

RETURN TO MAIN MENU

# **Recital/SQL**

## **Recital/SQL**

Recital Corporation,  
100 Cummings Center, Suite 318J  
Beverly, MA 01915

Recital may have patents and/or patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT ©1988-2006 Recital Corporation. All rights reserved. All Recital products are trademarks or registered trademarks of Recital Corporation, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Last updated August, 2006

## Index

---

OVERVIEW	1
DATA TYPES	2
<b>SYSTEM TABLES</b>	5
SYSACTIVEUSERS	6
SYSBESTROWIDENTIFIER	7
SYSCATALOGS	8
SYSCOLUMNCONSTRAINTS	9
SYSCOLUMNPRIVILEGES	10
SYSCOLUMNS	11
SYSCROSSREFERENCE	12
SYSEXPORTEDKEYS	13
SYSIMPORTEDKEYS	14
SYSINDEXINFO	15
SYSIOSTATS	16
SYSLOGGING	17
SYSPRIMARYKEYS	18
SYSPROCEDURECOLUMNS	19
SYSPROCEDURES	20
SYSRESULTSET	21
SYSSCHEMAS	22
SYSTABLECONSTRAINTS	23
SYSTABLEPRIVILEGES	24
SYSTABLES	25
SYSTABLETYPES	26
SYSTYPEINFO	27
SYSUDTS	28
SYSVERSIONCOLUMNS	29
<b>PSEUDO COLUMNS</b>	30
CURRVAL	31
NEXTVAL	32
ROWID	33
ROWNUM	34
SQLCNT	35
SQLCODE	36
SYNCNUM	37
<b>OPERATORS</b>	38
NUMERIC OPERATORS	39
STRING OPERATORS	40
DATE OPERATORS	41
LOGICAL OPERATORS	42
RELATIONAL OPERATORS	44

<b>PREDICATES</b>	46
BETWEEN	47
IN	48
LIKE	49
NULL	50
<b>TABLE CONSTRAINTS</b>	51
CHECK	52
ERROR	53
FOREIGN KEY	54
INDEX	55
KEY	56
ONCLOSE	57
ONDELETE	58
ONINSERT	59
ONOPEN	60
ONROLLBACK	61
ONUPDATE	62
PRIMARY KEY	63
UNIQUE	64
<b>COLUMN CONSTRAINTS</b>	65
AUTO_INCREMENT	66
AUTOINC	67
CALCULATED	68
CHECK	69
DEFAULT	70
DESCRIPTION	71
ERROR	72
FOREIGN KEY	73
NOCPTRANS	74
NOT NULL	75
NULL	76
PRIMARY KEY	77
RANGE	78
RECALCULATE	79
REFERENCES	80
SET CHECK	81
UNIQUE	82
<b>COMMANDS</b>	
ADD TABLE	83
ALTER INDEX	84
ALTER TABLE	85
BACKUP DATABASE	88
BEGIN TRANSACTION	90
CLOSE	92
CLOSE DATABASES	93
CLOSE TABLES	94
COMMIT	95
COMPILE DATABASE	96
CREATE BRIDGE	97

CREATE CONNECTION	100
CREATE CURSOR	103
CREATE DATABASE	105
CREATE INDEX	106
CREATE PROCEDURE	107
CREATE TABLE	109
CREATE TRIGGER	112
CREATE VIEW	114
DECLARE CURSOR	115
DELETE FROM	116
DELETE TRIGGER	117
DISPLAY DATABASE	118
DISPLAY TABLES	120
DROP BRIDGE	122
DROP CONNECTION	123
DROP CURSOR	124
DROP DATABASE	125
DROP INDEX	126
DROP PROCEDURE	127
DROP TABLE	128
DROP VIEW	129
END TRANSACTION	130
EXEC SQL	132
EXECUTE	133
EXECUTE IMMEDIATE	134
FETCH	135
GRANT	137
INSERT	139
LIST DATABASE	141
LIST TABLES	143
LOCK TABLE	145
OPEN	146
OPEN DATABASE	147
PACK DATABASE	150
PREPARE	151
REBUILD DATABASE	152
REINDEX DATABASE	153
RESTORE DATABASE	154
REVOKE	156
ROLLBACK	158
SAVE TRANSACTION	160
SAVEPOINT	161
SELECT	162
UPDATE	170
USE	172
<b>DATABASE EVENTS</b>	173
DBC_CLOSEDATA	174
DBC_OPENDATA	176

<b>AGGREGATE FUNCTIONS</b>	178
AVG()	179
COUNT()	180
MAX()	181
MIN()	182
SUM()	183
<b>REMOTE DATA CONNECTIVITY FUNCTIONS</b>	184
SQLCANCEL()	185
SQLCOLUMNS()	187
SQLCOMMIT()	189
SQLCONNECT()	191
SQLDISCONNECT()	193
SQLEXEC()	194
SQLGETPROP()	196
SQLMORERESULTS()	197
SQLPREPARE()	199
SQLROLLBACK()	200
SQLSETPROP()	202
SQLSTRINGCONNECT()	204
SQLTABLES()	206
<b>FUNCTIONS</b>	
ADATABASES()	208
BETWEEN()	209
BITAND()	210
BITCLEAR()	211
BITLSHIFT()	212
BITNOT()	213
BITOR()	214
BITRSHIFT()	215
BITSET()	216
BITTEST()	217
BITXOR()	218
CAST()	219
CDOW()	221
CLEARRESULTSET()	222
CMONTH()	223
CONNECTED()	224
CTOT()	225
CURSORNAME()	226
DATABASE()	227
DATE()	228
DATETIME()	229
DAY()	230
DBUSED()	231
DMY()	232
DOW()	233
DTOC()	234
DTOS()	235
EMPTY()	236
FINDCURSOR()	237

GATEWAY()	238
GETRESULTSET()	239
GOMONTH()	240
HOUR()	241
HOURS()	242
ICASE()	243
IF()	244
IIF()	245
ISNULL()	246
LIKE()	247
MINUTE()	248
MINUTES()	249
MONTH()	250
NVL()	251
QUARTER()	252
REFERENCES()	253
SEC()	254
SECONDS()	255
SECS()	256
SEQNO()	257
SETRESULTSET()	258
SQLVALUES()	259
TIME()	260
TTOC()	261
TTOD()	262
TXNISOLATION()	263
TXNLEVEL()	264
TYPE()	265
VARTYPE()	267
YEAR()	269
<b>SET COMMANDS</b>	
SET...	270
SET AUTOCATALOG	271
SET CENTURY	272
SET GATEWAY	273
SET HOURS	274
SET NULL	275
SET NULLDISPLAY	277
SET SECONDS	278
SET SEQNO	279
SET SQL	280
SET SQLPROMPT	282
SET SQLROWID	283
SET TCACHE	284
SET TRANSACTION	286
SET XMLFORMAT	288

## OVERVIEW

---

Recital SQL is SQL ANSI 92 compliant and compatible with both Microsoft® Visual FoxPro® and MySQL™ SQL. Where Recital, MySQL and Visual FoxPro differ in their SQL syntax, the SET SQL TO <dialect> command can be used to select the syntax to be used.

Recital SQL commands may be used interactively at the command prompt in Recital Terminal Developer products, embedded within Recital/4GL programs, issued from '.sql' programs or sent from SQL clients for direct execution by the Recital Database Server.

### Recital Terminal Developer Interactive Mode

Recital SQL statements may be typed interactively at the command prompt. The command SET SQL ON must first be issued. This changes the command prompt to the following: 'Recital/SQL>'.

When in interactive SQL mode, Recital SQL and Recital/4GL commands can be intermixed. The following Recital SQL commands take precedence over their Recital/4GL equivalents: CLOSE, DELETE, INSERT, SELECT, UPDATE

In interactive SQL mode, Recital SQL statements must be terminated with a semi-colon (;). Pressing [RETURN] without terminating the statement with a semi-colon continues the statement onto the next line. NOTE: If SQL is set to VFP, the semi-colon is not required.

The default Recital/SQL> prompt may be changed with the SET SQLPROMPT TO <expC> command.

### Embedded Recital SQL

SQL statements may be embedded in Recital/4GL programs. If SQL is set to the default RECITAL, SQL statements must be preceded by the EXEC SQL command and terminated with a semi-colon (;).

```
// Embedded Recital SQL statement in program
exec sql
select * from accounts
where account_no = "00010";
dialog box [end of select]
//
```

All lines that follow the EXEC SQL line up to the terminating semi-colon are treated as a single SQL statement. SET SQL ON may be used as an alternative to EXEC SQL.

### .sql Programs

Programs consisting entirely of SQL statements may be given a '.sql' file extension to differentiate them from standard Recital/4GL programs (.prg files). When a '.sql' file is run, SQL is automatically set ON and SQL is set to MYSQL. Compiled '.sql' files are given a '.sqo' file extension.

### Database Server execution

Database Server SQL clients such as the Recital Universal JDBC Driver and the Recital Universal ODBC Driver, send individual SQL commands for execution and process the results.

```
rs = stmt.executeQuery("SELECT * from jdbctest");
```

## DATA TYPES

---

### Class

SQL Applications

### See Also

ALTER TABLE, CONSTRAINTS, CREATE TABLE, SELECT, INSERT, UPDATE

### Description

The Recital database engine recognizes the following data types.

Data Types	Description
BIGINT[(p[,s])] [UNSIGNED]	Same as NUMERIC.
BIT	Same as LOGICAL.
CHAR(size) [BINARY]	Fixed length character data of length size bytes. The maximum size is 254 bytes and the minimum size is 1 byte. If BINARY is specified then column values are sorted and compared in case-sensitive fashion according to their ASCII order. If a BINARY flagged column is used in an expression, the whole expression is evaluated as a BINARY value.
CURRENCY	Stores double precision floating numbers corresponding to the double data type in C. The precision is fixed at 25. The scale is fixed at 4. Internal storage is 8 bytes.
DATE	Internal storage size of 4 bytes. Valid date range from January 1, 1900 to January 1, 3000.
DATETIME	Internal storage size of 8 bytes. Valid date range from January 1, 1900 to January 1, 3000. Valid time range from 12:00:00 AM to 11:59:59 PM.
DEC[IMAL][(p[,s])] [UNSIGNED]	Same as NUMERIC.
DOUBLE [PRECISION][(p[,s])] [UNSIGNED]	Same as NUMERIC.
FLOAT[(p[,s])] [UNSIGNED]	Stores double precision floating numbers corresponding to the double data type in C. The precision p can range from 1 to 25. The scale s can range from 0 to 9. The precision and scale only affect display, a FLOAT is always stored internally as 8 bytes. If the scale is omitted, it defaults to 0. If the precision is omitted, it defaults to 25 and the scale defaults to 6. If UNSIGNED is specified, only positive values are allowed.
GENERAL	Same as OBJECT.
INT[EGER][(p[,s])] [UNSIGNED]	Stores whole numbers from -2147483647 to +2147483647. The precision p can range from 1 to 25. The scale s can range from 0 to 9. The precision and scale only affect display, an INTEGER is always stored as 4 bytes. If the scale is omitted, it defaults to 0. If the precision is omitted, it defaults to 11 and the scale defaults to 0. If UNSIGNED is specified, only positive values are allowed.
LOGICAL	Fixed length of 1 byte. This is a Boolean value that takes .T. for True or .F. for False.
LONG VARCHAR	Same as MEMO.
LONG VARBINARY	Same as OBJECT.
MEDIUMINT[(p[,s])] [UNSIGNED]	Stores whole numbers from -2147483647 to +2147483647. The precision p can range from 1 to 25. The scale s can range from 0 to 9. The precision and scale only affect display, an INTEGER is always stored as 4 bytes. If the



	scale is omitted, it defaults to 0. If the precision is omitted, it defaults to 11 and the scale defaults to 0. If UNSIGNED is specified, only positive values are allowed.
MEMO	Variable length character data. Length up to 2 gigabytes, or 2 to the power 31. An 8 byte pointer is stored in the record, the data is stored in an associated file.
NUM[ERIC]([p],[s]) [UNSIGNED]	Stores whole numbers from -2147483647 to +2147483647 depending on p. The precision p can range from 1 to 25. The scale s can range from 0 to 9. The precision and scale affect the internal storage size. If the scale is omitted, it defaults to 0. If the precision is omitted, it defaults to 25 and the scale defaults to 6. If UNSIGNED is specified, only positive values are allowed.
OBJECT	Variable length binary data. Length up to 2 gigabytes, or 2 to the power 31. An 8 byte pointer is stored in the record, the data is stored in an associated file.
REAL([p],[s]) [UNSIGNED]	Stores single precision floating numbers corresponding to the float data type in C. The precision p can range from 1 to 25. The scale s can range from 0 to 9. The precision and scale only affect display, a REAL is always stored internally as 4 bytes. If the scale is omitted, it defaults to 0. If the precision is omitted, it defaults to 25 and the scale defaults to 6. If UNSIGNED is specified, only positive values are allowed..
SHORT([p],[s]) [UNSIGNED]	Stores whole numbers from -32768 to +32767. The precision p can range from 1 to 25. The scale s can range from 0 to 9. The precision and scale only affect display, a SHORT is always stored as 2 bytes. If the scale is omitted, it defaults to 0. If the precision is omitted, it defaults to 6 and the scale defaults to 0. If UNSIGNED is specified, only positive values are allowed.
SMALLINT([p],[s]) [UNSIGNED]	Stores whole numbers from -32768 to +32767. The precision p can range from 1 to 25. The scale s can range from 0 to 9. The precision and scale only affect display, a SMALLINT is always stored as 2 bytes. If the scale is omitted, it defaults to 0. If the precision is omitted, it defaults to 6 and the scale defaults to 0. If UNSIGNED is specified, only positive values are allowed.
TEXT	Same as MEMO.
TIME	An 8 character field for use to store time data in the format "HH:MM:SS".
TIMESTAMP	A 19 character field for use to store timestamp data in the format "YYYY-MM-DD HH:MM:SS".
TINYINT([length]) [UNSIGNED]	Fixed length of 1 byte. Stores whole numbers from 0 to 255. Display width is 3. The (length) and UNSIGNED are included for compatibility reasons only.
VARCHAR(size) [BINARY]	Variable length character data of length size bytes. The maximum size is 254 bytes and the minimum size is 1 byte. If BINARY is specified then column values are sorted and compared in case-sensitive fashion according to their ASCII order. If a BINARY flagged column is used in an expression, the whole expression is evaluated as a BINARY value.

Data Type Abbreviations:

Abbreviations	Data Type	Numeric DATA_TYPE
B	TINYINT/DOUBLE	-6
C	CHARACTER	1
D	DATE	91
F	FLOAT	8
G	LONG VARBINARY/GENERAL	-4
I	INTEGER	2
L	LOGICAL/BIT	-7
M	LONG VARCHAR/MEMO	-4
N	NUMERIC	2
P	PACKED DECIMAL	0
Q	QUAD	0
R	REAL	7
S	SHORT	5
T	DATETIME	93
U	UNKNOWN	0
V	VAXDATE	93
Y	CURRENCY	0
Z	ZONED	0

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSTEM TABLES

---

### Class

SQL Applications

### Purpose

System-defined read-only views

### See Also

SELECT, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement. The following is an alphabetical reference of the System Tables accessible to all users.

Table	Remarks
SYSACTIVEUSERS	Description of currently active users on the system
SYSBESTROWIDENTIFIER	Description of a table's optimal set of columns that uniquely identifies a row
SYSCATALOGS	Catalog names available in the database
SYSCOLUMNCONSTRAINTS	Description of the constraints for a table's columns
SYSCOLUMNPRIVILEGES	Description of the access rights for a table's columns
SYSCOLUMNS	Description of the table columns available in the catalog
SYSCROSSREFERENCE	Description of how one table imports the keys of another table
SYSEXPORTEDKEYS	Description of the foreign key columns that reference the primary key columns
SYSIMPORTEDKEYS	Description of the primary key columns that are referenced by the foreign key
SYSINDEXINFO	Description of a table's indices and statistics
SYSIOSTATS	Facility for monitoring table and index file I/O operations
SYSLOGGING	System Logging information
SYSPRIMARYKEYS	Description of the primary key columns in the table
SYSPROCEDURECOLUMNS	Description of the input, output and results associated with certain stored procedures available
SYSPROCEDURES	Description of the stored procedures available in the catalog
SYSRESULTSET	Used to return the singleton result from any Recital expression
SYSSCHEMAS	Schema names available in the database
SYSTABLECONSTRAINTS	Description of the constraints for each table available in the catalog
SYSTABLEPRIVILEGES	Description of the access rights for each table available in the catalog
SYSTABLES	Description of the tables available in the catalog
SYSTABLETYPES	Table types available in the database system
SYSTYPEINFO	Description of all data types supported by the database
SYSUDTS	Description of the user-defined types (UDTs) defined in the schema
SYSVERSIONCOLUMNS	Description of the columns in a table that are automatically updated when any row is updated

## **SYSACTIVEUSERS**

---

### **Class**

SQL Applications

### **Purpose**

Description of currently active users on the system

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
ACTIVE	N	1	True if Active, False if exited uncleanly
GID	C	8	Group ID
UID	N	8	User ID
PID	C	8	Process ID
USER_NAME	C	25	User name
TERMINAL	N	25	Terminal name
CLIENT	C	25	Client connection
DATE	D	4	Login date
TIME	C	8	Login time

### **Example**

EXEC SQL

```
SELECT user_name
FROM sysactiveusers;
```

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSBESTROWIDENTIFIER**

---

### **Class**

SQL Applications

### **Purpose**

Description of a table's optimal set of columns that uniquely identifies a row

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
SCOPE	N	1	Actual scope of result
COLUMN_NAME	C	30	Column name
DATA_TYPE	N	2	SQL data type
TYPE_NAME	C	30	Data source dependent type name
COLUMN_SIZE	N	3	Precision
BUFFER_LENGTH	N	1	Reserved
DECIMAL_DIGITS	N	2	Scale
PSEUDO_COLUMN	N	1	Is this a pseudo column?

