# Table Of Contents

Introduction ........................................................................................................... 1  
Notices .................................................................................................................. 3  
Audience ............................................................................................................... 5  
Syntax Conventions .............................................................................................. 7  
Concepts ............................................................................................................... 9  
Alias ..................................................................................................................... 9  
Column ............................................................................................................... 9  
Comments ........................................................................................................... 10  
Condition .......................................................................................................... 11  
Data Types ........................................................................................................ 12  
DECLARE VARIABLE Statement ....................................................................... 13  
Expressions ....................................................................................................... 14  
Foreign Key ...................................................................................................... 15  
Literal ............................................................................................................... 15  
M Value .......................................................................................................... 17  
M Variable ..................................................................................................... 17  
Method ............................................................................................................. 17  
Names .............................................................................................................. 18  
Primary Key ...................................................................................................... 19  
READ ............................................................................................................. 19  
SET ............................................................................................................... 22  
Subquery .......................................................................................................... 22  
Table ............................................................................................................... 22  
Tests .............................................................................................................. 23  
Values .......................................................................................................... 27  
View .............................................................................................................. 27  
Wild Cards ..................................................................................................... 27  
Column Modifiers ............................................................................................ 28  
Event Blocks ................................................................................................... 32  
RUN .............................................................................................................. 37  
SET Statement ................................................................................................. 38  
Data Control Language .................................................................................... 47  
GRANT ........................................................................................................... 47  
REVOKE ....................................................................................................... 48  
Query Set ....................................................................................................... 48  
Table Set ..................................................................................................... 49  
Data Definition Language .............................................................................. 51  
ALTER .......................................................................................................... 51  
ALTER INDEX .............................................................................................. 51  
ALTER PROCEDURE .................................................................................... 52  
ALTER SCHEMA .......................................................................................... 52  
ALTER TABLE .............................................................................................. 53  
ALTER VIEW ................................................................................................. 56  
CREATE ....................................................................................................... 57
Introduction

The KB_SQL Syntax Guide details KB_SQL syntax, usage rules, and examples.
Notices

Copyright © 2005 Knowledge Based Systems, Inc. All Rights Reserved.

KB_SQL ® is the registered trademark of Knowledge Based Systems, Inc. All Rights Reserved.

All other trademarks or registered trademarks are properties of their respective companies.

Warning: This publication is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Knowledge Based Systems, Inc. shall not be liable for any errors, opinions, or damages of any kind in connection with the content of this publication. The information in this publication is subject to change without notice.

The information contained in this document is subject to change without notice. Knowledge Based Systems, Inc. does not warrant that this document is free of errors. If you find any problems in the documentation, please report them to us in writing.

Knowledge Based Systems, Inc.
43053 Midvale Court
Ashburn, Virginia 20147

Alternatively you can suggest the correction in the KBS eService Center at www.knowledgebasedsys.com/support.

Website: www.knowledgebasedsys.com

Email: info@knowledgebasedsys.com
Audience

This manual is intended for those who are familiar with the basics of SQL and want to use SQL to access data from database systems defined in the KB_SQL data dictionary.
## Syntax Conventions

This manual uses the following syntax conventions when explaining the syntax of a KB_SQL statement.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY WORDS</td>
<td>SELECT</td>
<td>A SQL key word that should be entered exactly as shown. (However, it is not necessary for you to capitalize key words. We do so for identification purposes only).</td>
</tr>
<tr>
<td>lowercase word</td>
<td>table</td>
<td>A language element; substitute a value of the appropriate element type.</td>
</tr>
<tr>
<td>or</td>
<td>table or view</td>
<td>A choice; enter either the item to the left or to the right of the or, but not both. If the or is on a separate line, enter either the line(s) above or the line(s) below.</td>
</tr>
<tr>
<td></td>
<td>LEFT</td>
<td>RIGHT</td>
</tr>
<tr>
<td>}</td>
<td>Column {.column}</td>
<td>The items within the braces form a required composite item. Do not enter the braces.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Table [AS] alias</td>
<td>The item(s) within the brackets form an optional composite item. Including this item may change the meaning of the clause. Do not enter the bracket.</td>
</tr>
<tr>
<td>...</td>
<td>Column 9,.column]...</td>
<td>An ellipsis indicates</td>
</tr>
</tbody>
</table>


that the item which proceeds the ellipsis may be repeated one or more times.

<table>
<thead>
<tr>
<th>(a) or (b) or (c)</th>
<th>ASCII</th>
<th>Character literals composed of one or more characters enclosed within quotes (e.g., 'abc').</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m) or (n)</td>
<td>CHAR(n)</td>
<td>Numeric literals e.g., 123 or 1.23)</td>
</tr>
</tbody>
</table>
Concepts

Alias

An alias provides an alternate runtime name for either a table or expression.

Example

- SELECT * from PROJECT alias-name

Rules

- The alias name must be a valid identifier.
- An expression alias may only be used in the SELECT clause of a SELECT statement. Each expression alias name must be distinct from all other expression aliases and column names used in the query.
- Table aliases may be used in DELETE statements, SELECT statements and UPDATE statements. Each table alias name must be distinct from all other table aliases and table names used in the statement.

Column

A column is a reference to a value in a table. The column may have three parts: a table prefix, a foreign key link, and a column name.

[ [schema.] table.] [foreign_key@]... column_name

The table prefix is composed of a table name (with an optional schema prefix) or a table alias followed by the period character. The table prefix can be omitted if the remainder of the column specification is unambiguous. If the table prefix is used, and a foreign key is used, the foreign key name must be part of the table identified by the table prefix. If the table prefix is used and no foreign key is present, the column name must be part of the table identified by the table prefix.

If you omit the table's prefix, the system assumes that the column or first foreign key refers to the first table (that it finds in the FROM clause) that contains a column or foreign key with the same name. The first table is determined by scanning the FROM clause from left to right, top to bottom.

The foreign key link is optional, but if present it must contain one or more foreign key names separated from each other and the column name by the '@' character. Each foreign key causes a change of context from the current table to the table referenced by the foreign key. The initial context is determined by the table prefix of the first foreign key. If any of the foreign key columns have a null value, or if the foreign key does not find a row in the referenced table that
KBSSQLSyntax

matches, any reference to columns from the table using the foreign key will return null.

The column name must exist in the table referenced by the last foreign key, or by the table prefix if no foreign key is specified, or by one of the tables listed in the FROM clause.

Rules

- If a table or foreign key reference is specified, there cannot be any spaces between the table, the period '.', the foreign key, the '@' character, or the column name. If necessary, a long reference may be broken into multiple lines automatically by the SQL Editor.

Examples

name The NAME column.
employees.name The NAME column from the EMPLOYEES table.
manager_link@name The NAME column from the row identified by the foreign key MANAGER_LINK.

Comments

Comments can be added to queries/drafts. Comments are a valuable tool to assist the user to document the details of the query for readability and maintenance of them in the future. Comments are also valuable for assisting in debugging queries by selectively commenting out parts of the query. For use in debugging refer to the one-plan and two-plan sample queries in Lesson 4: Using the Show Plan Feature of the KB_SQL SQL Reference Guide for examples of using comments.

To add a query you can use either block comments (multiple line) or single line comments.

A block comment starts with a slash star (/*) and the comment does not end until a closing start slash (*/) is encountered.

Example

```
SELECT *
FROM sql_test.employees
WHERE name in ('SMITH', /* this is a long comment that can span multiple lines */
```
A single line comment (--) can occur anywhere in a line, everything in the line after the single line comment is not included in the parsed query. This is only for comments of less than 80 characters in length.

Example

```
SELECT PROJ_NO, PROJECT, LEADER, BUDGET
FROM PROJECTS
-- Restrict results to rows with a budget over 50000
WHERE BUDGET > 50000
```

**Condition**

A combination of one or more tests (predicates), which are evaluated as a unit to either true, false, or unknown. In most situations, unknown is equivalent to false.

Valid conditions are:

- `test`
- `test AND condition`
- `test OR condition`
- `[NOT] (condition)`

**Rules**

- Conditions enclosed within parentheses are evaluated first. If multiple levels of parentheses are used, the innermost level is evaluated first, followed by the remaining levels, with the outermost parentheses evaluated last.
- If two or more tests are combined using the AND key word, the composite is true if all of the tests are true.
- If two or more tests are combined using the OR key word, the composite is true if any of the tests are true.
- The NOT key word is used to reverse the meaning of a condition.
- If both the AND and the OR key words are used together without parentheses, the AND's are evaluated first, and the results are combined using the OR's to produce the final result.
Data Types

A data type specifies the format of a value. Valid data types are:

- **CHARACTER** [(length)]
- **DATE**
- **FLAG**
- **INTEGER** [(length)]
- **MOMENT**
- **NUMERIC** [(length [,scale])]  
- **TEXT**
- **TIME**

Other data types may have been created by your DBA or vendor to facilitate the mapping of your database into the KB_SQL data dictionary. These data types are not available for use within the SQL language.

For a complete list of valid formats for each of the data types please use the DBA OPTIONS/DATA DICTIONARY/REPORTS/OUTPUT FORMAT PRINT option.

Rules

- The **CHARACTER** data type is the most liberal format, allowing any combination of characters. The length refers to the maximum length of the value (in characters).
- The **DATE** data type requires an exact date value, including year, month and day.
- The **FLAG** data type is used for Boolean values that have only two possible states, true or false.
- The **INTEGER** data type is used for whole numbers composed of the digits \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}. Integers may have a leading sign \{+, -\}.
- The **MOMENT** data type requires an exact date and time value including year, month, day, hour, and minute.
- The **NUMERIC** data type is used for real numbers that may contain one or more digits to the right of the decimal point. The numeric data type allows all of the features allowed by the **INTEGER** data type. It also allows a decimal point followed by additional digits. The scale refers to the number of digits allowed to the right of the decimal point.
The TEXT data type is used for very long character values. There is no specific length limitation associated with the TEXT data type, but the optional length value is used as the default output width when the TEXT value is displayed. There are many restrictions on the use of TEXT values. They cannot be used with queries that export data to host files or queries that use methods. They may only be used in the SELECT clause of a query, a WRITE statement, an INSERT statement, a SET clause within the UPDATE statement, with specific functions, and with specific tests within the WHERE clause. The valid functions are EXTRACT, FIND, LENGTH, SQL_FN_LCASE, SQL_FN_UCASE or SQL_FN_SUBSTRING. The valid tests are the IS NULL, IS NOT NULL, equals (=), not equals (<>), or LIKE.

The TIME data type is used for exact times including hours and minutes.

For portability, CHAR may be used in place of CHARACTER, INT and SMALLINT may be used in place of INTEGER, and DEC, DECIMAL, DOUBLE PRECISION, FLOAT, and REAL may be used in place of NUMERIC.

Example

```sql
CREATE TABLE employee_data
(name CHARACTER(30),
hire_date DATE,
full_time FLAG,
years_experience INTEGER(2),
last_review MOMENT,
salary NUMERIC(9,2),
lunch_hour TIME)
```

DECLARE VARIABLE Statement

The DECLARE VARIABLE statement specifies a data type for an M variable.

```sql
DECLARE [VARIABLE] M_variable data_type
[DEFAULT literal] [HEADING literal] [{LEFT|RIGHT|CENTER} [integer]]
```

The declaration statement above specifies a host variable data type. This statement is used to specify the data type of host variables that are either passed into the query or calculated by the query.

By default, KB_SQL will treat all unknown variables as character data types, and all computed variables as the resulting data type based on the date types and operators used in the calculation. This statement overrides the default data type definition.

Rules
• If you use the DEFAULT modifier, the literal you specify is displayed when the value of the variable is null.
• If you use the HEADING modifier, the literal you specify is used as the column's heading.
• If you specify a LEFT, RIGHT, or CENTER modifier, you may also supply an integer for the length of the variable.
• Each variable must be a valid M variable prefixed by the colon character.
• Each variable may be declared only once, and the specified data type will remain in effect for the entire query.
• The DECLARE VARIABLE statement must be positioned before any other statement that references the variable.

Examples

DECLARE :SQLAFD DATE

DECLARE :%ID INTEGER(3)

Expressions

Expressions are composite values combining values and operators.

[-] value [operator value]

(value)

WHEN condition

WHEN condition THEN value

CASE value {WHEN value THEN value}... [ELSE value] END

CASE {WHEN condition THEN value}... [ELSE value] END

Rules

• The effect of a NULL value in an expression depends on how KB_SQL is configured. If your DBA (using the CONFIGURATION/ SITE EDIT/ANSI INFO option) has set the Return null... prompt to YES, any expression that contains a null value will return null for the result of the expression (i.e., the value of an expression that contains an unknown is also unknown). Otherwise, if the prompt is set to NO, the expression will return a result more typical to M, by treating null values as empty strings for character operations or zero for numeric operations.
• If the WHEN condition syntax is used, the result is a FLAG data type. If the condition is true, the result is true (YES); otherwise the result is false
(NO). This expression type is useful for indicators and simple cross-tab reports using the COUNT function.

- If you use the WHEN THEN syntax, and if the condition is true, the result is the value of the value with the data type of value; otherwise the result is null.
- If you use the CASE value syntax, the result is the THEN value following the first WHEN value which is equal to the CASE value. If there are no matches, the result is null. The data type of the result of a CASE expression is always CHARACTER.
- If the CASE WHEN syntax is used, the result is the value following the first true condition. If none of the conditions are true, the result is null. The data type of the result of a CASE expression is always CHARACTER.
- TEXT values may not be used in the CASE expression in the THEN value or ELSE value clauses. This is due to the limitation that the CASE expression can only return a CHARACTER result. To work around this limitation, you can use the EXTRACT function to return the first portion of a TEXT value within the CASE expression. The only limitation is that the length of the extract must not exceed the maximum string length for your M implementation.

**Foreign Key**

A foreign key is a column (or columns) that represents a relationship to another (or same) table. The foreign key can point to a row in another table or to another row in the same table or even to the same row (as the primary key) in the same table.

**Literal**

A constant value. KB_SQL supports six literal types.

<table>
<thead>
<tr>
<th>Literal Types</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER</td>
<td>A string of any characters enclosed in quotes.</td>
</tr>
<tr>
<td>DATE</td>
<td>A date value enclosed in quotes.</td>
</tr>
<tr>
<td>FLAG</td>
<td>A true or false ('YES' or 'NO').</td>
</tr>
<tr>
<td>MOMENT</td>
<td>A date and time value enclosed in quotes.</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>A numeric value, with an optional sign (-) or decimal point.</td>
</tr>
<tr>
<td>TIME</td>
<td>A time value enclosed in quotes.</td>
</tr>
</tbody>
</table>

**Rules**
KBSSQLSyntax

- A numeric literal must be composed of only digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} with an optional leading sign {+, -} and a decimal point.
- All non-numeric literals must be enclosed within quotation marks. Either single or double quotation marks may be used to enclose literals, however the same type of quote character should be used on both ends of the literal.
- Quotation marks may be embedded within literals by using two quotation marks together, without any spaces or other characters between them, for each embedded quote character.
- The default type for any literal enclosed within quotes is CHARACTER. Character literals are evaluated as DATE, FLAG, MOMENT, or TIME literals only when they are compared to another value of the corresponding type.
- DATE literals must be a valid date format.
- FLAG literals must be either 'YES' or 'NO'.
- MOMENT literals must be in a valid date-times format.
- TIME literals must be in a valid time format.

Examples

'ABC 123' A character literal.
'12/31/91' A date literal.
'YES' A flag literal
'12/31/91@11:59  PM' A moment literal.
-1.23 A numeric literal.
'11:59  PM' A time literal.

Using Quotes with Literals

You may use either single or double quotes to enclose a literal. You may also embed single or double quotes within a literal, but to do so you must use two of the same type of quote. (See examples below.) The double quote character is not ANSI standard SQL and is converted internally to the single quote character, except when it is embedded within a literal.

Single quote examples

'abcdef' produces the literal: abcdef
'abc''def' produces the literal: abc'def
'abc''d''ef' produces the literal: abc''d''ef
'abc''''''def' produces the literal: abc''''''def

Double quote examples

"abcdef" produces the literal: abcdef
"abc""def" produces the literal: abc"def
"abc""d""ef" produces the literal: abc"d"ef
"abc""""def" produces the literal: abc""""def
M Value

A reference to an M value.

RULES:

- The M variable should be a simple variable composed of a name without subscripts.

EXAMPLE:

```
DECLARE :$P($H","",1) DATE
SELECT name, last_edit_date
FROM data_dictionary.table
WHERE last_edit_date > :$P($H","",1)-14
```

M Variable

An M variable.

RULES:

- The M variable should be a simple variable composed of a name without subscripts.

EXAMPLES:

```
:SQLAX
:ABC1

SET :ABC1= SALARY
```

Method

A way to transfer data between systems.

RULES:

- There are two types of methods: import and export.
KBSSQLSyntax

- Import methods are used by the INSERT statement to load data into KB_SQL tables (upload).
- Export methods are used to transfer the result of a query (SELECT) to another system (download).
- All methods must be created by your DBA using the EXPORT METHOD EDIT and IMPORT METHOD EDIT options from the CONFIGURATION menu. The DBA can also designate default methods for your site using the SITE EDIT option from the CONFIGURATION menu. To use the default import method, simply do not include the METHOD clause in the INSERT syntax. To use the default export method, do not include the METHOD parameter before the SELECT statement.
- Import methods may be used with COMMA, DELIMITED, DIF, and TAB formats.

As an alternative to transferring data, you may want to consider using KB_SQL's ODBC feature which lets you use Windows applications directly to access your M data. The KB_SQL ODBC Driver Install Guide gives you instructions for making this possible.

Examples

```
INSERT INTO budget (year, month, department, amount)
FROM 'C:\EXCEL\BUDGET.DAT' FORMAT COMMA
METHOD PROCOMM

SET FORMAT = TAB, METHOD = PROCOMM
SELECT name, salary, manager
FROM employees
```

Names

Names are used to identify data dictionary objects. You need to assign names for the following object types: aliases, columns, data types, export methods, foreign keys, formats, functions, import methods, indices, pseudo columns, queries, schemas, tables, user groups, and views.

It is wise to keep names as short as possible. We suggest you establish and use standard abbreviations (PAT for PATIENT). When necessary, shorten words by removing the vowels (A, E, I, O & U), e.g., DATE becomes DT, NAME becomes NM. Avoid redundant prefixes, for example, in the PATIENT table use NAME and SEX instead of PATIENT_NAME and PATIENT_SEX.

