A and Z or between 0 and 9. The short names E0 to E9 are reserved by Adabas and may, therefore, not be used. If a short name is a reserved word like AS, it has to be represented in string format i.e. SHORTNAME 'AS'.

The short name specification is not case sensitive.

The keyword FIXED may only be used in the following contexts :

– In a CREATE TABLE DESCRIPTION, CREATE CLUSTER and CREATE CLUSTER DESCRIPTION statements
– When used with the attribute NULL or NOT NULL, they must be combined with the attribute DEFAULT ADABAS
– May not be combined with the attribute SUPPRESSION

The keywords FIXED and SUPPRESSION may only be used in the CREATE TABLE DESCRIPTION, CREATE CLUSTER and CREATE CLUSTER DESCRIPTION statements when the underlying Adabas field is defined with these attributes.

When using a SUPPRESSION attribute, the PRIMARY KEY constraint is not permitted.

When using the SUPPRESSION attribute in conjunction with a UNIQUE constraint or HAVING UNIQUE INDEX clause, the attribute NOT NULL is not permitted. For details refer to the *Adabas SQL Server Programmer's Guide*, chapter **Introduction**, section **Conversion of Adabas Field Attributes to Adabas SQL Server Column Attributes**.

ANSI Specifics:

The column physical element is not part of the Standard.

Adabas SQL Server Specifics:

None.

Example:

This example shows how to store bonus and sales for each month in a multiple-value field (each month is one occurrence). Each column is then rotated to be seen in one table (each months is a column of this table). An example of such a table description with rotated columns is:

```
CREATE TABLE DESCRIPTION rotated_table
DATABASE demo     FILE NUMBER 53
(
id                CHAR(20) SHORTNAME    ”AA”,
january_bonus     INTEGER  SHORTNAME    ”DA”(1),
january_sales     INTEGER  SHORTNAME    ”DB”(1),
february_bonus    INTEGER  SHORTNAME    ”DA”(2),
february_sales    INTEGER  SHORTNAME    ”DB”(2)

 ...

december_bonus    INTEGER  SHORTNAME  ”DA”(12),
december_sales    INTEGER  SHORTNAME  ”DB”(12),
)
```

where: ”DA” and ”DB” are the short names for the fields within the periodic group.

Table Constraint Element

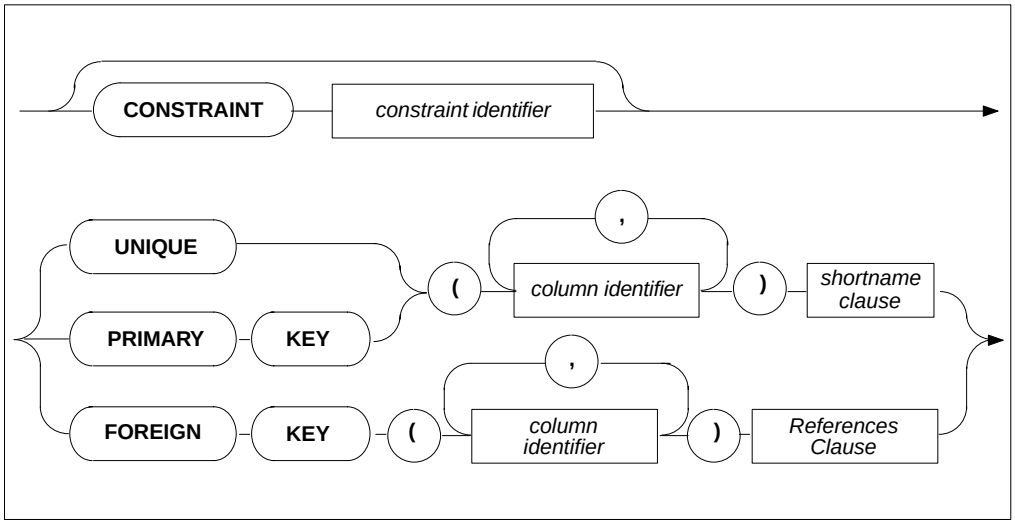
Function:

A table constraint specifies a constraint for a list of columns.

Invocation:

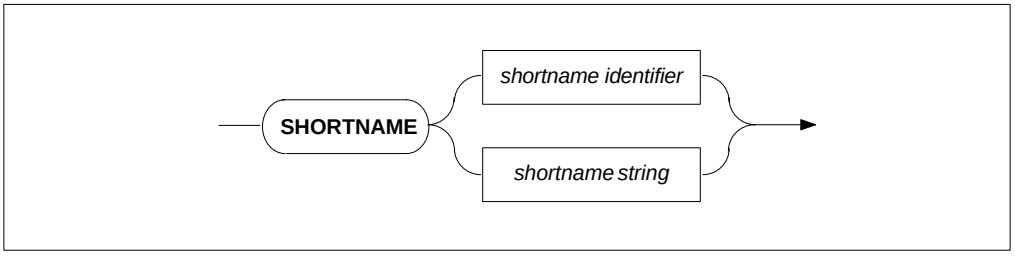
This element is part of the table element.

Syntax:



constraint name — a valid name for a constraint and must conform to the rules specified earlier in this chapter in section **Identifiers**.

Shortname clause:



Reference clause:

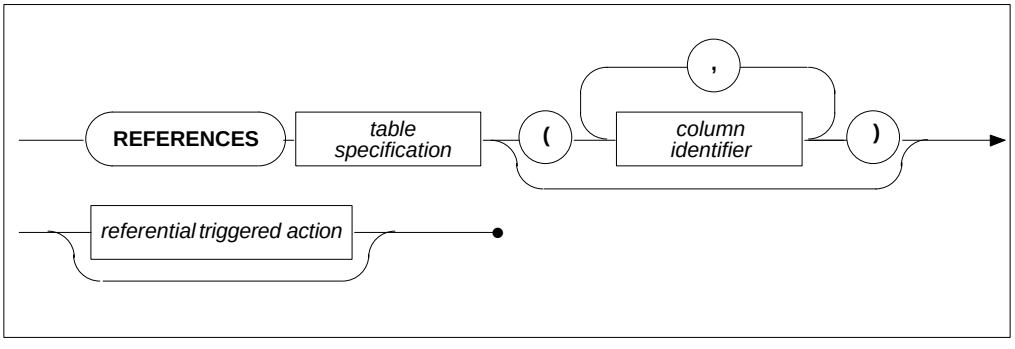
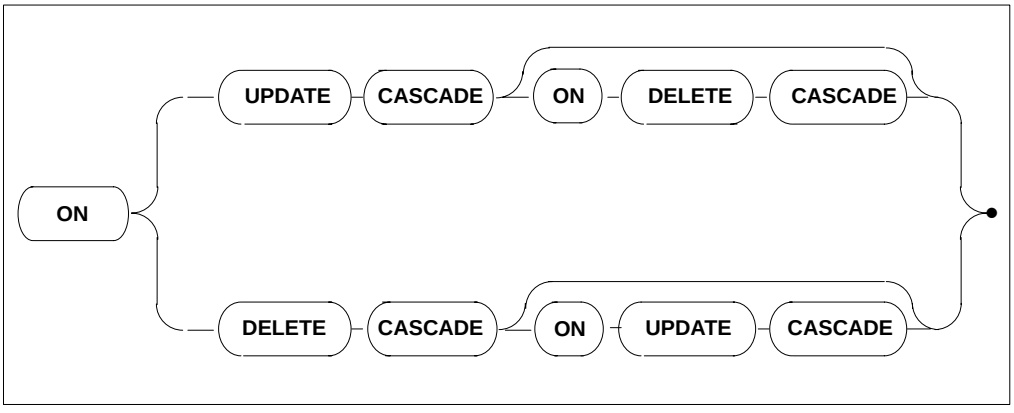


table specification	schema identifier:table identifier is the expected format
column identifier	optional specification of a list of referenced columns.
referential triggered action	optional, and specifies an action to be taken, if a row is updated or deleted. This clause has been added to reasons of conforming to the Standard, only.

Referential triggered action:



referential triggered action is optional and specifies an action to be taken if a row is updated or deleted. If no action is specified CASCADE is assumed.

Description:

UNIQUE and PRIMARY KEY constraints are called ”unique constraints”. A REFERENCES constraint is called ”referential constraint”. A table constraint element defines a constraint across one or more column(s).

The following conventions hold true for the following explanations:

– Let CL = (c1,... , cn) be the column list of one or more columns for which this table constraint element is specified.
– Let T be the table where the columns of CL reside.

- PRIMARY KEY:

  There may only be a maximum of one PRIMARY KEY definition in a base table.

  A PRIMARY KEY constraint ensures that there are no two rows of T having identical values in the columns of CL. When specifying a PRIMARY KEY constraint, it is mandatory for all the columns of CL to have the column attribute of NOT NULL, this therefore means that no columns of CL may contain the NULL value. A PRIMARY KEY constraint, implies an index definition, that, if not specified, will be generated.

PRIMARY KEY's on subtables of level one or two are limited to using all the columns of the associated FOREIGN KEY (the FOREIGN KEY that associates this level one or two table with its parent), plus a column of data type SEQNO on the current level and any other columns of this level. The important point here is that only PRIMARY KEY's with a column of data type SEQNO, for this table level, are classified as fulfilling the requirements for building a "unique constraint".

- UNIQUE:

  A UNIQUE constraint ensures that there are no two rows of T having identical values in the columns of C. Rows with NULL value(s), in any columns of C, do not effect this constraint. A UNIQUE constraint implies an index definition, that, if not specified will be generated.

- FOREIGN KEY:

  If specified, the REFERENCE's column list must conform to the following;

  – The number of columns in CL and the number of columns in the references column list must be equal.
  – The ith column of CL must be semantically the same as the ith column of the references list (i.e., the data type and attributes must match). The attributes UNIQUE and PRIMARY KEY should be converted to UQINDEX. The attribute REFERENCES is an exception to this rule.
  – The columns of the references clause must match those of a "unique constraint" in the referencing table.
  – All the columns of the "unique constraint" must have the attribute NOT NULL defined.

  If the REFERENCES column list is not specified, then the above must still hold true, but the checks are performed against the PRIMARY KEY constraints' columns for the referenced table.

  The Referential Triggered Action clause enables the specification of actions to be taken on a referential constraint, when a primary key of the referenced table changes.

  This clause is part of a references clause in column constraint element and table constraint element and exists for reasons of conforming to the ANSI standard, only. The default referential triggered action differs from the ANSI standard. The default is CASCADE and not NO ACTION.

  The following conventions hold true for the explanations below:

  – Let T be the referenced table and T0 be the referencing table of a foreign key.
  – Let further C be the referenced column and C0 the referencing column list of the foreign key.
  – Let r be a row of T to be updated or deleted with a value v in C.
    Let r0 be a row of T0 with the same value v in C0.

  If row r is updated where the value of C changes to vu, r0 in T0 is also updated and the value of C0 changes to the same value vu.

  If row r is deleted r0 is also deleted.

  For more details see the *Adabas SQL Server Programmers Guide* section **Describing Adabas Nested Data Structures**.

Limitations:

All columns of Cl must exist in the defining base table, and a column of T may not appear twice within Cl.

The FOREIGN KEY clause may only be used in a CREATE CLUSTER or CREATE CLUSTER DESCRIPTION statement.

The SHORTNAME clauses may only be used in a CREATE CLUSTER DESCRIPTION or CREATE TABLE DESCRIPTION statement.

There may be a maximum of one PRIMARY KEY for a bases table (this includes a column attribute of type PRIMARY KEY).

When using a PRIMARY KEY constraint the attribute SUPPRESSION is not permitted.

When using a UNIQUE constraint in conjunction with a SUPPRESSION attribute the attribute NOT NULL is not permitted. For details refer to the *Adabas SQL Server Programmer's Guide*, chapter **Introduction**, section **Conversion of Adabas Field Attributes to Adabas SQL Server Column Attributes**.

When using a UNIQUE or PRIMARY KEY constraint in conjunction with a DEFAULT ADABAS attribute, the attribute NULL is not permitted.

ANSI Specifics:

The columns of a UNIQUE constraint must under ANSI have the attribute NOT NULL specified.

The default referential triggered actions differs from that in the SQL standard, in that, the default action is CASCADE and not NO ACTION.

Adabas SQL Server Specifics:

None.

Table Index Element

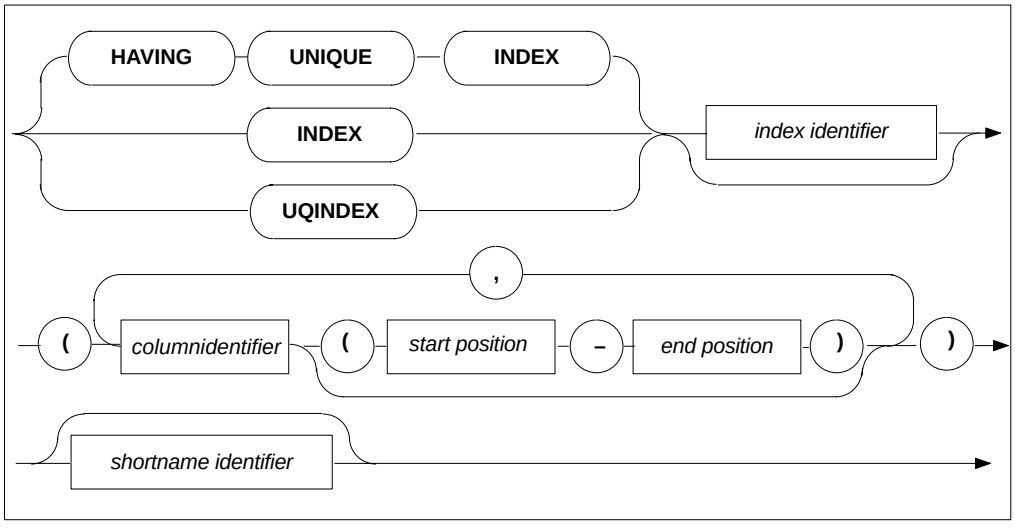
Function:

Specifies a set of columns as an index.

Invocation:

This element is part of the table element. The shortname identifier is only allowed in a CREATE TABLE DESCRIPTION or a CREATE CLUSTER DESCRIPTION statement.

Syntax:



index identifier	represents the name of an index and must conform to the rules specified earlier in this chapter in section **Identifiers**.
column identifier	name of a column to be used in the forming of an index.
start position	the start position within the column when defining an Adabas descriptor.
end position	the end position within the column when defining an Adabas descriptor. The end position must be greater than the start position.
shortname identifier	a valid specification of an Adabas short name. See section **Table Physical Element** earlier in this chapter for more details.

Description:

In order to improve the performance of an existing application, an index can be established for one or more column(s) of a base table.

The table index element allows for the creation of an Adabas descriptor, that reflects the capabilities of the Adabas database system's definition of descriptors. For a detailed discussion of Adabas descriptors, please refer to the Adabas documentation for your environment, in particular the **Database Design** chapter of *DBA Reference Manual*.

A ranges specification is when start and end positions are specified. This allows an index specification to be restricted to sub-elements of a column.

The following conventions hold true for the explanations below:

- Let Cl = (c1,... , cn) be a column list of one or more columns for which this table index element is specified.
- Let T be the table where the columns of Cl resides.

- INDEX:

  A INDEX is used to allow more efficient access to a base table. As such, the index is based on one or more column(s) of a base table, where the columns' listed are considered as an entity. If the number of columns in the column list is greater than one, then an Adabas Superdescriptor will be generated. If the index is over only one column and that column as no range specification, then the column to which this index references will have an Adabas Descriptor added. In the case of a range specification across only one column, then an Adabas Subdescriptor will be generated.

- HAVING UNIQUE INDEX:

  A HAVING UNIQUE INDEX ensures that there are no two rows of T having identical values in the columns of Cl. Rows with NULL value(s), in any columns of Cl, do not effect this index. A HAVING UNIQUE INDEX implies a UNIQUE constraint, and is stored in the Adabas SQL Server catalog as a UNIQUE constraint.

- UQINDEX:

  A UQINDEX is an index that will generate an Adabas unique sub- or superdescriptor on a column that is part of a subtable. This descriptor is not considered to be unique in SQL terms and can, therefore, not be represented by a normal "unique constraint".

  For more details see the section **Indexes and Constraints**, chapter **Adabas SQL Server Data Structures** in the *Adabas SQL Server Programmer's Guide*.

Limitations:

The shortname identifier is only used in a CREATE TABLE DESCRIPTION or a CREATE CLUSTER DESCRIPTION statement.

A specification of a UQINDEX is only valid for level 1 and level 2 base tables (subtables).

You are not allowed to specify a UNIQUE INDEX together with a UNIQUE constraint or a PRIMARY KEY.

When using a HAVING UNIQUE INDEX in conjunction with a SUPPRESSION attribute the attribute NOT NULL is not permitted. For details refer to the *Adabas SQL Server Programmer's Guide*, chapter **Introduction**, section **Conversion of Adabas Field Attributes to Adabas SQL Server Column Attributes**.

When using a HAVING UNIQUE INDEX in conjunction with a DEFAULT Adabas attribute, the attribute NULL is not permitted.

ANSI Specifics:

The table index element is not part of the Standard.

Adabas SQL Server Specifics:

None.

ORDER BY Clause

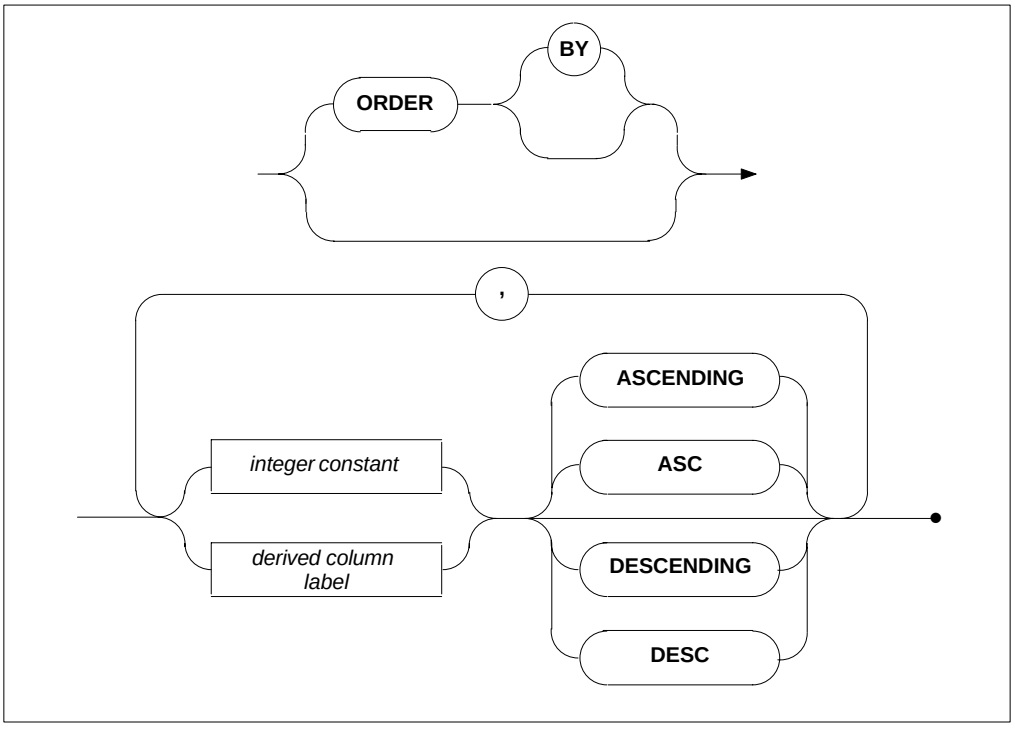
Function:

The ORDER BY clause enables the resultant table of a DECLARE CURSOR statement or a SELECT statement to be sorted in a user-defined sequence.

Invocation:

The ORDER BY clause is part of the DECLARE CURSOR statement and the dynamic or interactive SELECT statement.

Syntax:



integer constant	identifies a derived column of the resultant table
derived column label	identifies a derived column of the resultant table

Description:

The ORDER BY clause enables the resultant table to be sorted in a user-defined sequence. Generally, any resultant table is unordered, however, the ORDER BY clause sorts the rows according to the values of a particular column or columns. Rows are sorted by default in ascending order although descending may be explicitly specified. If more than one column is specified then the resultant table is sorted primarily according to the values in the first ordering column. Thereafter, rows which have the same value for that column are sorted amongst themselves according to the values of the second ordering column. The entire table is sorted according to all the columns specified in the ORDER BY clause.

Ordering columns may be specified either by derived column label or by an integer displacement representing their position in the derived column list.

Derived columns which do not have labels must be referenced by displacement whereas derived columns with labels may be referenced by either method. An integer displacement simply refers to a derived column by its position in the derived columns list as numbered from left to right, i.e. the first column would be referred to as 1, the second as 2, etc.

If a derived column label is used, then it must be sufficiently qualified in order to uniquely identify the required derived column. If the same column is specified more than once in the derived columns list, then it is impossible to specify a unique derived column label. In such a case, an integer displacement must be used to reference the required derived column.

The final order of a column of type character depends on the sorting order of the underlying hardware.

Limitations:

A derived column may not be specified more than once in the ORDER BY clause regardless of the type of reference used.

ANSI Specifics:

The keywords ASCENDING and DESCENDING are not part of the Standard.

Adabas SQL Server Specifics:

The keywords ASCENDING and DESCENDING are extensions.

Example:

To declare a cursor to select all cruise IDs , start harbors and cruise prices where the end resultant table lists all start harbors in ascending alphabetical order, and, for each 'group' of identical start harbors, the rows are then sorted by descending cruise price , the following syntax applies:

```
DECLARE cursor1 CURSOR FOR SELECT cruise_id,start_harbor,cruise_price
     FROM cruise
     ORDER BY start_harbor ASCENDING,cruise_price DESCENDING;
```

To declare a cursor where all resultant rows are ordered by the 6th and 7th columns (in this case start_harbor and destination_harbor) with the 7th column in descending order, the following syntax using derived column labels (6,7) applies:

```
DECLARE cursor1 CURSOR FOR SELECT *
     FROM cruise
     ORDER BY 6,7 DESCENDING;
```

To request all information about all cruises with a start date of December 22, 1991, with the result sorted in ascending alphabetical order of start harbors, and subsequently, any groups of rows with identical start harbors are sorted in ascending alphabetical order of destination harbors, the following syntax applies for an interactive SELECT statement.

```
SELECT *
     FROM cruise
     WHERE start_date=19911222
     ORDER BY start_harbor ASCENDING,destination_harbor ASCENDING;
```

USING Clause

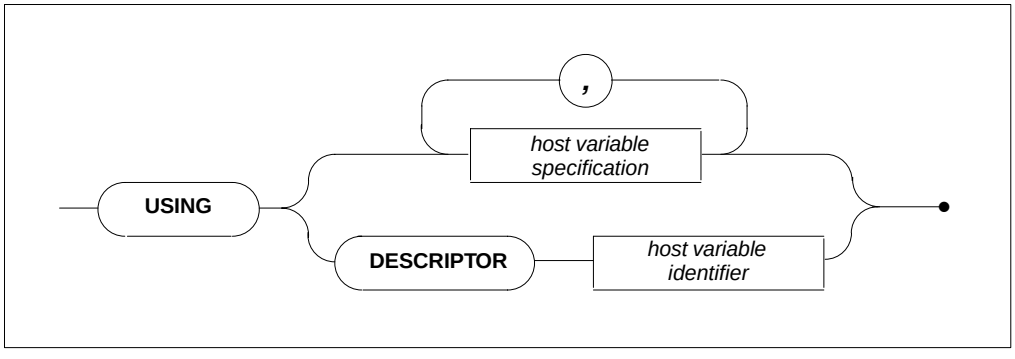
Function:

The USING clause is used to provide references to host variables for use in either a dynamic FETCH, OPEN or EXECUTE statement.

Invocation:

The USING clause is part of dynamic EXECUTE, FETCH or OPEN statements.

Syntax:



host variable specification	is a valid host variable specification and must have been defined in an application program. The host variable specification may reference a structure.
host variable identifier	is a valid single host variable identifier and which must be the address of an SQL descriptor area (SQLDA).

Description:

The USING clause defines a set of host variables for use either as value sources in a dynamic OPEN or EXECUTE statement or as target receptors in a dynamic FETCH statement.

A host variable specification, which references a host variable structure is equivalent to individual host variable specifications which reference all the elements of the structure singularly.

For a dynamic OPEN or EXECUTE statement, if the associated PREPAREd statement contained host variable markers, i.e., '?' then these markers must be satisfied by use of a USING clause. Prior to use the referenced host variables must have been assigned appropriate values. Each referenced host variable provides a value for its corresponding host variable marker. The user must make sure that the host variables are supplied with the correct values and formats in the correct order.

For a dynamic FETCH statement, the host variables provided are intended to receive the results of the statement.

Host variables within the USING clause can be provided in two ways:

- by explicitly specifying a list of host variables. The number, type and order of the required host variables must be known at compilation time of the host program.
- by providing an SQL descriptor area. This facility enables a more dynamic approach to be adopted. The DESCRIBE statement provides the necessary information in the SQLDA for each host variable marker or derived column. The user must then provide a pointer in each field description which references an appropriate host variable. The number, type and order of the host variables can be completely unknown at compilation time of the host program. An SQL descriptor area is identified by means of a host variable which contains the address of the SQLDA.

A host variable specification which references a host variable structure is equivalent to individual host variable specifications which reference all the elements of a structure singularly.

Limitations:

None.

ANSI Specifics:

The USING clause is not part of the Standard.

Adabas SQL Server Specifics:

This is an Adabas SQL Server extension.

Example:

If an EXECUTE statement requires the input of three values e.g ’SELECT * FROM contract WHERE price IN (?, ?, ?), then the USING clause will provide these values. The following syntax applies:

```
USING :hv1, :hv2, :hv3;
```

FOR UPDATE Clause

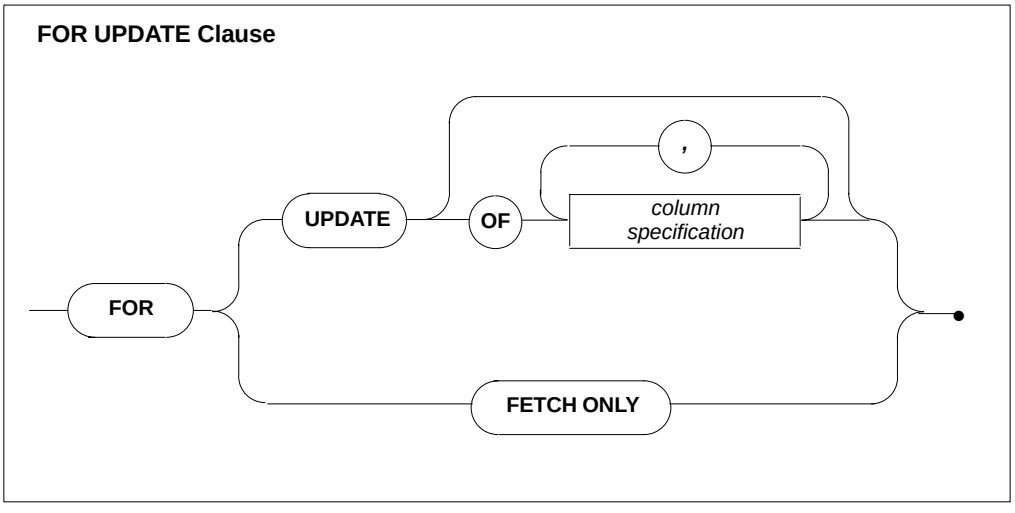
Function:

The FOR UPDATE clause indicates explicitly that the associated cursor is to be used in conjunction with either an UPDATE and/or DELETE statement making use of the WHERE CURRENT OF clause, or that the cursor is explicitly designated as not updatable.

Invocation:

The FOR UPDATE clause is part of the DECLARE CURSOR statement and the dynamic or interactive SELECT statement.

Syntax:



column specification is a valid column specification of the column which is to be updated or deleted.

Description:

A static cursor can be explicitly declared as being non-updatable by use of the FOR FETCH ONLY clause. In such a case, the use of positioned UPDATE or DELETE statements associated with the cursor is not allowed. Furthermore rows will not be locked once they are established regardless of the default locking specification.

Alternatively, a static cursor can be declared as FOR UPDATE, as long as it is updatable of course. In such a case, rows will be locked regardless of the default locking specification. In general, this clause need not be specified. However, if the associated UPDATE or DELETE statement is actually in a separate compilation unit, as is possible with Adabas SQL Server, then this clause is required in order to avoid a runtime error.

If neither a FOR FETCH ONLY clause nor a FOR UPDATE clause is specified and there are no associated UPDATE or DELETE statements within the same compilation unit, then the resulting rows will or will not be locked according to the system default locking specification.

Similar behavior can be ensured for a dynamic cursor by appending the clause to the dynamic SELECT statement. A column specification list is optional and indeed has no effect.

Limitations:

None.

ANSI Specifics:

The FOR UPDATE clause is not part of the Standard.

Adabas SQL Server Specifics:

This is an Adabas SQL Server extension.

Example:

To ensure that the cursor as declared in the first example can only be used for retrieval the following syntax applies:

```
DECLARE cursor1 CURSOR FOR
     SELECT cruise_id,start_date FROM cruise
          WHERE start_harbor = 'BARBADOS'
          FOR FETCH ONLY;
```

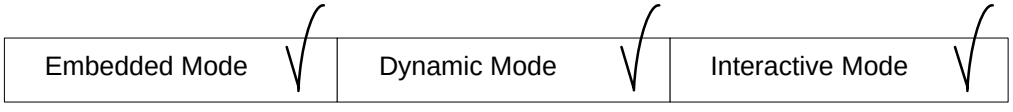
SQL STATEMENTS

ALTER TABLE

Function:

The ALTER TABLE statement changes the logical and physical structure of a base table.