### **Example**

EXEC SQL

```
SELECT column_name
FROM sysbestrowidentifier;
```

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSCATALOGS**

---

**Class**

SQL Applications

**Purpose**

Catalog names available in the database

**See Also**

SELECT, SYSTEM TABLES, DATA TYPES

**Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_CAT	C	30	Catalog name

**Example**

EXEC SQL

SELECT \*

FROM syscatalogs;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSCOLUMNCONSTRAINTS**

---

### **Class**

SQL Applications

### **Purpose**

Description of the constraints for a table's columns

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
COLUMN_NAME	C	32	Column name
CONSTRAINT_NAME	C	30	Constraint Name (see below)
CONSTRAINT	C	100	Constraint

Constraint names:

- AUTOINC NEXT
- AUTOINC STEP
- CALCULATED
- CHECK
- CHOICES
- DEFAULT
- ERROR
- HELP
- NOT NULL
- NULL SUPPORT
- PICTURE
- RANGE
- RECALCULATE
- WHEN

### **Example**

EXEC SQL

```
SELECT column_name
FROM syscolumnconstraints
WHERE table_name = 'example';
```

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSCOLUMNPRIVILEGES**

---

### **Class**

SQL Applications

### **Purpose**

Description of the access rights for a table's columns

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
COLUMN_NAME	C	32	Column name
GRANTOR	C	30	Grantor of access
GRANTEE	C	100	Grantee of access
PRIVILEGE	C	30	Name of access (SELECT, INSERT etc)
IS_GRANTABLE	C	3	"YES" if grantee is permitted to grant to others; "NO" if not; .NULL. if unknown.

### **Example**

EXEC SQL

```
SELECT column_name
FROM syscolumnprivileges
WHERE table_name = 'example';
```

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## SYSCOLUMNS

---

### Class

SQL Applications

### Purpose

Description of the table columns available in the catalog

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
COLUMN_NAME	C	25	Column name
DATA_TYPE	N	2	SQL data type
TYPE_NAME	C	30	Data source dependent type name
COLUMN_SIZE	N	3	Precision
BUFFER_LENGTH	L	1	Reserved
DECIMAL_DIGITS	N	2	Scale
NUM_PREC_RADIX	N	2	Radix (typically either 10 or 2)
NULLABLE	N	2	Is NULL allowed?
REMARKS	C	25	Comment describing column
COLUMN_DEF	C	30	Default value
SQL_DATA_TYPE	N	1	Reserved
SQL_DATETIME_SUB	N	1	Reserved
CHAR_OCTET_LENGTH	N	1	For char types the maximum number of bytes in the column
ORDINAL_POSITION	N	4	Index of column in table (starting at 1)
IS_NULLABLE	C	3	“NO” means column definitely does not allow NULL values; “YES” means the column might allow NULL values. An empty string means unknown.

### Example

```
EXEC SQL
  SELECT *
  FROM syscolumns
  WHERE type_name = 'DATETIME';
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSCROSSREFERENCE**

---

### **Class**

SQL Applications

### **Purpose**

Description of how one table imports the keys of another table

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
PKTABLE_CAT	C	30	Primary key table catalog
PKTABLE_SCHEM	C	30	Primary key table schema
PKTABLE_NAME	C	30	Primary key table name
PKCOLUMN_NAME	C	30	Primary key column name
FKTABLE_CAT	C	30	Foreign key table catalog being exported
FKTABLE_SCHEM	C	30	Foreign key table schema being exported
FKTABLE_NAME	C	30	Foreign key table name being exported
FKCOLUMN_NAME	C	30	Foreign key column name being exported
KEY_SEQ	N	2	Sequence number within foreign key
UPDATE_RULE	N	2	What happens to foreign key when primary is updated
DELETE_RULE	N	2	What happens to the foreign key when primary is deleted
FK_NAME	C	30	Foreign key name
PK_NAME	C	30	Primary key name
DEFERRABILITY	N	2	Can the evaluation of foreign key constraints be deferred until commit?

### **Example**

EXEC SQL

SELECT \*

FROM syscrossreference;

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSEXPORTEDKEYS

---

### Class

SQL Applications

### Purpose

Description of the foreign key columns that reference the primary key columns

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
PKTABLE_CAT	C	30	Primary key table catalog
PKTABLE_SCHEM	C	30	Primary key table schema
PKTABLE_NAME	C	30	Primary key table name
PKCOLUMN_NAME	C	30	Primary key column name
FKTABLE_CAT	C	30	Foreign key table catalog being exported
FKTABLE_SCHEM	C	30	Foreign key table schema being exported
FKTABLE_NAME	C	30	Foreign key table name being exported
FKCOLUMN_NAME	C	30	Foreign key column name being exported
KEY_SEQ	N	2	Sequence number within foreign key
UPDATE_RULE	N	2	What happens to foreign key when primary is updated
DELETE_RULE	N	2	What happens to the foreign key when primary is deleted
FK_NAME	C	30	Foreign key name
PK_NAME	C	30	Primary key name
DEFERRABILITY	N	2	Can the evaluation of foreign key constraints be deferred until commit?

### Example

EXEC SQL

SELECT \*

FROM sysexportedkeys;

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSIMPORTEDKEYS**

---

### **Class**

SQL Applications

### **Purpose**

Description of the primary key columns that are referenced by the foreign key

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
PKTABLE_CAT	C	30	Primary key table catalog being imported
PKTABLE_SCHEM	C	30	Primary key table schema being imported
PKTABLE_NAME	C	30	Primary key table name being imported
PKCOLUMN_NAME	C	30	Primary key column name being imported
FKTABLE_CAT	C	30	Foreign key table catalog
FKTABLE_SCHEM	C	30	Foreign key table schema
FKTABLE_NAME	C	30	Foreign key table name
FKCOLUMN_NAME	C	30	Foreign key column name
KEY_SEQ	N	2	Sequence number within foreign key
UPDATE_RULE	N	2	What happens to foreign key when primary is updated
DELETE_RULE	N	2	What happens to the foreign key when primary is deleted
FK_NAME	C	30	Foreign key name
PK_NAME	C	30	Primary key name
DEFERRABILITY	N	2	Can the evaluation of foreign key constraints be deferred until commit?

### **Example**

EXEC SQL

SELECT \*

FROM sysimportedkeys;

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSINDEXINFO

---

### Class

SQL Applications

### Purpose

Description of a table's indices and statistics

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
NON_UNIQUE	L	1	Can index values be non-unique?
INDEX_QUALIFIER	C	30	Index catalog
INDEX_NAME	C	30	Index name
TYPE	N	2	Index type
ORDINAL_POSITION	N	3	Column sequence number within index
COLUMN_NAME	C	30	Column name
ASC_OR_DESC	C	1	Column sort sequence, "A" is ascending, "D" is descending
CARDINALITY	N	10	The number of rows in the table or the number of unique values in the index, depending on the index type
PAGES	N	2	The number of pages used for the table or the number of pages used for the current index, depending on the index type
FILTER_CONDITION	C	30	Filter condition

### Example

EXEC SQL

```
SELECT *
FROM sysindexinfo;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSIOSTATS

---

### Class

SQL Applications

### Purpose

Facility for monitoring table and index file I/O operations

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_NAME	C	30	Table name
READS	N	10	# of row reads for table
UPDATES	N	10	# of row updates for table
DELETES	N	10	# of row deletes for table
RECALLS	N	10	# of row recalls for table
INSERTS	N	10	# of row inserts for table
DCACHEREADS	N	10	# of cache reads for table
DCACHEWRITES	N	10	# of cache writes for table
INDEXREADS	N	10	# of index reads for table
INDEXWRITES	N	10	# of index writes for table
ICACHEREADS	N	10	# of index cache reads for table
ICACHEWRITES	N	10	# of index cache writes for table

### Example

EXEC SQL

```
SELECT icachereads
FROM sysiostats;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSLOGGING

---

### Class

SQL Applications

### Purpose

Provides System Logging information. When SET SYSLOGGING is ON internal system logging is performed while the process is running. The information logged can be used to find performance problems or track down system errors.

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
LEVEL	N	1	Logging level: 0-FATAL, 1-ERROR, 2-WARNING, 3-INFORMATION, 4-LOGON/LOGOFF
PID	N	8	Process ID
USER_NAME	C	30	User name
DATE	D	4	Date logged
TIME	C	8	Time logged
FILE_NAME	C	10	Internal
LINE_NUMBER	N	6	Internal
OS_ERROR_NUMBER	N	4	OS error number
PRODUCT_NAME	C	28	Name of logging product
PATCH_LEVEL	C	30	Patch level of logging product
COMPILE_DATETIME	C	20	Compile date of logging product
MESSAGE	C	100	Textual information

### Example

EXEC SQL

```
SELECT user_name, level
FROM syslogging;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSPRIMARYKEYS**

---

**Class**

SQL Applications

**Purpose**

Description of the primary key columns in the table

**See Also**

SELECT, SYSTEM TABLES, DATA TYPES

**Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
COLUMN_NAME	C	30	Column name
KEY_SEQ	N	2	Sequence number within primary key
PK_NAME	C	30	Primary key name

**Example**

EXEC SQL

SELECT \*

FROM sysprimarykeys;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## **SYSPROCEDURECOLUMNS**

---

### **Class**

SQL Applications

### **Purpose**

Description of the input, output and results associated with certain stored procedures available

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
PROCEDURE_CAT	C	100	Procedure catalog
PROCEDURE_SCHEM	C	30	Procedure schema
PROCEDURE_NAME	C	30	Procedure name
COLUMN_NAME	C	30	Column/parameter name
COLUMN_TYPE	N	1	Kind of column/parameter
DATA_TYPE	N	2	SQL data type
TYPE_NAME	C	30	SQL type name
PRECISION	N	2	Precision
LENGTH	N	4	Length in bytes of data
SCALE	N	2	Scale
RADIX	N	1	Radix
NULLABLE	N	1	Can it contain NULL
REMARKS	C	30	Comment describing parameter/column

### **Example**

```
EXEC SQL
  SELECT *
  FROM sysprocedurecolumns;
```

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSPROCEDURES**

---

### **Class**

SQL Applications

### **Purpose**

Description of the stored procedures available in the catalog

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
PROCEDURE_CAT	C	100	Procedure catalog
PROCEDURE_SCHEM	C	30	Procedure schema
PROCEDURE_NAME	C	30	Procedure name
R1	L	1	Reserved
R2	L	1	Reserved
R3	L	1	Reserved
REMARKS	C	30	Comment describing procedure
PROCEDURE_TYPE	N	1	Kind of procedure

### **Example**

EXEC SQL

SELECT \*

FROM sysprocedures;

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSRESULTSET

---

### Class

SQL Applications

### Purpose

Used to return the singleton result from any Recital expression

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

The SYSRESULTSET system table is used to return the singleton result from any Recital expression.

Column	Data Type	Width
<expression1>	<result1 type>	<result1 width>
...	...	...
<expression256>	<result256 type>	<result256 width>

An error will occur if any expression is specified that returns more than one result.

### Example

EXEC SQL

set("EXCLUSIVE") as Excl, time() as Time from sysresultset;

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSSCHEMAS

---

**Class**

SQL Applications

**Purpose**

Schema names available in the database

**See Also**

SELECT, SYSTEM TABLES, DATA TYPES

**Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_SCHEM	C	30	Table schema

**Example**

EXEC SQL

SELECT \*

FROM sysschemas;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSTABLECONSTRAINTS

---

### Class

SQL Applications

### Purpose

Description of the constraints for each table available in the catalog

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
CONSTRAINT_NAME	C	30	Constraint name (see below)
CONSTRAINT	C	100	Constraint

Constraint names:

- CHECK
- CLOSE
- DELETE
- ERROR
- INSERT
- OPEN
- ROLLBACK
- UPDATE

### Example

```
EXEC SQL
  SELECT *
  FROM systableconstraints;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSTABLEPRIVILEGES

---

### Class

SQL Applications

### Purpose

Description of the access rights for each table available in the catalog

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
GRANTOR	C	30	Grantor of access
GRANTEE	C	100	Grantee of access
PRIVILEGE	C	30	Name of access (SELECT, INSERT etc)
IS_GRANTABLE	C	3	“YES” if grantee is permitted to grant to others; “NO” if not; .NULL. if unknown.

### Example

EXEC SQL

SELECT \*

FROM systableprivileges;

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSTABLES

---

**Class**

SQL Applications

**Purpose**

Description of the tables available in the catalog

**See Also**

SELECT, SYSTEM TABLES, DATA TYPES

**Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_CAT	C	30	Table catalog
TABLE_SCHEM	C	30	Table schema
TABLE_NAME	C	30	Table name
TABLE_TYPE	C	15	Table type
REMARKS	C	100	Comment describing table

**Example**

EXEC SQL

SELECT \*

FROM systables

WHERE table\_type = 'TABLE';

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSTABLETYPES

---

**Class**

SQL Applications

**Purpose**

Table types available in the database system

**See Also**

SELECT, SYSTEM TABLES, DATA TYPES

**Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TABLE_TYPE	C	15	Table type

**Example**

```
EXEC SQL
  SELECT *
  FROM systabletypes;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## SYSTYPEINFO

---

### Class

SQL Applications

### Purpose

Description of all data types supported by the database

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TYPE_NAME	C	30	Type name
DATA_TYPE	N	3	SQL data type
PRECISION	N	3	Maximum precision
LITERAL_PREFIX	C	1	Prefix used to quote a literal
LITERAL_SUFFIX	C	1	Suffix used to quote a literal
CREATE_PARAMS	C	30	Parameters used in creating the type
NULLABLE	N	1	Can you use NULL for this type?
CASE_SENSITIVE	L	1	Is it case sensitive?
SEARCHABLE	N	1	Can you use "WHERE" based on this type?
UNSIGNED_ATTRIBUTE	L	1	Is it unsigned?
FIXED_PREC_SCALE	L	1	Can it be a money value?
AUTO_INCREMENT	L	1	Can it be used for an auto-increment value?
LOCAL_TYPE	C	30	Localized version of type name
MINIMUM_SCALE	N	2	Minimum scale supported
MAXIMUM_SCALE	N	2	Maximum scale supported
SQL_DATA_TYPE	N	1	Reserved
SQL_DATETIME_SUB	N	1	Reserved
NUM_PREC_RADIX	N	2	Usually 2 or 10