Rules
A name must start with a letter followed by letters, numbers, or underscore characters [\_].
- The minimum length of a name is one letter.
- The maximum length of a name is 30 characters.
- For portability, limit the length of the name to 18 characters.
- Names must be different from all KB_SQL key words.
- In names, lowercase letters are equivalent to uppercase letters.

**Examples**

- projects
- emp_name
- addr1
- month_12
- total_3qtr

**Primary Key**

A primary key is a group of one or more columns that uniquely identifies a row in a table.

**READ**

Provides prompts for user-specified input prior to running a query.

The READ statement allows you to create prompts that are displayed to the user each time the query is run. Prompts are most often used to allow the user to specify search constraints. This allows a single query to be used for a variety of searches without editing or compiling the query.

**READ variable data_type [PROMPT literal] [DEFAULT literal] [lookup-specification] [HINT literal] [variable data_type [PROMPT literal] [DEFAULT literal] [lookup-specification] ]...**

\[
\text{[lookup-specification]}::= \text{LOOKUP} (\text{[table-lookup]} | \text{[procedure-lookup]})
\]

\[
\text{[table-lookup]}::= \text{TABLE} [\text{[schema_name]}.]\text{table_name} . \text{column_name}
\]

\[
\text{[procedure-lookup]}::= \text{PROCEDURE} [\text{[schema_name]}.]\text{procedure_name.procedure_column_name} (\text{[procedure-parameters]})
\]

**Rules**
Each read variable will create a separate prompt. All of the prompts are asked each time the query is run.
The length of the input area after the prompt will depend on the data type associated with the variable.
The cursor movement keys [up], [down], [left], and [right] can be used to move between prompts.
A query may not have more than nine read variables.
The PROMPT clause is used to specify the text that is displayed to the user. If no PROMPT clause is specified, KB_SQL will use the variable name as the prompt.
The DEFAULT clause is used to specify a default value for the prompt. If the user does not enter a new value for the prompt, the default value is used.
If a READ statement references a variable name, the variable name must conform to the logic established by the DBA in the CONFIGURATION/CUSTOM LOGIC/VARIABLE CHECK option.
If specified using the HINT clause, the [help] key will display the text. Otherwise, the [help] key will provide a domain specific suggestion for input.
If specified as a LOOKUP TABLE, the [list] key will display a list of values according to a query using the following template:

```
SELECT DISTINCT column_name FROM table_name WHERE column_name IS NOT NULL ORDER BY column_name
```

If specified as a LOOKUP PROCEDURE, the [list] key will display a list of values according to a query using the following template:

```
{CALL procedure_name(procedure_parameters)}
```

The procedure_parameters can include values of the following types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>string_literal</td>
<td>'hello, world'</td>
</tr>
<tr>
<td>numeric_literal</td>
<td>12345</td>
</tr>
<tr>
<td>host_expression</td>
<td>:XABC</td>
</tr>
<tr>
<td>pseudo_column</td>
<td>TODAY</td>
</tr>
</tbody>
</table>

Examples

**Using READ with PROMPT and DEFAULT**

```sql
READ :SQLAP INTEGER(3) PROMPT 'Project number'
READ :SQLAS CHARACTER(12) PROMPT 'Select status'
DEFAULT 'INCOMPLETE'
SELECT task_no, task
FROM tasks
WHERE proj_no = :SQLAP AND status = :SQLAS
```

**Using LOOKUP PROCEDURE**

```sql
READ :SQLAP INTEGER(3) PROMPT 'Project number'
```
LOOKUP PROCEDURE

sql_test.demo_sp_projects.project_number()
READ :SQLAS CHARACTER(12) PROMPT 'Select status'
DEFAULT 'INCOMPLETE'
SELECT task_no, task
FROM tasks
WHERE proj_no = :SQLAP AND status = :SQLAS

Note: The stored procedure is defined as follows:
-- DEMO_SP_PROJECTS
SET STORED_PROCEDURE ON
SET SCHEMA = SQL_TEST
SELECT PROJ_NO AS PROJECT_NUMBER
, PROJECT AS PROJECT_NAME
FROM SQL_TEST.PROJECTS
WHERE EXISTS (SELECT * FROM TASKS WHERE
TASKS.PROJ_NO = PROJECTS.PROJ_NO)
ORDER BY PROJ_NO

Use LOOKUP PROCEDURE whenever you want to add additional
display columns or additional row selection logic.

Using LOOKUP TABLE

READ :SQLAP INTEGER(3) PROMPT 'Project number'
LOOKUP TABLE sql_test.projects.proj_no
READ :SQLAS CHARACTER(12) PROMPT 'Select status'
DEFAULT 'INCOMPLETE'
LOOKUP TABLE sql_test.tasks.status
SELECT task_no, task
FROM tasks
WHERE proj_no = :SQLAP AND status = :SQLAS

Use LOOKUP TABLE for simple lists where no additional display
columns or row selection logic is required.

Using HINT

READ :SQLAP INTEGER(3) PROMPT 'Project number'
HINT 'Enter a valid 3-digit project number'
READ :SQLAS CHARACTER(12) PROMPT 'Select status'
HINT 'Enter a valid status description'
DEFAULT 'INCOMPLETE'
SELECT task_no, task
FROM tasks
WHERE proj_no = :SQLAP AND status = :SQLAS
SET

SET variable = expression [,variable = expression]...
Sets an M variable equal to an expression.

Rules
• All column references must refer to tables contained in the FROM clause of the SELECT statement.

Subquery

A clause that retrieves one column of information from one or more tables. Subqueries are usually used in conditions.

SELECT [DISTINCT] {value [AS alias]}
[WHERE condition]
[GROUP BY expression [, expression]...]
[HAVING condition]

Rules
• The result of a subquery is a single value or a single column of values.
• See the SELECT statement for additional rules.

Examples
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees)

SELECT name, salary
FROM employees a
WHERE salary > (SELECT AVG(b.salary)
FROM employees b
WHERE a.manager = b.manager)

Table

The name of a table in the data dictionary.

An index is a special kind of table that is linked to another table and provides an alternate path to that table.

[schema.] table_name
Rules

- The table name must be different from all other table, index, and view names in the same schema.
- If the schema prefix is not specified, the table name must either exist in your default schema, or be unique across all schemas.

Examples

```sql
sql_test.employees
employees
```

Tests

Tests provide information to assist in determining how the query will treat certain values.

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Flag (true/false)</td>
</tr>
<tr>
<td>value = value</td>
<td>Equal</td>
</tr>
<tr>
<td>value [ value</td>
<td>Contains test. The value before the [ contains the value after the [</td>
</tr>
<tr>
<td>value &lt;&gt; value</td>
<td>Not equal</td>
</tr>
<tr>
<td>value &lt; value</td>
<td>Less than</td>
</tr>
<tr>
<td>value &lt;= value</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>value &gt; value</td>
<td>Greater than</td>
</tr>
<tr>
<td>value &gt;= value</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>value [NOT] BETWEEN value AND value</td>
<td>Inclusive (exclusive) range check</td>
</tr>
<tr>
<td>value = [ANY] subquery</td>
<td>Equal to subquery result</td>
</tr>
<tr>
<td>value &lt;&gt; [ANY] subquery</td>
<td>Not equal to subquery result</td>
</tr>
<tr>
<td>value &lt; [ANY] subquery</td>
<td>Less than subquery result</td>
</tr>
<tr>
<td>value &lt;= [ANY] subquery</td>
<td>Less than or equal to subquery result</td>
</tr>
<tr>
<td>value &gt; [ANY] subquery</td>
<td>Greater than subquery result</td>
</tr>
<tr>
<td>value &gt;= [ANY] subquery</td>
<td>Greater than or equal to subquery result</td>
</tr>
<tr>
<td>value [NOT] IN (subquery)</td>
<td>Found (not found) in subquery result</td>
</tr>
<tr>
<td>value [NOT] IN (value [,value]...)</td>
<td>Found (not found) in value list</td>
</tr>
<tr>
<td>value [NOT] LIKE pattern</td>
<td>Match pattern. Pattern can include [ ] for any character, and [%] for any string of characters</td>
</tr>
<tr>
<td>value IS [NOT] NULL</td>
<td>Null value check</td>
</tr>
</tbody>
</table>
Rules

- The result of any test is either true, false, or unknown.
- If a value is used as a test, the value must have a data type associated with the FLAG data type.
- The data types of all values referenced by a test must be comparable.
- If any value referenced by any test, other than the NULL test, is null, then the result of the test is false (unknown).
- If the NOT key word is used, the conditions for true and false are reversed. The unknown value is not changed by the NOT key word.
- When comparing a value to a subquery result, list the value first, followed by the test, followed by the subquery. Listing the subquery first is not supported by the standard syntax and will not work.
- If the key word ANY is specified, and the test is true for at least one result row of the subquery, then the test is true. Otherwise, the test is false. Note: The key word SOME may be used in place of the key word ANY.
- If the key word ALL is specified, or if the test references a subquery, and the test is true for all rows returned by the subquery, then the test is true. Otherwise, the test is false.
- If the BETWEEN test is used, the first value after the key word is the lower limit, and the second value is the upper limit. A value that is greater than or equal to the lower limit, and less than or equal to the upper limit, is considered to be between those values.
- If the IN test is used, and a subquery is specified, and the value is equal to any result row of the subquery, then the test is true. Otherwise the test is false.
- If the IN test is used, and a value list is specified, and the value is equal to a value in the list, then the test is true. Otherwise, the test is false.
- If the NULL test is used, and the value is null, then the test is true. Otherwise, the test is false.
- If the EXISTS test is used, and the subquery returns at least one result row, then the test is true. Otherwise, the test is false.
- If the LIKE test is used, each underscore character '_' shall represent an arbitrary character, and each percent character '%' shall represent an arbitrary string of characters. If all other characters match exactly, then the test is true. Otherwise, the test is false.
- The percent and underscore characters can be used as literals in a LIKE test by preceding them with an escape character. The default LIKE escape character is '\'. The escape character must be specified using the escape clause syntax: escape '\'

Examples

```
SELECT key_word
FROM sql_key_word
```
WHERE key_word
LIKE 'sql\_%' escape '\'
literal. All key words that begin with
SQL_ will be selected.

SELECT name, visit_date
FROM patients
Where pregnant

SELECT name, salary
FROM employees
WHERE salary > 10

SELECT name, salary
FROM employees
WHERE salary <= 10

SELECT name, salary
FROM employees
WHERE salary > ANY
(SELECT salary
FROM employees
WHERE manager =
'416-82-6316')
True if the salary is larger than the
salary of anyone reporting to the
manager '416-82-6316'.

SELECT name, salary
FROM employees
WHERE salary > ALL
(SELECT salary
FROM employees
Where manager =
'416-82-6316')
True if the salary is higher than
everyone working for the manager
'416-82-6316'

SELECT name, salary
FROM employees
WHERE salary BETWEEN 8 AND
11

SELECT task, status
FROM tasks
WHERE proj_no IN (100,200)
True if the proj_no is 100 or 200.

SELECT task, status
FROM tasks
WHERE task_no NOT IN
(1,2,3)
True if task_no is not 1, 2 or 3.

SELECT task, status
True if the name starts with 'SM'.
KBSQLSyntax

FROM tasks
WHERE name LIKE 'SM%'

SELECT name, salary
FROM employees
WHERE name NOT LIKE 'S____'

SELECT name, salary
FROM employees
WHERE manager IS NULL

SELECT project
FROM projects
WHERE EXISTS
  (SELECT task FROM tasks
   WHERE projects.proj_no=
   tasks.proj_no
   AND status='INCOMPLETE')

SELECT name, salary
FROM employees
WHERE salary > 10
AND manager IS NOT NULL

SELECT name, salary
FROM employees
WHERE salary > 10
OR manager IS NOT NULL

SELECT project
FROM projects
WHERE status='INCOMPLETE'
OR proj_no = 100
AND budget > 40000

SELECT project
FROM projects
WHERE (status='INCOMPLETE'
OR proj_no = 100)
AND budget > 40000

A test that compares columns from more than one table is called a join. A join that uses the = operator in the WHERE clause is called an equijoin. For more information on joins, please refer to Lesson 2 in the KB_SQL SQL Reference Guide.
Values

A scalar entity. A value may be either a column, expression, function, literal, M expression, M variable, or pseudo column.

View

The name of a virtual table in the data dictionary.

[schema.] view_name

Rules

- The view name must be different from all other table, index, and view names in the same schema.
- If the view name exists in your default schema, or if the view name is different from all other table, index, and view names, then the schema prefix may be omitted.

Examples

proj_tasks

sql_test.proj_tasks

Wild Cards

By using wild cards you can specify a set of tables and queries on which to grant or revoke privileges. Valid wild cards include the match character (*), the range character (:), and the delete character (-).

Wild cards can be specified in multiples, separating set specifications from each other with the comma character (,).

Wild cards operate on the set of objects that the current user can access. In other words, the '*' for all queries will return only the set of queries that the current user may select. For DBAs, the '*' for queries means 'all' queries.

Examples

GRANT ALL on sql_test.* to users
(Grants to the users user group all privileges to tables beginning with sql_test.)

REVOKE ALL on a:z, -m to managers
(Revokes privileges from the managers user group for all tables 'a' through 'z' except table 'm'.)

Column Modifiers

Column modifiers control the display format of an expression in a SELECT or WRITE clause.

Rules

• Modifiers may be used after an expression in the SELECT clause or a WRITE statement.
• The CHANGED modifier cannot be used with TEXT data type values.
• If you specify a value to alter the column width (using the CENTER, LEFT, RIGHT, WRAP, or WORDWRAP modifiers), but the column header is longer than the width you specified, KB_SQL will default to the width of the column header.
• The HEADING modifier may be used only in the SELECT clause, or in a WRITE statement in the DETAIL, BREAK AT, BREAK AFTER, or FINAL event blocks. If the HEADING text is wider than the column value, the column width is expanded to the length of the heading.
• The modifiers CENTER, LEFT, and RIGHT are mutually exclusive; only one of the three modifiers may be used on a value.
• The modifiers COLUMN and CSPACE are mutually exclusive; only one of the modifiers may be used on a value.
• The modifiers WORDWRAP and WRAP are mutually exclusive; only one of the modifiers may be used on a value.
• If the SKIP modifier causes a new page, any remaining skip lines are ignored, and the value is printed on the first available line on the new page.
• The WRAP and WORDWRAP modifiers may be used only in the SELECT clause or in a WRITE statement. If you supply a value, you must also use either the CENTER, LEFT, or RIGHT modifier.
• You cannot use WRAP or WORDWRAP in INITIAL, FINAL, HEADER, or FOOTER event blocks.

IMPORTANT: When using the CHANGED modifier, you need to apply it to the column you want to affect and all columns to the left of that column in order to obtain the expected results. We suggest that if you do use the CHANGED modifier for any column that may have null values, you also use the DEFAULT modifier so you can distinguish between a null value and a duplicate value.

Example

SELECT manager CHANGED DEFAULT 'NONE',
name COLUMN 20,
salary RIGHT 10 HEADING '$'
FROM employees
ORDER BY 1,2

**BASE**
Displays the output of the expression in its base format. KB_SQL uses a specific set of base or internal formats for data.

**CENTER**

**CENTER [integer]**
Centers and truncates the value in a column integer wide.

**CHANGED**
Displays the value only when it is different from the value on the previous row.

**COLUMN**

**COLUMN integer**
Displays the value at tab position integer.

**CSPACE**

**CSPACE integer**
Overrides the default spacing between display columns. The value integer represents the number of spaces to insert between the previous value and the current value.

**DEFAULT**

**DEFAULT literal**
Displays the literal for NULL values.

**FORMAT**

**FORMAT literal**
KBSSQLSyntax

A type of host file format used by the INSERT (file read) and SELECT (file write) statements. KB_SQL supports the following file formats COMMA, DELIMITED, DIF, FIXED, HTML_TABLE, HTML_TEXT, MAPPER, REPORT and TAB.

Rules

- TEXT data type values cannot be used in file import or export operations.
- The COMMA format reads/writes files as unpadded columns separated by the comma character (,). If the data within a column is not numeric the value is enclosed in double quotation marks (").
- The DELIMITED format reads/writes files as columns separated by the system delimiter character (which the DBA specifies using the SITE EDIT/DEFAULT INFORMATION option) or as columns specified by the DELIMITER parameter (refer to the "Parameters Within a Set Statement" entry).
- The DIF format reads/writes a valid DIF format file. The format of a DIF file is beyond the scope of this document.
- The FIXED format writes files as character values left justified and space padded, and numeric values right justified and space padded.
- The HTML_TABLE format writes files that contain the query results in an html table that can be viewed by a browser. This format should be used for simple query results that retrieve well defined two dimensional table results, for example a select statement without event blocks.
- The HTML_TEXT format writes files that contain the query results in html text format that can be viewed by a browser. This format should be used for complex reports that contain event blocks with WRITE statements or complex multi-line formatting.
- The MAPPER format writes files as columns separated by the tab character ($C(9)). Above each column it writes a two-line header. The maximum line width is 130 characters.
- The REPORT format writes files in the same format that would be used if the results were printed on paper. The REPORT format may be used only when writing to host files; it cannot be used with export methods.
- The TAB format reads/writes files as columns separated by the tab character ($C(9)).
- The FIXED, HTML_TABLE, HTML_TEXT, MAPPER, and REPORT formats cannot be used for read operations.
- The maximum length of a row for the COMMA, DELIMITER, FIXED, HTML_TEXT, MAPPER, REPORT and TAB formats is limited by the maximum length of a M variable.

Examples

```
SET FILE='C:\DOWNLOAD\DATA.1', FORMAT=TAB
SELECT name, salary, manager
FROM employees

INSERT INTO budget (dept, year, month, amount)
FROM FILE 'C:\UPLOAD\DATA.2' FORMAT = TAB
```
HEADING

HEADING literal

Uses the literal for the column header. To create a multi-line header, include the vertical bar character (|) in the literal.

LEFT

LEFT [integer]

Left justifies and truncates the value in a column integer wide.

PAGE

PAGE [integer]

Starts a new page before displaying the value.

or

Starts a new page if less than integer lines remain on the current page.