Invocation:



Syntax:

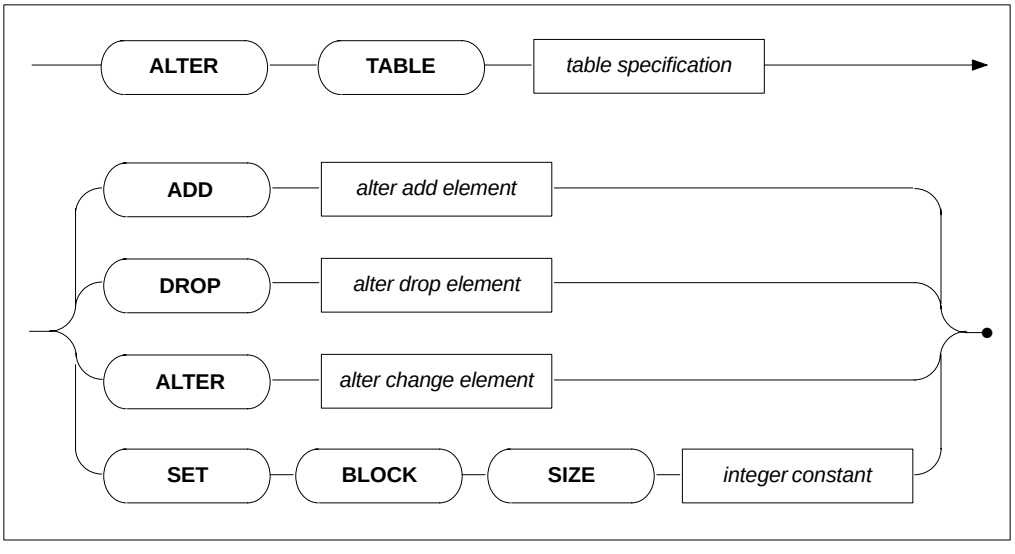
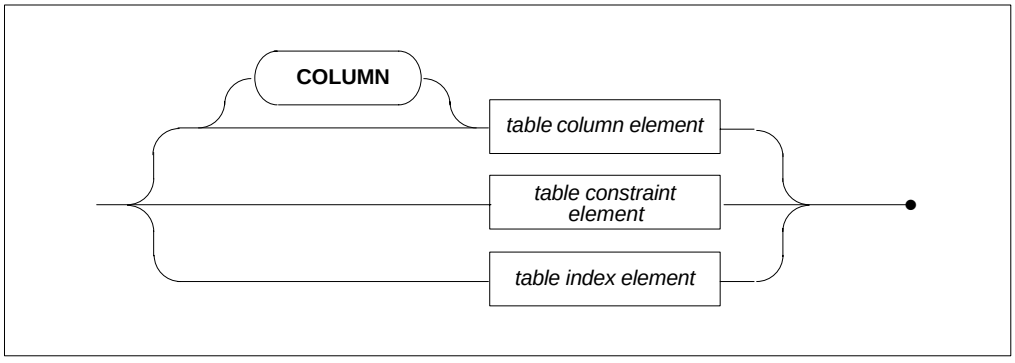
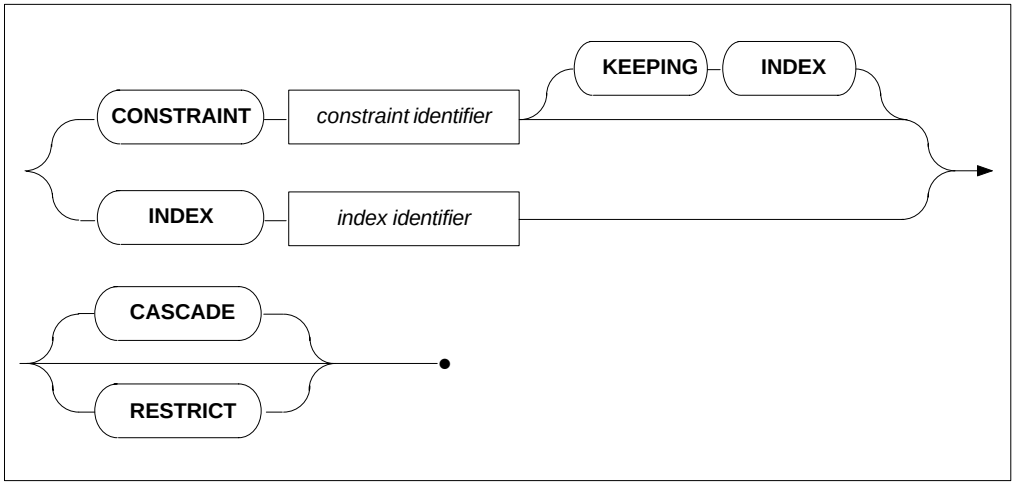


table specification — identifier of the base table to be modified, optionally qualified by the schema identifier.

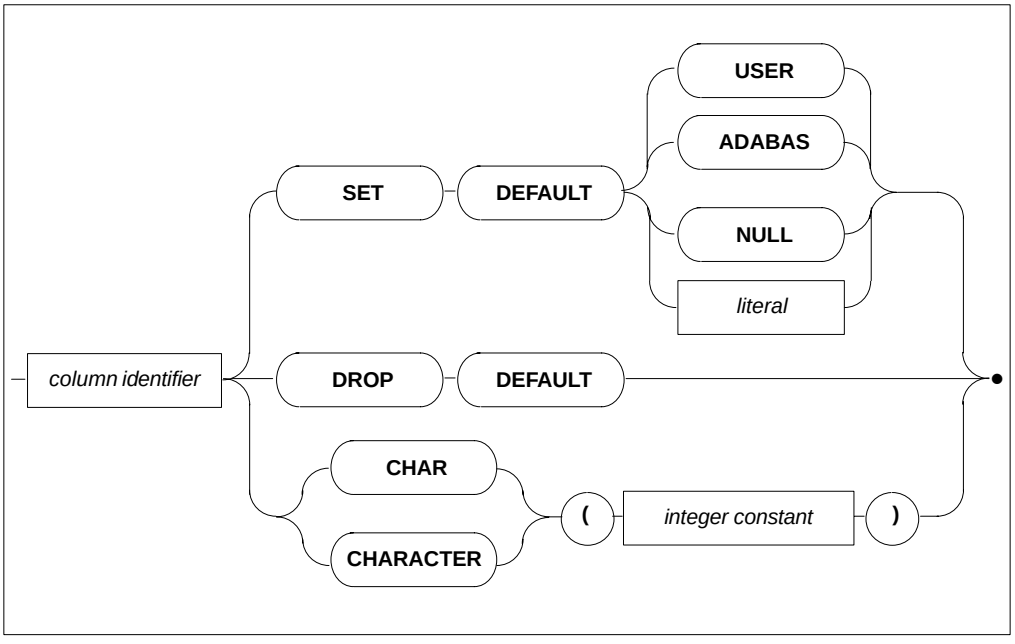
Alter add element:



Alter drop element:



Alter change element:



Description:

The ALTER TABLE statement allows the modification of characteristics of base tables. The following alteration can be made:

- Adding a new column to a base table.
  This function may only be performed on base tables. In the case of a cluster, a column may only be added to the master table.
- Changing the length of a character column.
  The changing of a character column length (only increasing the length), may be performed on any base table (including character columns of a subtable). It is though, only possible to increase a column's length up to either 253 characters for a column of less than this length or up to 16381 characters for a column that is originally greater than 253 characters in length.
- Changing/adding/dropping a column's default value.
  The deletion of a column's default value is not allowed for a column with a default value of ADABAS. For changing/adding a default value to a column see the section **Column Default Definition** in the section **Common Elements**.

- Creating/Dropping a Constraint or Index for the base table.
- Adding an Index or Constraint.
  In order to improve the performance of an existing application, an index or constraint can be established for one or more column(s) of a base table.
- Dropping an Index or Constraint.
  Allows the removal of an existing constraint or index from the specified base table. The KEEPING INDEX clause on dropping a constraint provides for the Adabas functionality of being able to remove the 'UQ' attribute from a descriptor.
- Changing the BLOCK SIZE of a Subtable in a Cluster.
  The ability to change the BLOCK SIZE of subtables allows the optimization regarding the number of occurrences that, if possible, will be multifetched in a MU/PE group. In the cases of a MU within a PE, the MULTIFETCH size is the multiplication of the level one (PE) table's BLOCK SIZE with that of the level 2 (MU). The following limitations apply to the changing of the BLOCK SIZE: default value = 7, minimum value = 1, maximum value = 191.

  The ALTER TABLE statement enables the creation of an Adabas descriptor, that reflects the capabilities of the Adabas database system's definition of descriptors. For a detailed discussion of Adabas descriptors, please refer to the Adabas documentation for your environment, in particular the **Database Design** chapter of the *DBA Reference Manual.*

  For more details on defining Indexes and Constraints, see the sections **Create Index Statement**, **Table Index Definition** or **Table Constraint Definition** in this manual and **Indexes and Constraints** in the *Adabas SQL Server Programmer's Guide*.

Limitations:

When adding a new column to a base or master table the following limitations exist in regard to the column attributes allowed within this statement. The specification of Column Indexes or Constraints are not allowed. The specification of NOT NULL without it being qualified with either SUPPRESSION or DEFAULT ADABAS.

When dropping a Constraint or Index, the following limitations exist. When attempting to drop an index, there may not be a constraint with the exact column list definition as the index (if there is, this index is classified as being the "defining index of the constraint").

Constraints and indexes that make up a FOREIGN KEY/REFERENCES constraint may not be dropped. FOREIGN KEY/REFERENCES constraints may not be dropped.

In the context of alter a character columns length, it is only possible to increase its length.

ANSI Specifics:

The following are not part of the SQL standard:

- When dropping a Constraint, the ability to use the ”KEEPING INDEX” suffix.
- The adding/dropping of Indexes.
- The in-ability to drop a FOREIGN KEY/REFERENCES constraint.
- The clause ”SET BLOCK SIZE”.
- The default value of ”ADABAS”.

Adabas SQL Server Specifics:

KEEP INDEX, SET BLOCK SIZE, INDEX and DEFAULT ADABAS are Adabas SQL Server extensions.

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To add one column to our base tables (cruise), the following syntax applies:

```
ALTER TABLE cruise ADD COLUMN fun_factor INTEGER;
```

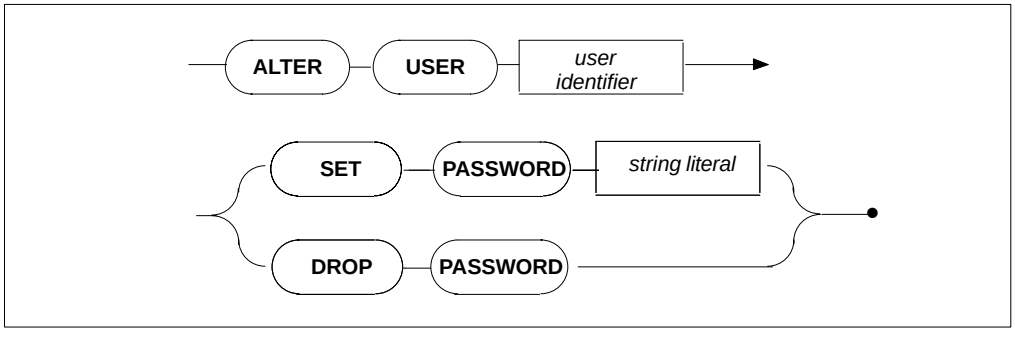
ALTER USER

Function:

With the ALTER USER statement the password for an existing user can be added, changed or dropped.

Invocation:

Syntax:



user identifier:	a unique identifier for a an existing user.
string literal:	the user's password.

Description:

A password for an existing user may be changed or added with the SET PASSWORD option. Naturally, from this point on, the user must provide the full user identification when connecting to Adabas SQL Server.

A password may be deleted with the DROP PASSWORD option. From this point on, connection to Adabas SQL Server will be granted with only the user ID.

Limitations:

Non-DBA users may only drop or set their own passwords. The DBA may execute this statement for all users.

ANSI Specifics:

The ALTER USER statement is not part of the Standard.

Adabas SQL Server Specifics:

DDL and D:CL statements may be mixed in one transaction. DML statements must not be mixed with DDL/DCL statements in the same transaction. For details, see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The existing user TIM wants to change his own password. The following syntax applies:

```
ALTER USER TIM SET PASSWORD 'XIYIZ';
```

The DBA wants to drop the password for the existing user PETER. The following syntax applies:

```
ALTER USER PETER DROP PASSWORD;
```

BEGIN DECLARE SECTION

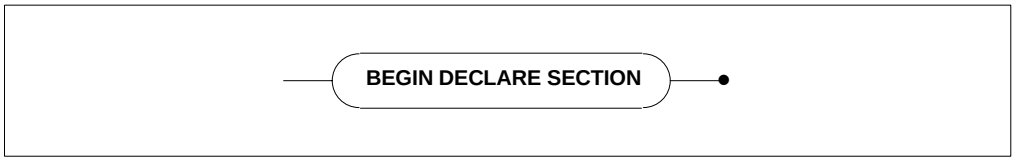
Function:

This statement is the starting delimiter for a host variable declaration block.

Invocation:

Embedded Mode √	Dynamic Mode	Interactive Mode

Syntax:



Description:

SQL application programs need to retrieve and provide values to and from Adabas SQL Server during runtime. This is achieved by using host variables which are specified in embedded SQL statements. During compilation, the nature of the host variables has to be known. To identify the relevant host variables it is suggested to declare these in a special section. This section is delimited by the BEGIN DECLARE SECTION and END DECLARE SECTION statements. These statements are always paired and can not be nested. The host variable declarations must be specified between the two statements and more than one of these sections are permitted. The statement does not result in an update of the SQLCA.

Limitations:

The positioning of the statement must conform to the rules governing the positioning of host variable declarations with the host applications. At least one host variable should be declared in such a block.

ANSI Specifics:

Any host variable referenced within an embedded SQL statement must have been declared with a host variable declaration section. Structures are not permitted in this context.

Adabas SQL Server Specifics:

For reasons of compilation efficiency it is recommended, although it is not absolutely necessary, to declare host variables in a host variable declaration section.

Example:

To specify the start of the host variable declaration section, the following syntax applies:

```
BEGIN DECLARE SECTION
    char a [5]
END DECLARE SECTION;
```

CLOSE

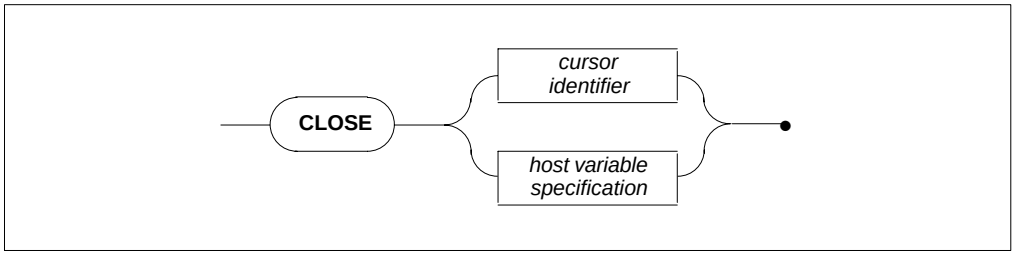
Function:

The CLOSE statement closes a cursor.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√		

Syntax:



cursor identifier	is a valid cursor identifier which identifies the cursor to be used in the CLOSE operation.
host variable specification	is a valid single host variable specification and must have been defined in the application program according to the host-language-dependent rules.

Description:

The CLOSE statement closes a cursor, i.e., releases resources allocated by an OPEN cursor statement. The value of the host variable must be a valid cursor identifier. A host variable can be used as cursor identifier only if the cursor is a dynamically declared cursor.

Limitations:

The cursor to be closed must have been opened.

ANSI Specifics:

All cursors opened within a transaction are automatically closed by a COMMIT or ROLLBACK statement. The associated DECLARE CURSOR statement must precede the CLOSE statement in the host program.

Adabas SQL Server Specifics:

The CLOSE statement does not have to be preceded by the associated DECLARE CURSOR statement. It may appear anywhere in the host program, even in another compilation unit.

Example:

To close a cursor that is assigned the identifier ‘cursor1’, the following syntax applies.

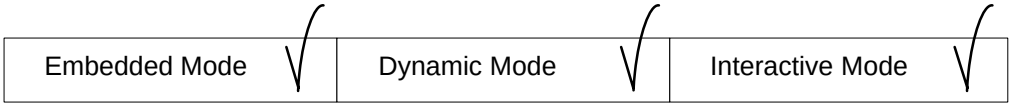
```
CLOSE cursor1;
```

COMMIT

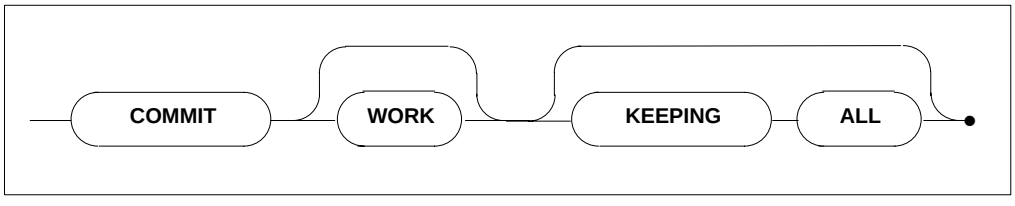
Function:

The COMMIT statement terminates a transaction and makes permanent all changes that were made to the database during the terminated transaction.

Invocation:



Syntax:



Description:

The COMMIT statement terminates the current transaction and starts a new transaction. All changes to the database that have been made during the terminated transaction are made permanent. All cursors that have been opened during the current transaction are closed.

If KEEPING ALL is specified, none of the currently opened cursors are closed after the execution of the COMMIT statement.

Limitations:

In DB2 mode, all statements which have been prepared during the current transaction are deleted.

In DB2 mode, when running under CICS, the use of the COMMIT statement is not permitted. In this environment, a COMMIT is automatically issued by the transaction system itself.

ANSI Specifics:

The keyword WORK is mandatory. The keywords KEEPING ALL are not part of the Standard.

Adabas SQL Server Specifics:

The keyword KEEPING ALL is an Adabas SQL Server extension.

Example:

To commit all changes made to the database in the current transaction, the following syntax applies:

```
COMMIT WORK;
```

CONNECT

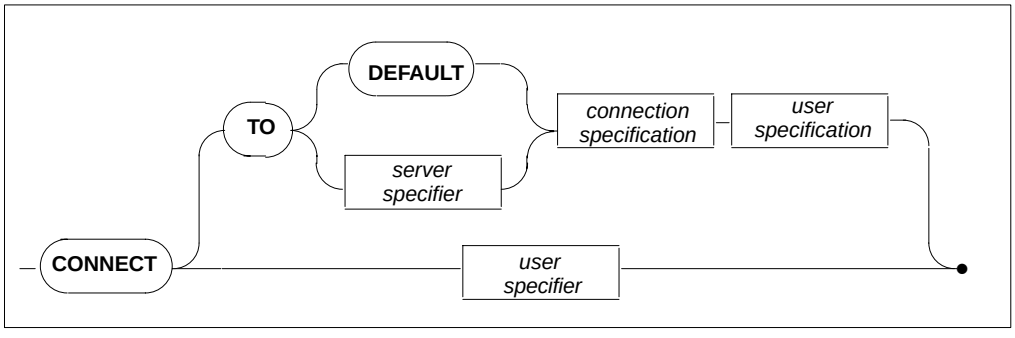
Function:

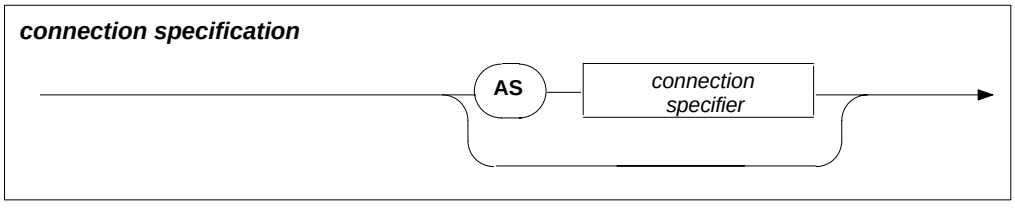
The CONNECT statement explicitly establishes an SQL session between a user application and Adabas SQL Server.

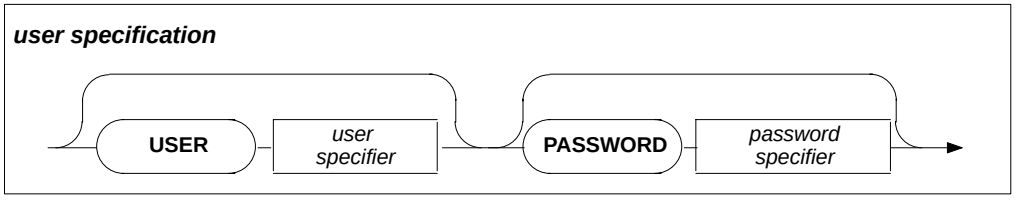
Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√		

Syntax:







The four specifiers can either be character-string constants or single host variable identifiers. The host variables must have been defined in the application program according to the host-language-dependent rules and their values must be character strings.

The lengths of these specifiers are as follows:

Server specifier	up to 8	case sensitive (see note below) or
Server specifier	up to 27 characters	case sensitive (see note below)
Connection specifier	up to 32 characters	case sensitive
User specifier	up to 32 characters	upper case
Password specifier	up to 32 characters	upper case

Note:
The server may be declared as a simple server specifier or as a server specifier (up to 8 characters) with communication mode (up to 9 characters) and node name (up to 8 characters) separated by blanks.

Description:

The CONNECT statement explicitly establishes an SQL session between the user application and Adabas SQL Server.

The user *user specifier* with the password *password specifier* is connected to Adabas SQL Server *server specifier* under the connection *connection specifier.*

The user specifier must exist on the server side. The user DBA is able to create other users with CREATE USER. If you do not specify the user specifier, the default user is the user identifier known by the operating system (login user). Exceptions to this rule apply for some mainframe environments as stated in the Limitation section below. If a password specifier is not specified, blanks will be generated as default password.

Adabas SQL Server checks authentication based on the user and password passed on.

If a server specifier is not specified, a default server is evaluated. The connection specifier is user-defined and will be used to set different connections. If a connection specifier is not specified, the server name will be used as connection name. For details, refer to the *Adabas SQL Server Programmer's Guide*, chapter **Client/Server Topics**.

Although an application can issue multiple CONNECT statements, it is set to only one Adabas SQL Server at a time. The Adabas SQL Server environment used in the most recently executed CONNECT statement is the active one. Multiple CONNECT statements can be issued to the same Adabas SQL Server to work with a few parallel connections at the same time.

If the application does not issue any CONNECT statement then the first SQL statement executes an implicit CONNECT to the default server. For further details regarding the default server refer to the *Adabas SQL Server Programmer's Guide*, chapter **Client/Server Topics**.

Limitations:

Adabas SQL Server determines the user ID for an implicit CONNECT as follows:

– in batch or under TSO
  The security logon identification is used.
  When this is not available, the job name is taken.
  Under VSE, the job name is used as User ID.

– under CICS
  The user (or default user) identification is used.
  When the user identification is not available, for example for asynchronous tasks, a user identifier is constructed using the four-byte CICS System Identification (SYSID) and the four-byte CICS Transaction ID.
  For Example:
  CICS System ID: 'fct4'
  Transaction ID: 'na22'
  User ID: 'fct4na22'

– under Com-plete
  the Com-plete logon user identification is used.

Note:
When you are using the external security interface, the user and password identifiers are limited to a maximum of 8 characters.This is a restriction of the external security products; e.g. RACF, ACF2, Top–Secret.

ANSI Specifics:

The CONNECT statement is not part of the Standard.

Adabas SQL Server Specifics:

The Version 1.2 syntax of the CONNECT statement (CONNECT USER *user identifier*) is still supported and is logically equal to the Version 1.3 and higher counterpart (CONNECT TO DEFAULT USER *user identifier*).

Example:

To establish an SQL session identified by the connection MYSESSION between the server ESQUNIX and user John the following syntax applies:

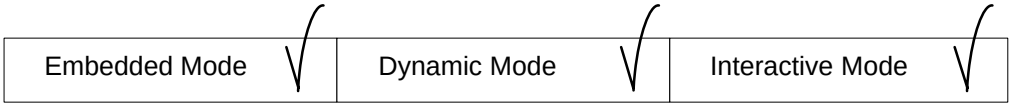
```
CONNECT TO :esqunix AS :mysession USER :john PASSWORD :xyz;
```

CREATE CLUSTER

Function:

The CREATE CLUSTER statement is used to combine a number of base tables in one internal table.

Invocation:



Syntax:

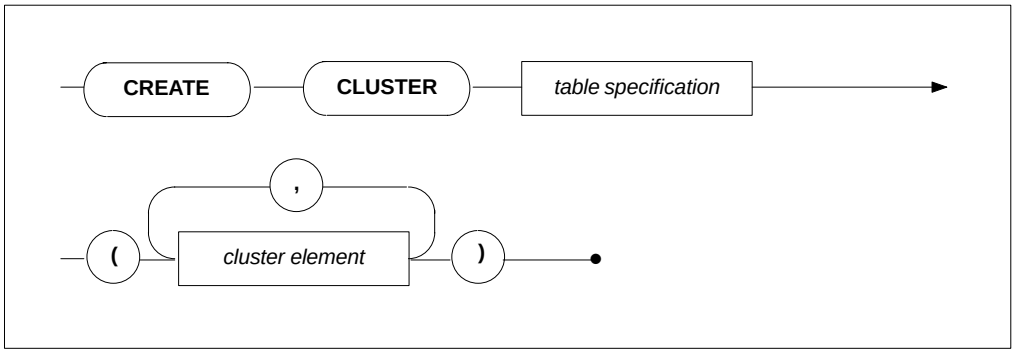


table specification	specifier of the master table to be created for this cluster, qualified by the schema identifier
cluster element	specifies a base table in the described cluster.

Cluster element:

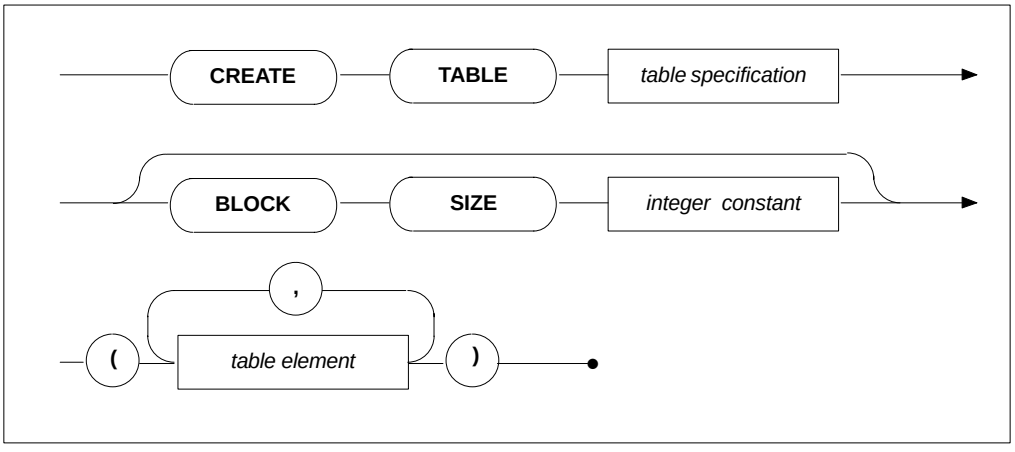


table specification	identifier of the subtable to be created for this cluster, qualified by the schema identifier
integer constant	a value between 1 and 191 is expected.
table element	see section **Table Elements** in chapter **Common Elements** of this manual.

Description:

A CREATE CLUSTER statement is used to combine a set of base tables into one Adabas file.

The BLOCK SIZE defined in a cluster element specifies the number of occurrences that will be multi-fetched in a MU/PE group. In the case of a MU within a PE, that Multifetch size is the multiplication of the level one (PE) table BLOCK SIZE with that of the level 2 (MU). A good value for a BLOCK SIZE is either a prime number or odd number. The BLOCK SIZE has the following limitations: default value = 7, minimum value = 1, maximum value = 191.

Each subtable within the CLUSTER represents either a PE group or an MU field. It is also possible to group MU's together into one single subtable, this assumes that all MUs have the same number of occurrences and that when accessing them, the occurrence number of each MU will be equal.

There is a different file creation behaviour on mainframe platforms compared to open systems platforms due to the possibility of keeping the FDTs of unloaded files:

- When using the FILE parameter, only the specified file number will be used to create the corresponding Adabas file. If a file with the specified file number already exists an error occurs. This is true for all platforms. On mainframe platforms only; if the specified file number is that of an unloaded file where the FDT has been kept, the FDT will be overwritten.
- When using the ADABAS FREE FILE SEARCH RANGE method, an unloaded file with existing FDT is not classified as a valid candidate for the CREATE TABLE or CREATE Cluster statement.

For a detailed description see the chapter **Adabas SQL Server Data Structures**, section **Describing Adabas Nested Data Structures** in the *Adabas SQL Server Programmer's Guide*.

Limitations:

The column attributes/table clause SHORTNAME definition may not be specified in this statement.

A CREATE CLUSTER statement will always represent tables of level one as a PE group within Adabas. Following rules apply:

1. Foreign keys reference only unique constraints. A subtable contains exactly one foreign key.
2. The same rules apply for the columns, constraints and indexes of the master table as for a CREATE TABLE statement.
3. Columns which are not an element of a foreign key and not of a SEQNO type are called data columns. The limitations under rules 4 – 7 apply to data columns in subtables.
4. The data columns of a level 1 table correspond only to fields of a single PE group.
5. The data columns of a level 2 table correspond to MU fields within a specific PE group – the group containing those fields which the data columns in the referenced table correspond to.
6. Not more than one data column may correspond to each field (with rotated fields, each subscript counts as its own field).
7. With parallel MU fields, it is assumed that in all Adabas records, the respective counter values are the same.
8. For x=1 or x=2, a unique constraint of a level x table encompasses the elements of the foreign keys and a column of the type SEQNO(x). Other unique constraints on subtables are not allowed.
9. For indexes to subtables, the same rules apply as for level-0 tables, plus the following additional constraints:
   - HAVING UNIQUE INDEX is not allowed. In order to model the Adabas UQ option, UQINDEX is used.

   Note:
   A unique constraint is defined as either a UNIQUE or PRIMARY KEY constraint.
10. All level 0 columns must be grouped within one CREATE TABLE of a cluster.

    Note:
    In the case of a PE data structure containing MU fields only, it is necessary to use an Adabas short name on the SEQNO(1) of the PE–subtable.