### Example

```
EXEC SQL
  SELECT *
  FROM systypeinfo;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYSUDTS

---

### Class

SQL Applications

### Purpose

Description of the user-defined types (UDTs) defined in the schema

### See Also

SELECT, SYSTEM TABLES, DATA TYPES

### Description

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

Column	Data Type	Width	Description
TYPE_CAT	C	30	Type catalog
TYPE_SCHEM	C	30	Type schema
TYPE_NAME	C	30	Type name
CLASS_NAME	C	30	Class name
DATA_TYPE	N	2	SQL data type
REMARKS	C	30	Comment describing type

### Example

EXEC SQL

SELECT \*

FROM sysudts;

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **SYSVERSIONCOLUMNS**

---

### **Class**

SQL Applications

### **Purpose**

Description of the columns in a table that are automatically updated when any row is updated

### **See Also**

SELECT, SYSTEM TABLES, DATA TYPES

### **Description**

System Tables are system defined read-only tables. You can query these tables using the SELECT statement.

<b>Column</b>	<b>Data Type</b>	<b>Width</b>	<b>Description</b>
SCOPE	N	1	Reserved
COLUMN_NAME	C	30	Column name
DATA_TYPE	N	2	SQL data type
TYPE_NAME	C	30	Data source-dependent type name
COLUMN_SIZE	N	3	Precision
BUFFER_LENGTH	N	1	Length of column value in bytes
DECIMAL_DIGITS	N	2	Scale
PSEUDO_COLUMN	N	1	Is this a pseudo column?

### **Example**

EXEC SQL

SELECT \*

FROM sysversioncolumns;

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## PSEUDO COLUMNS

---

### Class

SQL Applications

### Purpose

Provide extra information about a SELECT row set

### See Also

INSERT, SELECT, UPDATE

### Description

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. They provide extra information about a row set.

Pseudo Columns	Description
CURRVAL	The CURRVAL Pseudo Column will return the current sequence number from the specified table. Sequence numbers can be used for primary and unique index keys.
NEXTVAL	The NEXTVAL Pseudo Column will return the next unique sequence number from the specified table. Sequence numbers can be used for primary and unique index keys.
ROWID	The ROWID Pseudo Column will return a number identifying the row's physical stored position in the table. The ROWID Pseudo Column can be used to perform singleton selects, or optimize updates of a known ROWID.
ROWNUM	The ROWNUM Pseudo Column will return a number indicating the order in which the rows are selected from the table.
SQLCNT	The SQLCNT Pseudo Column will return the number of rows affected by the last SQL statement. For example, after a SELECT statement, SQLCNT will contain the number of rows selected.
SQLCODE	The SQLCODE Pseudo Column will return a number indicating the result of the last SQL statement.
SYNCNUM	The SYNCNUM pseudo column will return the unique sequence number assigned to a row from the specified table.

## CURRVAL

---

### Class

Pseudo Columns

### Purpose

Return the current sequence number from the specified table

### Syntax

CURRVAL

### See Also

INSERT, SELECT, UPDATE

### Description

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. Pseudo Columns provide extra information about a SELECT row set.

The CURRVAL Pseudo Column will return the current sequence number from the specified table. Sequence numbers can be used for primary and unique index keys.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db
```

```
CREATE TABLE cust (acc_num INT , acc_name char(20));
INSERT INTO cust (acc_num, acc_name) VALUES (NEXTVAL, "Smith");
INSERT INTO cust (acc_name) VALUES ("Brown");
INSERT INTO cust (acc_num, acc_name) VALUES (CURRVAL+2, "Jones");
SELECT * from cust;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## NEXTVAL

---

### Class

Pseudo Columns

### Purpose

Return the next unique sequence number from the specified table

### Syntax

NEXTVAL

### See Also

INSERT, SELECT, UPDATE

### Description

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. Pseudo Columns provide extra information about a SELECT row set.

The NEXTVAL Pseudo Column will return the next unique sequence number from the specified table. Sequence numbers can be used for primary and unique index keys.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db
```

```
CREATE TABLE cust (acc_num INT , acc_name char(20));
INSERT INTO cust (acc_num, acc_name) VALUES (NEXTVAL, "Smith");
INSERT INTO cust (acc_name) VALUES ("Brown");
INSERT INTO cust (acc_num, acc_name) VALUES (CURRVAL+2, "Jones");
SELECT * from cust;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ROWID

---

**Class**

Pseudo Columns

**Purpose**

Return a number identifying the row's physical stored position in the table

**Syntax**

ROWID

**See Also**

INSERT, SELECT, UPDATE

**Description**

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. Pseudo Columns provide extra information about a SELECT row set.

The ROWID Pseudo Column will return a number identifying the row's physical stored position in the table. The ROWID Pseudo Column can be used to perform singleton selects, or optimize updates of a known ROWID.

**Example**

```
// Optimized update accounts row 35 with a 15% commission charge
EXEC SQL
    UPDATE accounts
        SET ord_value=ord_value*1.15, due_date = date()+30
        WHERE ROWID=35;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ROWNUM

---

**Class**

Pseudo Columns

**Purpose**

Return a number indicating the order in which the rows are selected from the table

**Syntax**

ROWNUM

**See Also**

INSERT, SELECT, UPDATE

**Description**

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. Pseudo Columns provide extra information about a SELECT row set.

The ROWNUM Pseudo Column will return a number indicating the order in which the rows are selected from the table.

**Example**

```
// Display all overdue accounts with 15% commission in  
// Sorted "name" and "paid date" order with the row number.
```

EXEC SQL

```
    SELECT ROWNUM, name, address, balance, cost*1.15  
    FROM accounts  
    WHERE paid_date < date()  
    ORDER BY name, paid_date;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## SQLCNT

---

### Class

Pseudo Columns

### Purpose

Return the number of rows affected by the last SQL statement

### Syntax

CURRVAL

### See Also

INSERT, SELECT, UPDATE

### Description

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. Pseudo Columns provide extra information about a SELECT row set.

The SQLCNT Pseudo Column will return the number of rows affected by the last SQL statement. For example, after a SELECT statement, SQLCNT will contain the number of rows selected.

### Example

```
// Display all overdue accounts with 15% commission in
// Sorted "name" and "paid date" order with the row number.
EXEC SQL
    SELECT ROWNUM, name, address, balance, cost*1.15
    FROM accounts
    WHERE paid_date < date()
    ORDER BY name, paid_date;
// Check return code and number of rows returned
EXEC SQL
    SELECT DISTINCT sqlcode, sqlcnt from accounts;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLCODE

---

### Class

Pseudo Columns

### Purpose

Return a number indicating the result of the last SQL statement

### Syntax

SQLCODE

### See Also

INSERT, SELECT, UPDATE

### Description

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. Pseudo Columns provide extra information about a SELECT row set.

The SQLCODE Pseudo Column will return a number indicating the result of the last SQL statement.

SQLCODE return values:

SQLCODE	Description
0	The SQL statement completed successfully
+100	No rows were found or the end of the set reached
<0	An error occurred

### Example

```
// Display all overdue accounts with 15% commission in
// Sorted "name" and "paid date" order with the row number.
```

```
EXEC SQL
```

```
    SELECT ROWNUM, name, address, balance, cost*1.15
    FROM accounts
    WHERE paid_date < date()
    ORDER BY name, paid_date;
```

```
// Check return code and number of rows returned
```

```
EXEC SQL
```

```
    SELECT DISTINCT sqlcode, sqlcnt from accounts;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SYNCNUM

---

### Class

Pseudo Columns

### Purpose

Return the unique sequence number assigned to a row from the specified table

### Syntax

SYNCNUM

### See Also

INSERT, SELECT, UPDATE

### Description

A Pseudo Column behaves like a table column, but is not actually stored in the table. You can select from Pseudo Columns, but they can not be updated. Pseudo Columns provide extra information about a SELECT row set.

The SYNCNUM pseudo column will return the unique sequence number assigned to a row from the specified table. Each new row inserted into a table will be assigned a unique sequence number for that table. Even if the row is deleted later or if all the rows are deleted from the table, that number will not be issued again.

Note: The SYNCNUM pseudo column for existing Recital 9 tables can be populated using the dbconvert utility and the CONVERT command.

### Example

```
// Display all overdue accounts with 15% commission in
// Sorted "name" and "paid date" order with the unique row sequence number.
```

EXEC SQL

```
SELECT SYNCNUM, name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date()
ORDER BY name, paid_date;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## OPERATORS

---

**Class**

SQL Applications

**Purpose**

Operators

**See Also**

INSERT, SELECT, UPDATE, PREDICATES

**Description**

The following operators are available:

<b>Operators</b>
Numeric operators
String operators
Date operators
Logical operators
Relational operators

## NUMERIC OPERATORS

---

### Class

SQL Applications

### Purpose

Numeric operators

### Syntax

<expression1> operator <expression2>[ operator <expression3>...]

### See Also

INSERT, SELECT, UPDATE

### Description

Operators used with Numeric expressions:

Operator	Operation
**	Exponentiation
*	Multiplication
/	Division
%	Modulus/Remainder
+	Addition
-	Subtraction

### Example

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() AND ord_value > 10000
ORDER BY name, paid_date;
```

EXEC SQL

```
SELECT account_no, ord_value – paid_value as “Outstanding Balance”
FROM accounts
ORDER BY account_no;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## STRING OPERATORS

---

### Class

SQL Applications

### Purpose

String operators

### Syntax

<expression1> operator <expression2>[ operator <expression3>...]

### See Also

INSERT, SELECT, UPDATE

### Description

Operators used with Character expressions:

Operator	Operation
	Concatenate the <expression2> to the end of the <expression1> string
+	Concatenate the <expression2> to the end of the <expression1> string
-	The <expression1> string is trimmed of trailing spaces, the <expression2> string is concatenated to the end of the <expression1> string and the previously trimmed spaces concatenated to the end of the <expression2> string

### Example

EXEC SQL

```
SELECT trim(title) || ' ' || last_name as "Full Name"
FROM customers
ORDER BY last_name;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DATE OPERATORS

---

### Class

SQL Applications

### Purpose

Date operators

### Syntax

<expression1> operator <expression2>[ operator <expression3>...]

### See Also

INSERT, SELECT, UPDATE

### Description

Operators used with Date expressions and Date/Numeric expressions:

Operator	Expression1	Expression2	Operation
+	Date	Numeric	Adds the <expression2> number of days to the date and returns a date
-	Date	Numeric	Subtracts the <expression2> number of days from the date and returns a date
-	Date	Date	Returns a numeric signifying the number of days between the two dates. If <expression2> is later, a negative number is returned

### Example

EXEC SQL

```
SELECT account_no, ord_date - rec_date as "Delivery Delay"
FROM accounts
ORDER BY account_no;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## LOGICAL OPERATORS

---

### Class

SQL Applications

### Purpose

Logical Operators

### Syntax

<condition1> AND <condition2>

NOT <condition>

<condition1> OR <condition2>

<condition1> XOR <condition2>

### See Also

INSERT, SELECT, UPDATE

### Description

Operator	Description
AND	True if <condition1> and <condition2> conditions are both true, otherwise False
NOT	True if the <condition> is false, otherwise False
OR	True if <condition1> or <condition2> is true or if both conditions are true, otherwise False
XOR	True if <condition1> or <condition2> is true but not both, otherwise False

### Example

// AND

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() AND ord_value > 10000
ORDER BY name, paid_date;
```

// NOT

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() AND NOT ord_value BETWEEN 0 AND 10000
ORDER BY name, paid_date;
```

// OR

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() OR ord_value > 10000
ORDER BY name, paid_date;
```



```
// XOR  
EXEC SQL  
  SELECT name, address, balance, cost*1.15  
  FROM accounts  
  WHERE paid_date < date() OR ord_value > 10000  
  ORDER BY name, paid_date;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## RELATIONAL OPERATORS

---

### Class

SQL Applications

### Purpose

Relational Operators

### Syntax

<expression1> operator <expression2>

### See Also

INSERT, SELECT, UPDATE

### Description

Relational Operators compare two expressions and evaluate to either True or False.

Operator	Description
=	True if <expression1> and <expression2> are equal, otherwise False
<	True if <expression1> is less than <expression2>, otherwise False
<=	True if <expression1> is less than or equal to <expression2>, otherwise False
>	True if <expression1> is greater than the <expression2>, otherwise False
>=	True if <expression1> is greater than or equal to <expression2>, otherwise False
<>	True if <expression1> and <expression2> are not equal, otherwise False
!=	True if <expression1> and <expression2> are not equal, otherwise False
#	True if <expression1> and <expression2> are not equal, otherwise False
\$	True if <expression1> is a sub string of <expression2>. Both <expression1> and <expression2> must be character strings.
	True if <expression2> is a sub string of <expression1>. Both <expression1> and <expression2> must be character strings.
==	True if <expression1> and <expression2> match, otherwise False. Both <expression1> and <expression2> must be character strings. The <expression2> can contain the wildcards below.

Wildcards for == pattern matching:

_	Any single character
%	Zero or more characters

### Example

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date = date()
ORDER BY name, paid_date;
```

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date <= date()
ORDER BY name, paid_date;
```

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date <> date()
ORDER BY name, paid_date;
```

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE name == "%inc%"
ORDER BY name, paid_date;
```

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE name | "inc"
ORDER BY name, paid_date;
```

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE name $ "BigCo inc, BigCo plc"
ORDER BY name, paid_date;
```

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## PREDICATES

---

**Class**

SQL Applications

**Purpose**

Special operators

**See Also**

INSERT, SELECT, UPDATE

**Description**

Predicates are special operators. They are used to evaluate an expression and return True (.T.), False (.F.) or Unknown (.NULL.).

Predicate	Description
BETWEEN	Evaluate whether an expression is between two other expressions.
IN	Evaluate whether an expression matches one of a set of values.
LIKE	Evaluate whether an expression is like another expression
NULL	Evaluate whether an expression is equal to .NULL.

## BETWEEN PREDICATE

---

**Class**

SQL Applications

**Purpose**

Special predicate

**Syntax**

<expression1> [NOT] BETWEEN <expression2> AND <expression3>

**See Also**

INSERT, SELECT, UPDATE

**Description**

BETWEEN is a special predicate to evaluate whether the specified <expression1> is greater than or equal to <expression2> and less than or equal to <expression3>. The data type must be the same for <expression1>, <expression2> and <expression3>.

The optional NOT evaluates whether <expression1> is not greater than or equal to <expression2> and not less than or equal to <expression3>.

**Example****EXEC SQL**

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() AND ord_value NOT BETWEEN 0 AND 10000
ORDER BY name, paid_date;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## IN PREDICATE

---

### Class

SQL Applications

### Purpose

Special predicate

### Syntax

<expression> [NOT] IN (<value list>)

<expression> [NOT] IN (<nested select>)

### See Also

INSERT, SELECT, UPDATE

### Description

IN is a special predicate to evaluate whether the specified <expression> is equal to one of the values in the <value list> or <nested select>. The <value list> contains a list of comma separated values. The <nested select> is an SQL SELECT clause.

The optional NOT evaluates whether the specified <expression> is not equal to any of the values in the <value list> or <nested select>.

The data type must be the same for the <expression> and all the values in <value list> or returned by the <nested select>.

### Example

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() AND ord_value IN (100, 200, 300)
ORDER BY name, paid_date;
```

EXEC SQL

```
SELECT name, address, balance, rep_id
FROM accounts
WHERE rep_id IN (SELECT emp_id from employees where location = 'MA')
ORDER BY name;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## LIKE PREDICATE

---

**Class**

SQL Applications

**Purpose**

Special predicate

**Syntax**

&lt;expression1&gt; [NOT] LIKE &lt;expression2&gt;

**See Also**

INSERT, SELECT, UPDATE

**Description**

LIKE is a special predicate to evaluate whether the specified <expression1> matches <expression2>. The <expression2> can contain the following wildcards:

_	Any single character
%	Zero or more characters

The optional NOT evaluates whether the specified <expression1> does not match <expression2>.

**Example****EXEC SQL**

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() AND name LIKE "%inc%"
ORDER BY name, paid_date;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## NULL PREDICATE

---

**Class**

SQL Applications

**Purpose**

Special predicate

**Syntax**

<expression> IS [NOT] NULL

**See Also**

INSERT, SELECT, UPDATE

**Description**

NULL is a special predicate to evaluate whether the specified <expression> is NULL. The optional NOT can be used to evaluate whether the specified <expression> is not NULL.

**Example**

EXEC SQL

```
SELECT name, address, balance, cost*1.15
FROM accounts
WHERE paid_date < date() AND ord_value IS NULL
ORDER BY name, paid_date;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## TABLE CONSTRAINTS

---

### Class

SQL Applications

### Purpose

To define rules that help to provide data integrity

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. There are two different types of constraints, TABLE constraints, which do not require any column information and column constraints, which are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

### Table Constraints

CHECK	ERROR	FOREIGN KEY
INDEX	KEY	ONCLOSE
ONDELETE	ONINSERT	ONOPEN
ONROLLBACK	ONUPDATE	PRIMARY KEY
UNIQUE		

### Column Constraints

AUTO_INCREMENT	AUTOINC	CALCULATED
CHECK	DEFAULT	DESCRIPTION
ERROR	FOREIGN KEY	NOCPTRANS
NOT NULL	NULL	PRIMARY KEY
RANGE	RECALCULATE	REFERENCES
SET CHECK	UNIQUE	

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CHECK

---

**Class**

Table Constraints

**Purpose**

Table constraint activated when an operation to insert, update or delete records in the table is called

**Syntax**

CHECK <condition>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The CHECK table constraint is activated when an operation to insert, update or delete records in the table is called. The <condition> specified must evaluate to True (.T.) for the operation to succeed. If the <condition> evaluates to False (.F.) the operation is abandoned and the ERROR table constraint message is displayed. If the ERROR table constraint has not been defined, a default error message is displayed.

**Example**

```
set sql to vfp
CREATE TABLE purchase_order (POid i PRIMARY KEY, SuppId i, POtotal n(10,2), ;
    CHECK callauth() ERROR [Not authorized])
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ERROR

---

### Class

Table Constraints

### Purpose

Table constraint to define an error message to be displayed when a validation check fails

### Syntax

ERROR <expC>

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ERROR table constraint is used to define an error message to be displayed when a validation check fails. The <expC> is a character string of up to 80 characters.