RIGHT

RIGHT [integer]

Right justifies and truncates the value in a column integer wide.

SKIP

SKIP [integer]

Displays integer blank lines before displaying the value.

WORDWRAP

WORDWRAP [integer]

Displays the value of the expression in one or more lines, breaking the lines at a space character instead of the column width. You can alter the width of the column by specifying a value for integer.
KBSSQLSyntax

WRAP

WRAP [integer]

Displays the value of the expression in one or more lines, with the length of each line no greater than the column width. You can alter the width of the column by specifying a value for integer.

Event Blocks

Event blocks specify a group of procedural statements that are to be executed when a particular event occurs. Each event block is composed of the event name, followed in most situations by one or more statements. In certain situations event blocks may be used without statements, usually to suppress the default formatting.

**Event blocks may be used only in conjunction with a SELECT statement. However, they can not be used when the SELECT statement is used as an argument to another function.**

Examples

```sql
SELECT TASK COLUMN 5, TASK_NO HEADING '#'
FROM PROJECTS, TASKS
WHERE PROJECTS.PROJ_NO = TASKS.PROJ_NO
 AND STATUS = 'INCOMPLETE'
ORDER BY PROJECT, TASK

HEADER
   WRITE 'Incomplete Task List' CENTER 60
FOOTER
   WRITE 'For internal use only' CENTER 60

BREAK AT PROJECT NEWPAGE
   WRITE PROJECT, 'Leader: ' || LEADER NEWLINE
BREAK AFTER PROJECT
   WRITE COUNT(* BY PROJECT) ||
   ' Tasks' COLUMN 5

FINAL
   WRITE COUNT(*) || ' Incomplete Tasks'
```

**This report will start a new page for each project and print a blank line before printing the detail lines.**
SELECT null
FROM employees
DETAIL
IF salary < 10
    SET :SQLATYPE='low'
ELSEIF salary < 15
    SET :SQLATYPE='mid'
ELSE
    SET :SQLATYPE='high'
ENDIF
WRITE name, salary, :SQLATYPE

SELECT name, salary
FROM employees
ORDER BY manager, name
BREAK AT 1
    WRITE manager
BREAK AFTER manager
    WRITE 'Total salary ',SUM(salary BY 1)
FINAL
    WRITE 'Total salary for all managers ',
        SUM(salary BY 0)

The use of '1' or manager in the example is equivalent.

FOOTER
WRITE 'For Internal Use Only' CENTER 80

HEADER
WRITE 'Employees Grouped by Job Title' CENTER 80
WRITE 'Printed ' | TODAY | ' at ' | NOW CENTER 80

INITIAL
SET :SQLAT=0, :SQLAS=0

Conditional Statements

Conditional statements specify the action to be performed each time the event occurs. They provide conditional logic.

IF condition
    event_statement...
[ELSEIF condition
KBSSQLSyntax

```sql
  event_statement[]{...}
  [ELSE
    event_statement[]{...}
  ]
  ENDIF
```

Rules
- All column references must refer to tables contained in the FROM clause of the SELECT statement.
- Each IF must have a corresponding ENDIF.
- If the condition following the IF key word is true, the statements following the condition are executed. When the corresponding ELSEIF or ELSE key word is encountered, execution will stop and resume after the corresponding ENDIF key word.
- If the condition following the IF key word is false, the statements following the condition are skipped. If any corresponding ELSEIF statements exist, and any ELSEIF condition is true, those statements following the first true condition are executed, and all other ELSEIF and ELSE statements are skipped.
- If the IF and ELSEIF conditions are all false, and if an ELSE key word is encountered, the statements following the ELSE are executed.
- If an ELSE key word is present, it must follow all of the ELSEIF key words.

BREAK AFTER

**BREAK AFTER expression**

```sql
  event_statement{...}
```

An event that occurs after each different ORDER BY value is processed. This event is usually used to print subtotals for ORDER BY keys. It can also be used with the NEWLINE and NEWPAGE statements to improve readability.

Rules
- The BREAK AFTER event may be used only with an ORDER BY clause.
- In some cases, KB_SQL will inform you that it can't match the BREAK AFTER event to the ORDER BY clause. For example, this may occur with some expressions and virtual columns. When this does happen we suggest you use a positional reference to the ORDER BY clause.
- The expression referenced by the BREAK AFTER must be an exact match to an expression from the ORDER BY clause, or an integer that corresponds to the position of an expression in the ORDER BY clause.
BREAK AT

BREAK AT expression
    event_statement...

An event that occurs before each different ORDER BY value has been processed. This event is usually used to print subheaders for ORDER BY keys.

Rules
- The BREAK AT event may be used only with an ORDER BY clause.
- In some cases, KB_SQL will inform you that it can't match the BREAK AT event to the ORDER BY clause. For example, this may occur with some expressions and virtual columns. When this does happen we suggest you use a positional reference to the ORDER BY clause.
- The expression (order by key) referenced by the BREAK AT must be an exact match to an expression from the ORDER BY clause, or an integer that corresponds to the position of an expression in the ORDER BY clause.

DETAIL

DETAIL
    [event_statement...]

An event that occurs once for each result row. This event is used to provide more complex logic than is allowed in the SELECT clause for each result row.

Rules
- The detail logic will override any display logic specified by the SELECT clause.
- A DETAIL event block without statements will suppress output of the expressions in the SELECT clause.

FINAL

FINAL
    event_statement...

An event that occurs after the last result row has been processed. This event is usually used to print report totals.

Rules
- The final logic may reference only values that will exist after all result rows have been processed; for example, aggregate functions for the entire query, variables, and literals.
• The FINAL event may be used only for queries that contain an ORDER BY clause.

FOOTER

event_statement...

An event that occurs at the bottom of each page.

Rules
• You may not reference any column values in the FOOTER logic.

HEADER

[event_statement...]

An event that occurs at the top of each page. This event is used to override the default header.

Rules
• A HEADER event block with no statements will suppress the default header.

INITIAL

event_statement...

An event used to perform logic before the first result row is processed.

Rules
• The INITIAL event block may not reference any column values.

NEWLINE

NEWLINE [n]

Forces a new line (or n new lines).

NEWPAGE

NEWPAGE [n]
Forces a new page. If n is specified, a new page is forced if less than n lines remain on the current page.

RUN

RUN query [,query]...

The RUN statement executes one or more stored queries. If you are running subqueries from within an event block, refer to the Event Blocks entry.

Rules
A query that contains a RUN statement may not contain any other commands other than DECLARE, READ, and SET.

Each RUN statement may execute one or more queries (i.e., RUN A, B, C). If a query encounters an error, all remaining queries for that command will be skipped. If another RUN statement follows the statement that encountered an error, its queries will be executed.

Examples
READ :SQLAF DATE PROMPT 'From date'
READ :SQLAT DATE PROMPT 'Thru date'
RUN stats_delete
RUN stats_insert, stats_report

RUN A, B, C
RUN D

⚠️ If query B returns an error, query C will be skipped because it is part of the same RUN statement, but query D will run because it is part of the next RUN statement.

SET

SET parameter = value [, parameter = value]...

A SET statement specifies an override value for a parameter. If you are setting M variables within an event block, refer to the "Event Statements" entry also.

Rules
KBSSQLSyntax

- All SET commands must be specified before any other statement other than the READ or DECLARE statements.
- The types of values allowed depends on the parameter that is specified.
- If a SET statement references a variable name, the variable name must conform to the logic established by the DBA in the CONFIGURATION/CUSTOM LOGIC/VARIABLE CHECK option.

WRITE

WRITE value [modifier]... [, value [modifier]]...

Writes the specified values.

Rules
- All column references must be to tables contained in the FROM clause of the SELECT statement.
- The HEADING modifier may be used in a WRITE statement in the DETAIL, BREAK AT, BREAK AFTER, or FINAL event blocks. If the HEADING text is wider than the column value, the column width is expanded to the length of the heading.
- All rules related to the use of modifiers in the SELECT clause also apply to values in the WRITE statement.

SET Statement

Rules
- All parameters must be set at the top of the query, before any commands that reference the parameter values.
- The DELIMITER parameter may be used only when the FORMAT parameter is set to DELIMITED.
- When using the START_DATE and START_TIME parameters, the user can specify both a date and time value to control the time when the query is started. Note that some custom start dates will work with the user input to accomplish the initial schedule. For example, the EVERYHOUR start date rule will default to the top of the hour if nothing is specified by the user. In other cases, when the time is not specified, the default of 12:00 AM is used.
- The SEARCH_LIMIT and SELECT_LIMIT are used during the query design process. The limits on rows searched and selected are designed for use when developing queries. The search limit is often used when the designer is trying to evaluate the query to see if it returns anything at all. Sometimes, a limit of rows will avoid having to wait a long time for the query to find nothing. The select limit is often used when working on
Concepts

formatting of report output. In these cases, the designer is trying to get the columns to line up, and doesn't need more than a handful of rows. If both values are used together, the first limit exceeded will terminate the query.

- Any parameter that accepts YES/NO as a value may also accept ON/OFF. When ON/OFF is used, the equal sign '=' is omitted.
- The REPORT format may not be used with the INSERT ... FROM file statement.
- The REPORT format may be used only for downloads that perform Open/Write commands.

Examples

```
SET SHOW_PLAN = 'YES'
Or
SET SHOW_PLAN ON

SET FILE='C:\DATA.DAT',
FORMAT='TAB'

SET SEARCH_LIMIT = 1000

SET RMARGIN = 132

SET FORMAT = DELIMITED
SET DELIMITER = 59

SET FILE_TYPE=DELIMITED
SET DELIMITER = '$C(1)'
```

**BMARGIN**

SET BMARGIN = integer

The number of blank lines to reserve at the bottom of the page (integer).

**CSPACE**

SET CSPACE = integer

The number of blank characters to display between columns (integer).

**DELIMITER**
KBSSQLSyntax

SET DELIMITER = integer or literal

Specifies the delimiter for the DELIMITED file format (integer value or 'C(value,...)').

DEVICE

SET DEVICE = literal

The name of a device that will print the query results (literal).

```
! The device name must be defined in the list of KB_SQL logical device names.
```

```
! This feature is available only to customers using the KB_SQL logical device definitions for output. If the customer uses custom device selection, this feature is not supported.
```

DISPLAY_END

SET DEVICE = yes/no

Displays "End>" message after printing query (YES/NO).

DISPLAY_HEADING

SET DISPLAY_HEADING = yes/no

Displays default column headings (YES/NO).

DISPLAY_LINE

SET DISPLAY_LINE = yes/no

Displays dashed line between the heading and the data (YES/NO).

DISPLAY_PAGE

SET DISPLAY_PAGE = yes/no

Displays the page number (YES/NO).
FILE

SET FILE = literal

A file is used to specify the name of a host operating system file including drive and directory.

Note: The valid format for a file depends on the particular host operating system and site-specific restrictions. Please contact your DBA for more information.

Rules
- The file name must be a valid file name for the intended operating system.
- The file name must be enclosed in single quotation marks.
- You cannot export TEXT values to or from host files. To work around this limitation while exporting, you can use the EXTRACT function to return the first portion of a TEXT value within the CASE expression. The only limitation is that the length of the extract must not exceed the maximum string length for your M implementation.

Examples
```
SET FILE = 'C:\DOWNLOAD\DATA.1'
SET FORMAT = 'COMMA'
SELECT task_no, emp_ssn, SUM(hours)
FROM charges
WHERE proj_no = 100

INSERT INTO budget (dept, year, month, amount)
FROM FILE 'C:\UPLOAD\DATA.2' FORMAT TAB
```

FORMAT

SET FORMAT = literal

Changes the default output format for the expression without altering its value.

Available Formats
- REPORT
- DELIMITED
- COMMA
- TAB
- FIXED
- DIF
- XML

HALT_CHECK

SET HALT_CHECK = integer
Number of rows to process before checking if the query has been halted (integer).

**INITIAL_ROUTINE**

**SET INITIAL_ROUTINE = literal**

An M routine to be executed each time the query is run. This routine is executed in the foreground after the device/file name prompt and just before any 'READ' prompts. If the screen is effected by the routine, the SQLWIN variable must be set to null to refresh the screen.

**ISOLATION_LEVEL**

**SET ISOLATION_LEVEL = 1, 2, or 3**

Sets transaction isolation level to 0 (read uncommitted), 1 (read committed) or 2 (repeatable read).

**LMARGIN**

**SET LMARGIN = integer**

The number of blank characters to reserve at the left edge of the page (integer).

**MAX_KEYS**

**SET MAX_KEYS = integer**

The maximum number of group by, order by, or primary keys in use before the alternate global structure is used (integer). Some queries involving many primary keys or very long key values may exceed the maximum global subscript length set by your M implementation. This modifier allows you to establish a limit on the number of values and primary keys that will be combined into a single global subscript structure. If the number of values exceeds the limit, an alternate global structure, which looks at the keys or values as a two piece structure, is used.

**METHOD**

**SET METHOD = literal**

A method for transferring (exporting) data. (literal).

**MULTIPLE_UPDATES**
SET MULTIPLE_UPDATES = yes/no

Allows the same row to updated more than once if a previous update causes the row to be selected again by the same UPDATE/WHERE clause. (flag).

PAGE_LENGTH

SET PAGE_LENGTH = integer

Sets number of lines per page regardless of the device type used for output (integer).

PARAMETERS

SET PARAMETERS = literal(,literal...)

A list of parameter names separated by commas. (literal).

POST_RUN_ROUTINE

SET POST_RUN_ROUTINE = literal

An M routine to be executed after the query is run. This routine is executed after the FINAL event.

PRE_RUN_ROUTINE

SET PRE_RUN_ROUTINE = literal

An M routine to be executed before the query is run. This routine is executed after any 'READ' prompts, but before the INITIAL event.

RESULT_LIMIT

SET RESULT_LIMIT = integer

Limits the number of result rows produced by the query (integer). This parameter can be used in conjunction with the GROUP BY or ORDER [,scope] clauses to produce reports that display the "top five" or "lowest ten" result rows.

RMARGIN

SET RMARGIN = integer

The maximum number of characters to display on a line (integer).
KBSSQLSyntax

ROW COMMIT

SET ROW_COMMIT = on/off

If set to YES, this parameter causes the INSERT, UPDATE, and DELETE commands to reject individual rows and continue to process successful rows. Otherwise, if set to NO, all rows must succeed or the entire transaction is rejected. It defaults to site level setting.

SCHEMA

SET SCHEMA = literal

The default schema for the query (schema_name).

SEARCH_LIMIT

SET SEARCH_LIMIT = integer

The maximum number of rows to search (integer). The minimum value is 2.

SELECT_LIMIT

SET SELECT_LIMIT = integer

The maximum number of rows to select (integer). The minimum value is 2.

SHOW_ALL_PLANS

SET SHOW_ALL_PLANS = yes/no

Shows all plans considered by the optimizer (YES/NO).

SHOW_COST

SET SHOW_COST = yes/no

Shows the estimated cost of running the query (YES/NO).

SHOW_PLAN

SET SHOW_PLAN = yes/no

Shows the data access plan (YES/NO).

SHOW_STATS
SET SHOW_STATS = yes/no
Shows search/select statistics at the end of query (YES/NO).

START_DATE

SET START_DATE = literal
Sets a start date (date_literal). You may also specify a day of the week, for example 'MONDAY' to run the query on every Monday, or 'ALL' to run the query every day.

START_TIME

SET START_TIME = literal
Sets a start time (time_literal).

STORED_PROCEDURE

SET STORED_PROCEDURE = yes/no
Identifies the query as a stored procedure. (flag).

SUBQUERY

SET SUBQUERY = yes/no
Specifies that the query can be used by the RUN event statement (YES/NO).

TITLE

SET TITLE = literal
Provides an alternative to the query name for the default query header. (literal).

TMARGIN

SET TMARGIN = integer
The number of blank lines to reserve at the top of the page (integer).
Data Control Language

GRANT

The GRANT statement gives privileges to user groups.

Rules
- You must have command qualifications for GRANT, the WITH GRANT OPTION, and the privileges for the specified table or query in order for you to execute the GRANT statement. If you created the table or query to which you are giving privileges, you automatically have the WITH GRANT OPTION. If you include the WITH GRANT OPTION clause in your GRANT statement, you give the WITH GRANT OPTION to the specified user group (i.e., anyone in that user group can grant privileges to that table or query).
- You may assign the WITH GRANT OPTION and the table or query privileges to a user group or to PUBLIC. If privileges are assigned to PUBLIC, all users receive the privileges.

GRANT {ALL or {{SELECT} [INSERT] [UPDATE][DELETE]}}
ON [TABLE] table_set TO user_group ,[user_group]...
[WITH GRANT OPTION]

Gives table privileges to one or more user groups. If you specify PUBLIC as the user group, all users will have privileges to the table.

Example
   GRANT ALL ON TABLE employees, table_a:table_d, task*
   TO finance, managers

GRANT {ALL or {{SELECT} [UPDATE]}}
ON QUERY query_set TO user_group ,[user_group]...
[WITH GRANT OPTION]

Gives privileges on a query to one or more user groups. If you specify PUBLIC as the user group, all users will have privileges to the table.

Example
   GRANT SELECT ON QUERY hours_billed, query_a:query_c,
   insert*
   TO managers
REVOKE

The REVOKE statement takes table or query privileges away from user groups.

Rules
- To execute the REVOKE statement, you must receive command qualifications for REVOKE, the WITH GRANT OPTION, and privileges for the specified table or query from your DBA. If you created the table or query from which you are removing privileges, you automatically have the WITH GRANT OPTION. If you include the WITH GRANT OPTION clause in your REVOKE statement, you take away the specified user group's ability to grant or revoke privileges on that table or query.

REVOKE {ALL or {[SELECT] [INSERT] [UPDATE] [DELETE]}} ON [TABLE] table_set FROM user_group [,user_group]... [WITH GRANT OPTION]

Removes table privileges from one or more user groups.

Examples
REVOKE ALL ON TABLE tasks, table_a:table_c, emp* FROM finance, managers

REVOKE SELECT ON bill_stats FROM PUBLIC

REVOKE {ALL or {[SELECT] [UPDATE]}} ON QUERY query_set FROM user_group [,user_group]... [WITH GRANT OPTION]

Removes query privileges from one or more user groups.