ANSI Specifics:

The CREATE CLUSTER statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To create a cluster named *city_guide*, enter the following syntax:

```
CREATE CLUSTER city_guide
(
  CREATE TABLE states
    (
      abbreviation     CHAR (2)  PRIMARY KEY NOT NULL DEFAULT ADABAS,
      state_name       CHAR (20) UNIQUE NOT NULL DEFAULT ADABAS,
      capital          CHAR (20) INDEX state_capital,
      population       INT
    ),
  CREATE TABLE cities
    (
      state_abbrev     CHAR (2)  UQINDEX NOT NULL DEFAULT ADABAS,
      city_seqno       SEQNO (1) NOT NULL,
      city_name        CHAR (20),
      population       INT,
      PRIMARY KEY (state_abbrev, city_seqno),
      FOREIGN KEY (state_abbrev) REFERENCES states (abbreviation),
      UQINDEX city_state (city_name, state_abbrev)
    ),
  CREATE TABLE buildings
    (
      state_abbrev     CHAR (2) UQINDEX NOT NULL DEFAULT ADABAS,
      city_seqno       SEQNO(1)  NOT NULL,
      building_seqno   SEQNO(2)  NOT NULL,
      building_name    CHAR (20) NOT NULL SUPPRESSION,
      height           INT       NOT NULL SUPPRESSION,
      PRIMARY KEY (state_abbrev, city_seqno, building_seqno),
      FOREIGN KEY (state_abbrev, city_seqno)
      REFERENCES  cities (state_abbrev, city_seqno)
    ),
```

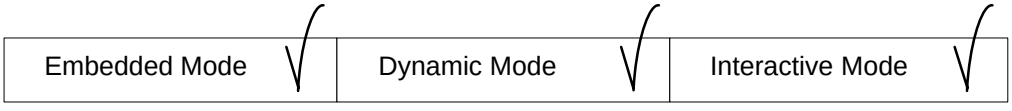
```
CREATE TABLE places
    (
      state_abbrev     CHAR (2) UQINDEX NOT NULL DEFAULT ADABAS,
      city_seqno       SEQNO(1)  NOT NULL,
      place_name       CHAR (20) NOT NULL SUPPRESSION,
      FOREIGN KEY (state_abbrev, city_seqno)
      REFERENCES  cities (state_abbrev, city_seqno)
    )
);
```

CREATE CLUSTER DESCRIPTION

Function:

This statement introduces an existing Adabas file including MU/PE fields to the SQL environment.

Invocation:



Syntax:

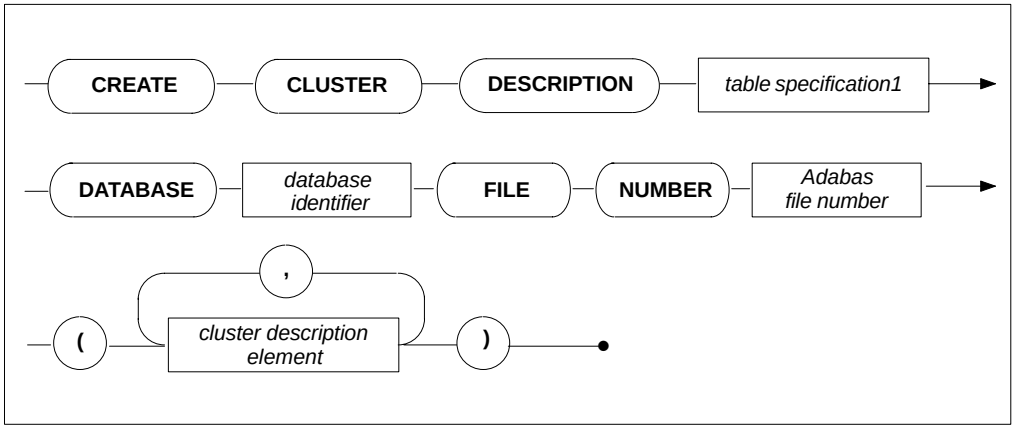


table specification1	represents the cluster to be created, qualified by the schema identifier
database identifier	represents the database in which the existing Adabas file resides.
Adabas file number	is a valid file number within an Adabas database system.

Cluster description element:

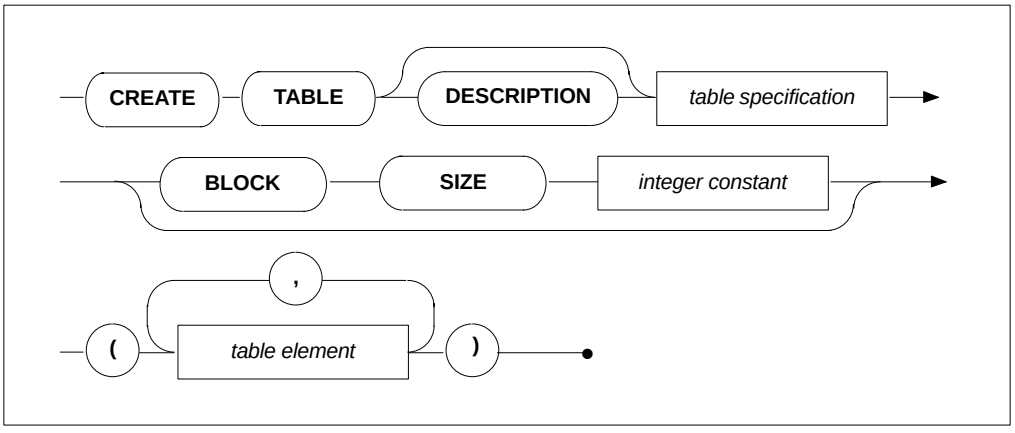


table specification	specifies the base tables (master tables and subtables) of the described cluster.
integer constant	a value between 1 and 191 is expected.
table element	see section Table Elements in chapter **Common Elements** of this manual.

Description:

A CREATE CLUSTER DESCRIPTION statement allows the description of an existing Adabas file that contains Adabas MU's and/or PE's, to Adabas SQL Server.

The BLOCK SIZE defined in a cluster element specifies the number of occurrences that will be multi-fetched in a MU/PE group. In the case of a MU within a PE, that multi-fetch size is the multiplication of the level one (PE) table BLOCK SIZE with that of the level 2 (MU). A good value for a BLOCK SIZE is either a prime number or odd number. The BLOCK SIZE has the following limitations: default value=: 7, minimum value = 1, maximum value = 191.

For further details of how to use this statement, refer to the *Adabas SQL Server Programmer's Guide*, chapter **Adabas SQL Server Data Structure**, section **Data in Table Clusters**.

Note:
The formulation of CREATE CLUSTER DESCRIPTION statements can be aided by the use of the Generate Table Description (ESQGTD) Utility.

Limitations:

Following rules apply:

1. Foreign keys reference only unique constraints. A subtable contains exactly one foreign key.
2. The same rules apply for the columns, constraints and indexes of the master table as for a CREATE TABLE DESCRIPTION statement.
3. Columns which are not an element of a foreign key and not of a SEQNO type are called data columns. The limitations under rules 4 – 7 apply to data columns in subtables.
4. The data columns of a level 1 table correspond either to MU fields which do not lie within a PE group, or to fields within a single PE group.
5. The data columns of a level 2 table correspond to MU fields within a specific PE group – the group containing those fields which the data columns in the referenced table correspond to.
6. Not more than one data column may correspond to each field (with rotated fields, each subscript counts as its own field).
7. With parallel MU fields, it is assumed that in all Adabas records, the respective counter values are the same.
8. For x=1 or x=2, a unique constraint of a level x table encompasses the elements of the foreign keys and a column of the type SEQNO(x). Other unique constraints on subtables are not allowed.
9. For indexes to subtables, the same rules apply as for level-0 tables, plus the following additional constraints:
   - HAVING UNIQUE INDEX is not allowed. In order to model the Adabas UQ option, UQINDEX is used.

   Note:
   A unique constraint is defined as either a UNIQUE or PRIMARY KEY constraint.
10. All level 0 columns must be grouped within one CREATE TABLE of a cluster.

    Note:
    In the case of a PE data structure containing MU fields only, it is necessary to use an Adabas short name on the SEQNO(1) of the PE–subtable.

ANSI Specifics:

This statement is not part of the ANSI standard

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

```
CREATE CLUSTER DESCRIPTION city_guide
DATABASE DB_214 FILE NUMBER 134
(
CREATE TABLE DESCRIPTION states (
   abbreviation     SHORTNAME 'AA' PRIMARY KEY DEFAULT ADABAS,
   state_name       SHORTNAME 'AB' UNIQUE NOT NULL DEFAULT ADABAS,
   capital          SHORTNAME 'AC' INDEX,
   population       SHORTNAME 'AD'
   ),
CREATE TABLE DESCRIPTION cities (
   state_abbrev     SHORTNAME 'AA',
   city_seqno       SEQNO(1)   NOT NULL DEFAULT ADABAS,
   city_name        SHORTNAME 'BA' INDEX NULL SUPPRESSION,
   population       SHORTNAME 'BB' NULL SUPPRESSION,
   PRIMARY KEY (state_abbrev, city_seqno),
   FOREIGN KEY (state_abbrev) REFERENCES states,
   UQINDEX ( city_name, state_abbrev)
   ),
CREATE TABLE DESCRIPTION buildings (
   state_abbrev     SHORTNAME 'AA',
   city_seqno       SEQNO(1)   NOT NULL DEFAULT ADABAS,
   building_seqno   SEQNO(2)   NOT NULL DEFAULT ADABAS,
   building_name    SHORTNAME 'CA' NOT NULL SUPPRESSION,
   height           SHORTNAME 'CB' NOT NULL SUPPRESSION,
   PRIMARY KEY (state_abbrev, city_seqno, building_seqno),
   FOREIGN KEY (state_abbrev, city_seqno) REFERENCES cities
   ),
CREATE TABLE DESCRIPTION places (
   state_abbrev     SHORTNAME 'AA' NOT NULL,
   city_seqno       SEQNO(1)   NOT NULL DEFAULT ADABAS,
   place_name       SHORTNAME 'DA' NULL SUPPRESSION,
   FOREIGN KEY (state_abbrev, city_seqno) REFERENCES cities
   )
);
```

Below is the corresponding Adabas FDT definition:

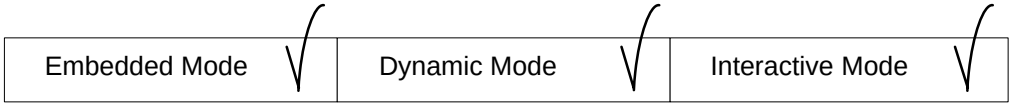
```
   Level  I Name I Length I Format I  Options  I  Comment
-------------------------------------------------------------------------------
  1       I  AA  I    2   I    A   I  DE,UQ    I  states.abbreviation
  1       I  AB  I   20   I    A   I  DE,UQ    I  states.state_name
  1       I  AC  I   20   I    A   I  DE,NC    I  states.capital
  1       I  AD  I    4   I    F   I  NC       I  states.population
  1       I  B0  I        I        I  PE       I  cities
   2      I  BA  I   20   I    A   I  DE,NU    I  cities.city_name
   2      I  BB  I    4   I    F   I  NU       I  cities.population
   2      I  CA  I   20   I    A   I  NU,MU    I  buildings.building_name
   2      I  CB  I    2   I    F   I  NU,MU    I  buildings.height
   2      I  DA  I   20   I    A   I  NU,MU    I  places.place_name
-------------------------------------------------------------------------------
   Type   I Name I Length I Format I  Options        I  Parent field(s)   Fmt
-------------------------------------------------------------------------------
 SUPER    I  X1  I   22   I    A   I  NU,UQ,PE       I  BA (   1 -  20 )   A
          I      I        I        I                  I  AA (   1 -   2 )   A
-------------------------------------------------------------------------------
```

CREATE DATABASE

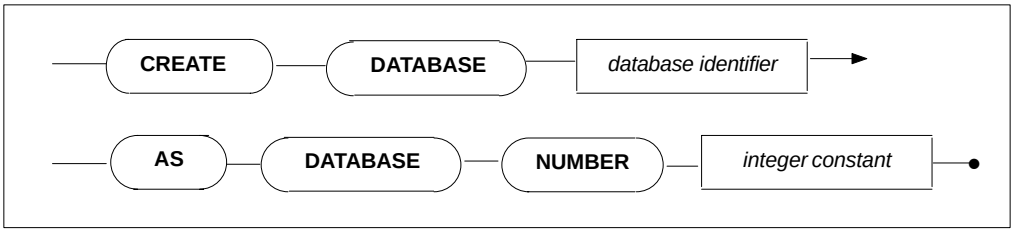
Function:

The CREATE DATABASE statement assigns a logical database identification to a physical database number.

Invocation:



Syntax:



database identifier	a valid database identifier representing a logical database name which must be known to Adabas SQL Server at runtime.
integer constant	the Adabas database number.

Description:

The CREATE DATABASE statement is used to assign a logical database identifier to databases otherwise defined by Adabas database numbers. In addition, the database number need not exist at compilation time.

Limitations:

The specified database identifier must be unique within the catalog. In addition, the database number must not already be associated with another database identifier within the catalog. The integer constant must be in the range from 1 to the maximum number of databases allowed by the underlying Adabas.

ANSI Specifics:

The CREATE DATABASE statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To assign the logical database identifier 'yachting' to an Adabas database number (75), the following syntax is used:

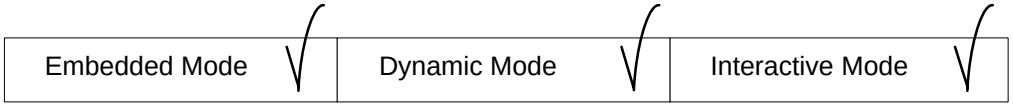
```
CREATE DATABASE yachting AS DATABASE NUMBER 75;
```

CREATE INDEX

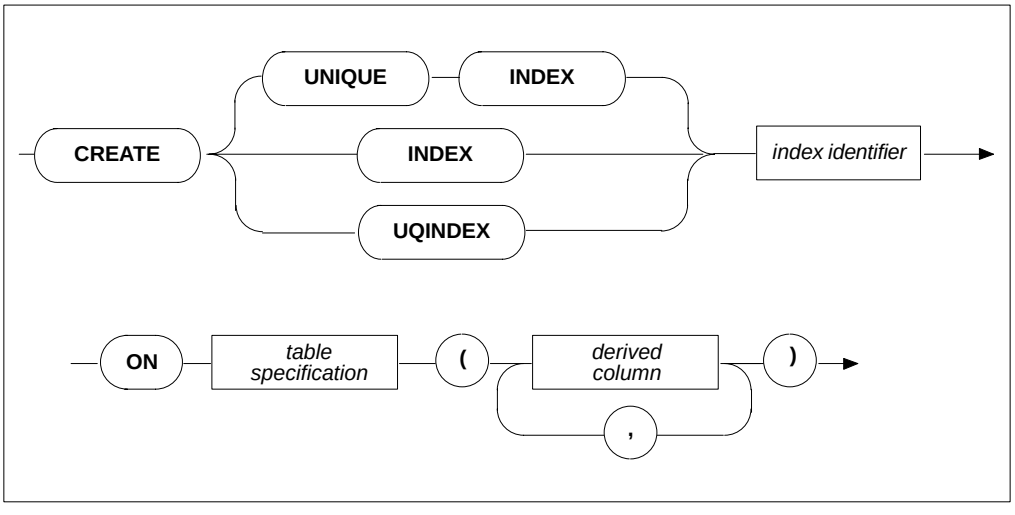
Function:

The CREATE INDEX statement establishes an index on one or more columns of an existing base table. Also, see section **Table Elements** in chapter **Common Elements.**

Invocation:



Syntax:



Description:

In order to improve the performance of an existing application, an index can be established for one or more column(s) of a base table.

The CREATE INDEX statement allows for the creation of an Adabas descriptor that reflects the capabilities of the Adabas C database system's definition of descriptors. For a detailed discussion of Adabas descriptors, please refer to the Adabas documentation for your environment, in particular the **Database Design** chapter of the *DBA Reference Manual*.

The following conventions hold true for the following explanations:

- Let C be the column for which this column index element is specified.
- Let T be the table where column C resides.

- INDEX:

  An INDEX is used to enable more efficient access to a base table. As such, the index is based on one or more column(s) of a base table, where the columns listed are considered to be an entity.

  - If the index is over only one column and that column has no range specification, then the column to which this index references will have an Adabas descriptor added.
  - If the number of columns in the column list is greater than one, then an Adabas superdescriptor will be generated.
  - In the case of a range specification across only one column an Adabas subdescriptor will be generated.

- UNIQUE INDEX:

  A UNIQUE INDEX ensures that there are no two rows of T having identical values in the columns of C. Rows with NULL value(s), in any columns of C, do not effect this index.

  A UNIQUE INDEX implies a UNIQUE constraint, and is stored in the Adabas SQL Server catalog as a UNIQUE constraint.

- UQINDEX:

  The UQINDEX is used to have a mapping for the Adabas UQ option in cases where it can not be mapped to an SQL UNIQUE constraint. UQINDEX will generate an Adabas unique descriptor on a column that is part of a subtable. This descriptor is not considered to be unique in SQL terms and, therefore, can not be represented by a normal 'UNIQUE constraint'.

  A UQINDEX specification of C results in the definition of an unique Adabas descriptor in the physical representation of T.

  For more details see the section **Indexes and Constraints**, chapter **Adabas SQL Server Data Structures** in the *Adabas SQL Server Programmer's Guide*.

Limitations:

The UQINDEX is allowed in subtables (tables of level 1 or 2) only.

ANSI Specifics:

The CREATE INDEX statement is not part of the ANSI Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To create a unique index on the column id_customer of the table contract, the following syntax applies:

```
CREATE UNIQUE INDEX contract_2 ON contract( id_customer );
```

CREATE SCHEMA

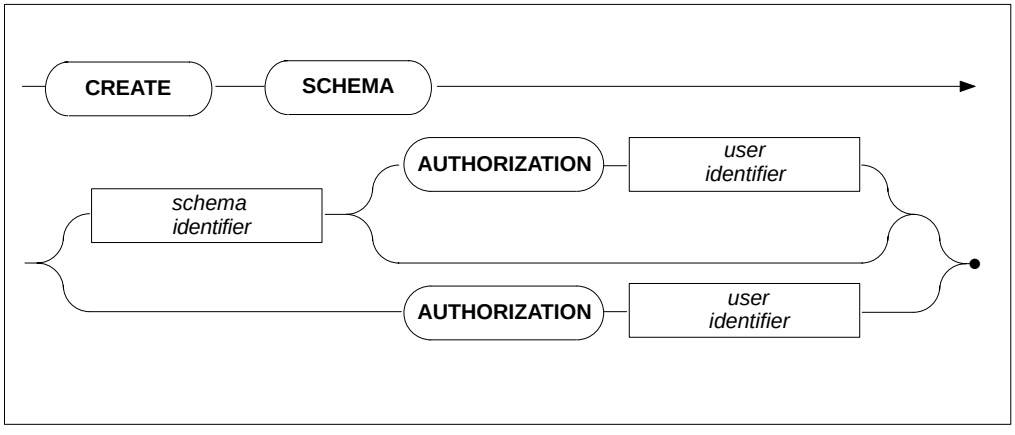
Function:

The CREATE SCHEMA statement creates creates an SQL-Schema which serves as a logical container for the subsequent creation of catalog resident objects e.g. tables, views etc.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√	√	√

Syntax:



schema identifier	is a valid schema identifier representing the schema to be created.
user identifier	is a valid user identifier of an existing user.

Description:

The CREATE SCHEMA statement causes a schema to be entered into the catalog. The name of the schema is:

- explicitly provided as the schema identifier in the CREATE SCHEMA statement or
- derived from the user identifier, if the schemas identifier has been omitted.

A schema must have an owner. This owner can be explicitly specified in the AUTHORIZATION clause as a user identifier. If this step is omitted, however, the user identifier is derived from the user identifier of the executor of the statement. In either case the resulting user identifier must be equal to a known user of the server.

If the statement is invoked statically, then during pre-compilation, any user identifier need not exist in the catalog. For successful execution, however, the user identifier must exist in the catalog, regardless of how it is invoked.

Limitations:

The resulting schema identifier must be unique within the catalog.

The user identifier must be equal to an already defined user identifier as defined using a CREATE USER statement.

Note:
The CREATE SCHEMA statement can only be executed by the DBA.

ANSI Specifics:

The CREATE SCHEMA statement provided by Adabas SQL Server is a cut-down version of that specified in the SQL-2 standard. In particular, it is not possible to specify the object that belongs to a schema in this statement (i.e., base tables, views and constraints).

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To create a schema with the name Wiltshire and assign it to the owner TIM the following syntax is used :

```
CREATE SCHEMA Wiltshire AUTHORIZATION TIM ;
```

To create a schema with the name TIM and assign it to the owner TIM the following syntax is used :

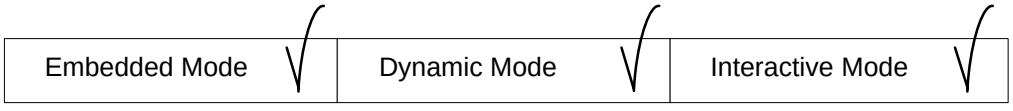
```
CREATE SCHEMA AUTHORIZATION TIM ;
```

CREATE TABLE

Function:

The CREATE TABLE statement defines a table in the catalog and physically creates the table in Adabas.

Invocation:



Syntax:

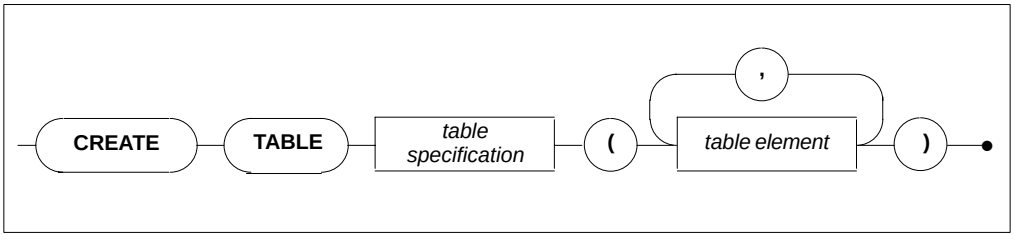


table specification — the expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified.

table element — either a table column element, table constraint element *or* table index element. For details, see chapter **Common Elements**, section **Table Elements**.

Description:

The CREATE TABLE statement defines the logical structure of a base table. From this logical structure the physical structure is derived. A stand-alone CREATE TABLE statement always allocates one physical table and defines one base table. The necessary physical table attributes are derived from the tablespace with the same table specification or from a default tablespace definition.

Compiling the statement does not create a table. Therefore, other statements cannot reference the table.

There is a different file creation behaviour on mainframe platforms compared to open systems platforms due to the possibility of keeping the FDTs of unloaded files:

- When using the FILE parameter, only the specified file number will be used to create the corresponding Adabas file. If a file with the specified file number already exists an error occurs. This is true for all platforms. On mainframe platforms only; if the specified file number is that of an unloaded file where the FDT has been kept, the FDT will be overwritten.
- When using the ADABAS FREE FILE SEARCH RANGE method, an unloaded file with existing FDT is not classified as a valid candidate for the CREATE TABLE or CREATE Cluster statement.

Limitations:

The table specification must be unique within a schema. A table must have at least one column. Adabas SQL Server does not permit tables to have more than 926 columns.

Adabas SQL Server adheres to the relational database theory where tables are strictly two dimensional and, therefore, periodic groups and multiple fields which are available in Adabas are not permitted for tables specified with the CREATE TABLE statement. For cases where nested data structures, i.e., MU/PE fields are involved, a special statement, CREATE CLUSTER, is provided.

The underlying Adabas database system has to run on the same machine as Adabas SQL Server and can not be accessed using Net-Work.

The following column attributes may not be specified in this statement:
UQINDEX
SUPPRESSION
FIXED
SHORTNAME definition
REFERENCES.

The following table clause(s) may not be specified in this statement:
UQINDEX
FOREIGN KEY.

ANSI Specifics:

The ANSI SQL Standard does not allow for independent table creation outside of an CREATE SCHEMA statement Therefore, this statement is not part of the ANSI SQL Standard.

Furthermore, the column default value of “ADABAS” is not part of the ANSI SQL Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming**, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The example below illustrates the syntax required to define and create the table CONTRACT (as defined in **Appendix A – The Sample Table** of the *Adabas SQL Server Programmer's Guide*).

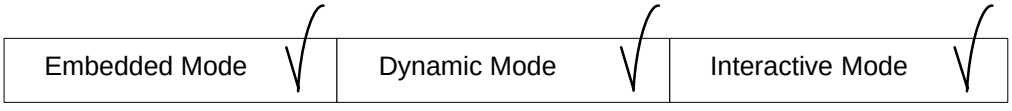
```
CREATE TABLE contract ( contract_id
    integer index ind_contract not null unique,
    price             numeric (13,3) not null,
    date_reservation  integer,
    date_booking      integer,
    date_cancellation integer,
    date_deposit      integer,
    amount_deposit    integer,
    date_payment      integer,
    amount_payment    numeric (13,3),
    id_customer       integer not null,
    id_cruise         integer not null );
```

CREATE DEFAULT TABLESPACE

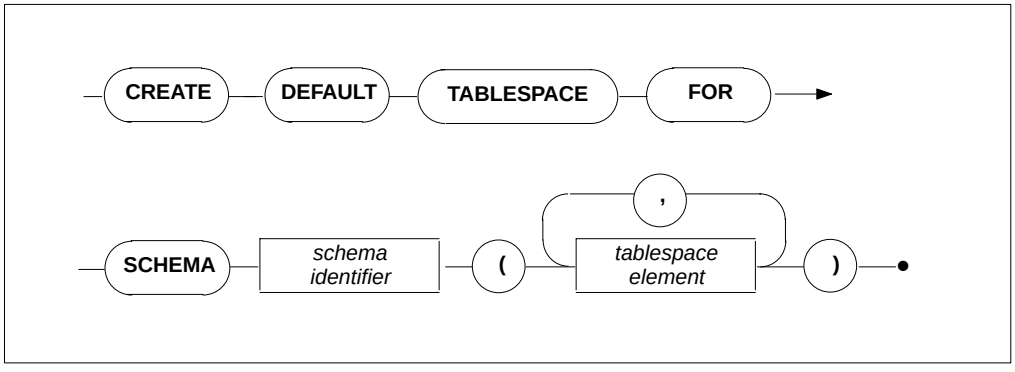
Function:

This statement introduces the definition of a default tablespace template for an Adabas file to the catalog. The definition is schema-specific and will be used as a default for all CREATE TABLE statements under this schema. These default settings can be modified for individual tables within this schema by using the CREATE TABLESPACE statement.

Invocation:



Syntax:



schema identifier identifies the SCHEMA for which this default tablespace is to be created.

tablespace element one element in a set of parameters describing the Adabas-specific file definitions. The syntax is listed below:

```
ASSOPFAC = numeric_int_literal
DATABASE = database_identifier
DATAPFAC = numeric_int_literal
DSDEV = numeric_int_literal
DSREUSE = ( YES | NO )
DSSIZE = numeric_int_literal [ BLOCK|MEGABYTE ]
ISNREUSE = ( YES | NO )
ISNSIZE = numeric_int_literal
MAXDS = numeric_int_literal
MAXISN = numeric_int_literal
MAXNI = numeric_int_literal
MAXRECL = numeric_int_literal
MAXUI = numeric_int_literal
MIXDSDEV
NISIZE = numeric_int_literal [ BLOCK|MEGABYTE ]
[ NO ] CONTIGUOUS_AC
[ NO ] CONTIGUOUS_DS
[ NO ] CONTIGUOUS_NI
[ NO ] CONTIGUOUS_UI
PGMREFRESH = ( YES | NO )
REUSE = ( ( ISN | DS ) [ , ( ISN | DS ) ] )
UISIZE = numeric_int_literal [ BLOCK | MEGABYTE ]
```

where:

[NO]	= optional, separated by a blank
(YES | NO)	= alternatives

Description:

The CREATE DEFAULT TABLESPACE statement describes the Adabas-specific file attributes to the SQL environment. The scope is not one single table, like the CREATE TABLESPACE statement, but it will define default settings for all tables to be created under the referenced schema.

The default settings defined by the CREATE DEFAULT TABLESPACE statement may be overridden for individual tables under that schema by explicitly using the CREATE TABLESPACE statement.

Compiling the statement does not create the table or tablespace. Therefore, other statements cannot reference the tablespace specified in the CREATE DEFAULT TABLESPACE statement until the statement has been successfully executed.

The following is a list of the parameters available for the individual platforms. A check for reasonable values is not executed. For further details, refer to the *Adabas Utilities Manual* for your platform.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

For general details about tablespaces see the chapter **The ADBAS SQL Server Data Structures** in the *Adabas SQL Server Programmer's Guide.*

On mainframe platforms, when using the FREE FILE SEARCH RANGE (specified using the parameter processing language PPL), a file number referencing an unloaded file with a kept FDT, will not be classified as a valid candidate in a CREATE TABLE or CREATE CLUSTER statement.

Parameter	MVS	VMS	UNIX	Description
ASSOPFAC	x	x	x	Associator padding factor
CONTIGUOUS_DS		x	x	Contiguous allocation for file loading
CONTIGUOUS_AC		x	x	Contiguous allocation for file loading
CONTIGUOUS_NI		x	x	Contiguous allocation for file loading
CONTIGUOUS_UI		x	x	Contiguous allocation for file loading
DATABASE	x	x	x	Database identifier (name)*****
DATAPFAC	x	x	x	Data storage padding factor
DSDEV	x			Data storage device type
DSREUSE	x	x	x	Data storage reusage
DSSIZE	x	x	x	Extent size for data storage
ISNREUSE	x	x	x	ISN reusage
ISNSIZE	x	x	x	ISN size in the normal index ******
MAXDS	x			Max. secondary allocation f. data storage
MAXISN	x	x	x	Highest ISN to be allocated
MAXNI	x			Max. secondary alloc. for normal index
MAXRECL	x			Max. compressed record length
MAXUI	**x**			Max. secondary alloc. for upper index
MIXDSDEV	x			Data storage mixed device types
NISIZE	x	x	x	Normal index size
PGMREFRESH	x			Program-generated file refresh
REUSE	x	x	x	Reusage of data storage or ISNs
UISIZE	x	x	x	Upper index size

x = available on marked platform

Note:
****** In contrast to Adabas, where DBID is specified as a number, the parameter DATABASE within the CREATE DEFAULT TABLESPACE/CREATE TABLESPACE statements is specified as a database identifier.*

Limitations:

If neither a CREATE DEFAULT TABLESPACE nor a CREATE TABLESPACE statement has been established before executing a CREATE TABLE statement, an Adabas SQL Server default setting will be used.