### Example

set sql to vfp

```
CREATE TABLE purchase_order (POid i PRIMARY KEY, SuppId i, POtotal n(10,2), ;
    CHECK callauth() ERROR [Not authorized])
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## FOREIGN KEY

---

### Class

Table Constraints

### Purpose

Table constraint to define a Foreign Key

### Syntax

```
FOREIGN KEY <expr> TAG <cTagName> [COLLATE <cCollateSequence>]
[REFERENCES <cTableName> [TAG <cTagName2>]]
```

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The FOREIGN KEY table constraint is used to define <expr> as a Foreign Key for a parent table. The <expr> can contain any valid index key. A tag index is built on the specified <expr>; it is given the name as defined in <cTagName>. A table may have more than one Foreign Key index.

The optional COLLATE <cCollateSequence> clause is included for Visual FoxPro language compatibility only.

The optional REFERENCES clause is used to create a relationship to an index key of another table. The value of the <expr> is validated by checking that it already exists as a value in the referenced index key.

The name of the referenced table is specified in <cTableName>. The index tag to reference is specified in <cTagName2>. If the optional TAG <cTagName2> clause is omitted, the primary index key of <cTableName> is used. If <cTableName> has no index tags, an error is generated.

### Example

set sql to vfp

```
CREATE TABLE supplier (SuppId i PRIMARY KEY, SuppName c(40) UNIQUE)
CREATE TABLE purchase_order (POid i PRIMARY KEY, SuppId i, POTotal n(10,2) )
ALTER TABLE purchase_order ADD FOREIGN KEY SuppID TAG SuppId REFERENCES supplier
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## INDEX

---

**Class**

Table Constraints

**Purpose**

Table constraint to define an index key

**Syntax**

INDEX [<cTagName>] (column1[column2,...])

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The INDEX table constraint is used to define an index key. The index is built on the column or columns specified and given the name as defined in <cTagName>. The KEY table constraint can be used in the same way.

**Example**

set sql to mysql

```
CREATE TABLE contact (LastName char(25), FirstName char(25),  
    INDEX FullName (LastName, FirstName));
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## KEY

---

**Class**

Table Constraints

**Purpose**

Table constraint to define an index key

**Syntax**

KEY [<cTagName>] (column1[column2,...])

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The KEY table constraint is used to define an index key. The index is built on the column or columns specified and given the name as defined in <cTagName>. The INDEX table constraint can be used in the same way.

**Example**

set sql to mysql

```
CREATE TABLE contact (LastName char(25), FirstName char(25),  
    KEY FullName (LastName, FirstName));
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ONCLOSE

---

**Class**

Table Constraints

**Purpose**

Table constraint activated when the table is closed

**Syntax**

ONCLOSE <procedure>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ONCLOSE table constraint is activated when the table is closed. The <procedure> specified must be a character expression evaluating to a procedure name. If no file extension is included, '.prg' is assumed. The specified procedure is run before the operation to close the table is executed.

**Example**

```
ALTER TABLE customer modify ONCLOSE "p_close";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ONDELETE

---

**Class**

Table Constraints

**Purpose**

Table constraint activated when an attempt is made to delete a record in the table

**Syntax**

ONDELETE <procedure>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ONDELETE table constraint is activated when an attempt is made to delete a record in the table. The <procedure> specified must be a character expression evaluating to a procedure name. If no file extension is included, '.prg' is assumed. The specified procedure is run before the operation to delete the record is executed and must return True (.T.) or the delete operation is cancelled.

**Example**

```
ALTER TABLE customer modify ONDELETE "p_delete";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## ONINSERT

---

**Class**

Table Constraints

**Purpose**

Table constraint activated when an attempt is made to insert a new record into the table

**Syntax**

ONINSERT <procedure>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ONINSERT table constraint is activated when an attempt is made to insert a new record into the table. The <procedure> specified must be a character expression evaluating to a procedure name. If no file extension is included, '.prg' is assumed. The specified procedure is run before the operation to insert the record is executed and must return True (.T.) or the insert operation is cancelled.

**Example**

```
ALTER TABLE customer modify ONINSERT "p_insert";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ONOPEN

---

**Class**

Table Constraints

**Purpose**

Table constraint activated when the table is opened

**Syntax**

ONOPEN <procedure>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ONOPEN table constraint is activated when the table is opened. The <procedure> specified must be a character expression evaluating to a procedure name. If no file extension is included, '.prg' is assumed. The specified procedure is run after the operation to open the table is executed.

**Example**

```
ALTER TABLE customer modify ONOPEN "p_open";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ONROLLBACK

---

**Class**

Table Constraints

**Purpose**

Table constraint activated when a forms based operation is abandoned

**Syntax**

ONROLLBACK <procedure>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ONROLLBACK table constraint is activated when a forms based operation is abandoned. The <procedure> specified must be a character expression evaluating to a procedure name. If no file extension is included, '.prg' is assumed. The specified procedure is run when the user presses the [ABANDON] key.

**Example**

```
ALTER TABLE customer modify ONROLLBACK "p_rollback";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ONUPDATE

---

**Class**

Table Constraints

**Purpose**

Table constraint activated when an attempt is made to update a record in the table

**Syntax**

ONUPDATE <procedure>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ONUPDATE table constraint is activated when an attempt is made to update a record in the table. The <procedure> specified must be a character expression evaluating to a procedure name. If no file extension is included, '.prg' is assumed. The specified procedure is run before the operation to update the record is executed and must return True (.T.) or the update operation is cancelled.

**Example**

```
ALTER TABLE customer modify ONUPDATE "p_update";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## PRIMARY KEY

---

### Class

Table Constraints

### Purpose

Table constraint to define table's Primary Key

### Syntax

PRIMARY KEY <expr> TAG <cTagName> [COLLATE <cCollateSequence>]

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The PRIMARY KEY table constraint is used to define <expr> as the table's Primary Key. The <expr> can contain any valid index key. A unique tag index is built on the specified <expr>; it is given the name as defined in <cTagName>. If a table already has a primary key defined, an error will be returned.

The optional COLLATE <cCollateSequence> clause is included for Visual FoxPro language compatibility only.

### Example

```
set sql to vfp
CREATE TABLE newcust (acc_ref char(5) default strzero(seqno(),5), acc_name char(20))
ALTER TABLE newcust ADD PRIMARY KEY acc_ref+acc_name TAG RefName
list structure index
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## UNIQUE

---

### Class

Table Constraints

### Purpose

Table constraint to define a candidate index

### Syntax

UNIQUE <expr> TAG <cTagName> [COLLATE <cCollateSequence>]

UNIQUE [<cTagName>] (column1[column2,...])

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. TABLE constraints apply to table-based operations. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

UNIQUE <expr> TAG <cTagName>

The UNIQUE table constraint is used to define <expr> as a candidate index for the table. The <expr> can contain any valid index key. A unique tag index is built on the specified <expr>; it is given the name as defined in <cTagName>. A table may have more than one candidate index.

COLLATE <cCollateSequence>

The optional COLLATE <cCollateSequence> clause is included for Visual FoxPro language compatibility only.

UNIQUE [<cTagName>] (column1[column2,...])

The UNIQUE table constraint is used to define a unique index key. The index is built on the column or columns specified and given the name as defined in <cTagName>.

### Example

set sql to vfp

```
CREATE TABLE newcust (acc_ref char(5) default strzero(seqno(),5), acc_name char(20))
```

```
ALTER TABLE newcust ADD UNIQUE acc_ref+acc_name TAG RefName
```

list structure index

set sql to mysql

```
CREATE TABLE contact (ContRef char(5), LastName char(25), FirstName char(25),
```

```
    UNIQUE FullName (LastName, FirstName, ContRef));
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## COLUMN CONSTRAINTS

---

### Class

SQL Applications

### Purpose

To define rules that help to provide data integrity

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. There are two different types of constraints, TABLE constraints, which do not require any column information and column constraints, which are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

### Table Constraints

CHECK	ERROR	FOREIGN KEY
INDEX	KEY	ONCLOSE
ONDELETE	ONINSERT	ONOPEN
ONROLLBACK	ONUPDATE	PRIMARY KEY
UNIQUE		

### Column Constraints

AUTO_INCREMENT	AUTOINC	CALCULATED
CHECK	DEFAULT	DESCRIPTION
ERROR	FOREIGN KEY	NOCPTRANS
NOT NULL	NULL	PRIMARY KEY
RANGE	RECALCULATE	REFERENCES
SET CHECK	UNIQUE	

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## AUTO\_INCREMENT

---

**Class**

Column Constraints

**Purpose**

Column constraint to auto increment the value of a column

**Syntax**

AUTO\_INCREMENT

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The AUTO\_INCREMENT column constraint is used to auto increment the value of a column whenever a new record is inserted. The first record to be inserted has a column value of 1 and for each new record the value increments by 1. A column with the AUTO\_INCREMENT constraint set is not read only; values can be inserted into the field, but it will default to auto incrementing if no value or a .NULL. is specified.

The AUTO\_INCREMENT value increases on a per-table basis, using the SEQNO() function in the DEFAULT constraint of the column. Only one column per table can have the AUTO\_INCREMENT constraint set.

**Example**

set sql to mysql

```
CREATE TABLE newcust (acc_num INT AUTO_INCREMENT, acc_name char(20));
INSERT INTO newcust (acc_name) VALUES ("Smith");
INSERT INTO newcust (acc_name) VALUES ("Jones");
SELECT * FROM newcust;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## AUTOINC

---

### Class

Column Constraints

### Purpose

Column constraint to auto increment the value of a column

### Syntax

AUTOINC [NEXTVALUE <NextValue> [STEP <StepValue>]]

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The AUTOINC column constraint is used to auto increment the value of a column whenever a new record is inserted. The NEXTVALUE <NextValue> clause may optionally be used to set the column value of the next inserted record to the integer <NextValue>. The optional STEP <StepValue> clause determines the integer amount the value should be incremented each time a new record is inserted. By default, the first record to be inserted has a column value of 1 and for each new record the value increments by 1.

Specifying the AUTOINC column constraint causes the column to be read only.

NOTE: AUTOINC is column based and multiple columns in the same table can have the AUTOINC column constraint set. The Recital SEQNO() function works on a per table basis.

### Example

set sql to vfp

```
CREATE TABLE newcust (acc_num INT AUTOINC NEXTVALUE 10 STEP 5, acc_name char(20))
INSERT INTO newcust (acc_name) VALUES ("Smith")
INSERT INTO newcust (acc_name) VALUES ("Jones")
SELECT * FROM newcust
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CALCULATED

---

**Class**

Column Constraints

**Purpose**

Column constraint to calculate the value of a column

**Syntax**

CALCULATED <expr>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The CALCULATED column constraint is used to calculate the value of a column whenever it is accessed. The <expr> specified must evaluate to the same data type as the target column. Specifying the CALCULATED column constraint causes the column to be read only. If the <expr> is based on other columns from the same table, these columns must have the RECALCULATE column constraint set, so that any changes to their values cause calculated fields to be recalculated.

**Example**

```
ALTER TABLE customer
  ALTER COLUMN available CALCULATED limit-balance
  ALTER COLUMN limit RECALCULATE
  ALTER COLUMN balance RECALCULATE;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CHECK

---

**Class**

Column Constraints

**Purpose**

Column constraint to validate a change to the value of a column

**Syntax**

CHECK <condition>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The CHECK column constraint is used to validate any changes made to the value of a column. The <condition> specified must evaluate to True (.T.) for the modification operation to succeed. If the <condition> evaluates to False (.F.) the column value remains unchanged and the ERROR column constraint message is displayed. If the ERROR column constraint has not been defined, a default error message is displayed.

**Example**

```
set sql to recital
exec sql
```

```
    ALTER TABLE customer ADD COLUMN timeref char(8) CHECK validtime(timeref)
        ERROR "Not a valid time string";
```

```
use customer
edit
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DEFAULT

---

**Class**

Column Constraints

**Purpose**

Column constraint to set a default value for the specified column

**Syntax**

DEFAULT <expr>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The DEFAULT column constraint is used to set a default value for the specified column. The <expr> must evaluate to the same data type as the target column. The column's value can subsequently be updated.

**Example**

set sql to recital

exec sql

ALTER TABLE customer ADD COLUMN dateref date DEFAULT date();

use customer

list dictionary

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DESCRIPTION

---

### Class

Column Constraints

### Purpose

Column constraint to set the column description for the specified column

### Syntax

DESCRIPTION <expC>

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The DESCRIPTION column constraint is used to set the column description for the specified column. The <expC> is a character expression of up to 25 characters.

### Example

set sql to recital

exec sql

```
ALTER TABLE customer ADD COLUMN dateref date
      DEFAULT date() DESCRIPTION "Date Reference";
```

use customer

list structure

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ERROR

---

**Class**

Column Constraints

**Purpose**

Column constraint to define an error message to be displayed when a validation check fails

**Syntax**

ERROR <expC>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The ERROR column constraint is used to define an error message to be displayed when a validation check fails. The <expC> is a character string of up to 80 characters.

**Example**

```
set sql to recital
```

```
exec sql
```

```
    ALTER TABLE customer ADD COLUMN timeref char(8) CHECK validtime(timeref)
```

```
        ERROR "Not a valid time string";
```

```
use customer
```

```
edit
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## FOREIGN KEY

---

**Class**

Column Constraints

**Purpose**

Column constraint to define a column as a Foreign Key for a parent table

**Syntax**

FOREIGN KEY [COLLATE <cCollateSequence>]

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The FOREIGN KEY column constraint is used to define the column as a Foreign Key for a parent table. A tag index is built on the specified column; it is given the same name as the column. More than one column in the table can be defined as a Foreign Key.

The optional COLLATE <cCollateSequence> clause is included for Visual FoxPro language compatibility only.

**Example**

set sql to recital

```
CREATE TABLE orderitem (ord_ref char(5) FOREIGN KEY, item_ref char(5) PRIMARY KEY);
```

use orderitem

list structure index

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## NOCPTRANS

---

**Class**

Column Constraints

**Purpose**

Column constraint to prevent code page translation for character and memo fields

**Syntax**

NOCPTRANS

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The NOCPTRANS column constraint is used to prevent code page translation for character and memo fields.

**Example**

```
ALTER TABLE customer  
  ALTER COLUMN notes NOCPTRANS;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## NOT NULL

---

**Class**

Column Constraints

**Purpose**

Column constraint to disallow NULL values

**Syntax**

NOT NULL

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The NOT NULL column constraint is used to specify that the column cannot contain NULL values. The NULL column constraint can be used to allow NULL values.

**Example**

set sql to recital

exec sql

```
ALTER TABLE customer ADD COLUMN custref char(5) NULL;
```

exec sql

```
ALTER TABLE customer ALTER COLUMN custref NOT NULL;
```

use customer

list dictionary

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## NULL

---

**Class**

Column Constraints

**Purpose**

Column constraint to allow NULL values

**Syntax**

NULL

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The NULL column constraint is used to specify that the column can contain NULL values. The NOT NULL column constraint can be used to disallow NULL values.

**Example**

```
set sql to recital
exec sql
ALTER TABLE customer ADD COLUMN custref char(5) NULL;
exec sql
ALTER TABLE customer ALTER COLUMN custref NOT NULL;
use customer
list dictionary
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## PRIMARY KEY

---

**Class**

Column Constraints

**Purpose**

Column constraint to define table's Primary Key

**Syntax**

PRIMARY KEY [COLLATE <cCollateSequence>]

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The PRIMARY KEY column constraint is used to define the column as the table's Primary Key. A unique tag index is built on the specified column; it is given the same name as the column. If a table already has a primary key defined, an error will be returned.

The optional COLLATE <cCollateSequence> clause is included for Visual FoxPro language compatibility only.

**Example**

set sql to vfp

```
CREATE TABLE newcust (acc_ref char(5) default strzero(seqno(),5) PRIMARY KEY, ;  
    acc_name char(20))
```

list structure index

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## RANGE

---

**Class**

Column Constraints

**Purpose**

Column constraint to specify minimum and maximum values for a date or numerical column

**Syntax**

RANGE <expr1>,<expr2>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The RANGE column constraint is used to specify minimum and maximum values for a date or numerical column. The minimum value is specified in <expr1>, the maximum in <expr2>. Attempting to update the column with a value outside this range generates an error.

**Example**

set sql to recital

```
CREATE TABLE orderhead (ord_week INT RANGE 1,52 ,  
    ord_date DATE RANGE date(),gomonth(date(),12));
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## RECALCULATE

---

**Class**

Column Constraints

**Purpose**

Column constraint to force recalculation of calculated columns when a column's value changes

**Syntax**

RECALCULATE

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The RECALCULATE column constraint is used to force recalculation of calculated columns when a column's value changes. Any column which forms part of the calculation expression of a CALCULATED column must have the RECALCULATE column constraint set.