Example
REVOKE SELECT ON QUERY hours_billed, query_a:query_d, insert* FROM managers

Query Set

Designates the query or queries on which the GRANT and REVOKE commands are to be applied.

{query_name or {query_name:query_name} or {query_name*}} [{query_name or {query_name:query_name} or {query_name*}}]...

Example
hours_billed, query_a:query_c, insert*
Table Set

Designates the table or tables on which the GRANT and REVOKE commands are to be applied.

{[schema.] table_name or {table_name:table_name} or {table_name*}}
[,]{{[schema.] table_name or {table_name:table_name} or {table_name*}}}...

Example
   EMPLOYEES, TABLE_A:TABLE_D, TASK*
Data Definition Language

ALTER

The ALTER statement is used to change the definition of an index, procedure, schema, table, or view. The ALTER statement can only be used to modify definitions that were created by the CREATE statement. This means tables and schemas defined using the DBA options MAP EXISTING GLOBALS and SCHEMA EDIT cannot be changed by the ALTER statement.

Rules

- To execute the ALTER INDEX or ALTER TABLE clauses, you must have the Create table or index command qualification. To execute the ALTER SCHEMA clause, you must have the Create schema command qualification. To execute the ALTER VIEW clause, you must have the Create view command qualification.

ALTER INDEX

ALTER INDEX index COMMENT literal

Changes the description of an index.

Rules

- The literal must be a character literal enclosed in quotation marks.

Example

    ALTER INDEX emp_by_name
    COMMENT 'Index to employees by name'
    Type topic text here.

ALTER INDEX index RENAME name

Changes the name of an index.

Rules

- The new index name must be different from any other table, view, or index name within the same schema.

Example

    ALTER INDEX emp_by_name RENAME employees_by_name
    .
ALTER PROCEDURE

ALTER PROCEDURE procedure [COMMENT literal]
[(parameter [data_type] [=default]
{,parameter [data_type] [=default]...})]
AS sql_statement

Changes a procedure definition.

Rules

- If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used by various display options to help identify the procedure.
- The parameter data_type may not be TEXT.
- The default values must be either character or numeric literals.

Example

ALTER PROCEDURE SP_GET_NAME (:InputSSN CHARACTER(11))
AS SELECT NAME FROM EMPLOYEES WHERE EMP_SSN=:InputSSN

ALTER SCHEMA

ALTER SCHEMA schema COMMENT literal

Changes the description of a schema.

Rules

- The literal must be a character literal enclosed in quotation marks.

Example

ALTER SCHEMA temp_statistics
COMMENT 'Temporary statistics tables'

ALTER SCHEMA schema RENAME name

Changes the name of a schema.

Rules

- The new schema name must be different from all other schema names.
Example

ALTER SCHEMA sql_test RENAME demo
ALTER TABLE table

ALTER TABLE

ADD COLUMN column_name [SINGLE] [COMMENT literal] [HEADING literal]{data_type or {REFERENCES table}} [NOT NULL]

Adds a new column to an existing table.

Rules

• If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used by various display options to help identify the column.
• If the HEADING clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used as the default column header in SELECT and WRITE clauses. The literal may contain one or more vertical bar characters ( | ), which are used to produce a multi-line header, and may contain both upper and lower case characters, as well as embedded spaces and other punctuation marks.
• The column must have either a data type or a REFERENCES clause.
• If the column has the REFERENCES table clause, the referenced table must have a single primary key column. In this situation the new column can be used as both a value and a foreign key. When used as a value, the column has the same data type as the primary key of the referenced table.
• The NOT NULL clause causes the system to reject any INSERT or UPDATE statement which would save a row with a null value for this column. If the table already contains one or more rows, and a column is added with a NOT NULL clause, the system will add the column even though the existing rows will contain a null value. To maintain database integrity you should use an UPDATE statement to add a value for all existing rows.
• If the SINGLE key word is used, the SQL engine reserves a global node for the specified column name (ignoring the pack logic for the specified column).

Examples

ALTER TABLE employees ADD COLUMN birthday DATE

ALTER TABLE employees ADD COLUMN dept REFERENCES departments NOT NULL
ALTER TABLE table ADD FOREIGN KEY foreign_key [COMMENT literal](column_name[,column_name]...) REFERENCES table

 Adds a new foreign key to an existing table.

 Rules

• If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks.
• The foreign key name must be different from all other column and foreign key names in the table.
• The referenced table can have one or more primary key columns; however, the number of column names specified within the parentheses must match the number and sequence of primary key columns for the referenced table.

 Example

 ALTER TABLE charges ADD FOREIGN KEY task_link (proj_no, task_no) REFERENCES tasks

 ALTER TABLE table ALTER COLUMN column_name [COMMENT literal] [HEADING literal] data_type [NOT [NULL]]

 Changes the description or data type of a column.

 Rules

• If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used by various display options to help identify the column.
• If the HEADING clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used as the default column header in SELECT and WRITE clauses. The literal may contain one or more vertical bar characters ( | ), which are used to produce a multi-line header.
• The column’s data type definition will be changed to the new specification.

 Example

 ALTER TABLE employees ALTER COLUMN emp_name CHARACTER(20) NOT NULL

 ALTER TABLE table ALTER COLUMN column_name RENAME name

 Changes the name of a column.

 Rules

• The new name must be different from all other column and foreign key names in the table.
Example

ALTER TABLE employees ALTER COLUMN name
RENAME emp_name

ALTER TABLE table ALTER FOREIGN KEY foreign_key
[COMMENT literal]

Changes the description of a foreign key.

Rules
• The literal must be a character literal enclosed within quotation marks.

Example

ALTER TABLE employees ALTER FOREIGN KEY mgr_link
COMMENT 'Link to the employees manager'

ALTER TABLE table ALTER FOREIGN KEY foreign_key RENAME name

Changes the name of a foreign key.

Rules
• The foreign key name must be different from all other column and foreign
  key names in the table.

Example

ALTER TABLE employees ALTER FOREIGN KEY mgr_link RENAME
manager_link

ALTER TABLE table COMMENT literal

Changes the description of a table.

Example

ALTER TABLE TASKS COMMENT 'Task description and status'

ALTER TABLE table DROP COLUMN column_name

Removes a column from a table.

⚠️ When a column is dropped, its contents are deleted and cannot be restored.

Rules
• The column cannot be part of the primary key.
• If the column is part of a foreign key, the foreign key will also be deleted.
KBSSQL Syntax

Example
ALTER TABLE employees DROP COLUMN birthday

ALTER TABLE table DROP FOREIGN KEY foreign_key

Removes an existing foreign key definition.

Rules
• The foreign key cannot be part of the primary key.

Example
ALTER TABLE employees DROP FOREIGN KEY mgr_link

ALTER TABLE table MERGE INTO schema

 Associates a table with a different schema.

⚠️ The contents of the table is not copied or changed in any way; only the table's schema is changed. If the old schema and the new schema have different globals for data storage, the data will exist only in the global associated with the old (original) schema. To move the data, you must create a new table in the new schema, copy the old table data to the new table using an INSERT statement, and then drop the old table to delete the old data.

Rules
• The table name and its associated index names must be different from all other table, index, and view names in the new schema.

Example
ALTER TABLE employees MERGE INTO training

ALTER TABLE table RENAME name

Changes the table name.

Rules
• The new name must be different from all other table, index and view names defined in the schema.

Example
ALTER TABLE employees RENAME emp

ALTER VIEW
ALTER VIEW view MERGE INTO schema

Associates a view with a different schema.

Rules
- The view name must be different from all other table, index and view names in the new schema.

Example
ALTER VIEW proj_tasks MERGE INTO training

ALTER VIEW view RENAME name

Changes the name of a view.

Rules
- The new name must be different from all other table, index and view names defined in the schema.

Example
ALTER VIEW proj_tasks RENAME ptasks

ALTER VIEW view [COMMENT literal] 
[(column_name [,column_name]...)] AS select

Changes the definition of a view.

⚠️ This clause completely deletes the old definition and replaces it with the new definition. It is similar to performing a DROP VIEW statement followed by a CREATE VIEW statement, except that all user group privileges will be preserved by the ALTER VIEW statement.

Example
ALTER VIEW proj_tasks AS
SELECT project, budget, task, status
FROM projects, tasks
WHERE projects.proj_no = tasks.proj_no

CREATE

The CREATE statement is used to add new indices, procedures, schemas, tables, and views to the data dictionary.

Rules
To execute the CREATE INDEX or CREATE TABLE clauses, you must have the Create table or index command qualification. To execute the CREATE SCHEMA clause, you must have the Create schema command qualification. To execute the CREATE VIEW clause, you must have the Create view command qualification.

CREATE INDEX

CREATE [UNIQUE] INDEX index {ON or FOR} table [COMMENT literal] (column_name [,column_name]...)  

Adds a new index for an existing table.

Indices are used by KB_SQL to optimize query execution. Indices can also be used as tables, by referencing the index name in the FROM clause. Typically indices with only one column provide the best results for query optimization. If a new index is created for a table with existing rows, the CREATE statement will add all valid rows to the index.

Rules

- The UNIQUE key word causes the database to restrict inserts and updates to rows with a unique set of column values. Any change to a row that violates this constraint is rejected.
- The table name may contain a schema prefix. The index name may not contain a schema prefix.
- The index name must be different from all other table, index, and view names within the schema.
- If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used by various display options to help identify the index.
- The index columns must be defined in the table specified in the ON or FOR clause. Each index column will have the same definition as the corresponding table column.
- The index columns may not be TEXT data type values.
- The index columns may not be foreign keys created by the FOREIGN KEY clause.
- If any of the columns that compose the primary key for the table are not included in the index column list, they are automatically added to the index.

Examples

CREATE INDEX emp_by_mgr_sal FOR employees (manager, salary)
CREATE UNIQUE INDEX emp_by_name FOR employees (name)

CREATE PROCEDURE

CREATE PROCEDURE procedure [COMMENT literal] [(parameter [data_type] [=default] {.parameter [data_type] [=default]}...)] AS sql_statement

Creates a procedure definition.

Rules

• The procedure name may contain a schema prefix.
• The procedure name must be different from all other procedure names.
• If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used by various display options to help identify the procedure.
• The parameter data_type may not be TEXT.
• The default values must be either character or numeric literals.

Example

ALTER PROCEDURE SP_GET_NAME (:InputSSN CHARACTER(11)) AS SELECT NAME FROM EMPLOYEES WHERE EMP_SSN=:InputSSN

CREATE SCHEMA

CREATE SCHEMA [AUTHORIZATION] schema [COMMENT literal]

Adds a new schema to the data dictionary.

Usually schemas are added by your DBA using the SCHEMA EDIT option from the DBA menu.

Rules

• The schema name must be different from all other schema names.
• The AUTHORIZATION key word has no effect. It is included for ANSI compatibility.

Example

CREATE SCHEMA monthly_stats COMMENT 'Monthly Statistics Extract Tables' topic text here.
CREATE TABLE

CREATE TABLE table [COMMENT literal] [PACK [USING {literal or integer}]]
(
    column_name [SINGLE] [COMMENT literal] [HEADING literal] [PRIMARY [AVG DISTINCT numeric]]
    {data_type or {REFERENCES table}} [NOT NULL] [,
    column_name [SINGLE] [COMMENT literal] [HEADING literal] [PRIMARY [AVG DISTINCT numeric]]
    {data_type or {REFERENCES table}} [NOT NULL]]...
    [,PRIMARY KEY (column_name [AVG DISTINCT numeric] [,column_name
    [AVG DISTINCT numeric]]...)]
    [,FOREIGN KEY foreign_key [COMMENT literal] (column_name
    [,column_name]...)
    REFERENCES table]...
)

Adds a new table to the data dictionary.

⚠️ The new table will not contain any rows of data. Rows may be added to the table using the INSERT statement.

Rules

- If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used by various display options to help identify the table, column, or foreign key.
- If the HEADING clause is used, the literal must be a character literal enclosed within quotation marks. The literal is used as the default column header in SELECT and WRITE clauses. The literal may contain one or more vertical bar characters ( | ), which are used to produce a multi-line header.
- Each column and foreign key name must be different from all other column and foreign key names.
- Columns with the data type TEXT may not be used as part of the primary key or and foreign key.
- If the PRIMARY clause is used in a column definition, that column must be the table's only primary key. If the table has more than one primary key column, the PRIMARY KEY clause must be used. Primary key columns are always not null.
- If the AVG DISTINCT clause is used, the numeric value represents the average number of distinct values for the key. If the table contains more than one primary key column, then the average distinct value refers to the number of key values expected within all previously primary key columns. All of the average distinct values multiplied together should be equal to the...
number of rows in the table. For example, a table with two keys, the first with ten average distinct values and the second with five, would contain fifty rows.

- If the REFERENCES clause is used in a column definition, the referenced table must have a single primary key. The column inherits the data type from the referenced table’s primary key column.

- If the FOREIGN KEY clause is specified, the number of column names listed must match the number of primary keys for the referenced table.

- If the PACK key word is used, the SQL engine packs multiple column values into the same M global node.

- The USING clause specifies the delimiter to use between column values. If the PACK key word is used without the USING clause, the column values will be separated by the value your DBA entered for the Delimiter prompt (using the CONFIGURATION/ SITE EDIT/DEFAULT INFO option).

- If the SINGLE key word is used, the SQL engine reserves a global node for the specified column name (ignoring the pack logic for the specified column).

**Examples**

```sql
CREATE TABLE employees COMMENT 'Employee data'
  (emp_ssn COMMENT 'Employee''s Social Security Number' HEADING 'Social|Security|Number' CHARACTER(11) PRIMARY,
   name COMMENT 'Employee''s name' CHARACTER(20), salary NUMERIC(8,2))
```

In the example above, two single quotes (not a double quote) are used in the COMMENT literal.

```sql
CREATE TABLE proj_status (proj_no INTEGER(3) NOT NULL,
   status_date DATE, budget NUMERIC(8,2), actual NUMERIC(8,2),
   PRIMARY KEY(proj_no, status_date),
   FOREIGN KEY proj_link (proj_no) REFERENCES projects))
```

**Illustration of packed and unpacked columns**

**Packed**

```sql
^SQLxx(KEY_ID,ROW,1) = COL_1 <del> COL_2 <del>...COL_N
```

**Unpacked**

```sql
^SQLxx(KEY_ID,ROW,1) = COL_1
   ,2) = COL_2
   ,N) = COL_N
```

CREATE VIEW
CREATE VIEW view [COMMENT literal]
    [(column_name [COMMENT literal] [HEADING literal]
    [,column_name [COMMENT literal] [HEADING literal]]...)]
AS select

Creates a virtual table.

The contents of a virtual table are not stored in the database, rather the table is computed as needed.

Rules

- The view name must be different from all other table, index, and view names in the schema.
- The view is updatable if:
  1. it is based on a single updatable table
  2. all the columns that the view returns come from that single table
  3. the CREATE VIEW statement does not reference aggregate functions
  4. HAVING and GROUP BY clauses are not used in the CREATE VIEW statement
- If the COMMENT clause is used, the literal must be a character literal enclosed within quotation marks. Use the COMMENT clause to change the column's default comment to reflect a meaning more appropriate to the way the column is being used in this view. Use the HEADING clause if you want to change the column's default header.
- If the SELECT statement contains any values other than column names, then the view definition must specify corresponding view column names for these values. This can be accomplished one of two ways. Refer to the last two examples.
- The SELECT statement referenced by the CREATE VIEW statement cannot contain an ORDER BY clause, or any events, or any modifiers.

Examples

CREATE VIEW proj_tasks AS
SELECT project, budget, task, status
FROM projects, tasks
WHERE projects.proj_no = tasks.proj_no

CREATE VIEW new_payroll COMMENT 'Projected payroll with a 10% increase in salary' AS
SELECT name, salary * 1.1 AS new_sal
FROM employees

CREATE VIEW task_count (project COMMENT 'Project name',
nbr_tasks HEADING 'Number of Tasks') AS
SELECT project, COUNT(*)
FROM projects, tasks
WHERE projects.proj_no = tasks.proj_no
GROUP BY proj_no

The second example specified view column names because the COUNT(*) value could not be used as a view column name.

In the last two examples following, at least one of the values in the SELECT statement is not a column name; therefore, names must be assigned to these values.

The first example uses a column list in the CREATE statement to assign column names. If you use this method, each value in the SELECT statement must have a corresponding name in the CREATE statement, even those values that are already column names.

CREATE VIEW new_view (budget, lnk_ssn, lnk_projno) AS
SELECT  budget, leader_link@emp_ssn,
        project_link@proj_no
FROM projects, tasks

The second example uses aliases to assign names to each value in the SELECT statement that is not a column name. We recommend the use of aliases; you have to specify names only for those values that need them, and the placement of each column name alongside of its expression lends to easier reading during debugging.

CREATE VIEW test_view AS
SELECT  budget,
        leader_link@emp_ssn AS lnk_ssn,
        project_link@proj_no AS lnk_projno
FROM projects, tasks

DROP

The DROP statement removes an object from the data dictionary. To execute this statement, your you must have command qualifications for DROP TABLE, DROP SCHEMA, and DROP VIEW. Only database objects created using the CREATE statement may be dropped.

DROP INDEX
**KBSSQLSyntax**

**DROP INDEX index**

Removes an index from the data dictionary.

> All defined index rows will be deleted.

**Rules**  
- The index name may contain a schema prefix.

**Example**

```
DROP INDEX emp_by_name
```

**DROP PROCEDURE**

**DROP PROCEDURE procedure**

Removes a procedure from the data dictionary.

**Example**

```
DROP PROCEDURE sp_get_employee
```

**DROP SCHEMA**

**DROP SCHEMA schema**

Removes a schema from the data dictionary.

**Rules**  
- The schema cannot contain any tables or views. If tables or views do exist, KB_SQL will display an error message.

**Example**

```
DROP SCHEMA training
```

**DROP TABLE**

**DROP TABLE table**

Removes a table definition from the data dictionary.

> All rows in the table will be deleted, and all indices for the table will also be dropped.
Rules
- The table name may contain a schema prefix.

Example
DROP TABLE backup_emp

DROP VIEW

DROP VIEW view

Removes a view definition from the data dictionary.

Rules
- The view name may contain a schema prefix.

Example
DROP VIEW proj_tasks
Data Manipulation Language

DELETE

DELETE FROM table [[AS] alias] [WHERE condition]

The DELETE statement removes one or more rows from a table. It is supported by SQL1.