The REUSE parameter can only be used to activate, not to deactivate, DS and/or ISN re–usage. For example, if REUSE = (ISN) is specified, then an implicit REUSE = DS is generated by default. To deactivate the unwanted DS re–usage the explicit specification of the parameter DSREUSE = NO is necessary.

ANSI Specifics:

The CREATE DEFAULT TABLESPACE statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter General Concepts of SQL Programmaing, section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The example below illustrates how to define possible attributes of a schema-specific DEFAULT TABLESPACE.

For details on the Adabas file definition elements, refer to the *Adabas Utilities Manual* for your platform.

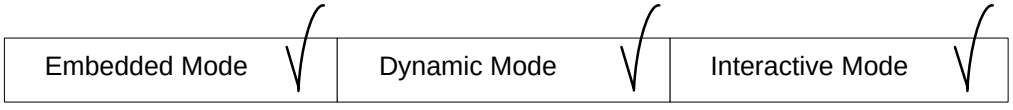
```
CREATE DEFAULT TABLESPACE FOR SCHEMA sagtours
                  (
                  ASSOPFAC=3,
                  DATAPFAC=2,
                  DSSIZE=20BLOCK,
                  NISIZE=10BLOCK,
                  UISIZE=10BLOCK,
                  MAXISN=2000,
                  REUSE=(ISN,DS)
                  );
```

CREATE TABLESPACE

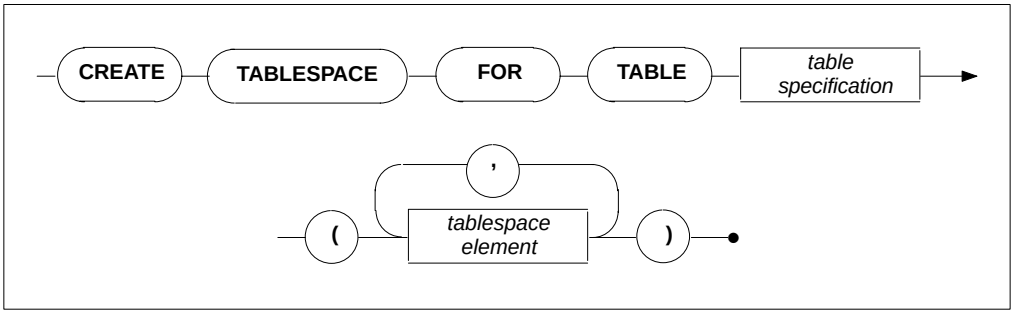
Function:

This statement introduces the specific definitions of an Adabas file to the catalog. These definitions must be known to Adabas SQL Server before the related CREATE TABLE statement may be executed.

Invocation:



Syntax:



tablespace element

one element in a set of parameters describing the Adabas-specific file definitions. For details/limitations (for example: FILENAME= max. of 16 chars) also refer to the *Adabas Utilities Manual* for your platform. The syntax is listed below:

ACRABN = numeric_int_literal
ASSOPFAC = numeric_int_literal
DATABASE = database_identifier
DATAPFAC = numeric_int_literal
DSDEV = numeric_int_literal
DSRABN = numeric_int_literal
DSREUSE = (YES | NO)
DSSIZE = numeric_int_literal [BLOCK|MEGABYTE]
FILE = numeric_int_literal
FILENAME = string_literal
ISNREUSE = (YES | NO)
ISNSIZE = numeric_int_literal
MAXDS = numeric_int_literal
MAXISN = numeric_int_literal
MAXNI = numeric_int_literal
MAXRECL = numeric_int_literal
MAXUI = numeric_int_literal
MIXDSDEV
NIRABN = numeric_int_literal
NISIZE = numeric_int_literal [BLOCK|MEGABYTE]
[NO] CONTIGUOUS_AC
[NO] CONTIGUOUS_DS
[NO] CONTIGUOUS_NI
[NO] CONTIGUOUS_UI
PGMREFRESH = (YES | NO)
REUSE = ((ISN | DS) [, (ISN | DS)])
UIRABN = numeric_int_literal
UISIZE = numeric_int_literal [BLOCK | MEGABYTE]

[NO] = optional, separated by a blank
(YES | NO) = alternatives

Description:

The following is a list of the parameters available for the individual platforms. A check for reasonable values is not executed. For further details, refer to the *Adabas Utilities Manual* for your platform.

Parameter	MVS	VMS	UNIX	Description
ACRABN	x	x	x	Starting RABN for address converter
ASSOPFAC	x	x	x	Associator padding factor
CONTIGUOUS_DS		x	x	Contiguous allocation for file loading
CONTIGUOUS_AC		x	x	Contiguous allocation for file loading
CONTIGUOUS_NI		x	x	Contiguous allocation for file loading
CONTIGUOUS_UI		x	x	Contiguous allocation for file loading
DATABASE	x	x	x	Database identifier (name) *
DATAPFAC	x	x	x	Data storage padding factor
DSDEV	x			Data storage device type
DSRABN	x	x	x	Starting RABN for data storage
DSREUSE	x	x	x	Data storage reusage
DSSIZE	x	x	x	Extent size for data storage
FILE	x	x	x	File number
FILENAME	x	x	x	File name (limited to 16 characters)
ISNREUSE	x	x	x	ISN reusage
ISNSIZE	x	x	x	ISN size in the normal index **
MAXDS	x			Max. secondary allocation f. data storage
MAXISN	x	x	x	Highest ISN to be allocated
MAXNI	x			Max. secondary alloc. for normal index
MAXRECL	x			Max. compressed record length
MAXUI	x			Max. secondary alloc. for upper index
MIXDSDEV	x			Data storage mixed device types
NIRABN	x	x	x	Starting RABN for normal index
NISIZE	x	x	x	Normal index size
PGMREFRESH	x			Program-generated file refresh
REUSE	x	x	x	Reusage of data storage or ISNs
UIRABN	x	x	x	Starting RABN for upper index
UISIZE	x	x	x	Upper index size

x = available on marked platform

Note:
***** *In contrast to Adabas, where DBID is specified as a number, the parameter DATABASE within the CREATE DEFAULT TABLESPACE/CREATE TABLESPACE statements is specified as a database identifier.*

Note:
****** *Under MVS, the parameter ISNSIZE is valid only for Adabas Version 6.1 and higher.*

The CREATE TABLESPACE statement describes the Adabas-specific file attributes to the SQL environment.

If the default settings of file attributes for a given schema, as specified using the CREATE DEFAULT TABLESPACE statement, do not match the table-specific requirements, then the CREATE TABLESPACE statement can be used.

Compiling the statement does not create the table or tablespace. Therefore, other statements cannot reference the tablespace specified in the CREATE TABLESPACE statement until the statement has been successfully executed.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

There is a different file creation behaviour on mainframe platforms compared to open systems platforms due to the possibility of keeping the FDTs of unloaded files:

– When using the FILE parameter, only the specified file number will be used to create the corresponding Adabas file. If a file with the specified file number already exists an error occurs. This is true for all platforms. On mainframe platforms only; if the specified file number is that of an unloaded file where the FDT has been kept, the FDT will be overwritten.
– When using the ADABAS FREE FILE SEARCH RANGE method, an unloaded file with existing FDT is not classified as a valid candidate for the CREATE TABLE or CREATE Cluster statement.

For general information about tablespaces see the section **Adabas SQL Server Data Structures** in the *Adabas SQL Server Programmer's Guide.*

Limitations:

If FILENAME is not specified, the qualified table identifier is used by default. Should the latter exceed the length of 16 characters, it will be cut, as the FILENAME parameter value is limited to 16 characters.

The table specified in the CREATE TABLESPACE statement must not exist at runtime.

The REUSE parameter can only be used to activate, not to deactivate, DS and/or ISN re–usage. For example, if REUSE = (ISN) is specified, then an implicit REUSE = DS is generated by default. To deactivate the unwanted DS re–usage the explicit specification of the parameter DSREUSE = NO is necessary.

ANSI Specifics:

The CREATE TABLESPACE statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The example below illustrates how to define possible attributes of an Adabas file to the catalog for the creation of the table CONTRACT.

For details on the Adabas file definition elements please refer to the *Adabas Utilities Manual* for your platform.

```
CREATE TABLESPACE FOR TABLE contract
                  (
                  DATABASE=yacht_db,
                  FILE=21,
                  FILENAME="CONTRACT",
                  ASSOPFAC=5,
                  DATAPFAC=5,
                  DSSIZE=10BLOCK,
                  NISIZE=10BLOCK,
                  UISIZE=10BLOCK,
                  MAXISN=300,
                  REUSE=(ISN,DS)
                  );
```

CREATE TABLE DESCRIPTION

Function:

Introduces an existing Adabas file to the SQL environment.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√	√	√

Syntax:

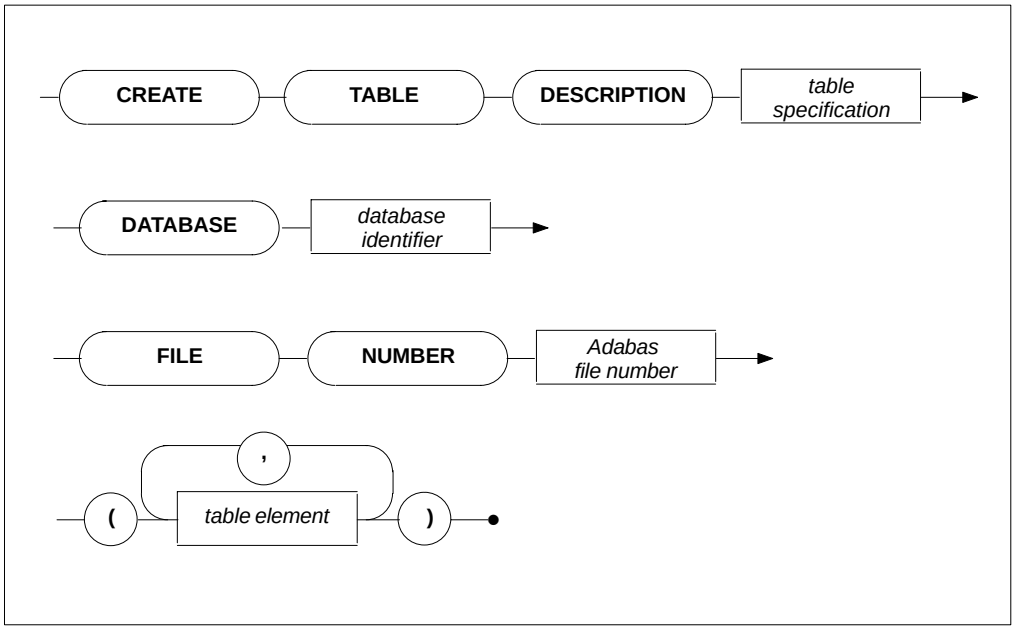


table specification	the expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified.
database identifier	as defined in the CREATE DATABASE statement with a length of 32 characters.
Adabas file number	is a valid file number within an Adabas database system.
table element	table column element, table unique element or table index element. For details, see chapter **Common Elements**, section **Table Elements**.

Description:

The CREATE TABLE DESCRIPTION statement is used to specify the already existing Adabas files for the SQL environment in the catalog. The statement consists of a table specification and a list of table elements. If a schema identifier is given in the table specification, then the table identifier will thus be explicitly qualified, otherwise the current default schema identifier will be used.

Compiling the statement does not create the table. Therefore, other statements can not reference the table specified in the CREATE TABLE DESCRIPTION statement until the statement has been successfully executed.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

This statement may also use the technique of rotating a MU/PE's fields into base columns. This allows each element of a MU/PE field, to be referenced as a separate column within a base table.

Because this statement executes on an already existing Adabas file, it is possible to specify a minimal of information, the rest will be generated for you by the SQL compiler. The minimal information that must be specified in this statement is the column identifier and the Adabas short name for this column. All other information will be generated from the underlying Adabas file.

Note:
If you specify more than the minimal information, it will be checked for correctness against that of the underlying Adabas file. More specifically, if you specify the column attributes NULL or NOT NULL on a field that does not have NC or NN,NC attributes, then the extra qualification of SUPPRESSION or DEFAULT ADABAS is needed.

For more details on what Adabas field attributes represent which Adabas SQL Server column attributes, see the section: **Data Structures**, sub-heading **Converting Adabas Fields To Adabas SQL Server Columns** in the *Adabas SQL Servers Programmer's Guide*.

Limitations:

The table specification must be unique within the catalog at runtime. A table must have at least one column. Adabas SQL Server does not permit tables to have more than 900 columns. The limit of 926 may be exceeded when rotated fields are being used.

The following column attributes may not be specified in this statement:
UQINDEX
REFERENCES

The following table clause(s) may not be specified in this statement:
UQINDEX
FOREIGN KEY

ANSI Specifics:

This statement is not part of the SQL standard.

The column default value of "ADABAS" is not part of the SQL standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Examples:

The examples below illustrates the syntax required to define and create the table CONTRACT (as defined in **Appendix A** of the *Adabas SQL Server Programmer's Guide*) where an Adabas file already exists but there is no table definition in the catalog.

```
1 a: Detailed format

CREATE TABLE DESCRIPTION contract DATABASE SAGTOURS FILE NUMBER 21
     (contract_id      integer           SHORTNAME AA
                                         index ind_contract not null unique,
     price             NUMERIC (13,3)    SHORTNAME AB not null,
     date_reservation  INTEGER           SHORTNAME AD,
     date_booking      INTEGER           SHORTNAME AG,
     date_cancellation INTEGER           SHORTNAME AH,
     date_deposit      INTEGER           SHORTNAME AJ,
     amount_deposit    NUMERIC (13,3)    SHORTNAME BA,
     date_payment      INTEGER           SHORTNAME BB,
     amount_payment    NUMERIC (13,3)    SHORTNAME BE,
     id_customer       INTEGER           SHORTNAME CA not null,
     id_cruise         INTEGER           SHORTNAME CD not null);

1 b: Minimal format

CREATE TABLE DESCRIPTION contract
DATABASE SAGTOURS FILE NUMBER 21
     (contract_id      SHORTNAME AA,
     price             SHORTNAME AB,
     .
     .
     id_cruise         SHORTNAME CD);
```

The next example shows how the elements of an MU may be rotated into base columns. The examples is for a table containing a sales persons bonuses for each month of the current year.

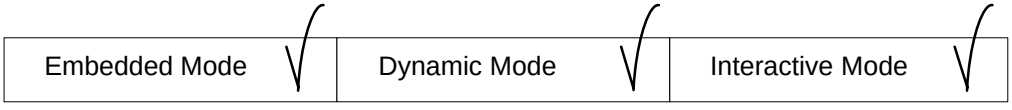
```
CREATE TABLE DESCRIPTION sales_bonuses
DATABASE ESQDEMO_203 FILE NUMBER 15
(    id           SHORTNAME ”AA”,
     surname      SHORTNAME ”AB”,
     first_name   SHORTNAME ”AC”,
     jan_bonus    SHORTNAME ”AD” ( 1 ),
     feb_bonus    SHORTNAME ”AD” ( 2 ),
     mar_bonus    SHORTNAME ”AD” ( 3 ),
     apr_bonus    SHORTNAME ”AD” ( 4 ),
     may_bonus    SHORTNAME ”AD” ( 5 ),
     jun_bonus    SHORTNAME ”AD” ( 6 ),
     jul_bonus    SHORTNAME ”AD” ( 7 ),
     aug_bonus    SHORTNAME ”AD” ( 8 ),
     sept_bonus   SHORTNAME ”AD” ( 9 ),
     oct_bonus    SHORTNAME ”AD” ( 10 ),
     nov_bonus    SHORTNAME ”AD” ( 11 ),
     dec_bonus    SHORTNAME ”AD” ( 12 ));
```

CREATE USER

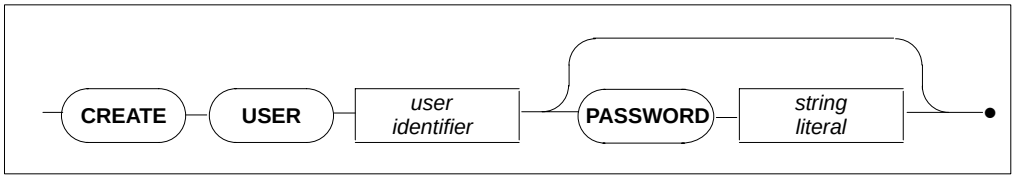
Function:

The CREATE USER statement establishes a user as a prerequisite to having access to Adabas SQL Server.

Invocation:



Syntax:



user identifier: a unique identifier for a user.

string literal: a valid password string with a maximum length of 20 bytes.

Description:

The CREATE USER statement installs a user with an optional password. If a password has been specified, it must be entered by the user to gain access to the system. If no password has been specified, there will be no password protection for this user. A password can be added later using the ALTER USER statement.

Note:
The password to be provided is visible in this statement. It will then be encrypted internally.

Limitations:

Only the designated DBA may execute this statement.

The user identifier must be unique.

ANSI Specifics:

The CREATE USER statement is not part of the Standard

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To install and define the user TIM with a password, the following syntax applies. If this user identification TIM already exists, an error message will be issued. The password must be typed in as it appears:

```
CREATE USER TIM PASSWORD ”&M%I?T”;
```

CREATE VIEW

Function:

The CREATE VIEW statement is used to create a viewed table derived from one or more base tables or views.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√	√	√

Syntax:

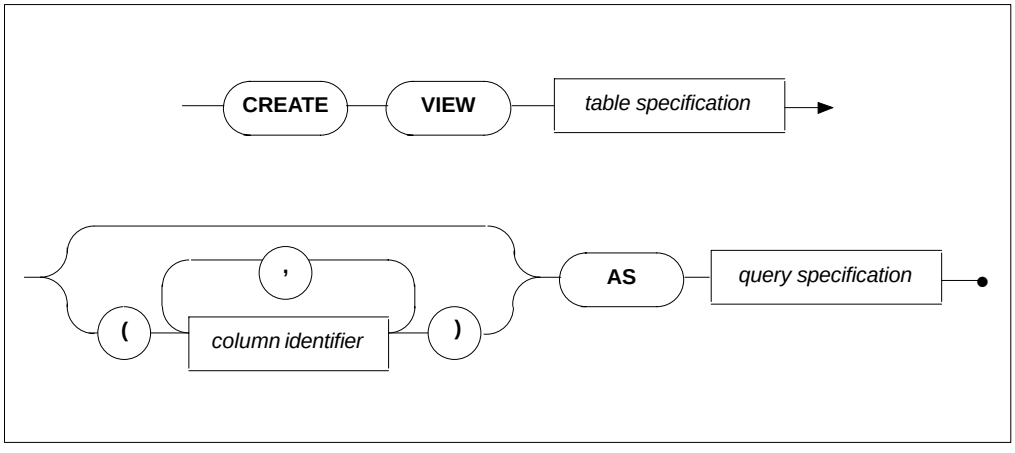


table specification	the expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified.
column identifier	specifies the column identifiers of a view and must not be longer than 32 characters.
query specification	must be any valid query specification. For details see chapter **Common Elements** section **Query Specification.**

Description:

The CREATE VIEW statement is used to specify a viewed table, also called view. A view is a virtual table and therefore, has no physical representation. Values are conceptually derived from base tables as the need arises. If a schema identifier is given in the table specification, than the table identifier will thus beexplicitly qualified, otherwise the current default schema identifier will be used.

The column identifier list specifies the number and order in which the columns will appear in the view. The number of column identifiers must equal the number of derived columns defined in the query specification. The *i*th column identifier represents the *i*th derived column and assumes its data type. Furthermore, two columns within the column identifier list may not be called the same.

If no column identifier list is specified, then the columns of the view are identified by the unqualified derived column labels of the query specification. If there is no label for a particular derived column, then the complete column identifier list must be specified.

- A view is called a **"joined view"** if more than one table has been specified or a joined view has been referenced in the FROM clause.
- A view is called a **"grouped view"** if the view is derived from a grouped query specification.
- A view is called a **"read-only view"** if the view is either grouped or joined or at least one of the derived columns does not have a label.

Only after successful execution of the statement is the view generally available. During execution the view description is stored in the catalog.

Limitations:

The table specification must be unique within an SQL environment, at runtime.

The number of column identifiers specified in the desired column list must be identical to the number of derived columns given in the query specification.

If no column identifier list is specified, all derived columns must have labels.

The query specification may not reference host variables.

The query specification may not reference the view which is the subject of the CREATE VIEW statement.

A view can not be updated when:

- it is a joined or grouped view, as described above
- a derived column is a literal
  (CREATE VIEW xyz AS SELECT col1, 'London' FROM table1)
- a derived column is an expression
  (CREATE VIEW xyz AS SELECT col1+3 FROM table1)

ANSI Specifics:

Within the ANSI concept the CREATE VIEW statement must be embedded in a CREATE SCHEMA statement. The SCHEMA as defined in ANSI is not fully supported by Adabas SQL Server.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

In this example a view named 'United States' is created which contains all of the information about people who live in United States.

```
CREATE VIEW united_states
     AS SELECT * FROM persons
     WHERE country = 'USA' ;
```

Once the above view is created it can be used to access information as if it is a normal table and the syntax below shows how to select the person ID of all the people living in PHILADELPHIA.

```
SELECT person_id
     FROM united_states
     WHERE united_states.city = 'PHILADELPHIA' ;
```

DEALLOCATE PREPARE

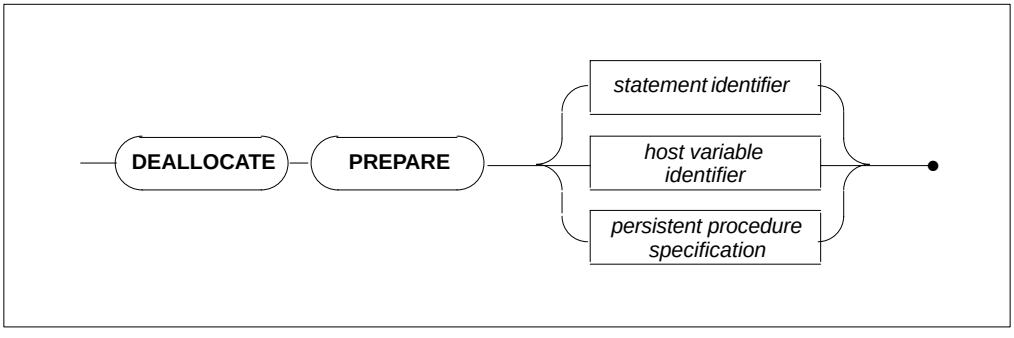
Function:

The DEALLOCATE PREPARE statement deallocates a prepared statement by releasing all associated resources. After the successful execution of a DEALLOCATE PREPARE statement the relevant prepared statement no longer exists and can, therefore, not be addressed anymore.

Invocation:

Embedded Mode ✓	Dynamic Mode	Interactive Mode

Syntax:



statement identifier	is a valid identifier used to identify the statement to be deallocated
host variable identifier	is a valid single host variable identifier and must hold the statement identifier.
persistent procedure specification	specifies the identification of the prepared statement that has been stored in the catalog. The persistent procedure specification must not include a VERSION clause. If the PROCEDURE clause is omitted all procedures of the specified module will be deallocated.

Description:

The effect of a DEALLOCATE PREPARE statement is that the identified statement will be destroyed.

If a persistent procedure specification is given the statement will be deleted from the catalog. In case a PROCEDURE clause is present, a single procedure will be deleted from the catalog. Otherwise, the execution affects one whole module.

The effect of a DEALLOCATE PREPARE statement is also achieved implicitly when an already existing prepared statement is specified in a PREPARE statement. For non-persistent statements, an implicit DEALLOCATE PREPARE statement also occurs in either of the following situations:

- a COMMIT or ROLLBACK executed in DB2 mode
- a DISCONNECT is issued to end a session.

Limitations:

All cursors must be closed before executing a DEALLOCATE PREPARE statement.

The relevant persistent procedure specification must not contain a VERSION clause.

If the relevant persistent procedure specification does not contain the PROCEDURE clause, all procedures of the specified module will be deallocated.

ANSI Specifics:

The DEALLOCATE PREPARE statement is not part of the Standard.

Adabas SQL Server Specifics:

This statement may be mixed with any other DML, DDL and/or DCL statements in the same transaction.

Example:

The statement identified by *statement_id* will be explicitly destroyed by:

```
DEALLOCATE PREPARE statement_id;
```

A persistent procedure identified by the names in the host variables mod and proc will be destroyed by following statement:

```
DEALLOCATE PREPARE MODULE :mod PROCEDURE :proc;
```

The whole module can be destroyed as follows:

```
DEALLOCATE PREPARE MODULE :mod;
```

DECLARE CURSOR

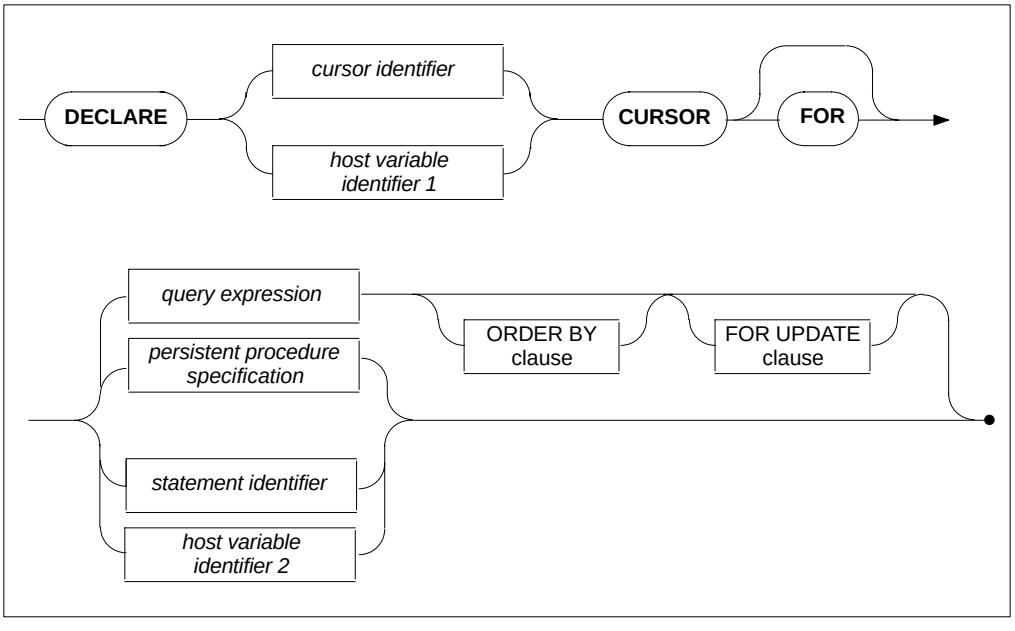
Function:

The DECLARE CURSOR statement associates a query expression and hence a resultant table with a cursor identifier. The statement only defines the contents of the resultant table; it does not establish it.

Invocation:

Embedded Mode ✓	Dynamic Mode	Interactive Mode

Syntax:



cursor identifier	is a valid identifier of no more than 18 characters and which has not previously been used as a cursor identifier within the same compilation unit.
host variable identifier 1	is a valid single host variable which is used to contain a unique dynamic cursor identifier.
query expression	is the specification of the resultant table associated with this cursor.
statement identifier	is a valid SQL identifier identifying a SELECT statement which has previously been prepared.
host variable identifier 2	is a valid single host variable. The value of the host variable must be the value returned by the PREPARE statement and thus identify the prepared statement.
persistent procedure specification	specifies the identification of the prepared statement that has been stored in the catalog. The persistent procedure specification must include both clauses.
ORDER BY clause	is the specification of a user-defined ordering of the resultant table. Otherwise the resultant table is not ordered.
FOR UPDATE clause	is the explicit indication that this cursor is to be used in conjunction with either an UPDATE and/or DELETE WHERE CURRENT OF *cursor-id* statement.

Description:

A cursor can be declared either as static using a static DECLARE CURSOR statement or as dynamic using a dynamic DECLARE CURSOR statement.

The static DECLARE CURSOR statement

A static DECLARE CURSOR statement associates a query expression and the definition of a resultant table with an SQL identifier, namely the cursor identifier. The DECLARE CURSOR statement is only a definition. The OPEN statement associated with this cursor establishes the resultant table at execution time.

Although the characteristics of the derived column list are completely defined, the actual number of rows returned is unknown until execution time. In other words the format of each row associated with the cursor is known but the number of rows established upon opening the cursor is not. This is in direct contrast to the SINGLE ROW SELECT where by definition only one row may be returned. The host program is, therefore, not in a position to receive all the data established upon opening and must sequentially execute associated FETCH statements in order to retrieve one row at a time. This is the classic DECLARE-OPEN-FETCH cycle. The cursor identifier can be thought of as a pointer into the resultant table identifying the row currently under consideration. In general, executing an associated FETCH statement advances the pointer by one row.

In addition to the OPEN and FETCH statements, other associated statements are positioned UPDATE, positioned DELETE and CLOSE.

The query expression defines the resultant table associated with the cursor. In theory, the expression can be unlimited in complexity. Certain query expressions are considered to be 'updatable', i.e. the positioned DELETE or UPDATE statements are valid for this cursor.

Updatable Cursors

For a cursor to be updatable the following rules must be observed:

- The specification of a UNION operator in a query expression is not allowed. Therefore, the expression must consist of only one query specification.
- Derived columns in the derived column list must be based on base tables not views. No operators, functions or literals are allowed in the derived column list.
- No column may be specified more than once in the derived column list of the query specification.
- The specification of DISTINCT in the derived column list is not allowed.
- A grouped or joined query specification is not allowed.
- If a subquery is specified, it may not reference the same table as that one referenced in the outer query, i.e the table which would be the subject of any amendment statement.
- If the query specification is derived from a view that view must be updatable.
- An ORDER BY clause is not specified.
- A FOR FETCH ONLY clause is not specified.

If the above conditions for a read-only cursor have been met, positioned UPDATE or DELETE statements will result in compilation errors.