**Example**

```
ALTER TABLE customer
  ALTER COLUMN available CALCULATED limit-balance
  ALTER COLUMN limit RECALCULATE
  ALTER COLUMN balance RECALCULATE;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## REFERENCES

---

### Class

Column Constraints

### Purpose

Column constraint to create a relationship to an index key of another table

### Syntax

REFERENCES <cTableName> [TAG <cTagName>]

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The REFERENCES column constraint is used to create a relationship to an index key of another table. The value of a column which has the REFERENCES column constraint set is validated by checking that it already exists as a value in the referenced index key.

The name of the referenced table is specified in <cTableName>. The index tag to reference is specified in <cTagName>. If the optional TAG <cTagName> clause is omitted, the primary index key of <cTableName> is used. If <cTableName> has no index tags, an error is generated.

### Example

```
set sql to vfp
CREATE TABLE supplier ;
    (SuppId i PRIMARY KEY, ;
     SuppName c(40) UNIQUE)
CREATE TABLE purchase_order ;
    (POid i PRIMARY KEY, ;
     SuppId i REFERENCES supplier TAG SuppId, ;
     POtotal n(10,2) )
list dictionary
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET CHECK

---

**Class**

Column Constraints

**Purpose**

Column constraint to validate a change to the value of a column

**Syntax**

SET CHECK <condition>

**See Also**

ALTER TABLE, CREATE TABLE

**Description**

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The SET CHECK column constraint is used to validate any changes made to the value of a column. The <condition> specified must evaluate to True (.T.) for the modification operation to succeed. If the <condition> evaluates to False (.F.) the column value remains unchanged and the ERROR column constraint message is displayed. If the ERROR column constraint has not been defined, a default error message is displayed.

**Example**

```
set sql to recital
exec sql
```

```
    ALTER TABLE customer ALTER COLUMN timeref SET CHECK validtime(timeref);
use customer
edit
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## UNIQUE

---

### Class

Column Constraints

### Purpose

Column constraint to define a candidate index for a table

### Syntax

UNIQUE [COLLATE <cCollateSequence>]

### See Also

ALTER TABLE, CREATE TABLE

### Description

A constraint is used to define rules that help to provide data integrity. Column constraints are specific to the column name specified. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

The UNIQUE column constraint is used to define the column as a candidate index for the table. A unique tag index is built on the specified column; it is given the same name as the column. More than one column can be defined as a candidate index for the table.

The optional COLLATE <cCollateSequence> clause is included for Visual FoxPro language compatibility only.

### Example

set sql to vfp

```
CREATE TABLE newcust (acc_ref char(5) default strzero(seqno(),5) UNIQUE, ;
    acc_name char(20) UNIQUE)
```

list structure index

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## ADD TABLE

---

### Class

Databases

### Purpose

Add the specified table to the currently active database

### Syntax

ADD TABLE <table>

### See Also

ALTER INDEX, ALTER TABLE, BACKUP DATABASE, BUILD, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INSTALL, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The ADD TABLE command is used to add an existing table to the currently active database. If no database is currently active an error will be generated.

Keywords	Description
table	This is the name of the table to be added to the database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can also be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

### Example

```
CREATE TABLE freetable (freeid char(10))
open database southwind
ADD TABLE freetable
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ALTER INDEX

---

**Class**

SQL Applications

**Purpose**

Rebuilds an existing index file for the specified table

**Syntax**

```
ALTER INDEX <index> ON <table> REBUILD [SHARED | EXCLUSIVE]
```

**See Also**

ALTER TABLE, CREATE INDEX, CREATE TABLE, DROP INDEX, SET TCACHE

**Description**

The ALTER INDEX command is used to rebuild an existing index file for the specified table. The table must be able to be locked for exclusive use during the operation.

Keywords	Description
index	This is the name of the index being rebuilt.
table	This is the name of the table for which the index will be rebuilt on.
REBUILD	Create the index anew using the existing index
SHARED	Allows read-only transactions on the table while the index is being rebuilt.
EXCLUSIVE	Prevents any transactions on the table while the index is being rebuilt.

**Example**

```
// Rebuild the index staff_no index on staff table
ALTER INDEX staff_no
  ON staff
  REBUILD
  SHARED;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ALTER TABLE

---

### Class

SQL Applications

### Purpose

Used to add, modify or delete table columns and constraints

### Syntax

```
ALTER [IGNORE] TABLE [<database>!]<table>
  ADD [COLUMN] (<column> <datatype> [<column constraints>] [...])
    | <table constraint>
  | ALTER | MODIFY [COLUMN] <column> [SET DEFAULT <value> | DROP DEFAULT]
    | (<column> <datatype> [<column constraint>] [...])
    | CONSTRAINT (<column> SET <column constraint> <value> [...])
    | <table constraint>
  | DROP [COLUMN] <column>
    | (<column> [...])
    | CONSTRAINT (<column> <column constraint> [...]) | <table constraint>
  | SET CHECK <condition> [ERROR <message>]
  | RENAME (<column>,<new column>)
```

### See Also

ADD TABLE, ALTER INDEX, CREATE TABLE, INSERT, SELECT, CONSTRAINTS, DATA TYPES, GETENV(), SET TCACHE

### Description

The ALTER TABLE command is used to add, modify or delete table columns and constraints and to rename columns. The ALTER TABLE statement automatically reloads the original data of the table back into the original columns. You must have ALTER privilege on the table. The table will be locked for EXCLUSIVE use during the operation.

Keywords	Description
IGNORE	If IGNORE is omitted and there are duplicate UNIQUE keys in the table, the ALTER TABLE is aborted with an error. If IGNORE is specified, records containing a duplicate UNIQUE key are deleted, leaving only the first row with that key.
database	The name of the database to which the table to be altered belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table to be altered.
ADD	This will insert one or more new columns into the table
COLUMN	Optional COLUMN keyword.

column	The name of the column to operate on.
datatype	The data type to be stored in the column, and the applicable length or precision.
column constraint	The column constraint.
table constraint	The table constraint.
ALTER   MODIFY	These are used to change existing column definitions and column and table constraints.
SET DEFAULT <expr>	Specify the DEFAULT column constraint to set a default value for the specified column. The <expr> must evaluate to the same data type as the target column. The column's value can subsequently be updated.
DROP DEFAULT	Remove the DEFAULT column constraint for the specified column.
DROP	This is used to delete existing column definitions and column and table constraints.
CONSTRAINT	This keyword is used if the constraint refers to a column.
SET	Precedes an existing column constraint whose value is being changed.
value	The new value for the specified column constraint.
SET CHECK <condition>	Specify CHECK table constraint. This validation is activated when an operation to insert, update or delete records in the table is called. The <condition> specified must evaluate to True (.T.) for the operation to succeed. If the <condition> evaluates to False (.F.) the operation is abandoned and the ERROR table constraint message is displayed. If the ERROR table constraint has not been defined, a default error message is displayed.
ERROR <message>	Specify ERROR table constraint. The <message> is the error message to be displayed if the CHECK table constraint evaluates to False (.F.).
RENAME	This is used to change the name of an existing column.
new column	The new name for the column.

**Example**

```
// Add new column with column constraints
```

```
EXEC SQL
```

```
    ALTER TABLE customer ADD COLUMN timeref char(8)
    CHECK validtime(timeref)
    ERROR "Not a valid time string";
```

```
// Alter existing columns to add column constraints
```

```
EXEC SQL
```

```
    ALTER TABLE customer
    ALTER COLUMN available CALCULATED limit-balance
    ALTER COLUMN limit RECALCULATE
    ALTER COLUMN balance RECALCULATE;
```

```
//or
```

```
EXEC SQL
```

```
    ALTER TABLE customer
    ALTER (available CALCULATED limit-balance,
    limit RECALCULATE,
    balance RECALCULATE);
```

```
// Add new column, add column constraint,
```

```

// modify column datatype and drop constraints then drop column
EXEC SQL
    ALTER TABLE customer ADD (timeref char(8));

EXEC SQL
    ALTER TABLE customer
    ALTER CONSTRAINT
    (timeref SET CHECK validtime(timeref)
    ERROR "Not a valid time string");

EXEC SQL
    ALTER TABLE customer
    ALTER (timeref datetime)
    DROP CONSTRAINT (timeref CHECK, timeref ERROR);

EXEC SQL
    ALTER TABLE customer DROP (timeref);

// Add an ONUPDATE table constraint
EXEC SQL
    ALTER TABLE customer
    MODIFY ONUPDATE "do check_update";

// Add and then remove CHECK table constraint
EXEC SQL
    ALTER TABLE customer SET CHECK checkit() error "Invalid operation";

set sql off
use customer
display dictionary
edit  && Save and Exit will call validation
use

EXEC SQL
    ALTER TABLE customer DROP CHECK;

// Rename column
EXEC SQL
    ALTER TABLE customer RENAME(first_name,forename);

```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BACKUP DATABASE

---

### Class

Databases

### Purpose

Exports bridge, table, and associated files from the current or specified database in ASCII format to allow them to be transferred to a binary incompatible platform

### Syntax

BACKUP DATABASE [<database name> | ?]

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, BUILD, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INSTALL, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The BACKUP DATABASE command issues a BUILD on the bridge files, tables and associated memo, dictionary and multiple index files from the currently open database or specified database. This exports these files into ASCII format to allow them to be transferred to a binary incompatible platform.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

If the <database name> is omitted, the BACKUP DATABASE command will operate on the active database. If no database is currently open, an error will be returned. If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select a database. The dialog defaults to the DB\_DATADIR directory. This is only applicable for Recital Terminal Developer: for Recital Database and Mirage Servers, the <database name> must be specified if the required database is not already open.

The ASCII files are created in a sub-directory of the Recital backup directory. The environment variable / symbol DB\_BACKUPDIR points to the current Recital backup directory and can be queried using the GETENV() function. The sub-directory is created automatically and has the same name as the database. If the sub-directory already exists, any files it previously contained are deleted. Once the BACKUP DATABASE has completed successfully, the sub-directory and its contents can be copied to the DB\_BACKUPDIR directory on another platform and the database recreated using the RESTORE DATABASE command.

**Example**

// On Source machine:

Recital/SQL> backup database southwind;

// Transfer southwind sub-directory from DB\_BACKUPDIR on source machine

// to DB\_BACKUPDIR on target machine

// On Target Machine

Recital/SQL> restore database southwind;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BEGIN TRANSACTION

---

### Class

SQL Applications

### Purpose

Flag the beginning of a transaction

### Syntax

```
BEGIN TRANSACTION [<transaction>]
```

```
<statements>
```

```
END TRANSACTION [<transaction>]
```

### See Also

ROLLBACK, SAVE TRANSACTION, SAVEPOINT, SET TRANSACTION, TXNISOLATION(), TXNLEVEL()

### Description

The BEGIN TRANSACTION statement is used to flag the beginning of a transaction. The END TRANSACTION statement is used to commit changes made during the transaction and close the transaction. The COMMIT statement and the ROLLBACK statement can also be used to close a transaction. The COMMIT statement will save the changes made and the ROLLBACK statement will discard the changes made.

Transactions can be nested by issuing a second or subsequent BEGIN TRANSACTION before an existing transaction has been closed. The TXNLEVEL() function returns the current transaction nesting level. When a transaction is closed, transactions nested within it are also closed.

Savepoints can be set during a transaction. These identify stages within the transaction which can subsequently be used as ROLLBACK points.

The optional <transaction> is a name for the transaction. This name can be used by the COMMIT and ROLLBACK statements.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db

// Transactions
BEGIN TRANSACTION trans1;
INSERT INTO customer
  (TITLE, LAST_NAME, FIRST_NAME, INITIAL, STREET,
   CITY, STATE, ZIP,LIMIT, START_DATE)
VALUES
  ('Ms', 'Jones', 'Susan', 'B', '177 High Street','Beverly', 'MA', '01915', 2000, date());
INSERT INTO accounts (ORD_VALUE) VALUES (30);
BEGIN TRANSACTION trans2;
INSERT INTO accounts (ORD_VALUE) VALUES (60);
// Rollback the trans1 transaction and any transactions
// nested in trans1
ROLLBACK TRANSACTION trans1;
```



```
END TRANSACTION;  
// End of program
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CLOSE

---

**Class**

SQL Applications

**Purpose**

Closes a cursor

**Syntax**

CLOSE &lt;cursor&gt;

**See Also**

DECLARE CURSOR, DROP CURSOR, OPEN, SELECT

**Description**

The CLOSE command closes the specified cursor, releasing all resources and locks allocated when the cursor was opened. A cursor is a pointer to a logical table. A logical table is a temporary collection of data that satisfy conditions specified in a SELECT statement. After a cursor has been CLOSED, it may be accessed again by issuing another OPEN statement. The cursor is not released until a DROP CURSOR statement is issued. This command can only be used in Embedded SQL. The cursor must already be open.

Keywords	Description
cursor	The name of the cursor to be closed.

**Example**

```
// Close the cursor
EXEC SQL
    CLOSE accounts;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CLOSE DATABASES

---

### Class

Databases

### Purpose

Closes the currently open database

### Syntax

CLOSE DATABASES [ALL]

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DATABASE EVENTS

### Description

The CLOSE DATABASES command closes the currently open database and its tables. If no database is currently open, all tables and their associated files are closed.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

The CLOSE DATABASES command triggers the DBC\_CLOSEDATA database event. If a dbc\_closedata.prg program file exists in the database's directory, this will be run. If the dbc\_closedata.prg program returns .F. (False), the CLOSE DATABASES operation will be abandoned.

Databases can have an associated procedure library that is activated automatically when the database is opened. If a program file with the name dbc\_<database>\_library.prg, exists in the database's directory, e.g. dbc\_southwind\_library.prg for the southwind demo database, a SET PROCEDURE...ADDITIVE is issued for this procedure library when the database is opened. When the database is closed, the procedure library is also closed.

CLOSE DATABASES does not close gateway sessions. The SET GATEWAY TO or CLOSE ALL commands can be used for this purpose.

### Example

```
Recital/SQL> set sql to vfp
VFP/SQL> OPEN DATABASE hr EXCLUSIVE
VFP/SQL> SELECT staff_no, lastname from staff
VFP/SQL> CLOSE DATABASES
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CLOSE TABLES

---

### Class

Databases

### Purpose

Closes the currently open tables and their associated files in the active database

### Syntax

CLOSE TABLES [ALL]

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, CLOSE DATABASES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The CLOSE TABLES command closes the currently open tables and their associated files in the active database. The database itself remains open. If no database is currently open, all tables and their associated files are closed.

If the ALL keyword is included, all tables and their associated files will be closed.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

### Example

```
Recital/SQL> set sql to vfp
VFP/SQL> OPEN DATABASE hr EXCLUSIVE
VFP/SQL> use staff
VFP/SQL> CLOSE TABLES
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## COMMIT

---

### Class

SQL Applications

### Purpose

Ends the current transaction, saving changes

### Syntax

COMMIT [TRANSACTION <transaction>] [WORK]

### See Also

BEGIN...END TRANSACTION, ROLLBACK, SAVE TRANSACTION, SAVEPOINT, SET TRANSACTION, TXNISOLATION(), TXNLEVEL()

### Description

The COMMIT statement ends the current or specified transaction and any transactions that are nested within it and makes permanent all changes performed in the transaction or transactions.

TRANSACTION <transaction>

The optional TRANSACTION <transaction> is used to specify the name of the transaction to be committed.

### WORK

The optional WORK keyword is included for SQL ANSI 92 compatibility. COMMIT WORK and COMMIT operate in the same way.