Rules

• This statement can be used only on tables that were created using the CREATE TABLE statement.
• To execute this statement, you must have command qualifications for DELETE and DELETE privileges for the specified table.
• The table may contain a schema prefix.
• The table specified cannot be an index. It must be either an updatable base table or an updatable view.
• The ‘table AS alias’ clause is used to provide an alternate name for a table. When an alias is used, the alias becomes the name of the table for this query, and the original table name cannot be used as a reference for the table. When specifying a table alias, the AS key word is optional, but its use is recommended for readability. The table alias must be different from any other table name referenced by the statement.
• If the WHERE clause is not specified, all rows in the table are deleted.
• The condition may not contain a subquery that references the table to be deleted.
• If the rows specified by the WHERE condition are unavailable for DELETE, or if the rows to be deleted violate an integrity rule or cause a serialization error, the DELETE will not be successful. Otherwise, the rows specified by the WHERE condition are deleted. For the API, when the DELETE is not successful, the SQLCODE variable is set to -1 and the SQLSTATE variable is set to a value specific to the error; and when the DELETE is successful, the SQLCODE variable is set to 0 and the SQLSTATE variable is set to 00000.

Examples

DELETE FROM year_to_date

DELETE FROM emp_data WHERE emp_ssn = '123-45-6789'

INSERT
The **INSERT** statement adds rows to a table. This statement is often used to create summary statistics, or take snapshots of transient data, or import data from applications outside of M.

**Rules**
- In order for you to execute the **INSERT** statement, your DBA must give you command qualifications for **INSERT** and **INSERT** privileges for the specified table.
- If the table is a mapped table, then your DBA must define a table filer that can support Inserts.
- The table specified cannot be an index. It must be either an updateable base table or an updateable view.

**Results**
If the table specified is unavailable for **INSERT**, or if the row(s) to be inserted violate an integrity rule or cause a serialization error, the **INSERT** will not be successful. Otherwise, the new row(s) will be added to the specified table.

**INSERT INTO** table *(column_name [,column_name]...)*
**VALUES** *(value [,value]...)*

Adds one row to a table. This version of the **INSERT** statement is often used to add rows to small dictionary tables.

**Rules**
- The number of column names must be equal to the number of values.
- The values must be either literals, M variables, or pseudo columns.
- The data types of the values must be compatible with the data types of the associated columns.

**Examples**
```
INSERT INTO tasks (task_no, proj_no, task, status)
VALUES (13,300,'INQUIRY','INCOMPLETE')

READ :XNAME CHARACTER(20) PROMPT 'Name'
READ :XSALARY NUMERIC(9,2) PROMPT 'Salary'
INSERT INTO emp_data (name, salary)
VALUES (:XNAME, :XSALARY)
```

**INSERT INTO** table *(column_name [,column_name]...)*
**FROM** file [**FORMAT** format] [**METHOD** method]

Adds one or more rows to a table from the specified host file. This version of the **INSERT** statement is used to upload data into M from non-M applications.

**Rules**
Data Manipulation Language

- The file must be in the specified format, and the number of column names must match the number of columns in the file.
- The values in the file must be in a valid format for the data types associated with the column names.
- If a FORMAT clause is specified, the format must be either COMMA, DELIMITED, DIF, or TAB. If the FORMAT clause is not specified, the default format defined by the DBA in the SITE EDIT option is used.
- The optional METHOD clause provides the ability to load from a source other than a host file, or to preprocess the file or data.

Example

```sql
INSERT INTO budget (year, month, department, amount)
FROM 'C:\EXCEL\BUDGET.DAT' FORMAT 'COMMA'
```

INSERT INTO table (column_name [,column_name]...) SELECT statement

Adds one or more rows from the results of the SELECT statement. This form of the INSERT statement is used to create summary statistics and snapshot tables.

Rules

- Because the SELECT statement is used as an argument to another function, you cannot use event blocks.
- The number of column_names must match the number of expressions in the SELECT statement.
- The data types of the columns returned by the SELECT statement must be compatible with the data types for the specified column names.

Example

```sql
INSERT INTO incomplete_tasks (proj_no, task_no, task)
SELECT proj_no, task_no, task
FROM tasks
WHERE status = 'INCOMPLETE'
```

SELECT

The SELECT statement is used to retrieve rows and columns from one or more tables, as well as to download data to host files or other structures using export methods. Restricted versions of the SELECT statement are also used by other commands to select information from tables.

```sql
SELECT [DISTINCT] *
[WHERE condition]
[GROUP BY expression [, expression]...]
[HAVING condition]
```
SELECT [DISTINCT] {{table.*} or NULL or {expression [AS alias][modifier...]}} [, {table.*} or NULL or {expression [AS alias] [modifier...]}]... FROM table [+] [AS alias] [,table [AS alias]]... [WHERE condition] [GROUP BY expression [, expression]...] [HAVING condition] [ORDER BY expression [DESC] [, expression [DESC]]...] [events]

The events parameter does not apply to the KB_APIs. Also, events can not be used if the SELECT statement is used as an argument to another function (e.g., within the INSERT statement).

Rules
- To execute this statement, you must have command qualifications for SELECT and SELECT privileges for the specified table.
- If the DISTINCT key word is specified, the result set will contain only one row for each unique combinations of the expressions listed in the SELECT clause. Otherwise, duplicate rows may exist.

SELECT clause

The asterisk (*) in the SELECT clause causes the result to contain every column (except those marked as 'conceal on select') from every table referenced in the FROM clause.

The table.* in the SELECT clause causes the result to contain every column, except those marked as 'conceal on select', from the specified table.

The use of NULL in the SELECT clause includes a null value for the corresponding result column. If the only column in a SELECT clause is the NULL value, the query will not print any result rows.

The 'value [AS] alias' clause provides an alternate name for the value. The alias name must be different from any column or foreign key name in any table referenced in the FROM clause. The value alias is used instead of the column name as the default column header.

If a value in the SELECT clause is anything other than a column, literal, pseudo column, or variable, the default column heading may be blank or incomplete. For this situation, an alias clause or HEADING modifier should be used to create a more meaningful column heading.
FROM clause

The '+' character following a table name in the FROM clause will cause that table to be "outer joined" with all of the other tables in the FROM clause. Only one table in each FROM clause may have the '+' character, all other tables will be outer joined to the one table with the '+' character. The table with the '+' character is the outer join table. All rows in the outer join table that pass conditions other than join conditions will appear in the result. If a join condition between the outer join table and any other table fails, the outer join table row will be joined with null values. A join condition is a test (or group of tests within parentheses) that involves columns from two or more tables.

The ' table AS alias' clause is used to provide an alternate name for a table. When an alias is used, the alias becomes the name of the table for this query, and the original table name cannot be used as a reference for the table. When specifying a table alias the AS key word is optional, but its use is recommended for readability. The table alias must be different from any other table name referenced by the query.

Other clauses

Each optional clause (WHERE, GROUP BY, HAVING, and ORDER BY) is applied to the results of the previous clauses.

The WHERE clause limits which rows are selected.

The GROUP [,scope] clause causes the result to be aggregated by the GROUP BY values, returning only one row for each set of distinct GROUP BY values. The GROUP BY is performed after the WHERE clause. If the expression listed in the GROUP [,scope] clause is a positive integer, then the value that will be used for the GROUP BY function will be the corresponding entry from the SELECT clause.

The HAVING clause limits which rows are returned. Since the HAVING clause is performed after the GROUP [,scope] clause, tests in the HAVING clause may check the results of the GROUP [,scope] clause, including GROUP BY aggregate functions, but not ORDER BY aggregate functions.

The ORDER [,scope] clause causes the result to be sorted by the specified values. The ORDER [,scope] clause is performed after the HAVING clause, and may sort by any value except ORDER BY aggregate functions.

The key word DESC following an ORDER BY expression causes the normal sort order to be reversed. The normal sort order is from low to higher numbers, and
from A to Z. If the expression listed in the ORDER [,scope] clause is a positive integer, then the value that will be used for the ORDER BY function will be the corresponding entry from the SELECT clause.

**Examples**

```sql
SELECT name, manager, salary
FROM employees
WHERE salary > 10

SELECT manager, AVG(salary), SUM(salary)
FROM employees
GROUP BY manager

SELECT manager, COUNT(*), SUM(salary)
FROM employees
GROUP BY 1
HAVING COUNT(*) > 1
ORDER BY 3
```

**UPDATE**

The UPDATE statement changes rows in a table.

```sql
UPDATE table [[AS] alias]
SET column_name = expression or NULL
[,column_name = expression or NULL]...
[WHERE condition]
[GROUP BY expression [, expression]...]
[HAVING condition]
```

**Rules**

- Your DBA must give you command qualifications for UPDATE and UPDATE privileges for the specified table in order for you to execute this statement.
- The table specified cannot be an index. It must be either an updatable base table or an updatable view.
- The condition in the WHERE clause may not contain a subquery that references the table specified in the UPDATE clause.
- The ' table AS alias' clause is used to provide an alternate name for a table. When an alias is used, the alias becomes the name of the table for this query, and the original table name cannot be used as a reference for the table. When specifying a table alias the AS key word is optional, but its
use is recommended for readability. The table alias must be different from any other table name referenced by the query.

- If a FROM clause is used, the table identified in the UPDATE clause must also be specified in the FROM clause, and if an alias has been used in the UPDATE clause, it must also be used in the FROM clause.
- If the WHERE condition is omitted, then every row in the table is updated, and the specified columns will not be changed.
- If an updated row violates a unique constraint, the update row is rejected.

Results
If the table specified is unavailable for update, or if the rows to be updated violate an integrity rule or cause a serialization error, the UPDATE will not be successful. Otherwise, the rows specified by the WHERE condition are updated.

Examples
UPDATE employees
SET salary = salary * 1.05

UPDATE tasks
SET status='COMPLETE'
WHERE proj_no = 100 AND task_no = 2
# Appendix

## ASCII Chart

<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>space</td>
<td>81</td>
<td>Q</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>82</td>
<td>R</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>83</td>
<td>S</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>84</td>
<td>T</td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td>85</td>
<td>U</td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td>86</td>
<td>V</td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td>87</td>
<td>W</td>
</tr>
<tr>
<td>39</td>
<td>'</td>
<td>88</td>
<td>X</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>89</td>
<td>Y</td>
</tr>
<tr>
<td>41</td>
<td>)</td>
<td>90</td>
<td>Z</td>
</tr>
<tr>
<td>42</td>
<td>*</td>
<td>91</td>
<td>[</td>
</tr>
<tr>
<td>43</td>
<td>+</td>
<td>92</td>
<td>\</td>
</tr>
<tr>
<td>44</td>
<td>,</td>
<td>93</td>
<td>]</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>94</td>
<td>^</td>
</tr>
<tr>
<td>46</td>
<td>.</td>
<td>95</td>
<td>–</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>96</td>
<td>a</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>97</td>
<td>b</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td>98</td>
<td>c</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>99</td>
<td>d</td>
</tr>
<tr>
<td>51</td>
<td>3</td>
<td>100</td>
<td>e</td>
</tr>
<tr>
<td>52</td>
<td>4</td>
<td>101</td>
<td>f</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>102</td>
<td>g</td>
</tr>
<tr>
<td>54</td>
<td>6</td>
<td>103</td>
<td>h</td>
</tr>
<tr>
<td>55</td>
<td>7</td>
<td>104</td>
<td>i</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>105</td>
<td>j</td>
</tr>
<tr>
<td>57</td>
<td>9</td>
<td>106</td>
<td>k</td>
</tr>
<tr>
<td>58</td>
<td>:</td>
<td>107</td>
<td>l</td>
</tr>
<tr>
<td>59</td>
<td>;</td>
<td>108</td>
<td>m</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>109</td>
<td>n</td>
</tr>
<tr>
<td>61</td>
<td>=</td>
<td>110</td>
<td>o</td>
</tr>
<tr>
<td>62</td>
<td>&gt;</td>
<td>111</td>
<td>p</td>
</tr>
<tr>
<td>63</td>
<td>?</td>
<td>112</td>
<td>q</td>
</tr>
<tr>
<td>64</td>
<td>@</td>
<td>113</td>
<td>r</td>
</tr>
<tr>
<td>65</td>
<td>A</td>
<td>114</td>
<td>s</td>
</tr>
<tr>
<td>66</td>
<td>B</td>
<td>115</td>
<td>t</td>
</tr>
<tr>
<td>67</td>
<td>V</td>
<td>116</td>
<td></td>
</tr>
</tbody>
</table>
Data Type Formats

The following data type formats are provided with KB_SQL. Your DBA has the authority to add data type formats that are specific to your site.

<table>
<thead>
<tr>
<th>Data Type Format Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Type</strong></td>
</tr>
<tr>
<td>CHARACTER</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>FLAG</td>
</tr>
</tbody>
</table>
### KB_SQL Key Words

All key words are reserved words, which means you cannot use them for object names (names of tables, columns, etc.). KB_SQL uses them to specify command syntax and to identify SQL system objects.

The following change in key words rule is backwards compatible:

KB_SQL will not reject any existing object names that use reserved words. But, it is strongly suggest that you eventually rename any object names that use key words because problems may arise if you use KB_SQL to interact with other DBMS applications.

Key words may be entered in uppercase or lowercase or a combination of both.

<table>
<thead>
<tr>
<th>KB_SQL Key Words</th>
<th>KB_SQL Key Words</th>
<th>KB_SQL Key Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSOLUTE</td>
<td>INNER</td>
<td>SQL_FN_IDENTIFIER</td>
</tr>
<tr>
<td>ACCESS</td>
<td>INPUT</td>
<td>SQL_FN_IFNULL</td>
</tr>
<tr>
<td>ADA</td>
<td>INSENSITIVE</td>
<td>SQL_FN_INITIAL_CAPS</td>
</tr>
<tr>
<td>ADD</td>
<td>INSERT</td>
<td>SQL_FN_INSERT</td>
</tr>
<tr>
<td>AFTER</td>
<td>INT</td>
<td>SQL_FN_LCASE</td>
</tr>
<tr>
<td>SQL_FUNCTION</td>
<td>SQL_INTEGER</td>
<td>SQL_NUMERIC</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_LTRIM</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_MINUTE</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_MOD</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_MONTH</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FNMONTHNAME</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_NEXT_MONTH_FIRST</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_NEXT_MONTH_LAST</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_NEXT_WEEKDAY</td>
</tr>
<tr>
<td>SQL_FN_LEFT</td>
<td>SQL_FN_LENGTH</td>
<td>SQL_FN_NEXT_WEEKEND_DAY</td>
</tr>
<tr>
<td>SQL_FN_NOW</td>
<td>SQL_FN_PI</td>
<td>SQL_FN_POWER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_RANDOM</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_REPEAT</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_REPLACE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_RIGHT</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_ROUND</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_RTRIM</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_SECOND</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_SIGN</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_SOUNDEX</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_SPACE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_T month</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_UCASE</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_USER</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_X week</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPADD</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPDIFF</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TIMESTAMPPART</td>
</tr>
<tr>
<td>SQL_FN_PI</td>
<td>SQL_FN_QUARTER</td>
<td>SQL_FN_TRUNCATE</td>
</tr>
<tr>
<td>Current Time</td>
<td>Partial</td>
<td>SQL Type Time</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Current Timestamp</td>
<td>Pascal</td>
<td>SQL Type Timestamp</td>
</tr>
<tr>
<td>Cursor</td>
<td>Piece</td>
<td>SQL Varchar</td>
</tr>
<tr>
<td>Date</td>
<td>PLI</td>
<td>SQL Version</td>
</tr>
<tr>
<td>Date Only</td>
<td>Position</td>
<td>SUBSTRING</td>
</tr>
<tr>
<td>Date Time Now</td>
<td>Precision</td>
<td>SUM</td>
</tr>
<tr>
<td>Day</td>
<td>Prepare</td>
<td>SYNONYM</td>
</tr>
<tr>
<td>DBA</td>
<td>Preserve</td>
<td>SYSTEM</td>
</tr>
<tr>
<td>Deallocate</td>
<td>Previous</td>
<td>SYS_ADDRESS1</td>
</tr>
<tr>
<td>Dec</td>
<td>Primary</td>
<td>SYS_ADDRESS2</td>
</tr>
<tr>
<td>Decimal</td>
<td>Prior</td>
<td>SYS_ADDRESS3</td>
</tr>
<tr>
<td>Declare</td>
<td>Privileges</td>
<td>SYS_ADDRESS4</td>
</tr>
<tr>
<td>Default</td>
<td>Procedure</td>
<td>SYS_NAME</td>
</tr>
<tr>
<td>Deferrable</td>
<td>Prompt</td>
<td>SYS_NODE</td>
</tr>
<tr>
<td>Deferred</td>
<td>Public</td>
<td>TABLE</td>
</tr>
<tr>
<td>Delete</td>
<td>Query</td>
<td>Temporary</td>
</tr>
<tr>
<td>Desc</td>
<td>RAVG</td>
<td>Text</td>
</tr>
<tr>
<td>Describe</td>
<td>RCOUNT</td>
<td>Then</td>
</tr>
<tr>
<td>Descriptor</td>
<td>Read</td>
<td>Then</td>
</tr>
<tr>
<td>Detail</td>
<td>Real</td>
<td>Time</td>
</tr>
<tr>
<td>Device</td>
<td>References</td>
<td>Time</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>Replace</td>
<td>Time</td>
</tr>
<tr>
<td>Dictionary</td>
<td>RENAME</td>
<td>Timezone</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Restrict</td>
<td>Timezone Minute</td>
</tr>
<tr>
<td>Displacement</td>
<td>REVOKE</td>
<td>Timezone Only</td>
</tr>
<tr>
<td>Distinct</td>
<td>Right</td>
<td>TO</td>
</tr>
<tr>
<td>Does</td>
<td>RMAX</td>
<td>TODAY</td>
</tr>
<tr>
<td>Domain</td>
<td>RMIN</td>
<td>TO_DATE</td>
</tr>
<tr>
<td>Double</td>
<td>Rollback</td>
<td>TO_FLAG</td>
</tr>
<tr>
<td>Drop</td>
<td>Round</td>
<td>TO_MOMENT</td>
</tr>
<tr>
<td>Else</td>
<td>Row</td>
<td>TO_NUMERIC</td>
</tr>
<tr>
<td>Elseif</td>
<td>Rowid</td>
<td>TO_TIME</td>
</tr>
<tr>
<td>End</td>
<td>ROWNUM</td>
<td>TRANSACTION</td>
</tr>
<tr>
<td>Endif</td>
<td>Rows</td>
<td>TRANSLATE</td>
</tr>
<tr>
<td>Escape</td>
<td>RPAD</td>
<td>TRANSLATION</td>
</tr>
<tr>
<td>Except</td>
<td>RSUM</td>
<td>TRIGGER</td>
</tr>
<tr>
<td>Exception</td>
<td>Run</td>
<td>TRUE</td>
</tr>
<tr>
<td>Exec</td>
<td>SCHEMA</td>
<td>UNION</td>
</tr>
<tr>
<td>Execute</td>
<td>Scroll</td>
<td>UNIQUE</td>
</tr>
<tr>
<td>Exists</td>
<td>Second</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>External</td>
<td>Section</td>
<td>UPDATE</td>
</tr>
<tr>
<td>Extract</td>
<td>SELECT</td>
<td>USAGE</td>
</tr>
<tr>
<td>False</td>
<td>SEQUENCE</td>
<td>USER</td>
</tr>
<tr>
<td>Fetch</td>
<td>SET</td>
<td>USER_GROUP</td>
</tr>
<tr>
<td>File</td>
<td>SIZE</td>
<td>USING</td>
</tr>
<tr>
<td>Final</td>
<td>SKIP</td>
<td>VALUE</td>
</tr>
<tr>
<td>Find</td>
<td>SMALLINT</td>
<td>VALUES</td>
</tr>
<tr>
<td>First</td>
<td>SOME</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Flag</td>
<td>SQL</td>
<td>VARIABLE</td>
</tr>
<tr>
<td>Float</td>
<td>SQLCA</td>
<td>VARYING</td>
</tr>
<tr>
<td>Footer</td>
<td>SQLCODE</td>
<td>VIEW</td>
</tr>
<tr>
<td>For</td>
<td>SQLERROR</td>
<td>WHEN</td>
</tr>
<tr>
<td>Foreign</td>
<td>SQLSTATE</td>
<td>WHENEVER</td>
</tr>
<tr>
<td>Format</td>
<td>SQLWARNING</td>
<td>WHERE</td>
</tr>
<tr>
<td>Fortran</td>
<td>SQL_BIGINT</td>
<td>WITH</td>
</tr>
<tr>
<td>Found</td>
<td>SQL_BIT</td>
<td>WORDWRAP</td>
</tr>
<tr>
<td>Footer</td>
<td>SQL_VERSION</td>
<td>WORK</td>
</tr>
<tr>
<td>Footer</td>
<td>SQL_VARCHAR</td>
<td>WRAP</td>
</tr>
</tbody>
</table>
KB_SQL supports seven value operators.