Non-Updatable Cursors

A static cursor can be explicitly declared as being non-updatable by use of the FOR FETCH ONLY clause. In such a case, the use of positioned UPDATE or DELETE statements associated with the cursor is not allowed. Furthermore rows will not be locked once they are established regardless of the default locking specification.

Alternatively, a static cursor can be declared as FOR UPDATE, as long as it is updatable of course. In such a case, rows will be locked regardless of the default locking specification. In general, this clause need not be specified. However, if the associated UPDATE or DELETE statement is actually in a separate compilation unit, as is possible with Adabas SQL Server, then this clause is required in order to avoid a runtime error.

If neither a FOR FETCH ONLY clause nor a FOR UPDATE clause is specified and there are no associated UPDATE or DELETE statements within the same compilation unit, then the resulting rows will or will not be locked according to the system default locking specification.

Similar behavior can be ensured for a dynamic cursor by appending the clause to the dynamic SELECT statement. A column specification list is optional and indeed has no effect.

The Dynamic DECLARE CURSOR Statement

A dynamic DECLARE CURSOR statement associates a dynamically created and prepared SELECT statement with a cursor identifier. The prepared SELECT statement can be identified either by a hard-coded SQL identifier or by a host variable containing the unique statement identification provided by the relevant PREPARE statement. Note, the use of such a host variable is not supported in DB2 compatibility mode.

As an Adabas SQL Server extension the statement can also be identified by a persistent procedure specification in which case the PREPARE statement may have occurred within the same or a different session.

The dynamic DECLARE CURSOR statement thus associates this previously prepared SELECT statement with a cursor identifier. The cursor can be identified in the normal way or by a host variable, which Adabas SQL Server fills with a unique cursor identifier. Note, the use of such a host variable is not supported in DB2 compatibility mode.

Limitations:

The syntax elements host variable identifier 1, host variable identifier 2 and statement identifier are not valid within a static DECLARE CURSOR statement.

Within a dynamic DECLARE CURSOR statement such host variables must be of data type character-string.

Any ORDER BY clause, FOR UPDATE clause is part of the prepared SELECT and the use of these clauses is not valid within a dynamic cursor statement, but only in a static DECLARE CURSOR statement.

ANSI Specifics:

The use of the FOR UPDATE and FOR FETCH ONLY clauses as well as the use of the persistent procedure specification is not permitted.

The DECLARE CURSOR statement must precede any other associated statement in the source. All associated statements must be contained within one compilation unit.

Adabas SQL Server Specifics:

The physical order of the associated statements within a compilation unit is irrelevant. The OPEN statement must be present in the same compilation unit as the DECLARE statement although its relative position is irrelevant. Associated UPDATE, DELETE, FETCH and CLOSE statements need not be in the same compilation unit. However, such a program design is more error prone as full compilation checks cannot be performed.

The physical position of any associated PREPARE statement relative to the dynamic DECLARE CURSOR statement is irrelevant.

The function of a dynamic DECLARE CURSOR statement can also be accomplished by an extended OPEN statement. This saves one request to Adabas SQL Server, since a dynamic DECLARE CURSOR statement is an executable statement.

Example:

The following syntax applies when declaring a cursor to find all the cruise and start dates for every cruise that leaves BARBADOS.

```
DECLARE cursor1 CURSOR FOR
     SELECT cruise_id,start_date FROM cruise
          WHERE start_harbor = 'BARBADOS';
```

To declare a cursor to list all the start harbor's in ASCENDING alphabetical order and each related cruise id, for each cruise that costs less than 1000 the following syntax applies:

```
DECLARE cursor1 CURSOR FOR
     SELECT cruise_id,start_harbor FROM cruise
          WHERE cruise_price < 1000
          ORDER BY 2 ASCENDING;
```

To ensure that the cursor as declared in the first example can only be used for retrieval the following syntax applies:

```
DECLARE cursor1 CURSOR FOR
     SELECT cruise_id,start_date FROM cruise
      WHERE start_harbor = 'BARBADOS'
      FOR FETCH ONLY;
```

The following statement will declare a cursor for a statement that has been stored in the catalog using a persistent procedure specification:

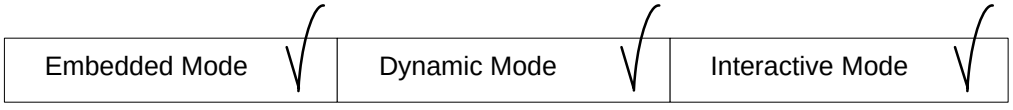
```
DECLARE cursor1 CURSOR FOR
     MODULE :mod PROCEDURE :proc Version :vers;
```

DELETE

Function:

The DELETE statement removes a particular row or set of rows from the target table. There are two forms of the statement, namely positioned DELETE and searched DELETE.

Invocation:



Syntax:

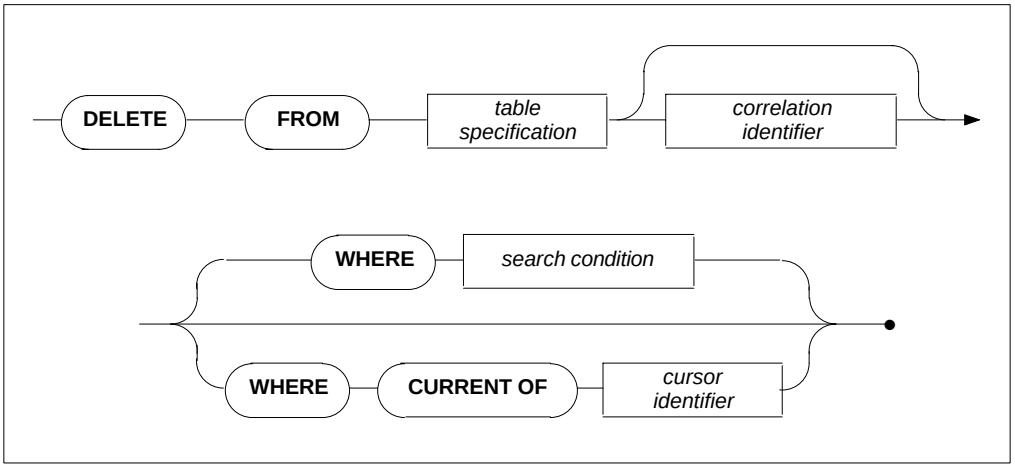


table specification is the table to be amended. The table must be defined at compilation time. If the table specification is a view reference, then the view must be updatable. See chapter **Common Elements**, section **Table Specification** for more details.

correlation identifier allows the table to be referenced by another SQL identifier. See chapter **Common Elements**, section **Correlation Identifier** for more details.

search condition	is the specification of a resultant table which is to be deleted from the target base table.
cursor identifier	is a valid identifier of no more than 18 characters and which has not previously been used as a cursor identifier within the same compilation unit.

Description:

A DELETE statement removes from the target table the row or rows identified in the WHERE clause.

Rows in Level 1 or level 2 tables can not be deleted directly using a DELETE statement. They can only be removed by deleting the associated level 0 row in the master table. The referencing level 1 and level 2 rows are automatically deleted with the level 0 row. This is analogous to a DELETE CASCADE in pure referential integrity terminology.

A DELETE statement with a WHERE CURRENT OF cursor identifier as its means of identifying the row to be deleted is called a positioned DELETE statement.

If the DELETE statement is positioned, then only the row to which the cursor is currently pointing is deleted. Hence, the cursor must be OPEN and pointing to a row otherwise a runtime error will occur. In addition, the cursor must be in itself updatable. See section **DECLARE CURSOR** for further details. Once the row has been deleted, the cursor is not advanced, it simply no longer points to a row.

A DELETE statement with a WHERE search condition is called a searched DELETE statement. If the DELETE is searched, a resultant table is established at execution time in a similar manor to a query specification. Each row in the target table which has a corresponding row in the resultant table is, then deleted. If no rows are identified for deletion, the field SQLCODE in the SQLCA is set to +100.

A DELETE statement without any WHERE clause is really a special case of the searched DELETE alternative as a resultant table is established which contains all the rows of the target table. In such a case, all rows of the table are deleted.

Limitations:

If the specified table is in fact a view, then that view must be updatable. See section **DECLARE CURSOR** for more details.

In a positioned DELETE CURSOR statement, the table reference must be identical to that referenced in the associated DECLARE CURSOR statement.

In addition, if the associated DECLARE CURSOR statement was defined in another compilation unit, then it must have been specified with the FOR UPDATE clause.

Also the associated cursor must be updatable, open and positioned on a row of the resultant table.

A positioned DELETE statement is not allowed in interactive SQL.

In a searched DELETE statement, any view referenced must be updatable.

The DELETE statement may not be applied to subtables directly but must always address the related master table (cascaded DELETE).

ANSI Specifics:

A positioned DELETE statement must appear in the same compilation unit as the associated DECLARE and OPEN and must appear physically after the DECLARE.

The use of correlation identifiers in this context is not supported in ANSI compatibility mode.

Adabas SQL Server Specifics:

A positioned DELETE statement can be in a different compilation unit to that of the associated DECLARE as long as a FOR UPDATE clause is specified. If the DELETE is in the same compilation unit as the associated DECLARE CURSOR statement, then there is no restriction as to the relative positions of the two statements.

The possibility to use a correlation identifier is an Adabas SQL Server extension.

DML statements must not be mixed with DDL/DCL statements in the same transaction. For details see the section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To delete all cruises that depart from VIRGIN ISLANDS, the following syntax is required.

```
DELETE FROM cruise
     WHERE start_harbor = 'VIRGIN ISLANDS';
```

To delete ALL information contained within table ‘cruise’, the following syntax is required.

```
DELETE FROM cruise;
```

To delete the row in table cruise to which a cursor named 'cursor1' is currently pointing, the following syntax is used.

```
DELETE FROM cruise
     WHERE CURRENT OF cursor1;
```

DESCRIBE

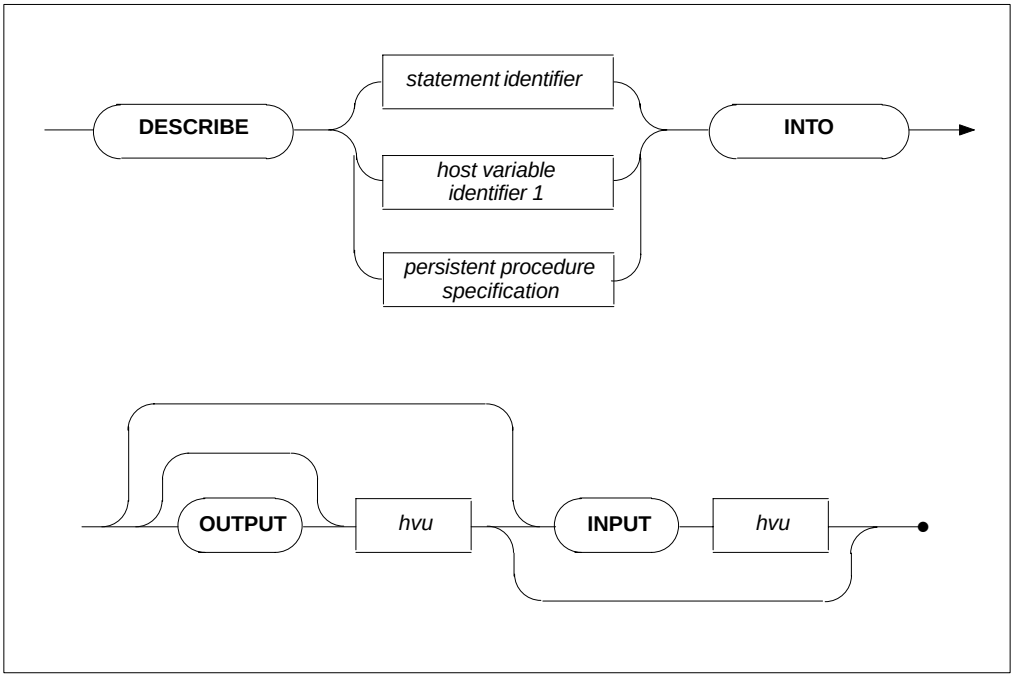
Function:

The DESCRIBE statement makes information about a prepared statement available to the application program.

Invocation:

Embedded Mode √	Dynamic Mode	Interactive Mode

Syntax:



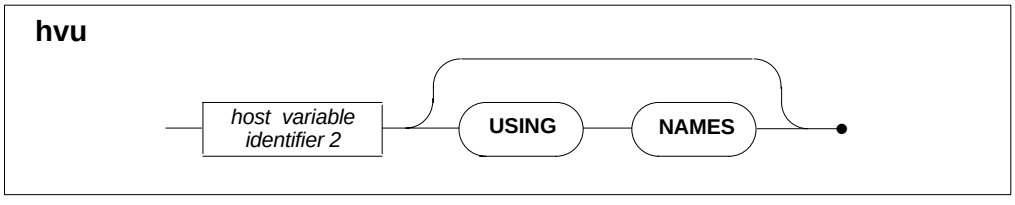
statement identifier — is a valid identifier denoting the prepared statement of which the information is to be retrieved.

host variable identifier — is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules. The value of the host variable must be the value returned by the PREPARE statement and thus identifying the prepared statement.

persistent procedure specification — specifies the identification of the prepared statement that has been stored in the catalog. The persistent procedure specification must include both the VERSION and PROCEDURE clause.

OUTPUT *hvu* — is the definition of the SQL descriptor area used to describe the expected output of the identified statement.

INPUT *hvu* — is the definition of the SQL descriptor area used to describe the expected input of the identified statement.



host variable identifier — is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules. The value of the host variable must be the address of an SQL descriptor area (SQLDA).

Description:

The DESCRIBE statement places information about the prepared statement identified by statement identifier or host variable identifier in one or two SQL descriptor areas.

The keyword OUTPUT is relevant only if the prepared statement is a SELECT statement. In this case, the SQL descriptor area indicated by host variable identifier 2 is filled with information concerning the elements in the derived column list of the SELECT statement. For each element in the derived column list, an element in the SQL descriptor area is filled. The elements in the derived column list are processed from left to right and the descriptive elements in the SQL descriptor area are filled in the that order.

The keyword INPUT is relevant only if the prepared statement contains host variable markers, i.e. '?'. In this case, the SQL descriptor area indicated by host variable identifier 2 is filled with information concerning the host variable markers used in the prepared statement. For each host variable marker, an element in the SQL descriptor area is filled. The host variable markers are processed in the order that they appear in the prepared statements. The descriptive elements in the SQL descriptor area are filled in that order.

If the prepared statement is a SELECT statement where host variable markers have been used, the usage of not only the OUTPUT clause but also the INPUT clause is recommended.

The USING NAMES option is currently not effective.

Limitations:

The statement indicated by statement identifier or host variable identifier 1 must be a successfully prepared statement.

If not enough elements have been provided in the SQL descriptor area to cater for the total number of elements that need to be described, all the elements that can be catered for are described, the rest of the information is ignored. The actual number of elements required is returned in field SQLN in the SQLDA.

ANSI Specifics:

The DESCRIBE statement is not part of the Standard.

Adabas SQL Server Specifics:

This statement may be mixed with any other DML, DDL and/or DCL statements in the same transaction.

Example:

To make available information about a statement that has been prepared, where the information required is of a dynamic statement's derived column list (i.e OUTPUT), the following syntax applies:

```
DESCRIBE statement_id INTO
     OUTPUT :sqlda_address ;
```

If the prepared statement resides in the catalog, a persistent procedure specification must be used:

```
DESCRIBE MODULE :mod PROCEDURE :proc
     INTO :sqlda_address;
```

DISCONNECT

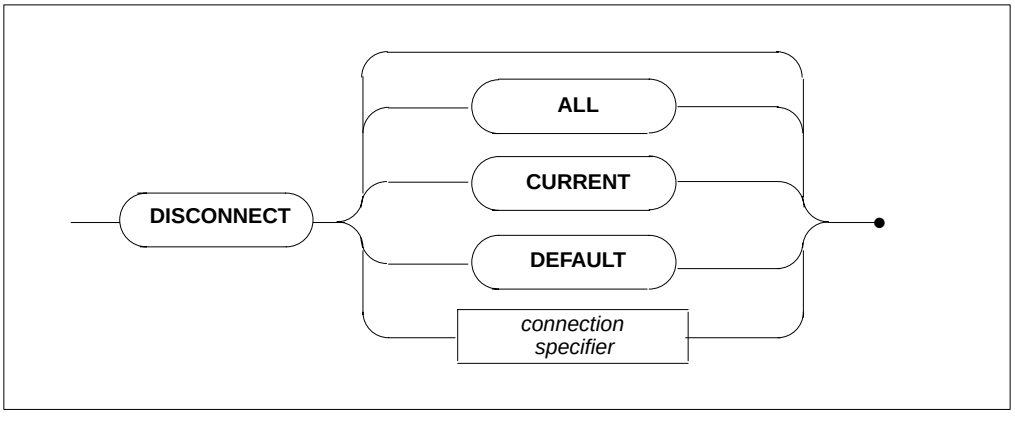
Function:

The DISCONNECT statement explicitly terminates an SQL session between a user and the Adabas SQL Server environment.

Invocation:

Embedded Mode √	Dynamic Mode	Interactive Mode

Syntax:



connection specifier can either be a character-string constant or single host variable identifier. The host variable must have been defined in the application program according to the host-language-dependent rules and its value must be a character string. The maximum length is 32 characters.

Description:

The DISCONNECT statement terminates an SQL session between an application program and Adabas SQL Server. The DISCONNECT statement performs an implicit ROLLBACK.

DISCONNECT/ DISCONNECT CURRENT:	are logically equal and terminate the current SQL session. The Version 1.2 syntax of the DISCONNECT statement is still supported and is represented in Version's 1.3 and higher as the DISCONNECT CURRENT statement.
DISCONNECT ALL:	terminates all SQL sessions A DISCONNECT ALL statement is performed automatically by the exit handler of Adabas SQL Server when terminating an application.
DISCONNECT DEFAULT:	terminates the SQL session with the default server
DISCONNECT *connection specifier*:	terminates the SQL session with the server specified by the connection identifier.

Limitations:

None.

ANSI Specifics:

The DISCONNECT statement is not part of the ANSI standard.

Adabas SQL Server Specifics:

The DISCONNECT statement is an Adabas SQL Server extension.

The Version 1.2 syntax of the DISCONNECT statement is still supported and is logically equal to Version's 1.3 counterpart: DISCONNECT CURRENT.

Example:

To disconnect from the session identified by the connection specifier MYSESSION the following syntax applies:

```
DISCONNECT :MYSESSION;
```

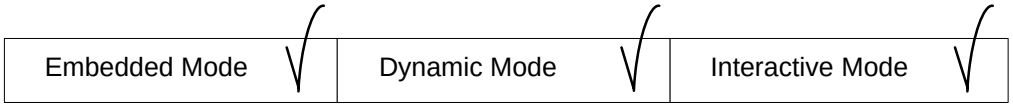
DROP CLUSTER

Function:

This statement deletes the logical and physical representation of a cluster.

Note:
Dropping a cluster causes all the data contained within to be destroyed. Once the statement has been executed, there is no way of recovering the data.

Invocation:



Syntax:

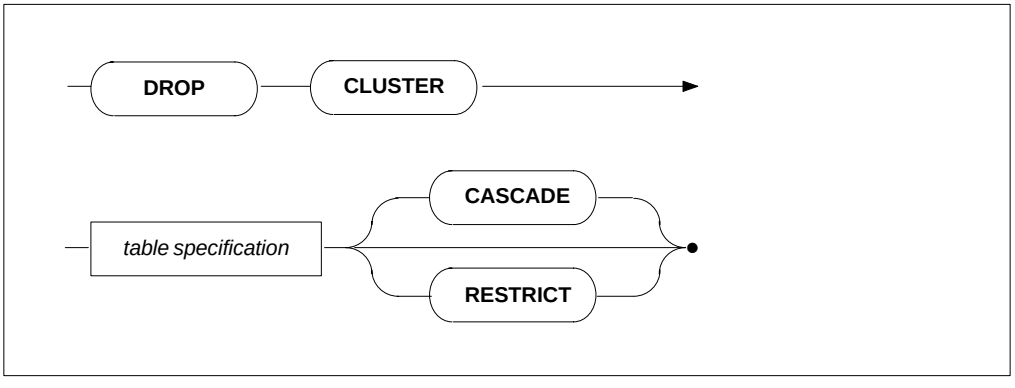


table specification — schema identifier and table identifier of the master table in the cluster to be dropped. The default schema name is assumed if not specified here.

Description:

A cluster and all associated information will be deleted from Adabas SQL Server's catalog and the underlying Adabas file will be deleted. Any other statements referencing this table will no longer be valid. In addition, any attempts to compile statements which reference this table will fail. Even if the table is re-specified, all previously compiled statements remain invalid.

If the CASCADE option is specified, all view descriptions based on the cluster to be dropped will be deleted as well.

If the RESTRICT option is specified, the statement execution will be rejected if there are dependent views.

If neither of these two options is specified, RESTRICT is assumed.

Limitations:

The specified table specification must denote an existing cluster at runtime.

ANSI Specifics:

The DROP CLUSTER statement is not part of the ANSI standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To delete all data and data structures (except the tablespace) of the cluster PRESIDENTS and all related views, the following syntax applies:

```
DROP CLUSTER presidents CASCADE;
```

DROP CLUSTER DESCRIPTION

Function:

This statement deletes the logical representation of a cluster but not the underlying Adabas file.

Note:
Dropping a cluster description does not cause all data contained within to be destroyed. Only the cluster description in the catalog is deleted.

Invocation:

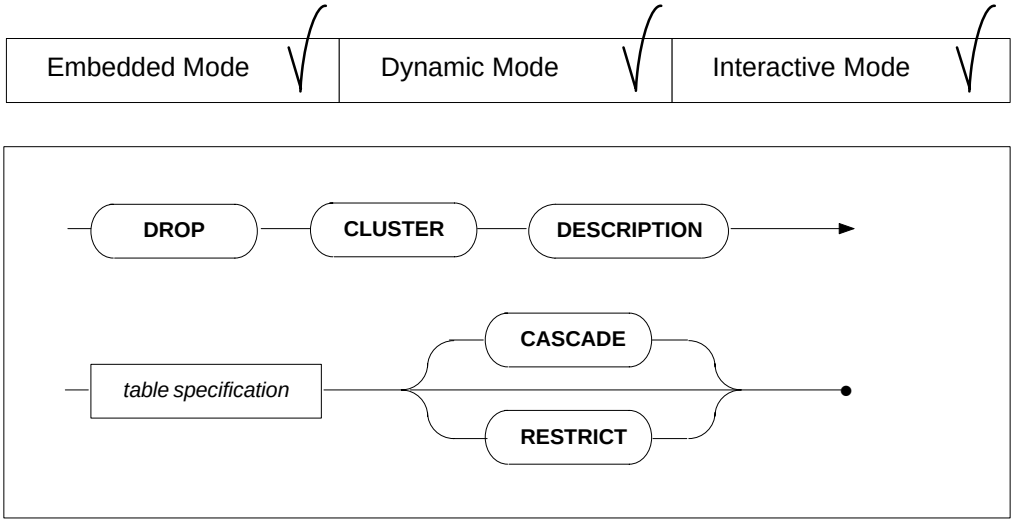


table specification — schema identifier and table identifier of the master table in the cluster description to be dropped. The default schema name is assumed if not specified here.

Description:

A cluster definition and all its associated information will be deleted from Adabas SQL Server's catalog. Any other statements referencing this cluster will no longer be valid. In addition, any attempts to compile statements which reference this cluster will fail. Even if the cluster description is re-specified, all previously compiled statements remain invalid. The tablespace with the same name remains unchanged and must be dropped separately if desired.

If the CASCADE option is specified, all dependent view descriptions based on the cluster to be dropped, will be deleted as well.

If the RESTRICT option is specified, the statement execution will be rejected if there are dependent views.

If neither of these two option is specified, RESTRICT is assumed.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

Limitations:

The specified table specification must denote an existing cluster at runtime.

ANSI Specifics:

The DROP TABLE DESCRIPTION statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To delete all cluster descriptions (except the tablespace) of the cluster PRESIDENTS and all related views, the following syntax applies:

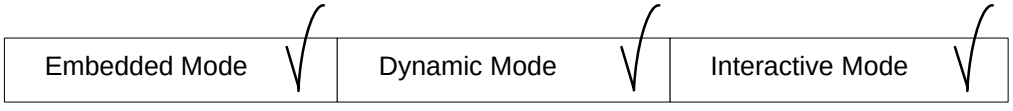
```
DROP CLUSTER DESCRIPTION presidents CASCADE;
```

DROP DATABASE

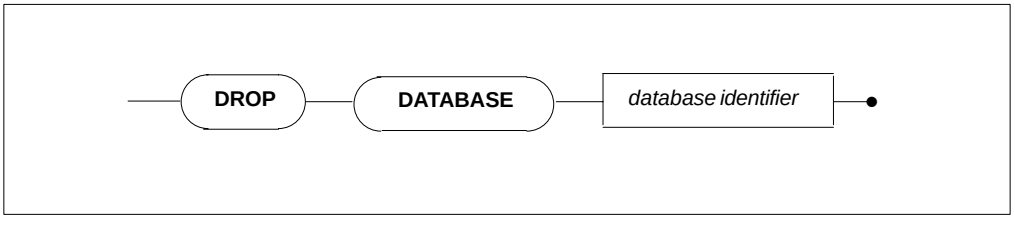
Function:

The statement deletes an existing logical database identifier but does not destroy the underlying data.

Invocation:



Syntax:



database identifier — a valid database identifier representing a logical database name which must be known to Adabas SQL Server at runtime.

Description:

The logical name of the specified database is removed from the catalog. Even though the database still exists, it can not be accessed by use of this name again.

The DROP DATABASE statement will fail if there are any objects remaining in the catalog which are dependent upon the database identifier to be deleted.

If the statement is invoked statically, then during pre-compilation, the database need not exist in the catalog. For successful execution, however, the database must exist in the catalog, regardless of how it is invoked.

Limitations:

The specified database name must exist.

ANSI Specifics:

The DROP DATABASE statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The following syntax applies when dropping a database named 'yachting':

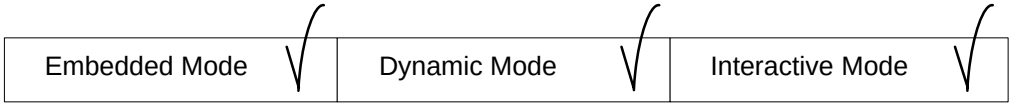
```
DROP DATABASE yachting ;
```

DROP INDEX

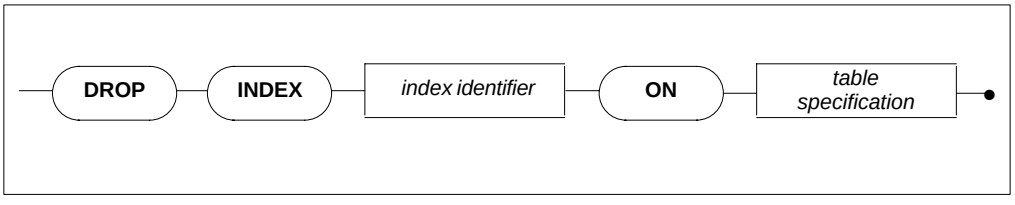
Function:

This statement removes an index from a base table.

Invocation:



Syntax:



index identifier	identifies the index to be dropped.
table specification	identifies the table from which an index is to be removed.

Description:

The specified index is removed from the specified base table. It does not matter whether the index was created during the table creation or later by a CREATE INDEX statement.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

Limitations:

The underlying Adabas database system has to run on the same machine as Adabas SQL Server and can not be accessed using Entire Net-Work.

ANSI Specifics:

This statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To drop an index named *iname* on the column *cruise_id* of our sample base table (cruise) the following syntax is used:

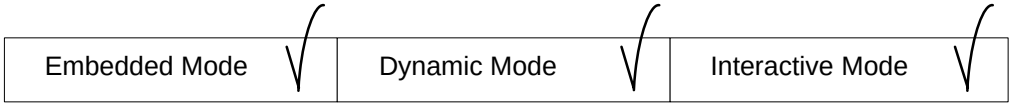
```
DROP INDEX iname ON cruise;
```

DROP SCHEMA

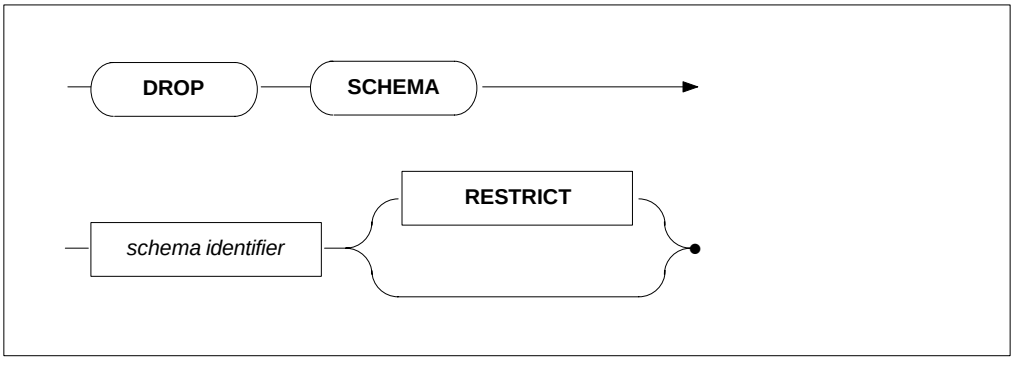
Function:

The DROP SCHEMA statement removes the schema from the catalog.

Invocation:



Syntax:



schema identifier a valid schema identifier representing the schema to be dropped.

Description

An SQL schema entry will be deleted from Adabas SQL Server's catalog.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

Limitations

The schema must be empty prior to deletion. Attempts to delete a non-empty schema will fail and a corresponding error message will be issued.

Only the designated DBA and not even the owner is permitted to drop a schema.

ANSI Specifics

ANSI requires that the statement is qualified with either of the keywords CASCADE or RESTRICT. The CASCADE functionality is not supported in Adabas SQL Server. However, when in ANSI mode, the presence of the keyword RESTRICT is mandatory .

Adabas SQL Server Specifics

The inclusion of the keyword RESTRICT is optional. The functionality is the same in both cases.