A transaction is a sequence of SQL statements that Recital treats as a single unit. A transaction begins with the first executable SQL statement after a BEGIN TRANSACTION. A transaction ends with a COMMIT, ROLLBACK or END TRANSACTION.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db

// Transactions
BEGIN TRANSACTION trans1;
INSERT INTO customer
  (TITLE, LAST_NAME, FIRST_NAME, INITIAL, STREET,
   CITY, STATE, ZIP,LIMIT, START_DATE)
VALUES
  ('Ms', 'Jones', 'Susan', 'B', '177 High Street', 'Beverly', 'MA', '01915', 2000, date());
INSERT INTO accounts (ORD_VALUE) VALUES (30);
// Commit the trans1 transaction
COMMIT TRANSACTION trans1;
END TRANSACTION;
// End of program
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## COMPILE DATABASE

---

### Class

Databases

### Purpose

Compile stored procedure files in the specified database or databases

### Syntax

COMPILE DATABASE <database name> | <skeleton>

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The COMPILE DATABASE command compiles all the stored procedure files or program source files in the specified database or databases. The name of the target database is specified in <database name>. Multiple databases can be specified using the skeleton and wild card characters. The database or databases need not be open when the COMPILE DATABASE command is issued.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

### Example

```
> compile database southwind
```

```
Recital/SQL> set sql to vfp
VFP/SQL> COMPILE DATABASE hr
```

```
Recital/SQL> COMPILE DATABASE Mirage_*;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CREATE BRIDGE

---

### Class

SQL Applications

### Purpose

Creates a Recital bridge file for connection to RMS or C-ISAM data files

### Syntax

```
CREATE BRIDGE <bridge>
TYPE <bridgetype> EXTERNAL <externalname> METADATA <metadataname> ALIAS <aliasname>
[INDEX <index> [, <index>]]
| AS <cKeyPairString>
```

### See Also

ALTER TABLE, CREATE TABLE, DROP BRIDGE, DROP TABLE

### Description

The CREATE BRIDGE command is used to create a bridge file to an external C-ISAM or RMS file. Recital clients can access Informix compatible C-ISAM files and the following fixed length RMS File types: RMS Indexed Sequential, RMS Relative and RMS Sequential. Data access is achieved through a bridge. This requires the creation of a bridge file and an empty Recital table that has a structure matching that of the external file. The empty Recital table can be created using the SQL CREATE command or the Recital Terminal Developer CREATE Development Tool. By convention, the empty structure file is given the file extension '.str' rather than the default '.dbf', which is often given to the bridge file instead.

Keywords	Description	Maximum Width (Characters)
bridge	The name of the bridge file. The file extension defaults to '.brg'.	10 + file extension
bridgetype	External data file type: CISAM, RMSIDX, RMSREL or RMSSEQ.	10
externalname	Name of the external data file.	80
metadataname	Name of the Recital structure table.	80
aliasname	The name to use to access the file.	10
index	RMS only. For RMS Indexed Sequential files specify the existing RMS index keys. For RMS Relative and Sequential file structures you can build Recital single indexes and associate them with the bridge file. In these cases, specify the full index filenames.	50

The AS <cKeyPairString> is defined using the following key pairs:

Key Pair	Description
TYPE=<bridgetype>	External data file type: CISAM, RMSIDX, RMSREL or RMSSEQ.
EXTERNAL=<externalname>	Name of the external data file.
METADATA=<metadataname>	Name of the Recital structure table.
ALIAS=<aliasname>	The name to use to access the file.
INDEXKEY1=<index> INDEXKEY2=<index> INDEXKEY3=<index> INDEXKEY4=<index> INDEXKEY5=<index> INDEXKEY6=<index> INDEXKEY7=<index>	RMS only. For RMS Indexed Sequential files specify the existing RMS index keys. For RMS Relative and Sequential file structures you can build Recital single indexes and associate them with the bridge file. In these cases, specify the full index filenames.

Each key pair is separated by a semi-colon, ‘;’.

### Example

```
exec sql
CREATE BRIDGE cisamdemo.dbf
TYPE "CISAM"
EXTERNAL "cisamdemo.dat"
METADATA "cisamdemo.str"
ALIAS "cisamdemo";
```

//or

```
exec sql
CREATE BRIDGE cisamdemo.dbf
AS "TYPE=CISAM;EXTERNAL=cisamdemo.dat;METADATA=cisamdemo.str;ALIAS=cisamdemo";
```

```
exec sql
CREATE BRIDGE rmsidxdemo.dbf
TYPE "RMSIDX"
EXTERNAL "rmsidx.dat"
METADATA "rmsidxdemo.str"
ALIAS "rmsidxdemo"
INDEX "acc_prefix+acc_no,acc_prefix+str(ord_total)";
```

//or

```
exec sql
CREATE BRIDGE rmsidxdemo.dbf
AS "type=RMSIDX;external=rmsidx.dat;metadata=rmsidxdemo.str;alias=rmsidxdemo;;
indexkey1=acc_prefix+acc_no;indexkey2=acc_prefix+str(ord_total)";
```



```
exec sql  
CREATE BRIDGE rmsreldemo.dbf  
TYPE "RMSRELX"  
EXTERNAL "rmsrel.dat"  
METADATA "rmsreldemo.str"  
ALIAS "rmsreldemo"  
INDEX "ind1.ndx,ind2.ndx,ind3.ndx";
```

//or

```
exec sql  
CREATE BRIDGE rmsreldemo.dbf  
AS "type=RMSREL;external=rmsrel.dat;metadata=rmsreldemo.str;alias=rmsreldemo;;  
indexkey1=ind1.ndx;indexkey2=ind2.ndx,indexkey3=ind3.ndx";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CREATE CONNECTION

---

### Class

Data Connectivity

### Purpose

Create a gateway connection file

### Syntax

```
CREATE CONNECTION <filename> | (<expC1>) | ?
[DATASOURCE <cNode> [USERID <cUser>] [PASSWORD <cPass>] [DATABASE <cDatabase>]]
| [CONNSTRING <cConnstring>]
| [AS <cKeyPairString>]
```

### See Also

DROP CONNECTION, REMOTE DATA CONNECTIVITY FUNCTIONS

### Description

The CREATE CONNECTION command is used to create a gateway connection file to an external SQL database. The connection file can be used by the SQLCONNECT() and SQLSTRINGCONNECT() remote data connectivity functions to establish a connection for SQL access to the database.

Keywords	Description
filename	The name of the gateway connection. If no file extension is specified, then .gtw is used.
(<expC>)	A character expression, enclosed in round brackets, that returns a valid filename for the gateway connection.
?	Displays a Select a File dialog allowing an existing connection file to be selected and modified. This is only applicable for Recital Terminal Developer.

If the optional clauses are not specified, the CREATE GATEWAY worksurface provides a full screen facility for gateway connection file creation in Recital Terminal Developer. The following elements can be defined for the gateway connection:

Server Element	Description
Name	The remote database server name, e.g. RECITAL
Network Node Name	The node name or IP address of the server
Protocol Type	The connection protocol, TCP/IP
Login Username	The login for the database server
Login Password	The login password for the database server
Database Name	The full name (including path if applicable) of the database

Information relating to individual tables is not required and is relevant only when gateway files are being used within the Recital forms system rather than with the remote data connectivity functions via cursor access.

Server Element	Description
Table Name	The table name from the database
Table Primary Key	The primary index key for the table
Table Restriction	An optional SQL SELECT...WHERE clause

Client Element	Description
Structure Name	The name of the matching Recital structure table
Alias Name	The alias name for the client table
Default Form Name	The default screen form to be used

The connection details can be supplied using the DATASOURCE or CONNSTRING clauses:

Keyword	Description
DATASOURCE <cNode>	Specify the node name or IP address of the server
USERID <cUser>	Specify the login for the database server
PASSWORD <cPass>	Specify the login password for the database server
DATABASE <cDatabase>	Specify the name of the database
CONNSTRING <cConnstring>	Specify an ODBC connection string

The remote database server name defaults to RECITAL.

The AS <cKeyPairString> is defined using the following key pairs:

Key Pair	Description
TYPE=<cType>	Specify the connection type: ODBC, JDBC, OLEDB, ORACLE, INGRES, MYSQL, POSTRGRESQL, RECITAL
NODE=<cNode>	Specify the node name or IP address of the server
USERID=<cUser>	Specify the login for the database server
PASSWORD=<cPass>	Specify the login password for the database server
DATABASE=<cDatabase>	Specify the name of the database
CONNECTION=<cConnstring>	Specify an ODBC, JDBC or OLEDB connection string
TABLE=<cTable>	Specify the database table name
PRIMARYKEY=<cKey>	Specify the primary key for the database table
SQLSELECT=<cSelect>	Specify an optional SQL SELECT...WHERE clause

Each key pair is separated by a semi-colon, ‘;’.

### Example

```
// config.db
set sql to vfp
set sql on
// end of config.db
```

```
CREATE CONNECTION conn1 DATASOURCE "server1" USERID "user1";
    PASSWORD "pass1" DATABASE "/usr/recital/data/demo"
nStatHand = SQLCONNECT("conn1.gtw")
if nStatHand < 1
    dialog box [Could not connect]
else
    nTabEnd = SQLTABLES(nStatHand)
    if nTabEnd = 1
        select sqlresult
        browse
    endif
    SQLDISCONNECT(nStatHand)
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## CREATE CURSOR

---

### Class

SQL Applications

### Purpose

Creates a database with the specified name

### Syntax

```
CREATE CURSOR <cursor>
[(<column> <datatype> [(<precision> [,<scale>])])
[NULL | NOT NULL]
[CHECK <expression> [ERROR <text>]]
[AUTOINC [NEXTVALUE <NextValue> [STEP <StepValue>]]]
[DEFAULT <expression>]
[UNIQUE [COLLATE <cCollateSequence>]]
[NOCPTRANS]
[, ...]]
| [FROM ARRAY <array>]
```

### See Also

ALTER INDEX, ALTER TABLE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DROP DATABASE, DROP INDEX, DROP TABLE, USE

### Description

The CREATE CURSOR command creates a temporary table with the specified name. Columns to be included in the table can be specified individually or details loaded from an existing array.

Keywords	Description
cursor	The name of the temporary table to be created.
column	The name of the column to be created.
datatype	The column's data type.
precision	The width of the column where not fixed.
scale	The column's decimal places where required.
NULL   NOT NULL	Specifies whether this column can have NULL values. NULL allows NULL values, NOT NULL prohibits NULL values.
CHECK <expression>	Validation rule for the column. The <expression> must evaluate to true (.T.), valid value or false (.F.), invalid value.
ERROR <text>	An optional error message, <text>, to be displayed when the CHECK <expression> validation fails.
AUTOINC	Enables auto incrementing for the column
NEXTVALUE <NextValue>	The specified <NextValue> is the numeric start value for the auto incrementing.
STEP <StepValue>	The specified <StepValue> determines the increment value. By default values are incremented by 1.
DEFAULT <expression>	The specified <expression> is used as the default value for the column.
UNIQUE	Creates a unique index on this column.
COLLATE <cCollateSequence>	The specified <cCollateSequence> is used as the index collating sequence.
NOCPTRANS	Disables code page translation for character and memo columns.
FROM ARRAY <array>	The table structure is taken from an existing array, whose name is

	specified in <array>. The array contents must be the column name, type, precision and scale for each column in the temporary table.
--	---

**Example**

```
CREATE CURSOR tempstaff
  (staff_no CHAR(6) NOT NULL UNIQUE,
   lastname CHAR(15) NOT NULL,
   firstname CHAR(10),
   hiredate DATE,
   location CHAR(15),
   supervisor CHAR(6),
   salary DECIMAL(6,0),
   picture VARBINARY,
   history LONG VARCHAR,
   commission DECIMAL(4,1));
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CREATE DATABASE

---

### Class

Databases

### Purpose

Creates a database with the specified name

### Syntax

CREATE DATABASE [IF NOT EXISTS] <database name>

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The CREATE DATABASE command creates a new database with the specified name. An error occurs if the database already exists unless the IF NOT EXISTS clause is specified.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. Since there are no tables in a database when it is initially created, the CREATE DATABASE statement creates an empty directory. The directory is created as a sub-directory of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

### Example

```
// cre_dat.sql
CREATE DATABASE hr;
USE hr;
CREATE TABLE staff
  (staff_no CHAR(6) NOT NULL UNIQUE,
   lastname CHAR(15) NOT NULL,
   firstname CHAR(10),
   hiredate DATE,
   location CHAR(15),
   supervisor CHAR(6),
   salary DECIMAL(6,0),
   picture VARBINARY,
   history LONG VARCHAR,
   commission DECIMAL(4,1));
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CREATE INDEX

---

### Class

SQL Applications

### Purpose

Creates an index for the specified table

### Syntax

```
CREATE [UNIQUE] INDEX <index> ON <table> (<column> [ASC | DESC] [...])
CREATE [UNIQUE] INDEX <index> ON <table> (<expression> [ASC | DESC])
```

### See Also

ALTER INDEX, ALTER TABLE, CREATE TABLE, DROP INDEX, SET TCACHE

### Description

The CREATE INDEX command creates an index file for the specified table. An index is a file which contains an entry for each value in the specified <column-name> of the CREATE INDEX statement. These values are known as key fields. Recital uses indexes to locate specific rows without reading the whole table. When changes are made to the table, such as column additions, deletions, or changes in key field values, the index will be updated as the table is updated. The table must be able to be locked for exclusive use during the operation.

Keywords	Description
UNIQUE	Specifies that the index being created cannot contain any duplicate keys.
index	This is the name of the index being created.
table	This is the name of the table for which the index will be created.
column	This is the name of the column that will be indexed. Multiple columns can be added from the same table separated with commas.
expression	The expression on which to index.
ASC	Ascending order is the default index creation.
DESC	This will create the index in descending order.

### Example

```
// Create an index on staff number
CREATE INDEX staff_no
  ON staff (staff_no);

// Create an index on descending hire date and ascending staff number
CREATE INDEX hiredate
  ON staff (hiredate DESC, staff_no ASC);

// Create an index on last name converted to lower case
CREATE INDEX lname
  ON staff (lower(lastname));
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## CREATE PROCEDURE

---

### Class

Databases

### Purpose

Creates a stored procedure in a database

### Syntax

```
CREATE PROCEDURE [<database>!]<procname> AS
<procedure source code>
ENDCREATE
```

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP PROCEDURE, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The CREATE PROCEDURE command creates a new stored procedure in a database. If the database name, <database>!, is specified, the stored procedure will be created in that database, otherwise the stored procedure will be created in the currently open database. If no database is open and no <database>! is specified, an error occurs. A prefix of 'sp\_' is added to the specified <procname> stored procedure name when the file is created. No file extension should be included in <procname>: the file will be given a '.prg' file extension. The SQL dialect must be set to VFP before issuing the CREATE PROCEDURE command. The SQL dialect can be set using the SET SQL TO <dialect> command.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

### Example

```
// config.db
set sql to vfp
set sql on
// end of config.db
```

```
// creaproc.prg
OPEN DATABASE southwind
CREATE PROCEDURE creaxml AS
    SELECT orders.orderid, orders.customerid, employees.employeeid, employees.lastname,
        employees.firstname, orders.orderdate, orders.freight, orders.requireddate, orders.shippeddate,
        orders.shipvia, orders.shipname, orders.shipaddress, orders.shipcity, orders.shipregion,
        orders.shippostalcode, orders.shipcountry, customers.companyname, customers.address,
        customers.city, customers.region, customers.postalcode, customers.country
    FROM orders INNER JOIN customers ON customers.customerid = orders.customerid,
        orders INNER JOIN employees ON orders.employeeid = employees.employeeid
    SAVE AS XML orderinfo
ENDCREATE
// end of creaproc.prg
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CREATE TABLE

---

### Class

SQL Applications

### Purpose

Creates a table

### Syntax

```
CREATE [TEMPORARY] TABLE | DBF [IF NOT EXISTS] [<database>!]<table>
[NAME <LongTableName>]
[FREE]
(<column> <datatype> [<column constraint> [...]][,...] [<table constraint> [...]])
[TYPE=CLIPPER | CLIPPER5 | RECITAL | DBASE3 | DBASE4 | FOXPLUS | FOXPRO | VFP]
|[FROM] XML <.xml file> [LOAD]
| FROM ARRAY <array>
```

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, DROP TABLE, INSERT, SELECT, CONSTRAINTS, DATA TYPES, SET XMLFORMAT, GETENV()

### Description

The CREATE TABLE and CREATE DBF commands are synonymous. Each creates a new table in the current database. The INSERT command can be used to populate the table with data. The ALTER TABLE command is used to change the table definition once it is created. CREATE INDEX can also be used to add new indexes and DROP INDEX to remove existing ones.

Keywords	Description
TEMPORARY	The table is created as a temporary table for this process and will be deleted when the process terminates.
IF NOT EXISTS	The table is only created if it does not already exist. An error occurs if the table already exists and the IF NOT EXISTS clause is not specified.
database	The name of the database in which the table should be created. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table to be created.
NAME<LongTableName>	Specify an alternative name for the table.
FREE	Specify that the table is not to be added to the currently active database.
column	This is the name of the column to be created.

datatype	The data type to be stored in that column, and applicable length and precision. See the data types section for additional information.
column constraint	A column constraint.
table constraint	A table constraint.
TYPE=CLIPPER   CLIPPER5   RECITAL   DBASE3   DBASE4   FOXPLUS   FOXPRO   VFP	The table's file format. The default is RECITAL or the current SET FILETYPE format if set.
[FROM] XML <.xml file> [LOAD]	The table structure is taken from the specified XML file. If the LOAD option is specified any data in the xml file is loaded into the newly created table. The XML file must be in ADO (Microsoft® ActiveX® Data Objects) format.
FROM ARRAY <array>	The table structure is taken from an existing array, whose name is specified in <array>. The array contents must be the column name, type, precision and scale for each column in the new table structure.

**Example**

// Create customer table with column and table constraints

EXEC SQL