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction or change sign</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>\</td>
<td>Integer division</td>
</tr>
<tr>
<td>#</td>
<td>Modulo</td>
</tr>
<tr>
<td>(or</td>
<td>)</td>
</tr>
</tbody>
</table>

**Rules**

- The concatenation operator is the only operator that you may use on CHARACTER, FLAG and TEXT values. The concatenation operator may be represented by either a single or double pipe (the ANSI standard).
- Only INTEGER and NUMERIC values may be combined using multiplication.
- Only DATE, INTEGER, NUMERIC, and TIME values may be combined using the modulo, division, or integer division operators.
- The resulting data type of any two values combined using the division operator is NUMERIC.
• The resulting data type of any two values combined using the integer division operator is INTEGER.
• The resulting date type of any two values combined using the concatenation operator will be TEXT if either value is a TEXT value, otherwise it is CHARACTER.
• Assuming two values (VALUE1 and VALUE2) are combined using the specified operators, the following tables indicate the data type of the result.

<table>
<thead>
<tr>
<th>Addition or Subtraction Value Combination Result Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE1</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>INTEGER</td>
</tr>
<tr>
<td>MOMENT</td>
</tr>
<tr>
<td>MOMENT</td>
</tr>
<tr>
<td>NUMERIC</td>
</tr>
<tr>
<td>NUMERIC</td>
</tr>
<tr>
<td>TIME</td>
</tr>
<tr>
<td>TIME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplication Value Combination Result Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE1</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>INTEGER</td>
</tr>
<tr>
<td>NUMERIC</td>
</tr>
<tr>
<td>NUMERIC</td>
</tr>
</tbody>
</table>

Modulo

<table>
<thead>
<tr>
<th>Modulo Value Combination Result Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE1</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>INTEGER</td>
</tr>
<tr>
<td>NUMERIC</td>
</tr>
<tr>
<td>NUMERIC</td>
</tr>
<tr>
<td>TIME</td>
</tr>
</tbody>
</table>

Examples

```sql
SELECT proj_no, 
    COUNT(WHEN status = 'INCOMPLETE' THEN 1) AS INCOMPLETE,
```
KBSSQLSyntax

```
ROUND(INCOMPLETE/COUNT(*)*100,2) HEADING '
INCOMPLETE',
COUNT(WHEN status = 'COMPLETE' THEN 1) AS COMPLETE,
ROUND(COMPLETE/COUNT(*)*100,2) HEADING '
COMPLETE'
FROM tasks
GROUP BY 1
HEADER
WRITE 'Project Task Status' LEFT 80
WRITE 'Printed | TODAY | ' at ' |NOW LEFT 80

SELECT patient_name,
admit_date # 7 as admit_day,
admit_date - birthdate \ 365.25 AS years,
rx_charges+lab_charges+op_charges+room_charges
AS total_charges
FROM patients

SELECT SUM(WHEN t_code IN (4,9) THEN amount) AS withdrawals,
SUM(WHEN t_code IN (1,2,5) THEN amount) AS deposits,
deposits-withdrawals AS balance
FROM transaction
WHERE t_date BETWEEN '1/1/91' AND '12/31/91'

Functions

A function creates a value derived from one or more values.

The following list of row functions includes only those functions that are included in the base system. There are two basic types of functions: Aggregate and scalar. Aggregate functions combine values from one or more rows to produce a single result value. Scalar functions operate on only data from one row. Scalar functions are further organized into the following categories: data type conversion, formatting, numeric, string, system, time and date, and miscellaneous.

Additional scalar functions may have been added to your system by your vendor or DBA. The on-line query, SQL_FUNCTION_BY_TYPE, provides you with a complete list of all functions.
An aggregate function produces a result by combining one or more rows into a single result value. There are two types of aggregate functions supported by KB_SQL: GROUP BY and ORDER BY.

The results of GROUP BY functions are based on either all of the values in the GROUP [,scope] clause or the entire result set, if there is no GROUP [,scope] clause.

ORDER BY functions can produce summary information based on any value in the ORDER [,scope] clause, and also for the entire query result. ORDER BY functions allow you to compute the same function for different ORDER BY keys. ORDER BY functions may be used in conjunction with GROUP BY functions to provide sophisticated summary information, or as an alternative to GROUP BY functions when multiple levels of summary functions are needed.

Rules
- The [,scope] clause can be used only with a SELECT command that has an ORDER [,scope] clause. If the [,scope] clause is specified, the function is reset each time the corresponding ORDER BY key changes. (An ORDER BY key is either an expression that must be an exact match to an expression from the ORDER [,scope] clause or an integer that corresponds to the position of an expression in the ORDER [,scope] clause.) In some cases if you are using an expression, KB_SQL will inform you that it can't match the value in the [,scope] clause to one of the arguments in the ORDER [,scope] clause. This may occur with some expressions and virtual columns. When this does happen we suggest you use a positional reference to the ORDER [,scope] clause.

  If zero (0) is used for the ORDER BY key, the function will return a value based on all of the result rows.

- If there is no ORDER [,scope] clause, and a GROUP [,scope] clause exists, the function is reset each time any of the GROUP BY values change. The [,scope] clause is not allowed with a GROUP [,scope] clause.

- If there is an ORDER [,scope] clause and a GROUP [,scope] clause, the [,scope] clause must be used to refer to functions that are reset by changes in the ORDER BY keys. In this situation, a GROUP BY aggregate function may be nested within an ORDER BY aggregate function. This phenomenon is allowed because the GROUP [,scope] clause and all GROUP BY aggregate functions are computed before the ORDER [,scope] clause is executed. An ORDER BY aggregate function cannot be nested within a GROUP BY aggregate function or another ORDER BY aggregate function.
KBSSQL Syntax

- If there is no ORDER BY or GROUP [,scope] clause, using an aggregate function will cause the query to return one row, which is computed from all result rows.

The comma in the [,scope] clause may be replaced with the BY key word.

Set Functions

A set function produces an aggregate result by combining one or more rows into a single result value. There are two types of set functions supported by KB_SQL: GROUP BY and ORDER BY.

The results of GROUP BY functions are based on either all of the values in the GROUP BY clause or the entire result set, if there is no GROUP BY clause.

ORDER BY functions can produce summary information based on any value in the ORDER BY clause, and also for the entire query result.

ORDER BY functions allow you to compute the same function for different ORDER BY keys. ORDER BY functions may be used in conjunction with GROUP BY functions to provide sophisticated summary information, or as an alternative to GROUP BY functions when multiple levels of summary functions are needed.

Rules

- The BY clause can be used only with a SELECT command that has an ORDER BY clause. If the BY clause is specified, the function is reset each time the corresponding ORDER BY key changes. (An ORDER BY key is either an expression that must be an exact match to an expression from the ORDER BY clause or an integer that corresponds to the position of an expression in the ORDER BY clause.) In some cases if you are using an expression, KB_SQL will inform you that it can’t match the value in the BY clause to one of the arguments in the ORDER BY clause. This may occur with some expressions and virtual columns. When this does happen we suggest you use a positional reference to the ORDER BY clause.

IMPORTANT: If zero (0) is used for the ORDER BY key, the function will return a value based on all of the result rows.

- If there is no ORDER BY clause, and a GROUP BY clause exists, the function is reset each time any of the GROUP BY values change. The BY clause is not allowed with a GROUP BY clause. If there is an ORDER BY...
clause and a GROUP BY clause, the BY clause must be used to refer to functions that are reset by changes in the ORDER BY keys. In this situation, a GROUP BY set function may be nested within an ORDER BY set function. This phenomenon is allowed because the GROUP BY clause and all GROUP BY set functions are computed before the ORDER BY clause is executed. An ORDER BY set function cannot be nested within a GROUP BY set function or another ORDER BY set function.

- If there is no ORDER BY or GROUP BY clause, using a set function will cause the query to return one row, which is computed from all result rows.

The BY key word may be replaced with a comma ',' character.

**AVG**

**AVG([DISTINCT] value [BY order_by_key])**

Averages all non-null values, or distinct non-null values.

**Rules**

- The average is computed by the sum of all non-null values (or distinct values) divided by the number of non-null values (or distinct non-null values).
- If the DISTINCT is specified, the function will average only unique non-null values for the value. Otherwise, the function will average all non-null values for the value.
- Formula: \( \text{AVG(value)} = \frac{\text{SUM(value)}}{\text{COUNT(value)}} \)

To compute an average including null values you can use the formula: \( \frac{\text{SUM(value)}}{\text{COUNT(*)}} \).

**Examples**

- **AVG(salary)**
  
The average of all non-null values for the salary column.

- **AVG(salary BY 1)**
  
The average of all non-null values for the salary column for each different value of the first ORDER BY value.

- **AVG(DISTINCT salary)**
  
The average of all distinct non-null values for the salary column.
AVG(DISTINCT salary BY 1)  The average of all distinct non-null values for the salary column for each distinct value of the first ORDER BY value.

COUNT

COUNT(* [BY order_by_key])
Counts all rows.

Rules
- The function will count all result rows.

  COUNT(*)  The number of rows.

  COUNT(* BY 1)  The number of rows for each different value of the first value in the ORDER [,scope] clause.

COUNT([DISTINCT] value [BY order_by_key])
Counts rows, non-null values, or distinct non-null values.

Rules (Also refer to the rules for Aggregate functions.)
- If the DISTINCT is specified, the function will count only unique non-null values for the value. Otherwise, the function will count only non-null values for the value.

Examples
COUNT(manager)  The number of rows with a non-null value for the manager column.

COUNT(manager BY 1)  The number of rows with a non-null value for manager column for each different value of the first ORDER BY value.

COUNT(DISTINCT manager)  The number of distinct non-null values for the manager column.

COUNT(DISTINCT manager BY 1)  The number of different non-null values for the manager column for each different value of the first ORDER BY value.
MAX

MAX(value [BY order_by_key])

Returns the maximum (highest) value for a value.

Rules
- The function will return the largest non-null value for the value. If the value is numeric, then higher numbers are considered larger. For example, 2 is larger than 1, and 0 is larger than -1. If the value has a character data type, then the largest value is the value that collates last. For example, B collates after A.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX(salary)</td>
<td>The maximum (highest) value for the salary column.</td>
</tr>
<tr>
<td>MAX(Salary BY 1)</td>
<td>The maximum (highest) value for the salary column for each different value of the first ORDER BY value.</td>
</tr>
<tr>
<td>MIN(value[BY order_by_key])</td>
<td>Returns the minimum non-null value for a value.</td>
</tr>
</tbody>
</table>

MIN

MIN([DISTINCT] value[,scope])

Returns the smallest value in a set.

Rules
- The function will return the smallest non-null value of value. If the value is numeric, the lowest value is returned. For example, 1 is lower than 2. If the value has a character data type, the result is the lowest collating value. For example, A is lower than B.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN(salary)</td>
<td>The minimum (lowest) non-null value for the salary column.</td>
</tr>
<tr>
<td>MIN(value[BY order_by_key])</td>
<td>Returns the minimum non-null value for a value.</td>
</tr>
<tr>
<td>MIN(salary BY 1)</td>
<td>The minimum (lowest) non-null value for the salary column for each</td>
</tr>
</tbody>
</table>
KBSSQLSyntax

different value of the first ORDER BY value.

SUM

SUM([DISTINCT] value [BY order_by_key])

Totals all non-null values for value, or totals all distinct non-null values.

Rules
• If the DISTINCT is specified, the function will total only unique non-null values for the value. Otherwise, the function will total all non-null values for the value.

Examples

SUM(salary) The total of all non-null values for the salary column.
SUM(salary BY 1) The total of all non-null values for the salary column for each different value of the first ORDER BY value.
SUM(DISTINCT salary) The total of all distinct non-null values for the salary column.
SUM(DISTINCT salary BY 1) The total of all distinct non-null values for the salary column for each different value of the first ORDER BY value.

Row Functions

Data Type Conversion Functions

These row functions are used to convert a value’s data type to something other than what it is in the data dictionary. They can also be used to format something that does not have a format.

CHARACTER

CHARACTER(value[,length])

Cast data type of value as CHARACTER data type.

DATE

DATE(value)

Cast the value as data type DATE.
DATE_ONLY

DATE_ONLY(moment_expression)

Return date portion of a moment value.

FLAG

FLAG(value)

Cast data type of value as FLAG.

INTEGER

INTEGER(value [,length])

Cast data type of value as INTEGER.

MOMENT

MOMENT(value)

Cast data type of value as MOMENT.

NUMERIC

NUMERIC(value [,length [,scale]])

Cast data type of value as NUMERIC.

TIME

TIME(value)

Cast data type of value as TIME.

TIME_ONLY

TIME_ONLY(moment_expression)

Return time portion of a moment value.

Formatting Functions

TO_DATE
KBSSQLSyntax

TO_DATE(date_exp[,format])
Formats a value of DATE data type.

TO_FLAG

TO_FLAG(value,true,false)
Formats a value of FLAG data type.

TO_MOMENT

TO_MOMENT(value,[date_format,[time_format]])
Formats a value of MOMENT data type.

TO_NUMERIC

TO_NUMERIC(value,format [,decimal])
Formats a value of NUMERIC data type.

TO_TIME

TO_TIME(value,format)
Formats a value of TIME data type.

Miscellaneous Row Functions

ASCII

ASCII(value[,position])
Returns the ASCII code corresponding to a character.

CHR

CHR(integer)
Return the character corresponding to the ASCII code value.

COALESCE

COALESCE(value[,value]...)
Returns the first non-null value.

**EXTRACT**

**EXTRACT(value, from [,to])**

Returns subset of characters from string.

**FIND**

**FIND(value,string[,start])**

Returns integer position of character after string.

**LENGTH**

**LENGTH(value[,value2])**

Returns the number of characters in string.

**LPAD**

**LPAD(value, length [,character])**

Returns left-justified string.

**PIECE**

**PIECE(value,delimiter[,from [,to]]**

Returns the substring of a value using a delimiter.

**ROUND**

**ROUND(value [,decimal])**

Returns value rounded to n places to right of decimal.

**RPAD**

**RPAD(value, length [,character])**

Returns a right-justified string.

**SUBSTRING**
KBSSQL Syntax

SUBSTRING(value, from, length)
Returns a subset of characters from a string.

TRANSLATE
TRANSLATE(value, from [,to])
Converts character values in a string.

Miscellaneous Functions

SQL_FN_ABS
SQL_FN_ABS(numeric_exp)
Returns absolute value.

SQL_FN_ASCII
SQL_FN_ASCII(string_exp)
Returns the ASCII code value of the leftmost character.

SQL_FN_CEILING
SQL_FN_CEILING(numeric_exp)
Returns smallest integer greater or equal to value.

SQL_FN_CHAR
SQL_FN_CHAR(code)
Returns the character of the ASCII code.

SQL_FN_CONCAT
SQL_FN_CONCAT(string_exp1, string_exp2)
Concatenates two values using SQL rules.

SQL_FN_CURDATE
returns DATE
Returns the current date.

**SQL_FN_CURTIME**
returns TIME

Returns the current time value.

**SQL_FN_DATABASE**
returns CHARACTER

Returns the name of the SQL Database.

**SQL_FN_DAYNAME**

**SQL_FN_DAYNAME**(date_exp)

Returns day name.

**SQL_FN_DAYOFMONTH**

**SQL_FN_DAYOFMONTH**(date_exp)

Returns day of month based on date value.

**SQL_FN_DAYOFWEEK**

**SQL_FN_DAYOFWEEK**(date_exp)

Returns day of week (1=Sunday).

**SQL_FN_DAYOFYEAR**

**SQL_FN_DAYOFYEAR**(date_exp)

Returns day of year.

**SQL_FN_DIFFERENCE**

**SQL_FN_DIFFERENCE**(string_exp1, string_exp2)

Difference in Soundex codes.

**SQL_FN_FLOOR**
KBSSQLSyntax

**SQL_FN_FLOOR(numeric_exp)**
Returns largest integer less than or equal to value.

**SQL_FN_HOUR**
**SQL_FN_HOUR(time_exp)**
Returns hour for time value.

**SQL_FN_IFNULL**
**SQL_FN_IFNULL(exp,value)**
Returns NULL or value.

**SQL_FN_INSERT**
**SQL_FN_INSERT(string_exp1, start, length, string_exp2)**
Overwrite characters in string.

**SQL_FN_LCASE**
**SQL_FN_LCASE(string_exp)**
Convert to lowercase.

**SQL_FN_LEFT**
**SQL_FN_LEFT(string_exp, count)**
Returns the leftmost characters in string.

**SQL_FN_LENGTH**
**SQL_FN_LENGTH(string_exp)**
Number of characters in string less trailing blanks.

**SQL_FN_LOCATE**
**SQL_FN_LOCATE(exp1,exp2[,exp3])**
Returns starting position of exp1 in exp2.
SQL_FNLTRIM

SQL_FNLTRIM(string_exp)
Strip leading blanks.

SQL_FN_MINUTE

SQL_FN_MINUTE(time_exp)
Returns minute of hour.

SQL_FN_MOD

SQL_FN_MOD(integer_exp1,integer_exp2)
Returns remainder (modulus) of integer_exp1 divided by integer_exp2.

SQL_FN_MONTH

SQL_FN_MONTH(date_exp)
Returns month based on date value.

SQL_FN_MONTHNAME

SQL_FN_MONTHNAME(date_exp)
Returns name of month.

SQL_FNEXT_MONTH_FIRST

SQL_FNEXT_MONTH_FIRST(<DateValue>)
Returns the date value of next first of month.

SQL_FNEXT_MONTH_LAST

SQL_FNEXT_MONTH_LAST
Returns the date value of the next month end date.

SQL_FNEXT_WEEKDAY

SQL_FNEXT_WEEKDAY(<DateValue>)
KBSSQLSyntax

Returns the date value of the next weekday.

**SQL_FN_NEXT_WEEKEND_DAY**

**SQL_FN_NEXT_WEEKEND_DAY(<DateValue>)**

Returns date value of next weekend day.

**SQL_FN_NOW**

returns MOMENT

Returns the current timestamp value.

**SQL_FN_PI**

returns NUMERIC

Returns the constant value of pi.

**SQL_FNPOWER**

**SQL_FNPOWER(numeric_exp,integer_exp)**

Returns value raised to power.

**SQL_FN_QUARTER**

**SQL_FN_QUARTER(date_exp)**

Returns quarter (1-4) for date value.

**SQL_FN_RAND**

**SQL_FN_RAND(integer_exp)**

Returns random number using value as seed.

**SQL_FN_REPEAT**

**SQL_FN_REPEAT(string_exp, count)**

Repeat string n-times.

**SQL_FN_REPLACE**
**Appendix**

**SQL_FN_REPLACE(string_exp1,string_exp2,string_exp3)**

Replace all occurrences of string_exp2 in string_exp1 with string_exp3.

**SQL_FN_RIGHT**

**SQL_FN_RIGHT(string_exp, count)**

Returns the rightmost count of characters from string.

**SQL_FN_ROUND**

**SQL_FN_ROUND(numeric_exp,integer_exp)**

Returns value rounded to n-places to right of decimal.

**SQL_FN_RTRIM**

**SQL_FN_RTRIM(string_exp)**

Returns the characters of string without trailing blanks.

**SQL_FN_SECOND**

**SQL_FN_SECOND(time_exp)**

Returns seconds for time value.

**SQL_FN_SIGN**

**SQL_FN_SIGN(numeric_exp)**

Returns -1 (neg), 0 (zero), or 1 (positive).

**SQL_FN_SOUNDEX**

**SQL_FN_SOUNDEX(string_exp)**

Returns soundex value of string.

**SQL_FN_SPACE**

**SQL_FN_SPACE(count)**

Returns string of n-spaces.
KBSSQLSyntax

**SQL_FN_SQRT**

**SQL_FN_SQRT(float_exp)**

Returns the square root of value.

**SQL_FN_SUBSTRING**

**SQL_FN_SUBSTRING(string_exp,start,length)**

Returns sub-component of string value.

**SQL_FN_TIMESTAMPADD**

**SQL_FN_TIMESTAMPADD(interval,integer_exp,timestamp_exp)**

Add interval to timestamp.

**SQL_FN_TIMESTAMPDIFF**

**SQL_FN_TIMESTAMPDIFF(interval,timestamp_exp1,timestamp_exp2)**

Returns integer number of intervals between m1 and m2.

**SQL_FN_TIMESTAMPPART**

**SQL_FN_TIMESTAMPPART(<Part>,<TimeStamp>)**

Returns a specified part of a timestamp.

**SQL_FN_TRUNCATE**

**SQL_FN_TRUNCATE(numeric_exp,integer_exp)**

Truncates numeric value.

**SQL_FN_UCASE**

**SQL_FN_UCASE(string_exp)**

Converts all lowercase characters to uppercase.

**SQL_FN_USER**

returns CHARACTER
Returns the user’s authorization name.

**SQL_FN_WEEK**

**SQL_FN_WEEK(date_exp)**

Returns week of year based on date value.

**SQL_FN_XML_ESCAPE**

**SQL_FN_XML_ESCAPE(string)**

Returns a string containing the XML-escaped version of the string.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>;</td>
<td>'</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

**Example**