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example

In order to drop the schema 'Wiltshire' the following syntax applies:

```
DROP SCHEMA Wiltshire ;
```

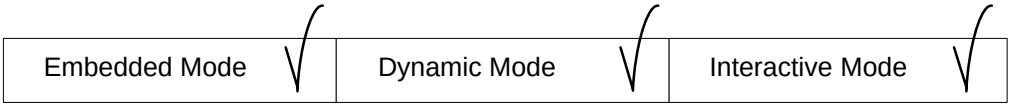
DROP TABLE

Function:

The statement removes a base table, all dependent views and all data.

Note:
Dropping a table causes all the data contained within to be destroyed. Once the statement has been executed, there is no way of recovering the data.

Invocation:



Syntax:

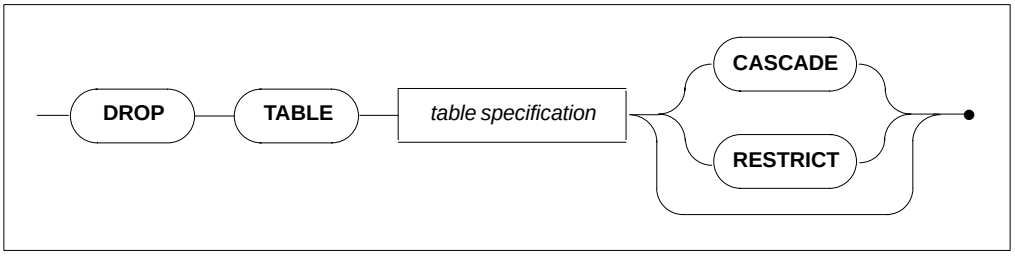


table specification — the expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified.

Description:

A table and all associated information will be deleted from Adabas SQL Server's catalog and the underlying Adabas file will be deleted. Any other statements referencing this table will no longer be valid. In addition, any attempts to compile statements which reference this table will fail. Even if the table is re-specified, all previously compiled statements remain invalid.

If the CASCADE option is specified, all view descriptions based on the table to be dropped, will be deleted. Statement execution will be rejected if attempts are made to drop a table with dependent views but without the CASCADE option.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must be existent in the catalog, regardless of how it is invoked.

Limitations:

The specified table specification must denote an existing table at runtime.

If a table has been created as a part of a cluster, then it can not be dropped individually. The cluster must be dropped in order to remove this table.

The underlying Adabas database system has to run on the same machine as Adabas SQL Server and can not be accessed using Net-Work.

ANSI Specifics:

ANSI requires that the statement is qualified with either of the keywords CASCADE or RESTRICT. The CASCADE functionality is not supported in Adabas SQL Server. However, when in ANSI mode, the presence of the keyword RESTRICT is mandatory .

.**Adabas SQL Server Specifics:**

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The following syntax applies when dropping the table 'cruise' and all dependent views.

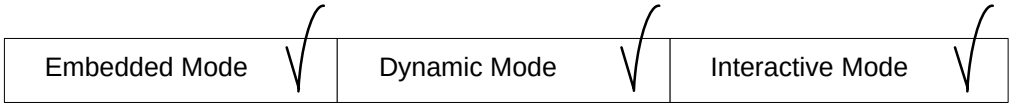
```
DROP TABLE cruise CASCADE;
```

DROP DEFAULT TABLESPACE

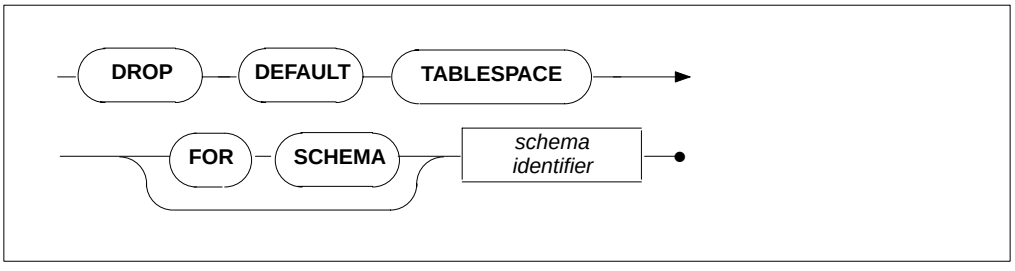
Function:

The DROP DEFAULT TABLESPACE statement removes the default tablespace definitions from the catalog but not the underlying Adabas file.

Invocation:



Syntax:



schema identifier a valid identifier describing an existing schema

Description:

The default settings as defined by the CREATE DEFAULT TABLESPACE statement will be removed from the catalog. The default settings will be removed regardless of any tables already created.

Limitations:

A previously defined tablespace must be present for the referred schema.

ANSI Specifics:

The DROP DEFAULT TABLESPACE statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The following syntax applies when dropping the default tablespace for the schema SAGTOURS.

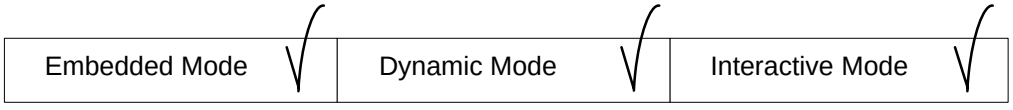
```
DROP DEFAULT TABLESPACE FOR SCHEMA sagtours;
```

DROP TABLESPACE

Function:

The statement removes the Adabas-specific file definitions from the catalog.

Invocation:



Syntax:

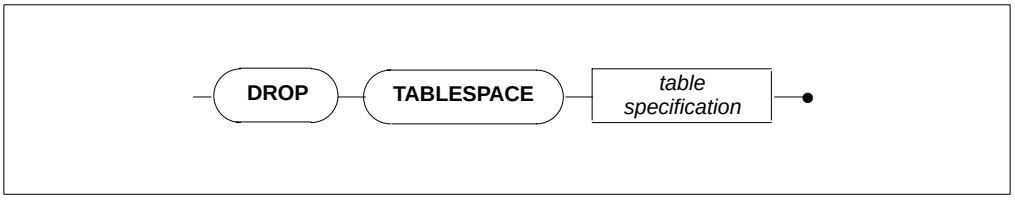


table specification — the expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified.

Description:

The Adabas-specific file definitions of an SQL table and all associated information are deleted from Adabas SQL Server. A tablespace may be dropped regardless of the related table specification.

If the statement is invoked statically, then during pre-compilation, the tablespace need not exist in the catalog. For successful execution, however, the tablespace must exist in the catalog, regardless of how it is invoked.

Limitations:

The specified table specification must denote an existing tablespace at runtime.

ANSI Specifics:

The DROP TABLESPACE statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The following syntax applies when dropping the Adabas-specific file definitions for the table 'cruise'.

```
DROP TABLESPACE cruise;
```

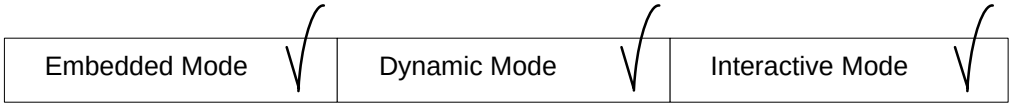
DROP TABLE DESCRIPTION

Function:

The statement removes a base table description in the catalog but does not destroy the data.

Note:
Dropping a table description does not cause all data contained within to be destroyed. Only the table description in the catalog is deleted.

Invocation:



Syntax:

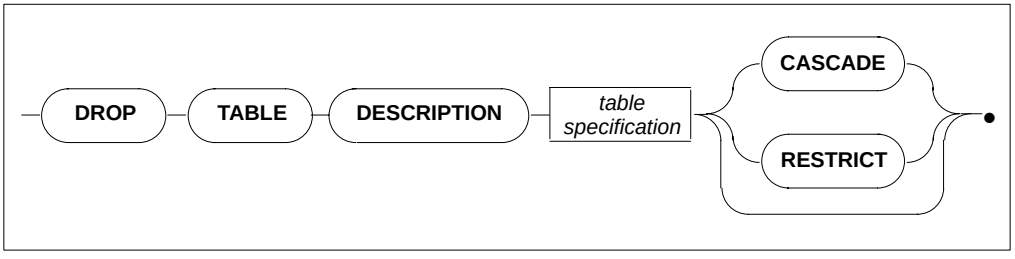


table specification — the expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified.

Description:

A table description and all its associated information will be deleted from Adabas SQL Server's catalog. Any other statements referencing this table will no longer be valid. In addition, any attempts to compile statements which reference this table will fail. Even if the table is re-specified, all previously compiled statements remain invalid.

If the CASCADE option is specified, all dependent view descriptions based on the table to be dropped, will also be deleted. Statement execution will fail when attempting to drop a table description with dependent view descriptions but without the CASCADE option.

If the statement is invoked statically, then during pre-compilation, the table description need not exist in the catalog. For successful execution, however, the table description must exist in the catalog, regardless of how it is invoked.

Limitations:

The specified table specification must denote an existing table at runtime.

If a table description has been created as a part of a cluster, then it can not be dropped individually. The cluster must be dropped in order to remove this table description.

ANSI Specifics:

The DROP TABLE DESCRIPTION statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The following syntax applies when dropping the description of the table 'cruise' and all dependent view descriptions.

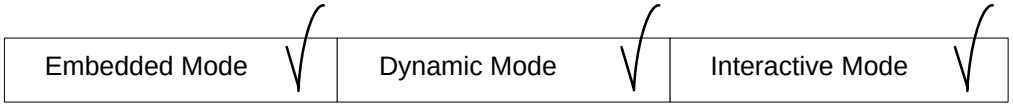
```
DROP TABLE DESCRIPTION cruise CASCADE;
```

DROP USER

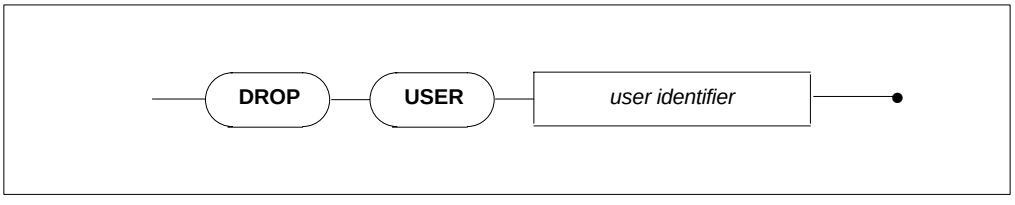
Function:

The DROP USER statement eliminates the user from Adabas SQL Server.

Invocation:



Syntax:



user identifier: an existing user identifier.

Description:

The DROP USER statement removes an existing user identifier and the associated password.

Limitations:

Before issuing this statement, the DBA will have to make sure that the user to be removed does not own any objects in the catalog. Otherwise, an error message will be issued. The statement may only be executed by the DBA.

ANSI Specifics:

The DROP USER statement is not part of the Standard

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The user PETER, who does not own any objects in the catalog, is to be removed from the system. The following syntax applies:

```
DROP USER PETER;
```

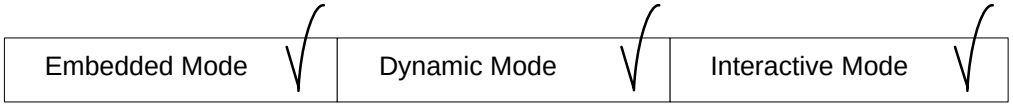
DROP VIEW

Function:

The statement deletes a view.

Note:
Dropping a view does not destroy any underlying data as a view is a logical table and not a physical or base table.

Invocation:



Syntax:

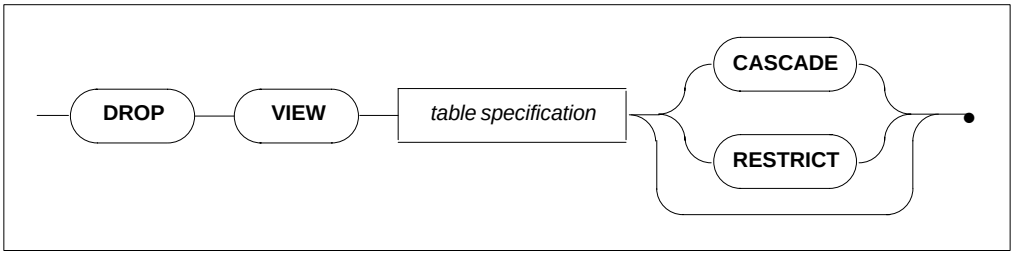


table specification — the expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified.

Description:

A view and its description in Adabas SQL Server is deleted. Any other statements referencing this view will no longer be valid. In addition, any attempts to compile statements which reference this view will fail. Even if the view is re-specified, all previously compiled statements remain invalid.

If the CASCADE option is specified, all views based on the view to be dropped, will be deleted. Statement execution will fail when attempting to drop a view description with dependent views without the CASCADE option.

If the statement is invoked statically, then during pre-compilation, the view need not exist in the catalog. For successful execution, however, the view must exist in the catalog, regardless of how it is invoked.

Limitations:

The specified table specification must denote an existing view, at runtime.

ANSI Specifics:

The DROP VIEW statement is not part of the Standard.

Adabas SQL Server Specifics:

The following transaction limitations apply to this statement:

- On successful execution of this statement, an implicit COMMIT will take place.
- DDL and DCL statements may not be mixed with DML statements within the same transaction.

For more details on transaction limitations see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The following syntax applies when dropping the view 'Canada' with all its related views.

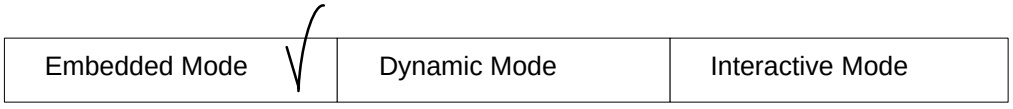
```
DROP VIEW Canada CASCADE;
```

END DECLARE SECTION

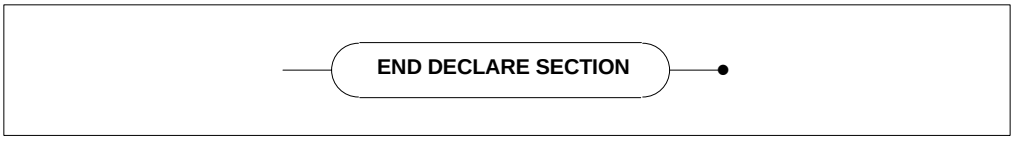
Function:

This statement is the terminating delimiter for a host variable declaration section.

Invocation:



Syntax:



Description:

This statement terminates a host variable declaration section. Please refer to the BEGIN DECLARE SECTION statement earlier in this chapter for more details.

Limitations:

Please refer to the BEGIN DECLARE SECTION statement earlier in this chapter for more details.

ANSI Specifics:

Please refer to the BEGIN DECLARE SECTION statement earlier in this chapter for more details.

Adabas SQL Server Specifics:

Please refer to the BEGIN DECLARE SECTION statement earlier in this chapter for more details.

Example:

To specify the end of the host variable declaration section, the following syntax applies:

```
BEGIN DECLARE SECTION
    char a [5]
END DECLARE SECTION;
```

EXECUTE

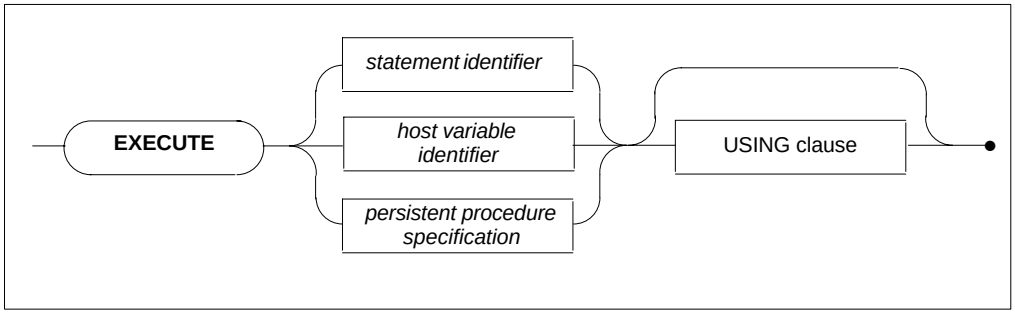
Function:

The EXECUTE statement executes a prepared statement.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√		

Syntax:



statement identifier	is a valid identifier denoting the name of the prepared statement which is to be executed.
host variable identifier	is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules. The value of the host variable must be the value returned by the PREPARE statement and thus identifying the prepared statement.
persistent procedure specification	specifies the identification of the prepared statement that has been stored in the catalog. The persistent procedure specification must include both the VERSION and PROCEDURE clause.
USING clause	defines an SQL descriptor area used to provide dynamic input variables if required by the statement to be executed.

Description:

The EXECUTE statement executes the prepared statement identified by a statement identifier or host variable identifier. If the prepared statement contains host variable markers, then values must be provided to satisfy these. In this case, a USING clause is required.

Limitations:

The statement indicated by statement identifier, host variable identifier or persistent procedure specification must be a successfully prepared statement.

A previously prepared SELECT statement cannot be submitted to the EXECUTE statement.

ANSI Specifics:

The EXECUTE statement is not part of the Standard.

Adabas SQL Server Specifics:

A host variable identifier or persistent procedure specification can be used to identify the prepared statement.

This statement may be mixed with any other DML, DDL and/or DCL statements in the same transaction.

Example:

The following syntax is required to execute an already prepared statement (with a statement id of ‘statement_id’).

```
EXECUTE statement_id ;
```

To execute a prepared statement that requires 3 values, then the following syntax applies:

```
EXECUTE statement_id USING :hv1, :hv2, :hv3 ;
```

To execute a prepared statement where the input information is stored in the SQLDA, the following syntax applies:

```
EXECUTE statement_id USING DESCRIPTOR :input_sqlda;
```

To execute a prepared statement that has been stored in the catalog the following syntax applies:

```
EXECUTE MODULE :mod PROCEDURE :proc VERSION :vers
     USING DESCRIPTOR :input_sqlda;
```

EXECUTE IMMEDIATE

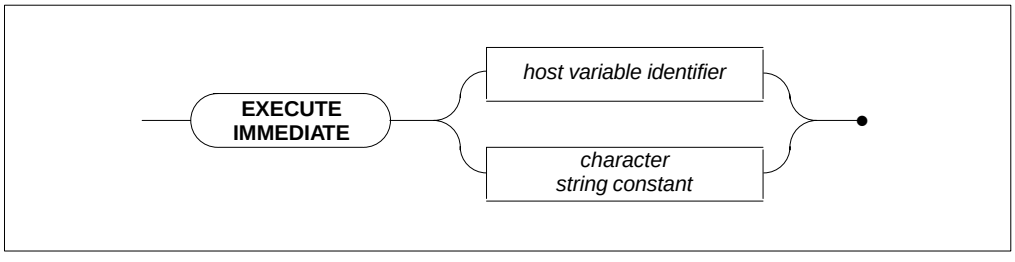
Function:

The EXECUTE IMMEDIATE statement prepares and executes an SQL statement for immediate execution. After execution, the prepared statement is deleted.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√		

Syntax:



host variable identifier is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules. The data type of the host variable must be a character-String.

character-string constant is a valid character-string constant.

Description:

The EXECUTE IMMEDIATE statement performs the following actions:

1 COMPILATION:
The SQL statement in a character-string representation is compiled into a prepared SQL statement. If an error is encountered by the SQL compiler which prevents the SQL statement being compiled successfully, an error is passed back to the application program in the SQLCODE field of the SQLCA. In this case, no prepared statement is created and the execution phase is not entered.

2 EXECUTION:
The prepared SQL statement is executed. If an error is encountered during the execution of the prepared statement, the error is passed back to the application program in the SQLCODE field of the SQLCA.

3 DELETION:
The prepared SQL statement is deleted. The prepared statement is deleted after execution. This means, that even if the exact same statement will have to be executed twice with two separate EXECUTE IMMEDIATE statements within the same transaction, it will have to be compiled twice.

Limitations:

The character-string must contain one of the following statements:

COMMIT, CREATE, DELETE, DROP, INSERT, ROLLBACK, or UPDATE.

Host variable markers or references are not permitted in the statement.

ANSI Specifics:

The EXECUTE IMMEDIATE statement is not part of the Standard.

Adabas SQL Server Specifics:

This statement may be mixed with any other DML, DDL and/or DCL statements in the same transaction.

Example:

To immediately execute the statement ‘DELETE FROM cruise’ the following syntax applies:

```
EXECUTE IMMEDIATE 'DELETE FROM cruise' ;
```

FETCH

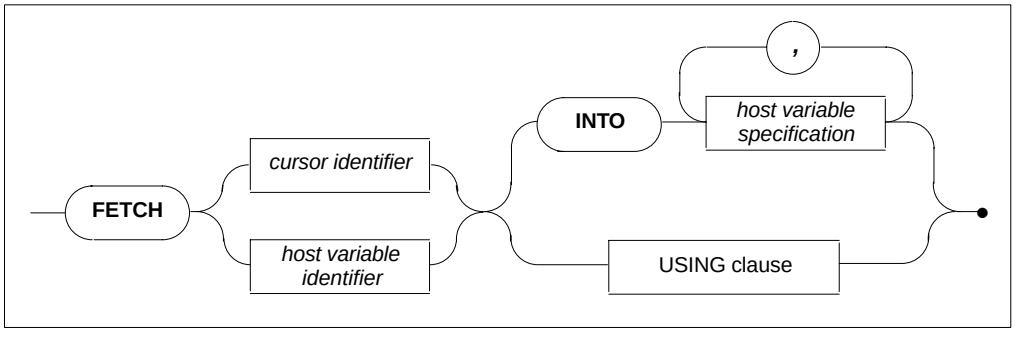
Function:

The FETCH statement positions the cursor on a row within the resultant table and makes the values of that row available to the application program.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√		

Syntax:



cursor identifier	identifies the cursor to be used in the FETCH operation.
host variable identifier	is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules. The value of the host variable must be a valid cursor identifier. A host variable can be used as cursor identifier only if the cursor is a dynamically declared cursor.
host variable specification	is a valid host variable specification and must reference a structure and must have been defined in the application program according to the host-language-dependent rules.
USING clause	defines an SQL descriptor area used to receive data from the associated dynamic cursor.

Description:

The FETCH statement performs two functions: it moves the cursor in the resultant table from top to bottom, one row at a time, and makes the relevant values of a row available to the application program according to the specification of the INTO clause or the USING clause. The mechanism used when the USING clause has been specified, is described in chapter **Common Elements,** section **USING Clause.**

The FETCH statement changes the position of the cursor as follows:

- If the cursor is positioned before the first row of the resultant table (as would be the case if the cursor had just been opened), it is moved to the first row.
- If the cursor is positioned on a row of the resultant table, it is moved to the next one.
- If the cursor is positioned on the last row of an resultant table, it is moved past the last row and the SQLCODE field in the SQLCA is set to +100.
- If the row on which the cursor is positioned is deleted, the cursor is, then positioned in front of the next row in the table.

A host variable specification which references a host variable structure is equivalent to individual host variable specifications which reference all the elements of a structure singularly.

Each *host variable* corresponds to a resultant column of the resultant table of the cursor in question, where the first *host variable* is passed with the first column and so on.

Each value of a resultant column is assigned to the corresponding host variable. The assignment operation follows the normal conversion rules as described in chapter **Common Elements,** section **Expressions**.

Limitations:

The cursor must have been prepared and opened prior to the execution of the FETCH statement.

The data type of a host variable must be compatible with the data type of its corresponding resultant column. If the data type is not compatible, an error occurs. The value of the unassigned host variable is unpredictable.

If the number of resultant columns is smaller than the number of host variables, as many host variables as possible are assigned the values of their corresponding resultant columns. The remaining host variables are left untouched.

If the number of resultant columns is greater than the number of host variables, an error message (warning) is generated.

A USING clause may only be used in association with a dynamic cursor.

ANSI Specifics:

An INTO clause is mandatory, the USING clause must not be used. Only single host variable specifications are permitted.

Adabas SQL Server Specifics:

A FETCH statement without an INTO or USING clause is legal. The effect of this FETCH statement is that the cursor is moved, but no data is exchanged. This enables the cursor to progress in conjunction with an UPDATE or DELETE statement without actually retrieving any data.

The OPEN statement and the FETCH statement can be in different compilation units (see also the section: **DECLARE CURSOR**).

For details about MULTIFETCH, refer to the *Adabas SQL Server Programmer's Guide*, chapter **The MULTIFETCH Feature**.

Example:

To fetch data from a cursor (identified as cursor_id) and place the data into 3 host variable the following syntax is applied.

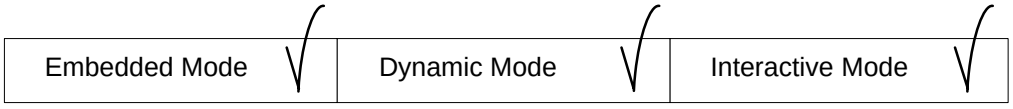
```
FETCH cursor_id
    INTO :host_var1,
         :host_var2,
         :host_var3 ;
```

GRANT

Function:

The GRANT statement gives users privileges to access tables or views.

Invocation:



Syntax:

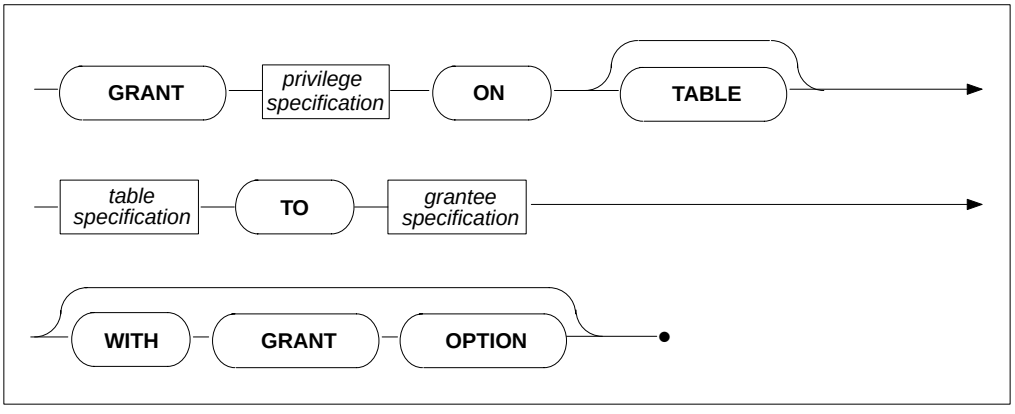


table specification	is an already existing table or view for which the grant is to be performed. The table or view name should only be specified once.
privilege specification	is a list of one or more privileges that are to be granted.
grantee specification	is a user, a list of users or PUBLIC for which the grant is to be executed. A user should only be specified once.

Note:
See chapter ***Common Elements*** *for more details.*

Description:

The GRANT statement gives the specified privileges to a user, a list of users or PUBLIC for the specified table(s) or view(s). Neither the user identifier nor the table or view identifier should be specified multiple times.

By default, owners of a table have all privileges for that table and should, therefore, not grant themselves rights on that table.

For details about what privileges are possible, what they mean and the constraints on them see chapter **Common Elements**, section **Privilege Specification**.

A privilege given with the WITH GRANT OPTION permits this user to grant other users privileges on the specified table(s) or view(s). The WITH GRANT OPTION can be specified for ALL PRIVILEGES, and so enables the grantee to grant all privileges to another user; or it can be specified for a particular set of privileges (see examples below).

Unsuccessful execution of the GRANT ALL PRIVILEGES statement results in response code 0, even though the ANSI Standard prescribes a Warning.

Limitations:

General rules:

- Each user can have a privilege only once. If, for example, Peter received the privilege SELECT on CRUISE from Tim, no one else can grant Peter the same SELECT on CRUISE privilege.
- Privileges on views are not automatically granted, just because privileges have been granted for the underlying base tables(s). For example, Peter has created an updatable view based on the base table CRUISE. He has SELECT and UPDATE privilege on this view. The granting of INSERT on the base table CRUISE to Peter will not result in INSERT privilege on the view.
- Granting UPDATE privileges on a table always means an implicit UPDATE on all columns of the table on which the grantor also has the GRANT option. In addition, a table privilege means that, when a column is added, all grantees that have the table privilege also receive the column privilege for the new column.

Authority to grant privileges:

- The grantor is the owner of the table or view.
- The grantor has been given the WITH GRANT OPTION by the owner of the table or view.

Granting privileges on Views:

- The creator of a view must have at least the SELECT privilege on all the base tables.
- In case of a read-only view, the SELECT privilege is the only privilege the owner has and may grant.
- The grantee must have at least a SELECT privilege, as above.
- If the above is not true, then the owner of the view must have at least the SELECT privilege plus the “WITH GRANT OPTION” to be able to grant privileges to other users for all base tables.

The execution of the GRANT statement is an atomic action that is closed by an implicit COMMIT and is, therefore, not capable of ROLLBACK.

ANSI Specifics:

The optional keyword TABLE in ON TABLE *table specification* is not supported.

Adabas SQL Server Specifics:

The keyword TABLE in ON TABLE *table specification* is optional.

Examples:

- Tim decided to GRANT ALL privileges to Peter on his table CRUISE.

```
GRANT ALL ON CRUISE TO PETER;
GRANT ALL PRIVILEGES ON CRUISE TO PETER; [ ANSI-specific grant ]
```

- Tim decided to GRANT the privilege SELECT to Anne on his table CRUISE.

```
GRANT SELECT ON CRUISE TO ANNE;
```

- Tim decided to GRANT the privileges SELECT, INSERT and DELETE to Martin with the ”WITH GRANT OPTION” on his table CRUISE. Martin then decides to GRANT the select privilege to Chris.

```
Tim : GRANT SELECT, INSERT, DELETE ON CRUISE TO MARTIN WITH GRANT OPTION;
Martin : GRANT SELECT ON CRUISE TO CHRIS;
```

- Peter decided to GRANT Roland the SELECT privilege on table CRUISE for which he has no “WITH GRANT OPTION”. Roland himself has no privileges for the table CRUISE.

```
GRANT SELECT ON CRUISE TO ROLAND;
```

This statement fails as Peter has no privileges to perform this operation.

- Peter decided to GRANT Roland the SELECT privilege on his view CRUISE_YACHTS.