```
CREATE TABLE customer
  (ACCOUNT_NO char(5) DESCRIPTION "Account Code"
    DEFAULT strzero(seqno(),5),
  TITLE char(3) DESCRIPTION "Personal Title",
  LAST_NAME char(16) DESCRIPTION "Customer's Last Name",
  FIRST_NAME char(10) DESCRIPTION "Customer's Given Name",
  INITIAL char(2) DESCRIPTION "Customer's Middle Initial",
  STREET char(25) DESCRIPTION "Street Number and Name",
  CITY char(12) DESCRIPTION "City",
  STATE char(2) DESCRIPTION "State Abbreviation"
    CHECK rlookup(customer.state,state)
    ERROR "Invalid State",
  ZIP char(10) DESCRIPTION "Zip Code",
  LIMIT decimal(11,2) DESCRIPTION "Credit Limit"
    RECALCULATE,
  BALANCE decimal(11,2) DESCRIPTION "Credit Balance"
    RECALCULATE,
  AVAILABLE decimal(11,2) DESCRIPTION "Credit Available"
    CALCULATED limit-balance,
  NOTES LONG VARCHAR DESCRIPTION "Customer Notes",
  START_DATE date DESCRIPTION "Customer Start Date"
    DEFAULT date(),
  ONOPEN "customer");
```

// Specify table format as Visual FoxPro

EXEC SQL

```
CREATE TABLE newtable
  (NEWID int AUTOINC,
  NEWNAME char(20))
  TYPE={VFP};
```

```
// Create table from XML file  
EXEC SQL  
    SELECT * FROM customer  
    SAVE AS XML cust.xml;
```

```
EXEC SQL  
    CREATE TABLE customer2  
    FROM XML cust.xml;
```

```
// Create table from array  
set compatible to vfp  
use vfptable  
afields(array1)
```

```
EXEC SQL  
    CREATE TABLE newtable FROM ARRAY array1;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CREATE TRIGGER

---

### Class

SQL Applications

### Purpose

Creates a trigger for a table

### Syntax

```
CREATE TRIGGER ON [<database>!]<table>
FOR UPDATE | INSERT | DELETE
AS <expression>
```

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, DROP TABLE, INSERT, SELECT, CONSTRAINTS, DATA TYPES, SET TCACHE, SET XMLFORMAT, GETENV()

### Description

The CREATE TRIGGER command is used to create a trigger for the specified table. Triggers cause the logical <expression> to be evaluated when certain operations are attempted.

Trigger	Operation
UPDATE	Attempt to modify an existing record
INSERT	Attempt to add a new record
DELETE	Attempt to delete an existing record

If the <expression> evaluates to False (.F.) the operation does not complete. If the <expression> evaluates to True (.T.) the operations does complete.

Keywords	Description
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table
FOR UPDATE   INSERT   DELETE	Specifies the type of trigger to be created.
expression	A logical expression to be evaluated

**Example**

USE accounts

CREATE TRIGGER ON customer FOR UPDATE AS .not. empty(CustName)

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CREATE VIEW

---

### Class

SQL Applications

### Purpose

Creates a logical view based on one or more tables

### Syntax

```
CREATE [SQL] VIEW <view> [(<alias> [...])] AS <sub-query>
```

### See Also

DROP VIEW, SELECT

### Description

The CREATE VIEW or CREATE SQL VIEW commands create a logical view based on one or more tables. A view is a logical table that allows you to access data from other tables. A view itself contains no data.

Keywords	Description
view	The name of the view to be created.
alias	This is an optional identifier that names a column in the view. The number of columns that you name must match the number of columns in the sub-query
AS sub-query	This identifies columns and rows of the tables from which the view is created. The sub-query is a SELECT statement.

When the view is created, the view definition is written into the sysodbc.ini file in the current directory. If no sysodbc.ini file currently exists, it will be created. The DROP VIEW command removes the view definition from this file. The view will be available until the DROP VIEW command is issued or the sysodbc.ini file is manually modified. The data extracted from the view is the data current at the time of the SELECT statement, not at the time of the view creation.

### Example

```
// Create a view based on price being over $10
CREATE VIEW OverTen AS
    SELECT * FROM orders WHERE price > 10;
```

```
// Create a view based on price being over $20
CREATE SQL VIEW OverTwenty AS
    SELECT * FROM orders WHERE price > 20;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## DECLARE CURSOR

---

### Class

SQL Applications

### Purpose

Declares a pointer to a logical table

### Syntax

```
DECLARE <cursor> [READ ONLY | INSERT ONLY]
    [TABLE] CURSOR FOR SELECT <statement>
```

### See Also

CLOSE, DROP CURSOR, FETCH, OPEN, SELECT

### Description

The DECLARE CURSOR command declares a cursor to represent the active set of rows specified by a SELECT or INSERT statement. It declares a cursor (a pointer to a logical table) to be processed in an application program. A logical table is a temporary collection of data that satisfy conditions specified in a SELECT statement. Declared cursors are opened with the OPEN statement and closed with the CLOSE statement. After a cursor has been CLOSED, it may be accessed again by issuing another OPEN statement. A cursor is not released until a DROP CURSOR statement is issued.

This command can only be used in Embedded SQL. The cursor cannot already be open.

Keywords	Description
cursor	The name of the cursor to be opened.
READ ONLY	The cursor is opened read only.
INSERT ONLY	The cursor is opened for inserts only.
TABLE	This is for compatibility only.
SELECT statement	This is a SELECT statement to be associated with the cursor. The select statement cannot contain an INTO clause.

### Example

```
// Declare the cursor to select records from the accounts table
EXEC SQL
    DECLARE accounts CURSOR FOR
    SELECT name,address,ord_value,balance
    FROM accounts
    ORDER BY name;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DELETE

---

### Class

SQL Applications

### Purpose

Deletes one or more rows from a table

### Syntax

```
DELETE FROM [<database>!]<table>
[WHERE CURRENT OF <cursor> | [CURRENT OF] <condition>]
```

### See Also

EXECUTE IMMEDIATE, INSERT, GETENV(), SET TCACHE

### Description

The DELETE command executes an immediate physical deletion of the specified records i.e., records are permanently removed from the table, and cannot be recalled.

To perform a DELETE operation you must have the DELETE privilege or be the owner of the table.

Keywords	Description
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table from which to delete the rows.
WHERE	Specifies which rows are to be deleted.
CURRENT OF cursor	Deletes only the row most recently fetched by the cursor.
condition	Deletes only the rows that satisfy the condition.

### Example

```
DELETE FROM staff
WHERE ord_date < date();
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DELETE TRIGGER

---

### Class

SQL Applications

### Purpose

Deletes a trigger from a table

### Syntax

```
DELETE TRIGGER ON [<database>!]<table>
FOR UPDATE | INSERT | DELETE
```

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, DROP TABLE, INSERT, SELECT, CONSTRAINTS, DATA TYPES, SET XMLFORMAT, GETENV()

### Description

The DELETE TRIGGER command deletes a trigger from the specified table. Triggers are used to evaluate a logical expression when certain operations are attempted.

Keywords	Description
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table
FOR UPDATE   INSERT   DELETE	Specifies the type of trigger to be deleted.

### Example

```
USE accounts
CREATE TRIGGER ON customer FOR UPDATE AS .not. empty(CustName)
DELETE TRIGGER ON customer FOR UPDATE
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DISPLAY DATABASE

---

### Class

Databases

### Purpose

Display information about the active database

### Syntax

DISPLAY DATABASE

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The DISPLAY DATABASE command is used to display information about the currently active database. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the SQL OPEN DATABASE command.

The DISPLAY DATABASE command displays the following information:

- Database Name, e.g. southwind
- Database Path, e.g. /usr/recital/data/southwind

and for each table in the database the equivalent of DISPLAY STRUCTURE INDEX followed by DISPLAY DICTIONARY:

- Table file name
- Number of records
- Date of creation
- Date of last update
- Encryption status
- Field names, types, sizes and description
- Total record length
- Production DBX file name
- Index tag names, keys, types and lengths
- Dictionary information

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

### Example

```
set sql to vfp
open database southwind
display database to file sw_info
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DISPLAY TABLES

---

### Class

Databases

### Purpose

Display table information about the active database

### Syntax

DISPLAY TABLES

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

### See Also

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The DISPLAY TABLES command displays the base name and file name including the full path for each table in the currently active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the SQL OPEN DATABASE command.

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".txt" will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

**Example**

VFP/SQL > OPEN DATABASE southwind

VFP/SQL > DISPLAY TABLES

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DROP BRIDGE

---

**Class**

SQL Applications

**Purpose**

Deletes a Recital bridge file

**Syntax**

DROP BRIDGE <bridge>

**See Also**

ADD TABLE, ADD TABLE, ALTER INDEX, ALTER TABLE, CREATE BRIDGE, CREATE TABLE, DROP TABLE

**Description**

The DROP BRIDGE command is used to delete a bridge file. Recital bridge files are used in connections to external C-ISAM or RMS files.

Keywords	Description
bridge	This is the name of the bridge file being deleted. If no file extension is specified, '.brg' will be used.

**Example**

```
exec sql  
DROP BRIDGE cisamdemo.dbf;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## DROP CONNECTION

---

### Class

Data Connectivity

### Purpose

Delete a gateway connection file

### Syntax

DROP CONNECTION <filename>

### See Also

CREATE CONNECTION, REMOTE DATA CONNECTIVITY FUNCTIONS

### Description

The DROP CONNECTION command is used to delete a gateway connection file to an external SQL database. Connection files are created by the CREATE CONNECTION command and can be used by the SQLCONNECT() and SQLSTRINGCONNECT() remote data connectivity functions to establish a connection for SQL access to a database.

Keywords	Description
filename	The name of the gateway connection. If no file extension is specified, then .gtw is used.

### Example

```
// config.db
set sql to vfp
set sql on
// end of config.db
```

```
CREATE CONNECTION conn1 DATASOURCE "server1" USERID "user1";
    PASSWORD "pass1" DATABASE "/usr/recital/data/demo"
nStatHand = SQLCONNECT("conn1.gtw")
if nStatHand < 1
    dialog box [Could not connect]
else
    nTabEnd = SQLTABLES(nStatHand)
    if nTabEnd = 1
        select sqlresult
        browse
    endif
    SQLDISCONNECT(nStatHand)
endif
DROP CONNECTION conn1
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DROP CURSOR

---

**Class**

SQL Applications

**Purpose**

Frees up all system resources allocated to a cursor

**Syntax**

DROP CURSOR <cursor>

**See Also**

CLOSE, DECLARE CURSOR, OPEN, SELECT

**Description**

The DROP CURSOR command frees up all system resources allocated to the specified cursor. Cursors may be closed with the CLOSE statement, and then re-opened with the OPEN statement. The DROP CURSOR statement should only be used if the cursor is no longer needed.

This command can only be used in Embedded SQL. The cursor must already be declared.

Keywords	Description
cursor	The name of a declared cursor to be dropped.

**Example**

EXEC SQL

    DROP CURSOR accounts;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DROP DATABASE

---

**Class**

Databases

**Purpose**

Removes the specified database and all its tables

**Syntax**

DROP DATABASE [IF EXISTS] <database name>

**See Also**

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV()

**Description**

The DROP DATABASE command removes the specified database and all its tables. The number of files deleted from the database is returned. An error occurs if the database does not exist unless the IF EXISTS clause is specified.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

**Example**

DROP DATABASE temp;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DROP INDEX

---

**Class**

SQL Applications

**Purpose**

Removes an index from a table

**Syntax**

DROP INDEX <index> ON <table>

**See Also**

ADD TABLE, ALTER INDEX, CREATE INDEX, SET TCACHE

**Description**

The DROP INDEX command removes an index from a table. When the index is dropped it frees the disk space which it occupied.

The table must be able to be locked for EXCLUSIVE use during the operation.

Keywords	Description
index	The name of the index to be dropped.
table	The name of the table from which to drop the index.

**Example**

DROP INDEX staff\_no ON staff;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DROP PROCEDURE

---

### Class

Databases

### Purpose

Removes a stored procedure from a database

### Syntax

DROP PROCEDURE [<database>!]<procname>

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE PROCEDURE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The DROP PROCEDURE command removes a stored procedure from a database and physically deletes the file. If the database name, <database>!, is specified, the stored procedure will be removed from that database, otherwise the stored procedure will be removed from the currently open database. If no database is open and no <database>! is specified, an error occurs. The <procname> can include the 'sp\_' prefix and '.prg' file extension, but if these are not specified they will be assumed.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

### Example

```
OPEN DATABASE southwind;
DROP PROCEDURE creaxml;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DROP TABLE

---

### Class

SQL Applications

### Purpose

Removes a table and all its data from the database

### Syntax

DROP TABLE [<database>!]<table>

### See Also

CREATE TABLE, DROP INDEX, DROP VIEW, GETENV()

### Description

The DROP TABLE command removes a table and all its data from the database. When you drop the table, it removes the specified table and its associated index files and frees the disk space that the files occupied.

You must have ALTER privilege on the table to issue the DROP TABLE command.

Keywords	Description
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table to be dropped.

### Example

DROP TABLE hr!staff

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DROP VIEW

---

**Class**

SQL Applications

**Purpose**

Frees up all system resources allocated to a view

**Syntax**

DROP VIEW <view>

**See Also**

CREATE VIEW, SELECT

**Description**

The DROP VIEW command frees up all system resources allocated to the specified view and removes the view definition from the sysodbc.ini file in the current directory.

Keywords	Description
view	The name of the view to be dropped.

**Example**

DROP VIEW OverTen

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## END TRANSACTION

---

### Class

SQL Applications

### Purpose

Commit changes made during a transaction and close the transaction

### Syntax

```
BEGIN TRANSACTION [<transaction>]
```

```
<statements>
```

```
END TRANSACTION [<transaction>]
```

### See Also

ROLLBACK, SAVE TRANSACTION, SAVEPOINT, SET TRANSACTION, TXNISOLATION(), TXNLEVEL()

### Description

The BEGIN TRANSACTION statement is used to flag the beginning of a transaction. The END TRANSACTION statement is used to commit changes made during the transaction and close the transaction. The COMMIT statement and the ROLLBACK statement can also be used to close a transaction. The COMMIT statement will save the changes made and the ROLLBACK statement will discard the changes made.

Transactions can be nested by issuing a second or subsequent BEGIN TRANSACTION before an existing transaction has been closed. The TXNLEVEL() function returns the current transaction nesting level. When a transaction is closed, transactions nested within it are also closed.

Savepoints can be set during a transaction. These identify stages within the transaction which can subsequently be used as ROLLBACK points.

The optional <transaction> is a name for the transaction. This name can be used by the COMMIT and ROLLBACK statements.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db

// Transactions
BEGIN TRANSACTION trans1;
INSERT INTO customer
  (TITLE, LAST_NAME, FIRST_NAME, INITIAL, STREET,
   CITY, STATE, ZIP, LIMIT, START_DATE)
VALUES
  ('Ms', 'Jones', 'Susan', 'B', '177 High Street', 'Beverly', 'MA', '01915', 2000, date());
INSERT INTO accounts (ORD_VALUE) VALUES (30);
BEGIN TRANSACTION trans2;
INSERT INTO accounts (ORD_VALUE) VALUES (60);
// Rollback the trans1 transaction and any transactions
// nested in trans1
ROLLBACK TRANSACTION trans1;
```



```
END TRANSACTION;  
// End of program
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## EXEC SQL

---

**Class**

SQL Applications

**Purpose**

Indicates that the statement that follows is an SQL statement

**Syntax**

EXEC SQL  
<statement>;

**See Also**

EXECUTE, EXECUTE IMMEDIATE, SET SQL

**Description**

If SQL is set to Recital, the EXEC SQL indicates that the statement that follows it is an SQL statement.