```
SQL_FN_ESCAPE('1 < 2') = 1 &lt; 2
```

**SQL_FN_XML_VALUE**

**SQL_FN_XML_VALUE(tag_name, value)**

Returns a well-formed XML tag containing the specified value. Also performs SQL_FN_XML_ESCAPE on value.

**Example**

```
SQL_FN_XML_VALUE('EMP_SSN', '111-22-3333') = <EMP_SSN>111-22-3333</EMP_SSN>
```

**SQL_FN_YEAR**

**SQL_FN_YEAR(date_exp)**

Returns year of date value.
KBSQLSyntax

**SQL_INT_TO_MASK**

**SQL_INT_TO_MASK**(integer)

Convert integer value to BitString.

**SQL_M_EXECUTE**

**SQL_M_EXECUTE**("D tag^rtn")

Execute M code; returns TRUE if successful, FALSE if failed.

**Pseudocolumns**

A system-wide variable that can be referenced by name. KB_SQL supplies you with the following pseudo columns. Your DBA has the ability to add site-specific pseudo columns.

**Rules**

- The value of a pseudo column is independent of any table row.

**Example**

```
SELECT proj_no, task_no, hours
FROM charges
WHERE chg_date = TODAY
```

**DATE_TIME_NOW**

returns MOMENT

Returns the current date and time.

**JOB_ID**

returns INTEGER

Returns the M $JOB value for the current process.

**KBSUID**

returns CHARACTER

Returns the unique identifier for the KB_SQL installation.

**NOW**
returns TIME

Returns the current time.

PAGE_NUMBER

returns INTEGER

Returns the current page number.

SQL_PRODUCT

returns CHARACTER

Returns the product name.

SQL_PRODUCT_VERSION

returns CHARACTER

Returns the product name and version number.

SQL_PRODUCT_VERSION_BUILD

returns CHARACTER

Returns the product version with build number.

SQL_QUERYNAME

returns CHARACTER

Returns the current query name.

SQL_VERSION

returns CHARACTER

Returns the product version number.

SYS_ADDRESS1

returns CHARACTER

Returns the first line of the site address.
SYS_ADDRESS2

returns CHARACTER

Returns the second line of the site address.

SYS_ADDRESS3

returns CHARACTER

Returns the third line of the site address.

SYS_ADDRESS4

returns CHARACTER

Returns the fourth line of the site address.

SYS_NAME

returns CHARACTER

Returns the site name.

SYS_NODE

returns CHARACTER

Returns the internal number of the system.

TODAY

returns DATE

Returns the current date.

USER

returns CHARACTER

Returns the name of the current user.

USER_GROUP

returns CHARACTER
Returns the name of the current user group.
# Index

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSOLUTE</td>
<td>BASE</td>
</tr>
<tr>
<td>ACCESS</td>
<td>BEGIN</td>
</tr>
<tr>
<td>ADA</td>
<td>BETWEEN</td>
</tr>
<tr>
<td>ADD</td>
<td>BIT</td>
</tr>
<tr>
<td>ADD COLUMN</td>
<td>BIT_LENGTH</td>
</tr>
<tr>
<td>AFTER</td>
<td>BitString</td>
</tr>
<tr>
<td>Aggregate</td>
<td>BMARGIN</td>
</tr>
<tr>
<td>Alias</td>
<td>Boolean</td>
</tr>
<tr>
<td>ALL</td>
<td>BREAK</td>
</tr>
<tr>
<td>ALLOCATE</td>
<td>BREAK AFTER</td>
</tr>
<tr>
<td>ALTER</td>
<td>BREAK AT</td>
</tr>
<tr>
<td>ALTER COLUMN column_name</td>
<td>BY</td>
</tr>
<tr>
<td>ALTER COLUMN column_name RENAME</td>
<td>BY order_by_key</td>
</tr>
<tr>
<td>ALTER FOREIGN KEY foreign_key</td>
<td></td>
</tr>
<tr>
<td>ALTER FOREIGN KEY foreign_key RENAME</td>
<td></td>
</tr>
<tr>
<td>ALTER INDEX</td>
<td></td>
</tr>
<tr>
<td>ALTER PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>ALTER SCHEMA</td>
<td></td>
</tr>
<tr>
<td>ALTER TABLE</td>
<td>51, 53</td>
</tr>
<tr>
<td>ALTER VIEW</td>
<td>51, 56</td>
</tr>
<tr>
<td>AND</td>
<td>11, 23, 77, 80</td>
</tr>
<tr>
<td>ANSI</td>
<td>15, 59</td>
</tr>
<tr>
<td>ANY</td>
<td>23, 77</td>
</tr>
<tr>
<td>ARE</td>
<td>77</td>
</tr>
<tr>
<td>AS</td>
<td>56, 61, 69, 77</td>
</tr>
<tr>
<td>ASC</td>
<td>77</td>
</tr>
<tr>
<td>ASCII</td>
<td>75, 77, 90, 92</td>
</tr>
<tr>
<td>ASCII Chart</td>
<td>75</td>
</tr>
<tr>
<td>ASCII Code</td>
<td>75</td>
</tr>
<tr>
<td>ASSERT</td>
<td>77</td>
</tr>
<tr>
<td>ASSERTION</td>
<td>77</td>
</tr>
<tr>
<td>Assign</td>
<td>47</td>
</tr>
<tr>
<td>AT</td>
<td>77</td>
</tr>
<tr>
<td>AUTHORIZATION</td>
<td>59, 77</td>
</tr>
<tr>
<td>AVG</td>
<td>69, 77, 85</td>
</tr>
<tr>
<td>AVG DISTINCT</td>
<td>60</td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>CASCADE</td>
<td>77</td>
</tr>
<tr>
<td>CASE</td>
<td>14, 77</td>
</tr>
<tr>
<td>CAST</td>
<td>77</td>
</tr>
<tr>
<td>CATALOG</td>
<td>77</td>
</tr>
<tr>
<td>CENTER</td>
<td>28, 29, 77</td>
</tr>
<tr>
<td>CHANGED</td>
<td>28, 29, 77</td>
</tr>
<tr>
<td>CHAR</td>
<td>12, 77</td>
</tr>
<tr>
<td>CHAR_LENGTH</td>
<td>77</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>15, 77, 88</td>
</tr>
<tr>
<td>CHARACTER_LENGTH</td>
<td>77</td>
</tr>
<tr>
<td>CHECK</td>
<td>77</td>
</tr>
<tr>
<td>CHR</td>
<td>77, 90</td>
</tr>
<tr>
<td>CLOSE</td>
<td>77</td>
</tr>
<tr>
<td>COALESCE</td>
<td>77, 90</td>
</tr>
<tr>
<td>COBOL</td>
<td>77</td>
</tr>
<tr>
<td>COLLATE</td>
<td>77</td>
</tr>
<tr>
<td>COLLATION</td>
<td>77</td>
</tr>
<tr>
<td>COLUMN</td>
<td>9, 28, 77</td>
</tr>
<tr>
<td>Column Modifiers</td>
<td>28</td>
</tr>
<tr>
<td>Column_name</td>
<td>53, 56, 61</td>
</tr>
<tr>
<td>COMMA</td>
<td>17, 41, 67</td>
</tr>
<tr>
<td>Comments</td>
<td>10</td>
</tr>
<tr>
<td>COMMIT</td>
<td>77</td>
</tr>
<tr>
<td>Condition</td>
<td>11</td>
</tr>
<tr>
<td>Conditional Statements</td>
<td>33</td>
</tr>
<tr>
<td>CONFIGURATION</td>
<td>60</td>
</tr>
<tr>
<td>CONNECT</td>
<td>77</td>
</tr>
<tr>
<td>CONNECTION</td>
<td>77</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>77</td>
</tr>
<tr>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>77</td>
</tr>
<tr>
<td>CONVERT</td>
<td>77, 88</td>
</tr>
<tr>
<td>CORRESPONDING</td>
<td>77</td>
</tr>
<tr>
<td>COUNT</td>
<td>77, 85, 86</td>
</tr>
<tr>
<td>CREATE</td>
<td>58, 61, 77</td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>57, 58</td>
</tr>
<tr>
<td>CREATE PROCEDURE</td>
<td>59</td>
</tr>
<tr>
<td>CREATE SCHEMA</td>
<td>57, 59</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>57, 60, 67</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>56, 57, 61</td>
</tr>
<tr>
<td>CSSPACE</td>
<td>28, 39, 77</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>76</td>
</tr>
<tr>
<td>CURRENT</td>
<td>77</td>
</tr>
<tr>
<td>CURRENT_DATE</td>
<td>77</td>
</tr>
<tr>
<td>CURRENT_TIME</td>
<td>77</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP</td>
<td>77</td>
</tr>
<tr>
<td>CURSOR</td>
<td>77</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D MONTH</td>
<td>76</td>
</tr>
<tr>
<td>Data Type Conversion Functions</td>
<td>88</td>
</tr>
<tr>
<td>Data Type Formats</td>
<td>76</td>
</tr>
<tr>
<td>Data Types</td>
<td>12, 76</td>
</tr>
<tr>
<td>Data_type</td>
<td>53, 60</td>
</tr>
<tr>
<td>DATE</td>
<td>15, 77, 88, 89, 102</td>
</tr>
<tr>
<td>DATE AT TIME</td>
<td>76</td>
</tr>
<tr>
<td>DATE_ONLY</td>
<td>77, 88</td>
</tr>
<tr>
<td>DATE_TIME_NOW</td>
<td>77, 100</td>
</tr>
<tr>
<td>DAY</td>
<td>77</td>
</tr>
<tr>
<td>DBA</td>
<td>76, 77</td>
</tr>
<tr>
<td>DBMS</td>
<td>77</td>
</tr>
<tr>
<td>DD/MM/YYYY</td>
<td>77</td>
</tr>
<tr>
<td>DD/MM/YYYYYYYY</td>
<td>76</td>
</tr>
<tr>
<td>DEALLOCATE</td>
<td>77</td>
</tr>
<tr>
<td>DEC</td>
<td>12, 77</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>12, 77</td>
</tr>
<tr>
<td>DECLARE</td>
<td>13, 77</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>28, 29, 77</td>
</tr>
<tr>
<td>DEFERRABLE</td>
<td>77</td>
</tr>
<tr>
<td>DEFERRED</td>
<td>77</td>
</tr>
<tr>
<td>DELETE</td>
<td>9, 67, 77</td>
</tr>
<tr>
<td>DELETE FROM</td>
<td>67</td>
</tr>
<tr>
<td>DELIMITED</td>
<td>17, 38, 41</td>
</tr>
<tr>
<td>DELIMITER</td>
<td>38, 39, 60</td>
</tr>
<tr>
<td>DESC</td>
<td>69, 77</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>77</td>
</tr>
<tr>
<td>DESCRIPTOR</td>
<td>77</td>
</tr>
<tr>
<td>DETAIL</td>
<td>28, 35, 77</td>
</tr>
<tr>
<td>DEVICE</td>
<td>28, 35, 77</td>
</tr>
<tr>
<td>DIAGNOSTICS</td>
<td>40, 77</td>
</tr>
<tr>
<td>DICTIONARY</td>
<td>77</td>
</tr>
<tr>
<td>DIF</td>
<td>17, 41, 67</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>77</td>
</tr>
<tr>
<td>DISPLACEMENT</td>
<td>77</td>
</tr>
<tr>
<td>DISPLAY_END</td>
<td>40</td>
</tr>
<tr>
<td>DISPLAY_HEADER</td>
<td>40</td>
</tr>
<tr>
<td>DISPLAY_LINE</td>
<td>40</td>
</tr>
<tr>
<td>DISPLAY_PAGE</td>
<td>40</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>69, 77, 86</td>
</tr>
<tr>
<td>DOES</td>
<td>77</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>77</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>77</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>12</td>
</tr>
<tr>
<td>DROP</td>
<td>63, 77</td>
</tr>
<tr>
<td>DROP COLUMN</td>
<td>53</td>
</tr>
<tr>
<td>DROP COLUMN column_name</td>
<td>53</td>
</tr>
<tr>
<td>DROP FOREIGN KEY</td>
<td></td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>63</td>
</tr>
<tr>
<td>DROP PROCEDURE</td>
<td>64</td>
</tr>
<tr>
<td>DROP SCHEMA</td>
<td>63, 64</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>63, 64</td>
</tr>
<tr>
<td>DROP VIEW</td>
<td>56, 63, 65</td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>Effect</td>
<td>14</td>
</tr>
<tr>
<td>ELAPSED_TIME</td>
<td>76</td>
</tr>
<tr>
<td>ELSE</td>
<td>32, 33, 77</td>
</tr>
<tr>
<td>ELSEIF</td>
<td>32, 33, 77</td>
</tr>
<tr>
<td>END</td>
<td>14, 77</td>
</tr>
<tr>
<td>ENDIF</td>
<td>32, 33, 77</td>
</tr>
<tr>
<td>Equal</td>
<td>14, 23</td>
</tr>
<tr>
<td>Equijoin</td>
<td>23</td>
</tr>
<tr>
<td>ESCAPE</td>
<td>77</td>
</tr>
<tr>
<td>Event Blocks</td>
<td>32</td>
</tr>
<tr>
<td>Event Statements</td>
<td>37</td>
</tr>
<tr>
<td>Event_statement</td>
<td>34, 35, 36</td>
</tr>
<tr>
<td>EXCEPTION</td>
<td>77</td>
</tr>
<tr>
<td>EXCEPTION</td>
<td>77</td>
</tr>
<tr>
<td>EXEC</td>
<td>77</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>77</td>
</tr>
<tr>
<td>Execute M</td>
<td>100</td>
</tr>
<tr>
<td>EXISTS</td>
<td>23, 77</td>
</tr>
<tr>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td>LOCAL</td>
<td>77</td>
</tr>
<tr>
<td>LOOKUP</td>
<td>19</td>
</tr>
<tr>
<td>LOWER</td>
<td>76</td>
</tr>
<tr>
<td>LPAD</td>
<td>77, 91</td>
</tr>
<tr>
<td>M</td>
<td>76</td>
</tr>
<tr>
<td>M42</td>
<td></td>
</tr>
<tr>
<td>M Variable</td>
<td>17</td>
</tr>
<tr>
<td>MATCH</td>
<td>35, 77</td>
</tr>
<tr>
<td>MAX</td>
<td>77, 86</td>
</tr>
<tr>
<td>MAX_KEYS</td>
<td>42</td>
</tr>
<tr>
<td>MERGE</td>
<td>77</td>
</tr>
<tr>
<td>MERGE INTO</td>
<td>53, 56</td>
</tr>
<tr>
<td>METHOD</td>
<td>17, 77</td>
</tr>
<tr>
<td>MIN</td>
<td>77, 86, 87</td>
</tr>
<tr>
<td>MINUS</td>
<td>77</td>
</tr>
<tr>
<td>MINUTE</td>
<td>77</td>
</tr>
<tr>
<td>MINUTES</td>
<td>76</td>
</tr>
<tr>
<td>MM.SS12</td>
<td>76</td>
</tr>
<tr>
<td>MM/DD/YY</td>
<td>76</td>
</tr>
<tr>
<td>MM/DD/YYYY</td>
<td>76</td>
</tr>
<tr>
<td>MM12</td>
<td>76</td>
</tr>
<tr>
<td>MM24</td>
<td>76</td>
</tr>
<tr>
<td>MODIFY</td>
<td>77</td>
</tr>
<tr>
<td>MODULE</td>
<td>77</td>
</tr>
<tr>
<td>Modulus</td>
<td>95</td>
</tr>
<tr>
<td>MOMENT</td>
<td>12, 76, 77, 89, 90</td>