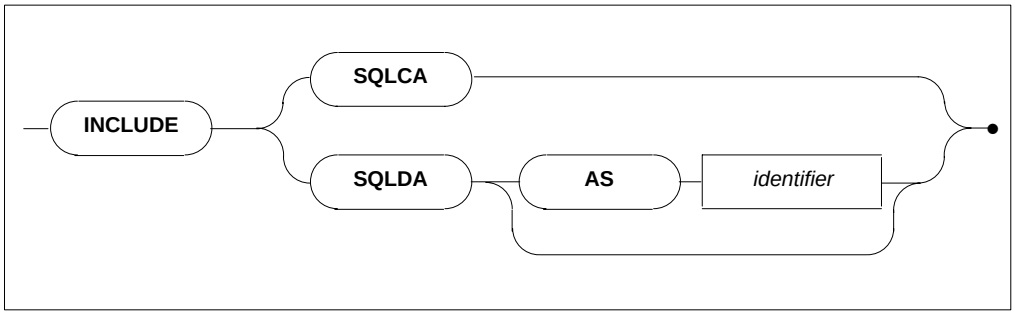
```
GRANT SELECT ON TABLE CRUISE_YACHTS TO ROLAND;
```

INCLUDE

Function:

This statement includes the data description for the SQLCA or SQLDA.

Syntax:



AS *identifier* — a host-language-specific identifier used to explicitly name the pointer variable to the SQLDA structure.

Description:

The application programs must be able to find out if an SQL statement has been successfully completed or if it failed. The respective control values are available in the host variable structure called SQLCA. Although, such a structure may be defined and declared explicitly, it is much easier to let Adabas SQL Server generate a definition and a declaration into the host program's source code. Such a generation will occur whenever the SQL statement INCLUDE SQLCA is specified. The position of this statement must conform to the rules of the declaration of host variables and the resulting structure will be represented by the identifier SQLCA. For a comprehensive explanation regarding SQLCA please refer to the *Adabas SQL Server Programmer's Guide*. The SQLCA is not updated as a result of an INCLUDE statement.

Note:
An explicit identifier can not be specified for an SQLCA structure.

Certain embedded dynamic SQL statements require the use of an SQLDA. Again, such a structure could be defined explicitly, but it is much easier to let Adabas SQL Server generate a definition into the host program's source code. Only an SQLDA structure definition is generated along with a declaration of a pointer to such a structure. The user must actually provide an appropriate structure himself. Please refer to the *Adabas SQL Server Programmer's Guide* for details. The generated pointer will, by default, be identified by SQLDA unless the AS clause is supplied in which case the given identifier is used. The use of such an identifier within an appropriate SQL statement identifies this instance of the SQLDA pointer variable.

Limitations:

This statement must be placed outside of a BEGIN DECLARE SECTION. It must also be positioned so that it obeys the rules regarding the declaration of host variables. In accordance with the host language rules governing the declaration of variables and their scope, any number of INCLUDE statements may be specified.

ANSI Specifics:

The INCLUDE statement is not part of the Standard.

Adabas SQL Server Specifics:

The AS clause is an Adabas SQL Server extension.

Example:

To enable Adabas SQL Server to generate a definition and declaration of the SQLCA structure, the following syntax applies:

```
INCLUDE SQLCA;
```

To enable Adabas SQL Server to generate a definition of the SQLDA structure and generate a pointer to the structure where the pointer name is specified by the user (in this case sql_pointer), the following syntax applies:

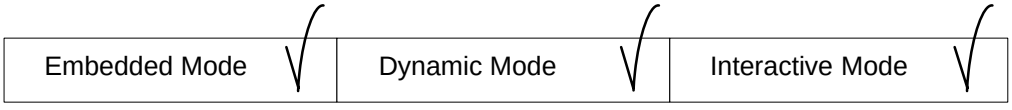
```
INCLUDE SQLDA AS sql_pointer;
```

INSERT

Function:

The INSERT statement inserts a new row into the target table using values derived from the row amendment expression.

Invocation:



Syntax:

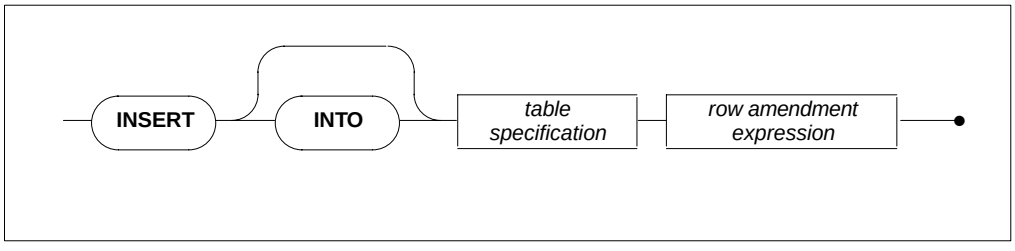


table specification is a qualified or unqualified identifier which refers to the table to be amended. The table must be defined at this statement's compilation time. If the table specification is a view reference this view must be updatable. See chapter **Common Elements**, section **Table Specification** for more details.

row amendment expression specifies the new values to which the columns in the row(s) under consideration will be assigned. See chapter **Common Elements**, section **Row Amendment Expression** for more details.

Description:

An INSERT statement simply inserts a number of new rows into the target table as specified by the row amendment expression.

If the target table in an INSERT statement is a subtable, then the values assigned to the foreign key columns must be equal to the values contained in the associated referenced key columns of the master table. This is compatible with the concepts of referential integrity. Adabas SQL Server uses these key values to identify the record, and in case of a level 2 target table the periodic group within the record, into which the new candidate row is to be inserted. The insertion of a row should not result in an insertion of a new record but rather in the insertion of a new occurrence. If the specified foreign key values do not correspond to any referenced key values, then a referential constraint violation is issued.

If the row amendment expression uses a query specification as its means of defining the input, then multiple rows may be inserted, otherwise the insertion of a single row will result.

If the query specification results in no rows, then no rows are inserted and the field sqlcode in the SQLCA is set to +100.

Limitations:

If the target table is a view, this view must be updatable as described in the section **DECLARE CURSOR** in this chapter.

It is only possible to insert rows into subtables, if the corresponding referenced key columns in the master table exit and are specified with the same value.

The special register SEQNO must not be specified as a target column. However, a value can be specified for a level 0 named SEQNO column. This value will then be the Adabas ISN. The values for level 1 and level 2 named SEQNO columns are occurrence numbers which are generated automatically and can not be specified in an insert operation. One exception is that the level 1 named SEQNO column may be assigned a value when the target table is a level 2 table and the value assigned to the level 1 named SEQNO column already exists.

An empty string or zero value can not be inserted into columns which have been defined with SUPPRESSION (i.e. the Adabas NU option) and with the NOT NULL option, as these two values actually represent the NULL value.

An empty string or zero value can not be inserted into a column that maps to an Adabas multiple-value field defined with SUPPRESSION, as these values are not representable under these conditions.

ANSI Specifics:

The use of a row amendment expression based on the SET format of the syntax diagram is not valid.

Adabas SQL Server Specifics:

The use of a row amendment expression based on the SET format of the syntax diagram is permitted.

DDL and DCL statements may be mixed in one transaction. DML statements must not be mixed with DDL/DCL statements in the same transaction. For details see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

DML statements must not be mixed with DDL/DCL statements in the same transaction. For details see the section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To insert a whole new row into the table cruise, the following syntax applies:

```
INSERT INTO cruise
SET cruise_id = 1234,
     start_date = 19920925,
     start_time = 12,
     end_date = 19921206,
     end_time = 14,
     start_harbor = 'ACAPULCO',
     destination_harbor = 'LIVERPOOL',
     cruise_price = 2050,
     bunk_number = 7
     bunks_free = 10
     id_yacht = 146,
     id_skipper = 244,
     id_predecessor = 5037,
     id_successor = 5039;
```

OPEN

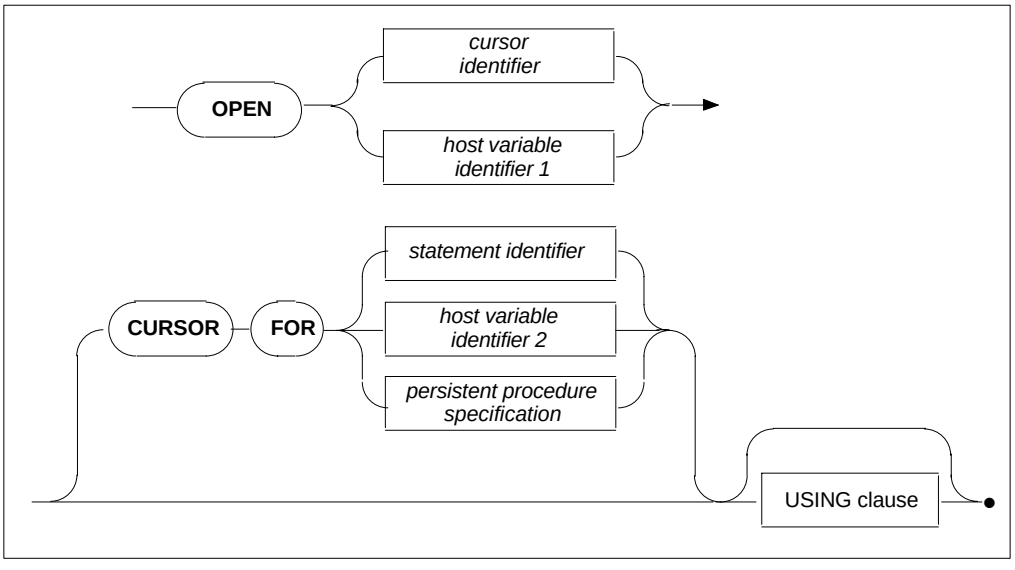
Function:

An OPEN statement establishes the contents of a cursor.

Invocation:

Embedded Mode √	Dynamic Mode	Interactive Mode

Syntax:



cursor identifier identifies the cursor to be used in the OPEN operation.

host variable identifier 1 is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules.

The value of the host variable must be a valid cursor identifier. A host variable can be used as cursor identifier only if the cursor is a dynamically declared cursor.

statement identifier	is a valid identifier denoting the name of the prepared statement.
host variable identifier 2	is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules. The value of the host variable must be the value returned by the PREPARE statement and thus identifying the prepared statement.
persistent procedure specification	specifies the identification of the prepared statement that has been stored in the catalog. The persistent procedure specification must include both the VERSION and PROCEDURE clause.
USING clause	defines an SQL descriptor area used to supply data to the associated dynamic cursor.

Description:

The OPEN statement causes the contents of the associated resultant table to be established. The cursor is initially positioned before to the first row. The cursor can be identified by use of a host variable only if the cursor is declared dynamically. Likewise, the USING clause can be used to provide input values only if the cursor is a dynamically declared cursor. Alternatively, values can be provided by the direct use of host variables.

Limitations:

The cursor to be opened must have been declared and must not be open. If the statement does not contain a CURSOR FOR clause the cursor must have been declared before.

The statement must be in the same compilation unit as the associated DECLARE CURSOR statement.

ANSI Specifics:

All cursors opened within a transaction are automatically closed by a COMMIT or ROLLBACK statement.

The persistent procedure specification and the USING clause must not be used.

The associated DECLARE CURSOR statement must physically precede the OPEN statement in the host program.

Adabas SQL Server Specifics:

The CLOSE statement is the only statement apart from the DISCONNECT statement that closes a cursor.

The cursor identifier can be given as a host variable if the cursor has been dynamically prepared.

The OPEN statement may appear anywhere in relation to the associated cursor statement in the host language.

DML statements must not be mixed with DDL/DCL statements in the same transaction. For details see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

The following syntax is required for opening a cursor of name cursor1.

```
OPEN cursor1 ;
```

The following syntax applies to opening a dynamic cursor and needing to supply that cursor with values within host variables (in this example 3 values).

```
OPEN cursor1 USING :hv1, :hv2, :hv3 ;
```

The following syntax is used to open a dynamic cursor using a persistent procedure specification:

```
OPEN cursor1 CURSOR FOR MODULE :mod PROCEDURE :proc Version :vers;
```

PREPARE

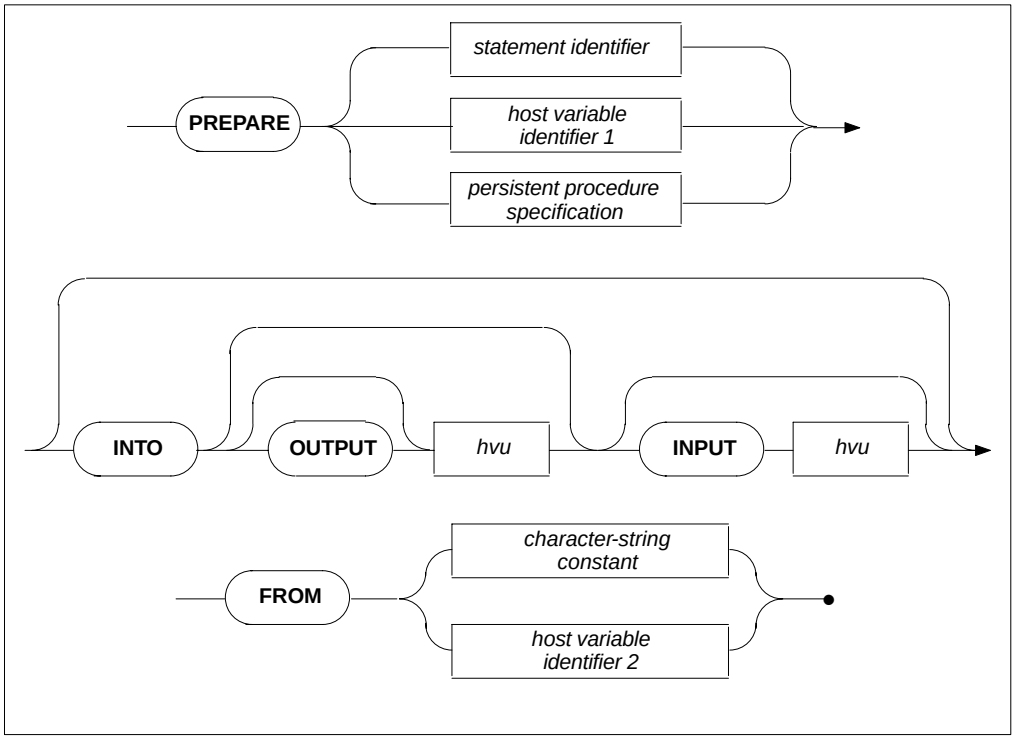
Function:

The PREPARE statement prepares an SQL statement for later execution.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√		

Syntax:



statement identifier — is a valid single identifier used to identify the statement to be prepared.

host variable identifier 1 — is a valid single host variable identifier of type character- string. It receives the unique value which is either generated by Adabas SQL Server or defined in the application program. This value identifies the statement to be prepared.

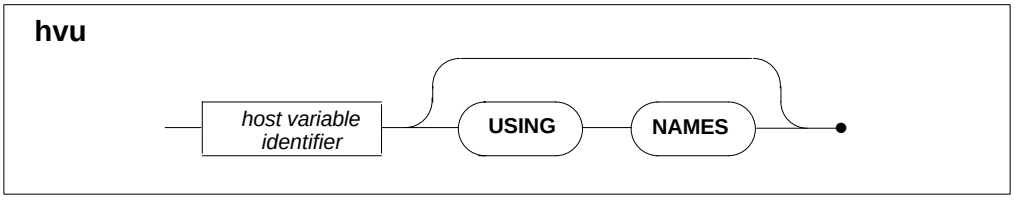
persistent procedure specification — specifies the identification of the prepared statement that has been stored in the catalog. The persistent procedure specification must include both the VERSION and PROCEDURE clause.

OUTPUT *hvu* — is the definition of the SQL descriptor area used to describe the expected output of the identified statement.

INPUT *hvu* — is the definition of the SQL descriptor area used to describe the expected input of the identified statement.

character-string constant — explicitly contains the source statement to be prepared

host variable identifier 2 — is a valid single host variable identifier which contains the character-string representation of the statement to be prepared.



host variable identifier — is a valid single host variable identifier and must have been defined in the application program according to the host-language-dependent rules. The value of the host variable must be the address of an SQL descriptor area (SQLDA).

Description:

The PREPARE statement performs the following actions:

1 COMPILATION:
An SQL statement in a character-string representation is compiled into an executable SQL statement which is called the prepared statement. If an error is encountered by Adabas SQL Server which prevents the SQL statement to be compiled successfully, an error is passed back to the application program in the SQLCODE field of the SQLCA. In this case no prepared statement is created.

2 IDENTIFICATION:
The prepared statement is kept for later execution. It is identified by the statement identifier provided by the application program or is generated by Adabas SQL Server and passed back into host variable 1. If it is intended that the statement identifier is to be generated by Adabas SQL Server it is necessary to initialize the variable with blanks or an empty string prior to execution. Otherwise, Adabas SQL Server will use the actual (non-blank) value of the variable. This identification will be used to refer to the prepared statement in a DESCRIBE, DECLARE CURSOR or EXECUTE statement.

3 RETENTION:
In DB2 mode a non-persistent prepared statement is kept for the duration of the transaction. This means that it can be executed repeatedly as long as this is done within the same transaction. The prepared statement is deleted by a COMMIT, ROLLBACK or DISCONNECT statement. A new statement can always be prepared using this identifier and deleting the originally prepared statement, except, when the identified statement refers to a SELECT statement whose cursor is currently open.

DEALLOCATE PREPARE may be used to explicitly delete a prepared statement.

4 DESCRIPTION:
The nature of the prepared statement can be determined and conveyed to the user by supplying appropriate SQL descriptor area variables. The functionality of a DESCRIBE statement can be incorporated into the PREPARE statement. For a full description of this functionality refer to the relevant passages of the section **DESCRIBE Statement** in this chapter.

Limitations:

The character-string must contain one of the following statements:

COMMIT, CREATE, DELETE, DROP, INSERT, ROLLBACK, SELECT, or UPDATE.

The statement string cannot contain host variables, instead it may contain host variable markers. A host variable marker is represented by a question mark (?). Host variable markers mark those places where values are to be inserted at the time the prepared statement is executed. For a description of how host variable markers are replaced by real values, see section **EXECUTE** in this chapter. In general, a host variable marker can be used in an SQL statement wherever a host variable can normally appear with the following restriction:

Note:
At compilation time, it must be possible to determine the data type resulting from the expression(s) contained in this statement.

ANSI Specifics:

The PREPARE statement is not part of the Standard.

Adabas SQL Server Specifics:

This statement may be mixed with any other DML, DDL and/or DCL statements in the same transaction.

Example:

To prepare an SQL statement, with an Id *'identifier1'* to remove all rows from the table cruise, the following syntax applies:

```
PREPARE identifier1 FROM
     'delete from cruise';
```

To prepare an SQL statement to delete a single row from the table cruise, where the row to be deleted is identified by it's cruise id given in a host variable, the following syntax is applied. Note the use of the host variable marker '?'.

```
PREPARE statement_id FROM
     'delete from cruise where cruise_id = ?';
```

To prepare a dynamic SELECT statement where the format of the derived columns is not known until runtime, and hence, the SQLDA needs to be used, the following syntax applies:

```
PREPARE statement_id INTO OUTPUT :sqlda
     FROM :dyn_select_id ;
```

REVOKE

Function:

The REVOKE statement removes privileges from a user, a list of users or PUBLIC on tables or views.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√	√	√

Syntax:

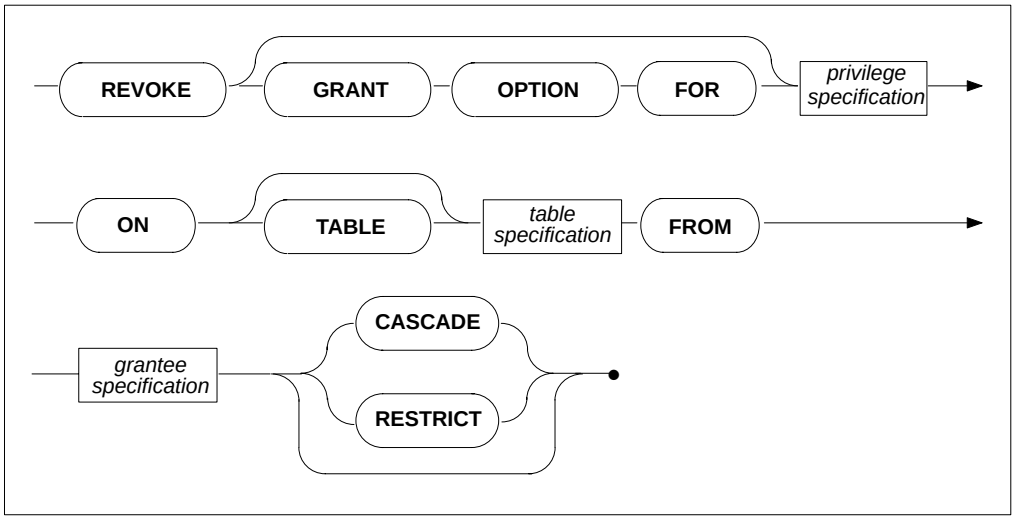


table specification	is an already existing table or view for which the revocation is to be performed. The table or view name should only be specified once.
privilege specification	is a list of one or more privileges that are to be revoked.
grantee specification	is a user, a list of users or PUBLIC for which the revocation is to be executed. A user should only be specified once.

Note:
*See chapter **Common Elements** for more details.*

Description:

The REVOKE statement revokes the specified privileges from a user, a list of users or PUBLIC for the specified table(s) or view(s). Neither the user identifier nor the table or view identifier should be specified more than once.

Unsuccessful execution of the REVOKE ALL PRIVILEGES statement results in response code 0, even though the ANSI Standard prescribes a Warning.

For details about what privileges are possible, what they mean and the constraints on them, see chapter **Common Elements**, section **Privilege Specification**.

Limitations:

General rules:

- If a revoke from PUBLIC is specified then only those privileges that have been granted to PUBLIC will be revoked.
- You can not revoke privileges from yourself.
- The keyword RESTRICT only affects the current user plus constraints.
- For the privilege UPDATE a revocation of the table privilege causes an implicit revocation of all column privileges for the specified table. If only the column privilege is revoked, an existing table privilege remains unaltered.
- REVOKE CASCADE is not supported yet. If the revokee has granted the privilege to a third grantee, the privilege cannot be revoked from the revokee unless he has revoked it from the third grantee. Trying to revoke these privileges will fail and result in an error condition.

Authority to revoke privileges:

- The revoker is the owner of the table or view.
- The revoker gave the privileges that are to be revoked.

Revoking privileges from Views:

Revoking privileges from a base table which would affect any view that relies upon that table will fail and result in an error conditions. To revoke these privileges, the view must be dropped first.

The execution of the REVOKE statement is an atomic action that is closed by an implicit COMMIT and can, therefore, not be rolled back.

ANSI Specifics:

It is mandatory to specify:

- ON specified table. The keyword TABLE in ON TABLE *table specification* is not supported.
- either CASCADE or RESTRICT.

Adabas SQL Server Specifics:

It is optional (and has no effect) to specify the keyword TABLE in ON TABLE *table specification.*

If neither CASCADE nor RESTRICT is specified, then RESTRICT is the default action.

The CASCADE functionality is not yet implemented.

Examples:

Simple revocation:

Tim has given Peter ALL privileges on table CRUISE. Tim then decides to revoke the DELETE privilege from Peter.

```
REVOKE DELETE ON CRUISE FROM PETER;
REVOKE DELETE ON CRUISE FROM PETER RESTRICT; [ ANSI-specific method ]
```

This has the effect of removing the DELETE privilege from Peter, but will still leave him with the SELECT, INSERT and UPDATE privileges for this table.

Simple revocation (no cascading):

Tim has given Peter ALL privileges including the "WITH GRANT OPTION" on table CRUISE. Peter then gives Anne the privileges to SELECT and DELETE on table CRUISE. Tim then decides to revoke the DELETE privilege from Peter.

```
Tim: REVOKE DELETE ON CRUISE FROM PETER;
Tim: REVOKE DELETE ON CRUISE FROM PETER RESTRICT; [ ANSI-specific method ]
```

This will fail and result in the error message that there are still dependent privileges. First, Peter has to revoke the privileges SELECT and DELETE from Anne:

```
Peter: REVOKE SELECT, DELETE ON CRUISE FROM ANNE;
Peter: REVOKE SELECT, DELETE ON CRUISE FROM ANNE RESTRICT;
     [ ANSI-specific method ]
```

After that, Tim can revoke the DELETE privilege from Peter.

Assume that Tim has also given Peter the UPDATE table privilege. Now he wants to revoke the UPDATE privilege on column xx from Peter.

```
Tim: REVOKE UPDATE ( XX ) ON CRUISE FROM PETER;
```

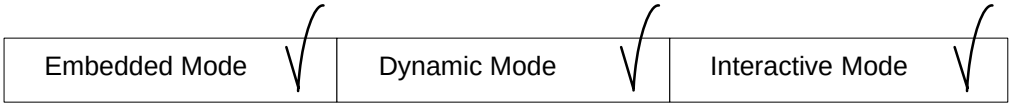
The result will be that the UPDATE table privilege still exists: only the column privilege for column xx is destroyed. If Peter then tries to grant the UPDATE privilege to Gary, this will have the effect that Gary also gets UPDATE column privileges for all columns of table CRUISE with the exception of column xx. That means Gary is allowed to update all columns in CRUISE except xx.

ROLLBACK

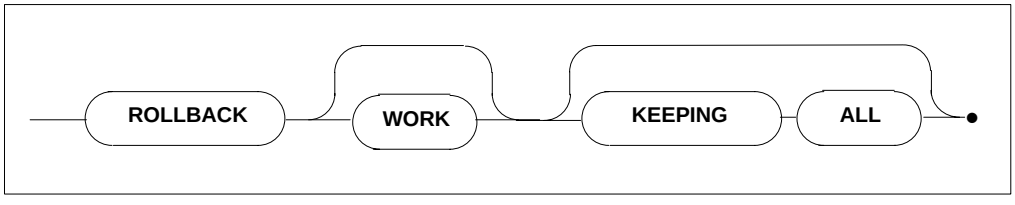
Function:

The ROLLBACK statement terminates a transaction and cancels all changes to the database that were made during the current transaction.

Invocation:



Syntax:



Description:

The ROLLBACK statement terminates the current transaction and starts a new transaction. All changes to the database that have been made during the transaction are cancelled and the situation is reinstated as it existed at the time the transaction was started. All cursors that have been opened during the current transaction are closed.

If KEEPING ALL is specified, none of the currently opened cursors are closed, i.e., all cursors can be processed further after the execution of the ROLLBACK statement.

Limitations:

In DB2 mode all statements that have been prepared during the current transaction are deleted.

ANSI Specifics:

The keyword WORK is mandatory. The keywords KEEPING ALL are not supported.

Adabas SQL Server Specifics:

The keyword WORK is optional.

Example:

To negate all changes in the current transaction the following syntax applies:

```
ROLLBACK WORK ;
```

SELECT (SINGLE ROW)

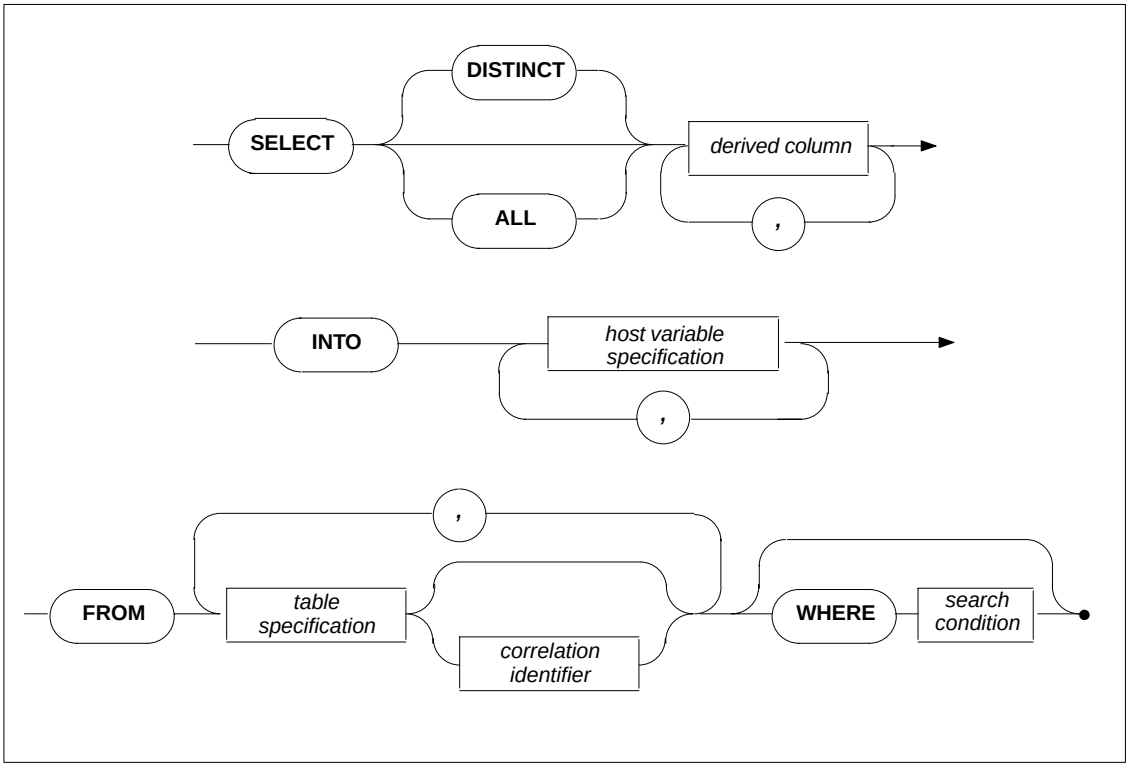
Function:

The single row SELECT statement obtains a single row of data from the database according to the specified conditions.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
√		

Syntax:



Elements which are also part of the query specification are described in detail in chapter **Common Elements** section **Query Specification**.

derived column	are the specification of the corresponding columns in the final resultant table derived by the query. Derived columns are separated by commas and all of them together are referred to as the derived column list.
*	is an abbreviated form of listing all derived columns of all tables in the table name list. In ANSI compatibility mode, it is not permitted to qualify the asterisk by using the correlation identifier or the table specification.
host variable specification clause)	is a valid single host variable and is only relevant for (INTO single row SELECT. The host variables are intended to receive the returned data as specified by the SELECT statement's derived column list.
table specification	a valid table specification as described in chapter **Common Elements** section **Table Specification**.
correlation identifier	is a means of giving an alternative name to a particular table for use within the query and subqueries which are in scope.
WHERE clause	is the specification of a search condition which candidate rows must fulfil in order to become part of the resultant table.

Description:

The single row SELECT statement is used to obtain a single row of data from the database. Please refer to the description of a query specification (chapter **Common Elements**, section **Query Specification)** for information on the processing of a SELECT statement.

The single row SELECT statement can only be embedded and can only return one or no rows. A negative error code is returned in the sqlcode field of the SQLCA, if the resultant table actually contains more than one row. This is because the specified host variables in the INTO clause can only receive one row of data. A host variable specification which references a host variable structure is equivalent to individual host variable specifications which reference all the elements of a structure singularly.

The single row SELECT statement is the only invocation of a SELECT statement where an INTO clause is allowed and required. The only other way to specify an INTO clause is as a part of the FETCH statement. For details refer to the section **FETCH Statement** of this chapter.

Limitations:

A maximum of one row may be returned. The use of a valid INTO clause is required.

ANSI Specifics:

None.

Adabas SQL Server Specifics:

DML statements must not be mixed with DDL/DCL statements in the same transaction. For details see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To find out how many persons the yacht no. 6230 can accommodate, the following syntax applies:

```
SELECT bunks
     INTO :bunks
          FROM yacht
          WHERE yacht_id = 6230;
```

SELECT

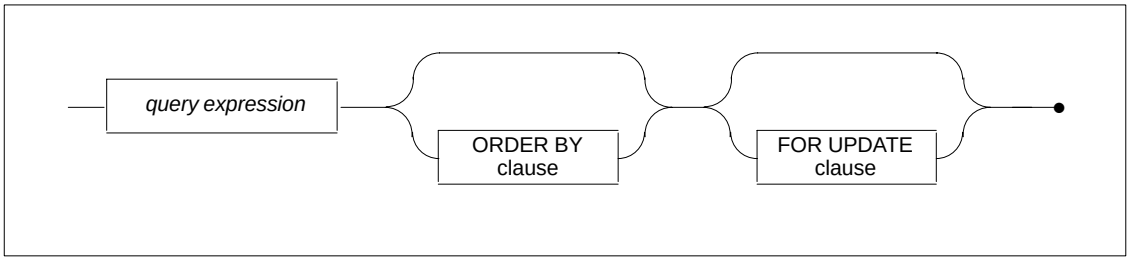
Function:

The SELECT statement obtains data from the database according to the specified conditions.

Invocation:

Embedded Mode	Dynamic Mode	Interactive Mode
	√	√

Syntax:



For details about the elements of query expression see chapter **Common Elements**.

ORDER BY clause — is the specification of a user-defined ordering of the resultant table. Otherwise the resultant table is not ordered (not valid within a single row SELECT statement).

FOR UPDATE clause — is the explicit indication that this cursor is to be used in conjunction with either an UPDATE and/or DELETE WHERE CURRENT OF CURSOR statement.

Description:

The SELECT statement is used to obtain data from the database. Please refer to the description of a query expression or query specification (chapter **Common Elements)** for information on the processing of a SELECT statement.

When submitted either dynamically the statement must be associated with a PREPARE statement and an associated dynamic cursor. The statement may then select more than one row.

When used interactively, the statement may again select more than one row. The use of the INTO clause is not permitted.

Limitations:

The use of an ORDER BY clause is only valid within a dynamic or interactive SELECT statement. Its use enables the resultant table to be sorted in a user-defined sequence.

The use of the FOR UPDATE clause is only valid within a dynamic or interactive SELECT statement.

ANSI Specifics:

The use of a FOR UPDATE clause is not supported.

Adabas SQL Server Specifics:

DML statements must not be mixed with DDL/DCL statements in the same transaction. For details see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To select the addresses of all persons living in Frankfurt the following syntax applies:

```
SELECT address_addition_1,address_addition_2
     FROM persons
     WHERE city = 'FRANKFURT';
```

To list all individual start and destination harbors, the following syntax applies:

```
     SELECT start_harbor
     FROM cruise
UNION
     SELECT destination_harbor
     FROM cruise ;
```

SET CONNECTION

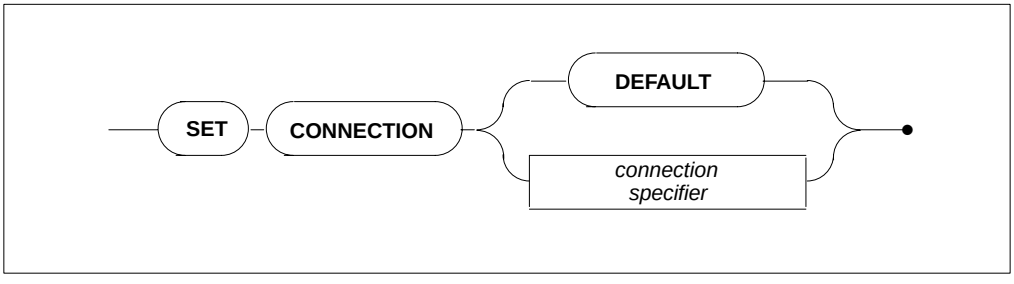
Function:

The SET CONNECTION statement is used to switch between SQL sessions.

Invocation:

Embedded Mode √	Dynamic Mode	Interactive Mode

Syntax:



connection specifier can either be a character-string constant or single host variable identifier. The host variable must have been defined in the application program according to the host-language-dependent rules and its values must be a character string. The maximum length is 32 characters.

Description:

The SET CONNECTION statement is used to switch from one SQL session to the next specified SQL session. The context of the Adabas SQL Server environment is restored to its exact state at the time of suspension.

Limitations:

None.

ANSI Specifics:

The SET CONNECTION statement is not part of the ANSI standard.

Adabas SQL Server Specifics:

None.

Example:

To switch from the current session to MYSESSION the following syntax applies:

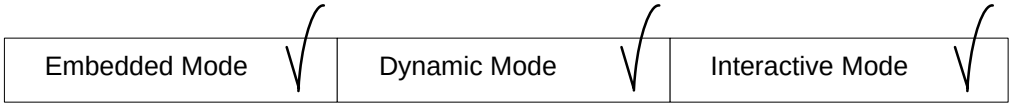
```
SET CONNECTION :MYSESSION;
```

UPDATE

Function:

The UPDATE statement modifies the data contained in a particular row or set of rows. There are two forms, namely positioned UPDATE and searched UPDATE.

Invocation:



Syntax:

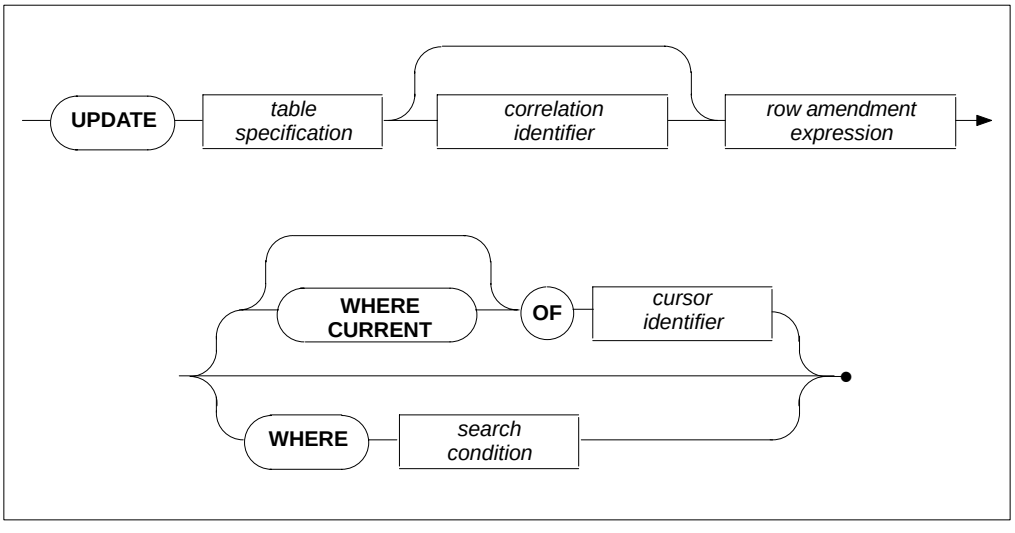


table specification	is a qualified or unqualified identifier which refers to the table to be amended. The table must be defined at this statement's compilation time. If the table specification is a view reference this view must be updatable. See chapter **Common Elements**, section **Table Specification** for more details.
correlation identifier	allows the table to be referenced by another identifier. See chapter **Common Elements**, section **Correlation Identifier** for more details.
row amendment expression	specifies the new values to which the columns in the row(s) under consideration will be assigned. See chapter **Common Elements**, section **Row Amendment Expression** for more details.
WHERE CURRENT OF *identifier*	signifies that the UPDATE is positioned. The cursor *cursor* identifier refers to a cursor which is currently open and pointing to a row.
WHERE *search condition*	signifies that the UPDATE statement is searched. Omission of the WHERE clause really equates to a special case of a searched UPDATE statement.

Description:

An UPDATE statement modifies the columns of the rows identified in the WHERE clause with the values specified in the row amendment expression.

Updates of key column values in the master table will be cascaded to the related subtables. All other columns of a subtable can be updated with new values as usual, provided that the values of foreign keys and SEQNOs remain the same as already stored.

If the UPDATE statement is positioned, then the UPDATE is only applied to the row to which the cursor is currently pointing. The cursor must be open and pointing to a row otherwise a runtime error will occur. In addition, the cursor must be updatable. See section **DECLARE CURSOR Statement** for further details. Updating does not alter the position of the cursor. In addition, any locks on the row are not released until either a COMMIT or a ROLLBACK statement is executed.

Alternatively, in case of a searched UPDATE statement, a resultant table is established at execution time in a similar manor to a query specification. The UPDATE, then occurs for each row in the resultant table as specified by the row amendment expression. All the rows of the resultant table are locked and are not released until either a COMMIT or a ROLLBACK is executed. If no rows are identified for updating, then the field SQLCODE of the SQLCA will be set to +100.

An UPDATE statement without a WHERE clause is really a special case of the searched alternative as a resultant table is established which contains all the rows of the target table.

Limitations:

If the table referenced is a view, then this view must be updatable. See the section **DECLARE CURSOR Statement** earlier in this chapter for more details.

In a positioned UPDATE statement the table reference must be identical to that referenced in the associated DECLARE CURSOR statement.

In addition, if the associated DECLARE CURSOR statement was defined in another compilation unit, then it must have been specified with the FOR UPDATE clause.

Also, the associated cursor must be updatable, open and positioned on a row of the resultant table.

A positioned UPDATE statement is not allowed in interactive mode.

For reasons of enforcing referential integrity it is not possible to change the value of foreign key columns in level 1 or level 2 tables. In a clustered environment this would require to physically move a row to a new location.

Restrictions which apply when updating views can be found in the **Limitation** section of the **CREATE VIEW** statement description.

A SEQNO column is not updatable. The SEQNO columns map to the information that is used for internal Adabas addressing, and no rows will be moved to a new location using an UPDATE statement.

An empty string or zero value can not be inserted into columns which have been defined with SUPPRESSION (i.e. the Adabas NU option) and with the NULL capability, as these two values are actually not representable under this condition. Same applies to columns with just the NULL capability, as the empty string or zero value represent the NULL value.

ANSI Specifics:

The use of the VALUES format in the row amendment expression is not permitted.

A positioned UPDATE statement must appear in the same compilation unit as the associated DECLARE and OPEN statements and must appear physically after the DECLARE statement.

The use of correlation identifiers in this context is not supported in ANSI compatibility mode.

Adabas SQL Server Specifics:

The use of either format (SET or VALUES) of the row amendment expression is permitted.

A positioned UPDATE statement can be in a different compilation unit to that of the associated DECLARE as long as a FOR UPDATE clause is specified. If the UPDATE is in the same compilation unit as the associated DECLARE CURSOR statement,, then there is no restriction as to the relative positions of the two statements.

The use of correlation identifiers is permitted.

DML statements must not be mixed with DDL/DCL statements in the same transaction. For details see the chapter **General Concepts of SQL Programming,** section **Transaction Logic** in the *Adabas SQL Server Programmer's Guide*.

Example:

To update all prices in the cruise table by adding 100 to the original cost, the following syntax is required.

```
UPDATE cruise
     SET cruise_price = cruise_price + 100 ;
```

To decrease a particular customers amount to pay by 100 (customer id = 816), the following syntax applies:

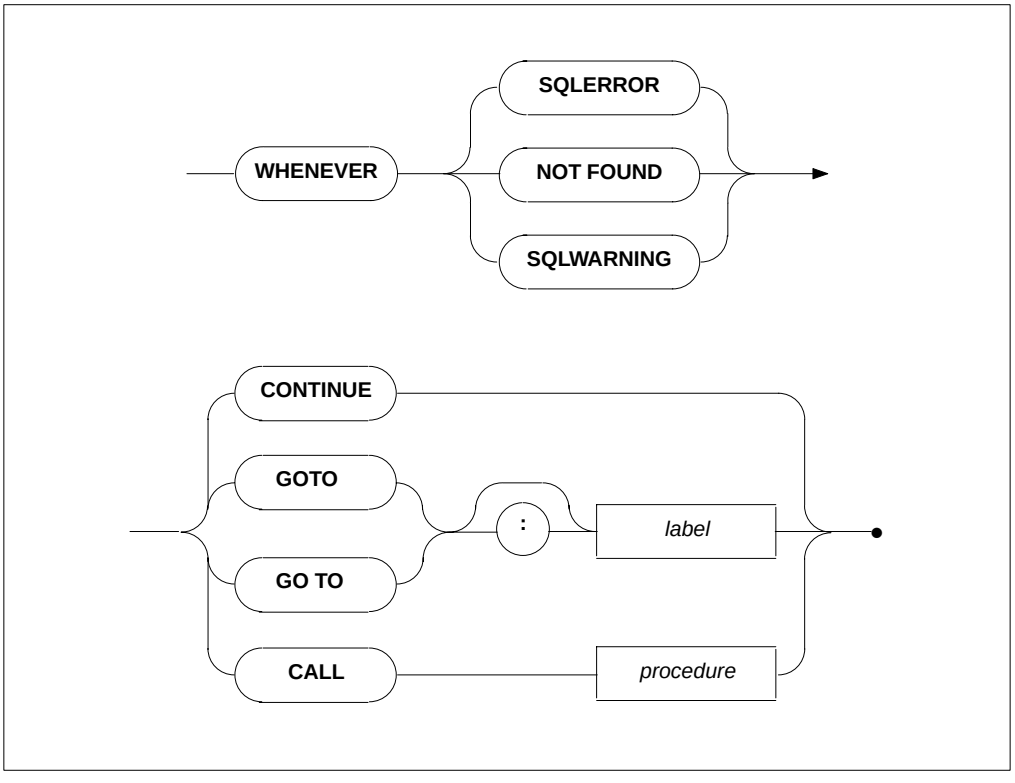
```
UPDATE contract
     SET amount_payment = amount_payment - 100
          WHERE id_customer = 816;
```

WHENEVER

Function:

This statement specifies the action to be performed when an SQL statement results in an exception condition.

Syntax:



label	a valid host language label
procedure	a valid host language procedure, routine or function identifier

Description:

The variables in the SQLCA are updated during program execution and should be verified by the application program. This may be done in two different ways:

- by explicitly testing the contents of the appropriate variable in the SQLCA, usually the SQLCODE field.
- by specifying the SQL statement WHENEVER.

Note:
If no testing takes place, the default action for errors is to continue with the application program.

An application program may contain any number of WHENEVER statements. The WHENEVER statement may appear anywhere after the declaration of an SQLCA. WHENEVER statements are pre-processed strictly in the order of their physical appearance in the source code, regardless of the execution order or conditional execution that the application program might imply. They will also only refer to the SQLCA which is currently in scope.

Should two or more WHENEVER statements contradict each other, then the statement which was physically specified last is relevant for a particular SQL statement.

The SQLCA is not updated as a result of a WHENEVER statement.

The condition is determined to be true according to the value of the variable SQLCODE and may be one of the following:

- NOT FOUND if the value is +100, indicating that no rows were found.
- SQLERROR if the value is negative, indicating an error.
- SQLWARNING if the value is positive other than +100, indicating a warning.

The action taken, should the condition be true, may be one of the following:

- CONTINUE ignores the exception condition and continues with the next executable statement.
- GOTO label continues the application program's logic with the statement identified by the label. The label must conform to the rules of the host language. The label may be prefixed with or without a ':' . GOTO may also be specified as GO TO.
- CALL procedure continues the application program's logic with the procedure identified by procedure. The procedure name must conform to the rules of host language. The procedure may not specify any host language parameters.

Generally, a single WHENEVER statement will be valid for all SQL statements in the program. If an error occurs, control can be passed to an error handling routine. If SQL statements are to be executed from within this error handling routine, they too are subject to the conditions of the relevant WHENEVER statement. This means, if an error occurs during execution of the called error handling routine, an attempt will be made to call this same routine again, because the initial WHENEVER statement is still valid. This situation can be avoided by having a second WHENEVER statement in the error handling routine which specifies the option CONTINUE. It is recommended to test the SQLCA explicitly within the error routine.

Limitations:

For the ANSI 74 standard (Precompiler setting COBOL II = off) every SQL statement is treated as if the optional period was coded. That means the generated code will always be terminated with a period. It is not possible to code more than one SQL statement in an IF statement. Also refer to the host language sections of the *Adabas SQL Server Programmers Guide.*

ANSI Specifics:

The SQLWARNING condition and the CALL option are not part of the Standard.

Adabas SQL Server Specifics:

This is an Adabas SQL Server extension.

Example:

To continue normal execution of a program if, after an SQL query returned no rows, the following syntax would apply:

```
WHENEVER NOT FOUND CONTINUE;
```

To continue a programs execution at another point (where that point is specified by a label) when an SQL statement produces a warning, the following syntax applies:

```
WHENEVER SQLWARNING GOTO label_name;
```

To divert a program's flow to a 'Procedure' when an SQL statement produces an error, the following syntax applies:

```
WHENEVER SQLERROR CALL procedure_name;
```

A

APPENDIX A — GLOSSARY

Adabas File

Each SQL base table – contained or not contained in a cluster – is represented by one Adabas file. Before issuing the CREATE TABLE or CREATE CLUSTER statements, this Adabas file must be tailored using parameters specified in a CREATE TABLESPACE statement. An existing Adabas file is introduced to the catalog using CREATE TABLE DESCRIPTION/ CREATE CLUSTER DESCRIPTION statements.

Adabas Short Name

A shortname identifier specifies the Adabas short name of the corresponding field in the underlying Adabas file.

Base Table

A base table (and the data contained herein) is directly physically present and is not computed as in the case of viewed tables (views). A base table is defined by a CREATE TABLE/CREATE TABLE DESCRIPTION statement or a corresponding substatement in a CREATE CLUSTER/ CREATE CLUSTER DESCRIPTION statement. A base table which is not part of a cluster is the sole representation of one Adabas file.

Boolean Operator

An operator with predicates as its operands constitute a boolean expression. It can be: AND, OR, NOT.

Boolean Expression

An expression of predicates linked with boolean operators delivering boolean results.

Candidate Row

Any row within an intermediate resultant table which is to be considered for the next resultant table.

Candidate Group

A set of candidate rows which are grouped together during the processing of a GROUP BY clause.

Case Sensitive

Differences between lowercase and uppercase strings are significant.

Catalog

The catalog is a group of Adabas files which hold all data objects and their descriptions for the SQL environment and including any meta programs. The catalog, therefore, contains all information necessary for the operation and maintenance of Adabas SQL Server. The DBA_SCHEMA and the INFORMATION_SCHEMA contain views extracting those data relevant for administration purposes.

Cluster

Refer to Table Cluster

Column

A column is a subobject of a table and is the smallest unit of a table that can be selected and updated. Adabas SQL Server supports column types ORDINARY, SEQ-NO., and SIMULATED-LONG. The order in which columns are specified during the creation of a table is the order in which the columns will be displayed in a SELECT * request. Unless explicitly specified, this is also the order which Adabas SQL Server expects when rows are to be inserted.

Column Level

The level of a column describes its Adabas realization. Level 1 means an multiple-value field (MU field) not in a periodic group (PE group), whereas Level 2 means an MU field in a PE group. Note that only those columns whose level is equal to the level of the table in which they are contained can be updated.

Compilation Unit

A user’s source code file containing embedded SQL commands, which is submitted to Adabas SQL Server compilation system.

Constraint

A constraint is a subobject of a base table which is defined to ensure the compliance of the actual data with the specified conditions. Adabas SQL Server knows four different types of constraints: NOT NULL, UNIQUE, PRIMARY KEY and FOREIGN KEY. Syntactically, a constraint referring to a simple column can be defined within a table column element. Constraints referring to more than one column have to be defined by a table constraint element. The name of a constraint is unique within the schema. It is automatically generated if not specified.

Cursor

The cursor concept was developed to aid 3GLs (e.g., COBOL) to process selected resultant tables. A cursor is essentially a pointer used to proceed through the rows of the resultant table.

Database

The database is the implementation of the SQL schema concept consisting of tables (Adabas files) which in turn consist of rows (Adabas records) and columns (Adabas fields).

Database Identifier

Adabas SQL Server enables a string identifier to be associated with an Adabas database identifier for use within certain statements.

Data Control Language (DCL)

DCL handles the data security aspects by providing statements for privilege granting and revoking.

Data Definition Language (DDL)

DDL handles the creation, alteration and deletion of SQL data structures.

Data Manipulation Language (DML)

DML handles the manipulation of SQL data structures.

Declare Cursor-Open-Fetch Cycle

These three statements are interdependent. Wherever a DECLARE CURSOR statement defines a resultant table, an associated OPEN statement establishes it and then successive FETCH statements retrieve rows. This is the classic method of retrieving data.

Default Schema Identifier

The qualifier which automatically prefixes an unqualified table or view name when attempting to resolve it.

Default Database

A table created using a CREATE TABLE statement is assigned an Adabas file number and an Adabas database ID. This is the default database for that table. When no override database (either local or global) has been specified, this is the current database. See **Appendix C** of the *Adabas SQL Server Programmer's Guide* for details.

Default Locking Specification

A default locking specification may be defined for Adabas SQL Server. This determines the locking of rows upon executing retrieval operations.

Derived Column

A column of a query's projection list (derived columns list) which is derived from one or more value sources.

Derived Column Label

Under certain circumstances, a derived column also has an associated derived column label which can be referenced from the ORDER BY clause, for instance.

Derived Column List

A list of derived columns in a query specification which define the resultant table's format.

Descriptor Area

See "SQLDA"

Dynamic Cursor

A cursor associated with a dynamic SELECT statement. See also: DECLARE CURSOR-OPEN-FETCH Cycle.

Dynamic SQL

An SQL statement which is generated at runtime only. Host programs submit this statement in the form of a string.

Embedded Statement

An SQL statement which is embedded in the host program directly rather then being interactively submitted to Adabas SQL Server.

Host Language

The language in which the host or application program is written (C, COBOL, etc) and into which SQL statements will be embedded.

Host Variable

A variable declared within the host program, which is the medium of exchange of data between the host program and Adabas SQL Server at runtime.

Host Variable Markers

A marker which may be specified in dynamic statements where a value is to be inserted by Adabas SQL Server at runtime.

Indicator Variable

A variable which primarily denotes whether the associated host variable contains the NULL value or not. Alternatively, it also denotes whether the associated host variable's contents have been truncated by Adabas SQL Server.

Index

An index is a subobject of a base table improving the performance of queries on columns which it is referring to. The current version of Adabas SQL Server supports the representation of Adabas (super-/subdescriptors). The Adabas UQ option leads to an UQ index. An index is called multiple, if an only if it refers to at least one column with a level greater than zero.

Joined Query

A query specification with more than one table specified in the FROM clause.

Master Table

A table cluster consists of one master table and one or more subtables. The relationship between master table and subtable is defined using a referenced key/foreign key relationship. The master table of a table cluster is the only table which contains no (clustering) referential constraint.

MU/Multiple-value Field

A field that can have a maximum of 199 multiple values (occurrences) within an Adabas record. It is preceded by an one-byte count field indicating the number of occurrences. For further details refer to the current Adabas C documentation.

NULL value

A special status of a field signifying that the value is unknown.

Outer Reference

A column in a subquery which is derived from a table which is declared in another query specification. This query specification in turn contains, either directly or through several levels, the subquery in question.

PE/Periodic Group

One of many fields that may repeat multiple times within an Adabas record and which is always preceded by an one-byte count field. For further details refer to the current Adabas C documentation.

Positioned UPDATE/DELETE

A DELETE or UPDATE statement in which the row to be considered in the base table is identified by the position of the associated cursor.

PREPAREd Statement

A statement which has been dynamically generated and submitted to the embedded static PREPARE statement for compilation.

Privileges

A privilege is the authorization to perform predefined operations (INSERT, UPDATE, DELETE, SELECT). This authorization is given by an owner of a database object to a particular user.

Query Expression

A query expression is an expression involving one or more query specifications connected using the UNION operator. It is used exclusively in a DECLARE CURSOR statement.

Query Specification

A query specification defines the resultant table specified in the projection list derived from the tables or views given in the table list, subject to the conditions imposed by the optional WHERE and/or HAVING clause and optionally grouped according to the GROUP BY clause.

Referential Constraint

A referential constraint is a constraint of type FOREIGN KEY. Adabas SQL Server supports this constraint type only in order to build table clusters. Therefore, the columns used by such a constraint, the referencing columns, are physically identical with the referenced columns. As a consequence the only referential triggered actions supported are the cascaded DELETE and UPDATE operations.

Resultant Table

A query specification conceptually produces a virtual table as its result. This is called a resultant table.

Rotated Field

If a MU field has a fixed number of occurrences, then this field can be rotated. Each occurrence is mapped to an SQL column.

Row

A row is the smallest unit of data that can be inserted into or deleted from a table. The order in which Adabas SQL Server returns rows is not necessarily consistent from query to query. If a specific order is desired, the ORDER BY clause must be used.

Runtime

Applications, after having been compiled at compile time are executed at runtime. Runtime is the point where SQL statements are executed and data is returned.

Runtime Error

During execution of a previously precompiled SQL statement, certain conditions may occur which result in runtime errors.

Schema

The Schema is a logical container within the catalog to group together subsequently created catalog resident objects. A Schema is a collection of data structures and objects defined by a set of DDL statements and privileges defined by DCL statements.

Scope

For example, he scope of a correlation identifier consists of the query specification or statement where it has been defined and all the subqueries present within that query specification or statement.

Search Criteria

A generic term for the search condition of a WHERE clause.

Searched UPDATE/DELETE

An UPDATE or DELETE statement, which establishes its own resultant table by means of an integral WHERE clause (see also: Positioned UPDATE/DELETE).

Search Expression

A collection of predicates linked by boolean operators.

Search Term

A predicate.

Select List

See “derived column list”.

SEQNO (Sequence Number)

The SEQNO is an SQL column or a special register. At table level 0 the SEQNO is used to retrieve or insert the Adabas ISN, at table level 1 and 2 the SEQNO is used to retrieve or insert the Adabas occurrence numbers in connection with MU/PE.

Short name

Refer to Adabas short name.

SQLCA

The SQL Communications Area is a host variable structure used to provide the programmer with comprehensive information about the success or failure of each SQL statement.

sqlcode Field

As part of the SQLCA, the sqlcode field is a special integer field and functions as a carrier of a status code, i.e. unsuccessful/successful execution of the command.

SQLDA

The SQLDA is a host variable structure defined by Adabas SQL Server and used to communicate information about a recently prepared dynamic statement. This SQL Descriptor Area provides the programmer with comprehensive information about each result column of a dynamic SELECT statement.

SQL Identifier

An identifier which, rather than refer to a variable in a host program, refers to an SQL entity pertinent to the program, e.g., cursor identifier, statement identifier.

Statement Id(entifier)

Any dynamic statement which is prepared for later processing needs a unique identifier in order to reference these prepared statement in other DESCRIBE, EXECUTE or dynamic DECLARE CURSOR statements.

Static SQL

An SQL statement which is processed at compilation time and is fixed thereafter.

Subquery (Subselect)

A query specification which is nested in another query specification, e.g., as part of an IN predicate.

Subtable

Subtables are all tables in a table cluster other than the master table. Each subtable contains exactly one (clustering) referential constraint. From a physical point of view, subtables represent the multiple structures of Adabas, the MU fields and PE groups. There are some update restrictions on subtables: No deletion of columns/rows, update of columns not referenced as foreign key, only and insertion of values for each referenced row up to the Adabas limit (191 or 99) only.

Table

A table is a subobject of a schema or a table cluster. It is the only structure for storing and accessing data. Adabas SQL Server supports the table types: base table and view.

Table Cluster

A table cluster is a subobject of a schema. It contains base tables which are interconnected by referential constraints. It is implemented by one Adabas file.

Table Level

A subtable referencing the master table is a level 1 table, whereas a subtable referencing another (level 1) subtable has level 2. All other base tables are level 0 tables.

Tablespace

A tablespace is a subobject of a schema. It contains the Adabas file attribute for a base table or a table cluster with the same name.

Transaction

A transaction is initiated as soon as the first SQL statement is being executed provided that no other transaction is currently active. Termination of a transaction requires a COMMIT or ROLLBACK statement.

Tri-State Logic

A predicate may return any of the following results: TRUE, FALSE, UNKNOWN.

UNIQUE CONSTRAINT

A unique constraint is a constraint of type UNIQUE or PRIMARY KEY. It implies a UQ index on the column list which it is defined for (without SEQNO columns).

UNKNOWN Status

The result of a predicate may be UNKNOWN if an operand equates to the NULL value.

User

A user is established as a result of a CREATE USER statement. It defines a particular user identification which must be supplied to the system upon connection using the CONNECT statement. The subsequent session then runs under this user identification. The user identification is furthermore used to establish ownerships in regard to schemas and associated database objects and in the evaluation of privileges.

Value Source

A value can result from several different origins: a constant, a host variable, a column, a function, an expression or a subquery.

View (Viewed Table)

Views are virtual tables not based upon their own, physically separate, distinguishable stored data. The view definition in terms of base tables is specified within the CREATE VIEW statement. Retrieval operations on a view are translated into equivalent operations on the underlying base table(s). Grouped views contain columns which are derived by using built-in functions like SUM, MAX, etc. Therefore, grouped views cannot be updated or joined with other tables.

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W