</tr>
<tr>
<td>MON</td>
<td>76</td>
</tr>
<tr>
<td>MON DD</td>
<td>76</td>
</tr>
<tr>
<td>MON YYYY</td>
<td>76</td>
</tr>
<tr>
<td>MONTH</td>
<td>76, 77</td>
</tr>
<tr>
<td>MONTH D</td>
<td>76</td>
</tr>
<tr>
<td>MONTH YYYY</td>
<td>76</td>
</tr>
<tr>
<td>MULTIPLE_UPDATES</td>
<td>42</td>
</tr>
<tr>
<td>MUMPS</td>
<td>77</td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>NAMES</td>
<td>18, 77</td>
</tr>
<tr>
<td>NATIONAL</td>
<td>77</td>
</tr>
<tr>
<td>NCHAR</td>
<td>77</td>
</tr>
<tr>
<td>NEWLINE</td>
<td>34, 36, 77</td>
</tr>
<tr>
<td>NEWPAGE</td>
<td>34, 36, 77</td>
</tr>
<tr>
<td>NEXT</td>
<td>77</td>
</tr>
<tr>
<td>NONE</td>
<td>28, 77</td>
</tr>
<tr>
<td>NOT LIKE</td>
<td>23</td>
</tr>
<tr>
<td>NOT NULL</td>
<td>53, 60</td>
</tr>
<tr>
<td>NOW</td>
<td>77, 100</td>
</tr>
<tr>
<td>NULL</td>
<td>14, 23, 69, 77</td>
</tr>
<tr>
<td>NULLIF</td>
<td>77</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>12, 15, 76, 77, 89, 90</td>
</tr>
<tr>
<td>OCTET_LENGTH</td>
<td>77</td>
</tr>
<tr>
<td>OF</td>
<td>77</td>
</tr>
<tr>
<td>OFF</td>
<td>77</td>
</tr>
<tr>
<td>ON</td>
<td>77</td>
</tr>
<tr>
<td>ON/OFF</td>
<td>76</td>
</tr>
<tr>
<td>ONLY</td>
<td>77</td>
</tr>
<tr>
<td>Only DATE</td>
<td>80</td>
</tr>
<tr>
<td>Only INTEGER</td>
<td>80</td>
</tr>
<tr>
<td>OPEN</td>
<td>77</td>
</tr>
<tr>
<td>Operators</td>
<td>80</td>
</tr>
<tr>
<td>OPTION</td>
<td>77</td>
</tr>
<tr>
<td>OR</td>
<td>11, 23, 77</td>
</tr>
<tr>
<td>ORDER</td>
<td>43, 69, 77, 82</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>34, 35, 82, 84, 86</td>
</tr>
<tr>
<td>OUTER</td>
<td>77</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>77</td>
</tr>
<tr>
<td>OVERLAPS</td>
<td>77</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>PACK</td>
<td>60</td>
</tr>
<tr>
<td>PACK key</td>
<td></td>
</tr>
<tr>
<td>PAGE</td>
<td>31, 77</td>
</tr>
<tr>
<td>PAGE_LENGTH</td>
<td>43</td>
</tr>
<tr>
<td>PAGE_NUMBER</td>
<td>77, 101</td>
</tr>
<tr>
<td>PARAMETERS</td>
<td>43</td>
</tr>
<tr>
<td>PARTIAL</td>
<td>77</td>
</tr>
<tr>
<td>PASCAL</td>
<td>77</td>
</tr>
<tr>
<td>Performing</td>
<td>56</td>
</tr>
<tr>
<td>PIECE</td>
<td>77, 91</td>
</tr>
<tr>
<td>PLI</td>
<td>77</td>
</tr>
<tr>
<td>POSITION</td>
<td>77</td>
</tr>
<tr>
<td>POST_RUN_ROUTINE</td>
<td>43</td>
</tr>
<tr>
<td>PRE_RUN_ROUTINE</td>
<td>43</td>
</tr>
<tr>
<td>PRECISION</td>
<td>77</td>
</tr>
<tr>
<td>PREPARE</td>
<td>77</td>
</tr>
<tr>
<td>PRESERVE</td>
<td>77</td>
</tr>
<tr>
<td>PREVIOUS</td>
<td>77</td>
</tr>
<tr>
<td>PRIMARY</td>
<td>77</td>
</tr>
<tr>
<td>Primary Key</td>
<td>19</td>
</tr>
<tr>
<td>PRIOR</td>
<td>77</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>77</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>19, 77</td>
</tr>
<tr>
<td>PROMPT</td>
<td>19, 77</td>
</tr>
<tr>
<td>Pseudocolumns</td>
<td>100</td>
</tr>
<tr>
<td>Term</td>
<td>Page Numbers</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>47, 77</td>
</tr>
<tr>
<td>Q</td>
<td>77</td>
</tr>
<tr>
<td>QUERY</td>
<td>77</td>
</tr>
<tr>
<td>RAVG</td>
<td>77</td>
</tr>
<tr>
<td>RCOUNT</td>
<td>77</td>
</tr>
<tr>
<td>READ</td>
<td>19, 37, 67, 77</td>
</tr>
<tr>
<td>REAL</td>
<td>12, 77</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>53, 60, 77</td>
</tr>
<tr>
<td>RENAME</td>
<td>52, 53, 56, 77</td>
</tr>
<tr>
<td>REPLACE</td>
<td>77</td>
</tr>
<tr>
<td>REPORT</td>
<td>38, 41</td>
</tr>
<tr>
<td>RESTRICT</td>
<td>77</td>
</tr>
<tr>
<td>RESULT_LIMIT</td>
<td>43</td>
</tr>
<tr>
<td>Results</td>
<td>67, 84</td>
</tr>
<tr>
<td>REVOKE</td>
<td>48, 49, 77</td>
</tr>
<tr>
<td>REVOKE_ALL</td>
<td>27</td>
</tr>
<tr>
<td>RIGHT</td>
<td>28, 31, 77</td>
</tr>
<tr>
<td>RMARGIN</td>
<td>43</td>
</tr>
<tr>
<td>RMAX</td>
<td>77</td>
</tr>
<tr>
<td>RMIN</td>
<td>77</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>77</td>
</tr>
<tr>
<td>ROUND</td>
<td>77, 91</td>
</tr>
<tr>
<td>ROW</td>
<td>77</td>
</tr>
<tr>
<td>ROW_COMMIT</td>
<td>44</td>
</tr>
<tr>
<td>ROWID</td>
<td>77</td>
</tr>
<tr>
<td>ROWNUM</td>
<td>77</td>
</tr>
<tr>
<td>ROWS</td>
<td>77</td>
</tr>
<tr>
<td>RPAD</td>
<td>77, 91</td>
</tr>
<tr>
<td>RSUM</td>
<td>77</td>
</tr>
<tr>
<td>RUN</td>
<td>37, 45, 77</td>
</tr>
<tr>
<td>Running</td>
<td>37</td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>SCHEMA</td>
<td>44, 77</td>
</tr>
<tr>
<td>SCROLL</td>
<td>77</td>
</tr>
<tr>
<td>SEARCH_LIMIT</td>
<td>38, 44</td>
</tr>
<tr>
<td>SECOND</td>
<td>77</td>
</tr>
<tr>
<td>SECONDS</td>
<td>76</td>
</tr>
<tr>
<td>SECTION</td>
<td>77</td>
</tr>
<tr>
<td>SELECT</td>
<td>22, 28, 32, 69, 77, 82</td>
</tr>
<tr>
<td>SELECT_LIMIT</td>
<td>38, 44</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>77</td>
</tr>
<tr>
<td>SET</td>
<td>22, 32, 37, 72, 77</td>
</tr>
<tr>
<td>SET DELIMITER</td>
<td>38</td>
</tr>
<tr>
<td>SET DISPLAY_PAGE</td>
<td>40</td>
</tr>
<tr>
<td>SET FILE</td>
<td>29, 38, 41</td>
</tr>
<tr>
<td>SET FILE_TYPE</td>
<td>38</td>
</tr>
<tr>
<td>SET FORMAT</td>
<td>17, 38, 41</td>
</tr>
<tr>
<td>Set Functions</td>
<td>84</td>
</tr>
<tr>
<td>SET INITIAL_ROUTINE</td>
<td>42</td>
</tr>
<tr>
<td>SET RMARGIN</td>
<td>38</td>
</tr>
<tr>
<td>SET SEARCH_LIMIT</td>
<td>38</td>
</tr>
<tr>
<td>SET SHOW_PLAN</td>
<td>38</td>
</tr>
<tr>
<td>SET SHOW_PLAN_ON</td>
<td>38</td>
</tr>
<tr>
<td>SET START_DATE</td>
<td>45</td>
</tr>
<tr>
<td>SET Statement</td>
<td>38</td>
</tr>
<tr>
<td>Show Plan Feature</td>
<td>10</td>
</tr>
<tr>
<td>SHOW_ALL_PLANS</td>
<td>44</td>
</tr>
<tr>
<td>SHOW_COST</td>
<td>44</td>
</tr>
<tr>
<td>SHOW_PLAN</td>
<td>44</td>
</tr>
<tr>
<td>SHOW_STATS</td>
<td>44</td>
</tr>
<tr>
<td>SINGLE</td>
<td>53, 60</td>
</tr>
<tr>
<td>SIZE</td>
<td>77</td>
</tr>
<tr>
<td>SKIP</td>
<td>28, 31, 77</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>12, 77</td>
</tr>
<tr>
<td>SOME</td>
<td>23, 77</td>
</tr>
<tr>
<td>Soundex</td>
<td>93</td>
</tr>
<tr>
<td>SQL</td>
<td>12, 53, 77</td>
</tr>
<tr>
<td>SQL Database</td>
<td>93</td>
</tr>
<tr>
<td>SQL Editor</td>
<td>9</td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td>77</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>77</td>
</tr>
<tr>
<td>SQL_CHAR</td>
<td>77</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>77</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FLOAT</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_ABS</td>
<td>77, 92</td>
</tr>
<tr>
<td>SQL_FN_ASCII</td>
<td>77, 92</td>
</tr>
<tr>
<td>SQL_FN_CEILING</td>
<td>77, 92</td>
</tr>
<tr>
<td>SQL_FN_CONCAT</td>
<td>77, 92</td>
</tr>
<tr>
<td>SQL_FN_CURRENT_DATE</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_CURRENT_TIME</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_CURRENT_TIMESTAMP</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_CURDATE</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_CURDATE</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_DATE</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_DAYNAME</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_DAYOFMONTH</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_DAYOFWEEK</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_DAYOFYEAR</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_CURTIME</td>
<td>77</td>
</tr>
<tr>
<td>SQL_FN_DATABASE</td>
<td>77, 93</td>
</tr>
<tr>
<td>SQL_FN_DAYNAME</td>
<td>77, 93</td>
</tr>
<tr>
<td>SQL_FN_DAYOFMONTH</td>
<td>77, 93</td>
</tr>
<tr>
<td>SQL_FN_DAYOFWEEK</td>
<td>77, 93</td>
</tr>
<tr>
<td>SQL_FN_DAYOFYEAR</td>
<td>77, 93</td>
</tr>
</tbody>
</table>
SQL_FN_DIFFERENCE .......... 77, 93
SQL_FN_ESCAPE ................. 99
SQL_FN_FLOOR .................. 77, 93
SQL_FN_HOUR .................... 77, 94
SQL_FN_IDENTIFIER ............. 77
SQL_FN_IFNULL .................. 77, 94
SQL_FN_INITIAL_CAPS .......... 77
SQL_FN_INSERT ................ 77, 94
SQL_FN_LCASE ................... 77, 94
SQL_FN_LEFT ..................... 77, 94
SQL_FN_LENGTH .................. 77, 94
SQL_FN_LOCATE ................ 77, 94
SQL_FN_LTRIM .................... 77, 94
SQL_FN_MINUTE .................. 77, 95
SQL_FN_MOD ...................... 77, 95
SQL_FN_MONTH ................... 77, 95
SQL_FN_MONTHNAME .......... 77, 95
SQL_FN_NEXT_MONTH_FIRST .... 77, 95
SQL_FN_NEXT_MONTH_LAST ....... 77, 95
SQL_FN_NEXT_WEEKDAY ....... 77, 95
SQL_FN_NEXT_WEEKEND_DAY .... 77, 95
SQL_FN_NOW ...................... 77, 96
SQL_FN_PI ........................ 77, 96
SQL_FN_POWER ................. 77, 96
SQL_FN_QUARTER .............. 77, 96
SQL_FN_RAND .................... 77, 96
SQL_FN_REPEAT ................ 77, 96
SQL_FN_REPLACE .............. 77, 96
SQL_FN_RIGHT ................. 77, 97
SQL_FN_ROUND .................. 77, 97
SQL_FN_RTRIM ................... 77, 97
SQL_FN_SECOND ............... 77, 97
SQL_FN_SIGN .................... 77
SQL_FN_SOUNDEX .............. 77, 97
SQL_FN_SPACE ................... 77, 97
SQL_FN_SQRT ................... 77, 97
SQL_FN_SUBSTRING .......... 77, 98
SQL_FN_TEXT_ROUTINE ......... 77
SQL_FN_TIMESTAMPADD ....... 77, 98
SQL_FN_TIMESTAMPDIFF ....... 77, 98
SQL_FN_TIMESTAMPPART ...... 77, 98
SQL_FN_TRUNCATE ............. 77, 98
SQL_FN_UCASE ................ 77, 98
SQL_FN_USER .................... 77, 98
SQL_FN_WEEK .................... 77, 98
SQL_FN_XML_ESCAPE .......... 77, 99
SQL_FN_XML_VALUE ............ 77, 99
SQL_FN_YEAR .................... 77, 99
SQL_FUNCTION_BY_TYPE ......... 82
SQL_INT_TO_MASK ............. 77, 99
SQL_INTEGER .................... 77
SQL_LONGVARCHAR .............. 77
SQL_M_EXECUTE ............... 77, 100
SQL_NUMERIC ................... 77
SQL_PRODUCT .................. 77, 101
SQL_PRODUCT_VERSION .... 77, 101
SQL_PRODUCT_VERSION_BUILD .......................... 77, 101
SQL_QUERYNAME ............... 77, 101
SQL_REAL ......................... 77
SQL_SMALLINT .................. 77
SQL_TINYINT .................... 77
SQL_TP_FKEY ..................... 77
SQL_TP_HADDR ................. 77
SQL_TP_INDEX_ADDR ........... 77
SQL_TP_VADDR .................. 77
SQL_TYPE_DATE .................. 77
SQL_TYPE_TIME .................. 77
SQL_TYPE_TIMESTAMP .......... 77
SQL_VARCHAR .................... 77
SQL_VERSION ................... 77, 101
SQLCA .......................... 77
SQLCODE .......................... 77
SQLERROR ......................... 77
SQLSTATE .......................... 77
SQLWARNING ..................... 77
START_DATE .................. 38, 45
START_TIME .................. 38, 45
STORED_PROCEDURE .......... 45
Subheaders .................... 35
Subqueries ....................... 37
Subquery .......................... 22, 23
SUBSTRING ..................... 77, 91
SUM .......................... 69, 77, 85, 88
SYNONYM ......................... 77
SYS_ADDRESS1 ................. 77, 101
SYS_ADDRESS2 ................. 77, 101
SYS_ADDRESS3 ................. 77, 102
SYS_ADDRESS4 ................. 77, 102