**Example**

```
set sql to recital
EXEC SQL
select account_no from customer;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

# EXECUTE

---

## Class

SQL Applications

## Purpose

Executes an SQL statement previously set up using the PREPARE command

## Syntax

EXECUTE <statement> USING :<variable>[,<variable2>[,...]]

## See Also

EXECUTE IMMEDIATE, PREPARE

## Description

The EXECUTE command is used to execute an SQL statement previously set up using the PREPARE command. Variables used as parameters in the SQL statement must be declared before execution.

Keywords	Description
statement	An identifier for the SQL statement previously set up using PREPARE
variable	The name of a variable to be used as the first parameter to the SQL statement
variable2...	The names of variables to be used as the subsequent parameters to the SQL statement

## Example

```
stmtbuf = 'SELECT * FROM customers WHERE contactnam = ?'
```

```
exec sql
```

```
    prepare mystmt from :stmtbuf;
```

```
gcAuthor = 'Ann Devon'
```

```
exec sql
```

```
    execute mystmt using :gcAuthor;
```

```
gcAuthor = 'Yang Wang'
```

```
exec sql
```

```
    execute mystmt using :gcAuthor;
```

```
stmtbuf = 'INSERT INTO customers (customerid, companynam, contactnam) VALUES (?, ?, ?)'
```

```
exec sql
```

```
    prepare mystmt from :stmtbuf;
```

```
buf1 = '00101'
```

```
buf2 = 'Recital'
```

```
buf3 = 'US'
```

```
exec sql
```

```
    execute mystmt using :buf1, :buf2, :buf3;
```

## Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## EXECUTE IMMEDIATE

---

**Class**

SQL Applications

**Purpose**

Prepares and executes immediately a DELETE, INSERT, or UPDATE SQL statement or any Recital/4GL command

**Syntax**

EXECUTE IMMEDIATE <statement>

**See Also**

DELETE, INSERT, UPDATE

**Description**

The EXECUTE IMMEDIATE command is used to prepare and execute immediately a DELETE, INSERT, or UPDATE SQL statement or any Recital/4GL command.

Keywords	Description
statement	The SQL DELETE, INSERT or UPDATE statement, or 4GL command to execute immediately.

**Example**

EXECUTE IMMEDIATE set deleted off;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## FETCH

---

### Class

SQL Applications

### Purpose

Fetches rows returned by the SELECT statement defined in a CURSOR

### Syntax

FETCH

```
[NEXT | PREVIOUS | FIRST | LAST | ABSOLUTE <row position> |
CURRENT | RELATIVE <row position>]
<cursor> INTO [<data variable> [, ...] | XML <xml filename>]
```

### See Also

DECLARE CURSOR, DROP CURSOR, OPEN, SELECT

### Description

The FETCH command fetches rows returned by the SELECT statement defined in a CURSOR. It reads the values in the specified row from the open cursor and places them in the bound data variables. If the variables do not exist, they are created. If they do exist, they are updated. The cursor must have already been opened with the OPEN statement.

Keywords	Description
NEXT	Retrieves the next row from the open cursor. This is the default if no direction is specified.
PREVIOUS	Retrieves the previous row from the open cursor.
FIRST	Retrieves the first row from the open cursor.
LAST	Retrieves the last row from the open cursor.
ABSOLUTE row position	Retrieves the absolute row position. Row positions start at 1 in the open cursor.
CURRENT	Retrieves the current row from the open cursor.
RELATIVE row position	Retrieves the row position relative to the current cursor position. The row position can be negative to move to previous rows, positive to move to next rows and zero to retrieve the current row.
cursor	The cursor that has been declared with the DECLARE CURSOR statement.
INTO data variable	Specifies a list of data variables to receive the extracted row values. The list may contain fewer variables than there are table columns, but may not contain more variables than there are table columns.
INTO XML <xml filename>	Specify the name of an XML file that will be created and populated with all the rows from the open cursor.

**Example**

// Declare the cursor to retrieve records from the accounts table

EXEC SQL

    DECLARE accounts CURSOR FOR  
    SELECT name, address, ord\_value, balance  
    FROM accounts  
    ORDER BY name;

// Open the cursor and establish a temporary set of records

EXEC SQL

    OPEN accounts;

// Retrieve each row from open cursor

DO WHILE sqlcode = 0

    EXEC SQL

        FETCH NEXT accounts

        INTO m\_name, m\_address, m\_ord\_value, m\_balance;

ENDDO

// Close the cursor

EXEC SQL

    CLOSE accounts;

// Free up any resources used for the cursor

EXEC SQL

    DROP CURSOR accounts;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## GRANT

---

### Class

SQL Applications

### Purpose

Grants access privileges for users to tables

### Syntax

GRANT

```
ALL | [SELECT [(<column> [...])]] [UPDATE [(<column> [...])]]
[INSERT] [DELETE] [ALTER] [READ ONLY [(<column> [...])]]
ON [<database>!] <table> TO '<user>,<group>' [...] | PUBLIC
```

### See Also

REVOKE, GETENV(), SET TCACHE

### Description

The GRANT command is used to grant access privileges for users to tables. It can extend user privileges but cannot limit existing privileges. Later GRANT statements do affect privileges already granted to a user. Privileges can only be removed with the REVOKE statement. To grant privileges you must be the owner of the table or have already been granted ALTER privileges.

Keywords	Description
ALL	All privileges are granted.
SELECT	The ability to name any column in a SELECT statement. The privilege can be restricted to one or more columns by listing them.
UPDATE	The ability to name any column in an UPDATE statement. The privilege can be restricted to one or more columns by listing them.
INSERT	The ability to INSERT rows into the table.
DELETE	The ability to DELETE rows from the table.
ALTER	The data type to be stored in that column, and the applicable length or precision.
READ ONLY	The ability to read from any column in a SELECT statement. The privilege can be restricted to one or more columns by listing them.
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table to which the privileges are granted.
user	The user access control string that will be granted the privilege. User access control strings are defined by the operating system.
group	The group access control string that will be granted the privilege. Group access control strings are defined by the operating system.
PUBLIC	All users and groups will be granted the privilege.

**Example**

// Grant update privilege for columns lastname and firstname and insert for the table

EXEC SQL

```
GRANT UPDATE (lastname, firstname) INSERT  
ON customer  
TO '[20,100]';
```

// Grant all privileges to all users

EXEC SQL

```
GRANT ALL ON test TO PUBLIC;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## INSERT

---

### Class

SQL Applications

### Purpose

Inserts one or more rows into a table

### Syntax

```
INSERT INTO [<database>!]<table> [(<column> [...])]
    VALUES(<expr> [, ] | NULL [, ] | <empty> [, ] [...])
    | <sub-query>
    | [FROM] XML <.xml file>
INSERT INTO [<database>!]<table>
    FROM ARRAY <array>
    | FROM MEMVAR
    | FROM NAME <ObjectName>
```

### See Also

EXECUTE IMMEDIATE, SELECT, PSEUDO COLUMNS, UPDATE, SET TCACHE, SET XMLFORMAT, GETENV()

### Description

The INSERT command inserts one or more rows into a table. An INSERT statement with a VALUES clause adds a single row to the table. An INSERT statement with a sub-query adds the rows returned by the query. To insert data you must be the owner of the table or have already been granted INSERT privileges.

Keywords	Description
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table into which to insert the rows. When data is being inserted into encrypted tables, the <table> can include the three-part encryption key, enclosed in angled brackets, appended to the table name. The SET ENCRYPTION command allows a default encryption key to be defined. If the key is not included in the <table>, this default key will be used. If the default key is not the correct key for the table, an error will be given. If no default key is active, a dialog box will be displayed in Recital Terminal Developer to allow the user to enter the key.
column	The column name from the table. In the inserted row each column in this list is assigned a value from the VALUES clause or the sub-query. If you omit the column list altogether, then you must supply values for each column in the table.

VALUES	Specifies the row of values to be inserted into the table. Each <expr> must be the same data type as the column it will update. If a column has a default value defined in the Applications Data Dictionary (.dbd), the value can be omitted and the default value will be inserted. Date constants can be specified as valid dates in the current format (SET DATE, SET CENTURY, SET MARK) or as a character string in the format “DD- MMM-YYYY”, e.g. “01-Sep-2002”.
sub-query	The sub-query is a SELECT statement that returns rows that are to be inserted into the table.
[FROM] XML <.xml file>	The values to be inserted into the table are taken from the specified XML file. The XML file must be in ADO (Microsoft® ActiveX® Data Objects) format.
FROM ARRAY <array>	The values to be inserted into the table are taken from an existing array, whose name is specified in <array>.
FROM MEMVAR	The values to be inserted into the table are taken from existing memory variables with the same names as the columns. If the corresponding memory variable does not exist, the column is left blank.
FROM NAME <ObjectName>	The values to be inserted into the table are taken from an object whose properties have the same names as the columns. If the corresponding property does not exist, the column is left blank.

**Example**

// Add a new row and update the column values.

EXEC SQL

```
INSERT INTO accounts!balances
(acc_prefix, acc_no, balance)
VALUES ('hmt', 'a12345', m_value*1.75);
```

// Encrypted table example.

EXEC SQL

```
INSERT INTO encbal<key_1,key_2,key_3>
(acc_prefix, acc_no, balance)
VALUES ('hmt', 'a12345', m_value*1.75);
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## LIST DATABASE

---

### Class

Databases

### Purpose

List information about the active database

### Syntax

LIST DATABASE

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The LIST DATABASE command is used to display information about the currently active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the SQL OPEN DATABASE command.

The LIST DATABASE command displays the following information:

- Database Name, e.g. southwind
- Database Path, e.g. /usr/recital/data/southwind

and for each table in the database the equivalent of LIST STRUCTURE INDEX followed by LIST DICTIONARY:

- Table file name
- Number of records
- Date of creation
- Date of last update
- Encryption status
- Field names, types, sizes and description
- Total record length
- Production DBX file name
- Index tag names, keys, types and lengths
- Dictionary information

<b>Keyword</b>	<b>Description</b>
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

**Example**

VFP/SQL > OPEN DATABASE southwind  
VFP/SQL > LIST DATABASE

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## LIST TABLES

---

### Class

Databases

### Purpose

List table information about the active database

### Syntax

LIST TABLES

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

### See Also

ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The LIST TABLES command displays the base name and file name including the full path for each table in the currently active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the SQL OPEN DATABASE command.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

**Example**

VFP/SQL > OPEN DATABASE southwind  
VFP/SQL > LIST TABLES

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## LOCK TABLE

---

### Class

SQL Applications

### Purpose

Locks a table to control access by other processes

### Syntax

LOCK TABLE [<database>!]<table> IN SHARE | EXCLUSIVE MODE [NOWAIT]

### See Also

UPDATE

### Description

To lock tables to control access by other processes. The LOCK TABLE statement locks an entire table to restrict access by other users or transactions. There are no prerequisites required to perform this operation.

Keywords	Description
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table to set the lock mode on.
SHARE	Share locks allow queries on locked tables, but prevent updates.
EXCLUSIVE	This lock denies access by any other process to the table
NOWAIT	This specifies that if a lock cannot be granted immediately during an update, then an error should be returned.

### Example

```
LOCK TABLE staff
  IN SHARE MODE;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## OPEN

---

**Class**

SQL Applications

**Purpose**

Opens a pointer to a logical table

**Syntax**

OPEN &lt;cursor&gt;

**See Also**

CLOSE, DECLARE CURSOR, DROP CURSOR, FETCH

**Description**

The OPEN statement opens a cursor (a pointer to a logical table) that has been declared with the DECLARE CURSOR statement. The DECLARE CURSOR statement declares a cursor which may be processed in an application program. A logical table is a temporary collection of data that satisfies conditions specified in a SELECT statement. Once a declared cursor has been opened, the FETCH statement may be used to obtain values from the rows of its logical table. The CLOSE statement is used to close an open cursor. Cursors may be opened and closed repeatedly until the cursor is released with the DROP CURSOR statement. Cursors are also released when RECITAL is exited. The OPEN statement opens the cursor on the first row of its logical table, regardless of which row it was on when last closed.

This command can only be used in Embedded SQL. The cursor must already be declared.

Keywords	Description
cursor	The name of the cursor to be opened.

**Example**

OPEN accounts;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## OPEN DATABASE

---

### Class

Databases

### Purpose

Sets the specified database as the default database for subsequent operations or SQL queries

### Syntax

OPEN DATABASE [<database name> | ? [EXCLUSIVE | SHARED] [NOUPDATE] [VALIDATE]]

### See Also

ALTER TABLE, ADD TABLE, ALTER INDEX, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DB\_MAXWKA, DATABASE EVENTS

### Description

The OPEN DATABASE command sets the specified database, <database name> as the default database for subsequent operations or SQL queries. The database remains current until the end of the session or until the CLOSE DATABASES or another OPEN DATABASE command is issued. Tables from other databases can still be accessed, but must be indicated by including the database name in the table reference, database!table.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

If the <database name> is omitted, a prompt will be displayed to enter the name of the database to be opened. If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select the database to be opened. The dialog defaults to the DB\_DATADIR directory. This is only applicable for Recital Terminal Developer: for Recital Database and Mirage Servers, the <database name> must be specified.

### EXCLUSIVE | SHARED

Determines whether the database is opened EXCLUSIVE, which prevents other users from opening the database, or SHARED, which allows other shared users of the database. If neither keyword is included, the access is determined by the active SET EXCLUSIVE setting.

### NOUPDATE

If the NOUPDATE keyword is included, the database is opened in read only mode.

### VALIDATE

The VALIDATE keyword is included for language compatibility.

The OPEN DATABASE command does not cause the current working directory to be changed. When a database is opened, its catalog file is also opened. If no catalog file exists, it is automatically created. The catalog is a Recital table with an associated index tag, which stores information about the files in the database. The catalog is named <database>.cat, its index <database>.cam and it is opened with the alias name \_<database>, e.g. the southwind database has the catalog southwind.cat, the index southwind.cam and the alias \_southwind. It is opened in the last available workarea as determined by the DB\_MAXWKA environment variable/symbol. If a database is open, this highest workarea is unavailable for use by other tables and an error will be returned if an attempt is made to open a table in this workarea.

The catalog has the following structure.

Field Name	Type	Width	Description	Index
PATH	Character	255	File directory	N
FILENAME	Character	32	File name	N
ALIAS	Character	32	Alias	Y
TYPE	Character	3	File type	N
TITLE	Character	80	File title	N
CODE	Numeric	3	File group	N
CATEGORY	Character	10	File category	N
ORDER	Character	1	File sort tag	N
DETAILS	Memo	8	File details	N

Each program or table has a record in the catalog, each index has multiple records. For a single index, there is one record for the file itself and one record for each component of the index key. For a production tagged index there is one record for each component of the index key.

Field Name	Description
PATH	The full directory path name for the file. This may or may not be the database's directory. Files added to the database catalog via AUTOCATALOG functionality may be located in other directories. This makes them accessible as part of the database without specifying the full path or using SET PATH.
FILENAME	The name of the file (tables, programs and single index files) e.g. example.dbf, exind.ndx. Or, a reference to a component of an index key. The reference is formatted as follows: <table-alias>-<tagname   single-index-filename><number>. e.g. for a tag called 'address' on the example.dbf table with a key of city+state, there would be two records with the filenames example-address01 and example-address02.
ALIAS	For program files, this is the program basename, e.g. 'test' for 'test.prg' and for all other files this is the associated table alias name.
TYPE	dbf - Recital, FoxPro, dBase or Clipper tables or Recital Bridge Files with '.dbf' extensions, e.g. cisamdemo.dbf. dbx - Recital, FoxPro, dBase or Clipper tagged index files with '.dbx', 'cdx' or '.mdx' file extensions. ndx - Single index files with '.ndx' and '.idx' file extensions. prg - Program source files with '.prg' file extension.
TITLE	The file description if available, 'Database Catalog' if not.
CODE	Code for internal use.
CATEGORY	Data - Recital, FoxPro, dBase and Clipper tables and Recital Bridge Files with '.dbf' extensions, e.g. cisamdemo.dbf. Index - Recital, FoxPro, dBase and Clipper tagged index files with '.dbx', 'cdx' or '.mdx' file extensions and single index files with '.ndx' and '.idx' file extensions. Program - Program source files with '.prg' file extension.
ORDER	Order for internal use, tables (1), programs (6), indexes (7).
DETAILS	File details: database, index keys, table names etc.

All files in the catalog become accessible when the database is opened, whether they are in the database directory itself or in alternative paths. Single index files included in the catalog will be opened when their associated table is opened. If a single index appears in the database catalog, but its file no longer exists, it will be removed from the catalog when its associated table is next opened. New tables, tagged indexes and single indexes created while the database is open, are added automatically to the database catalog.

The database catalog can be rebuilt using the REBUILD DATABASE command. The PACK DATABASE command issues the PACK command for all tables in the database catalog. The REINDEX DATABASE command rebuilds all indexes in the database catalog.

The OPEN DATABASE command triggers the DBC\_OPENDATA database event. If a dbc\_opendata.prg program file exists in the database's directory, this will be run. If the dbc\_opendata.prg program returns .F. (False), the OPEN DATABASE operation will be abandoned.

Databases can have an associated procedure library that is activated automatically when the database is opened. If a program file with the name dbc\_<database>\_library.prg, exists in the database's directory, e.g. dbc\_southwind\_library.prg for the southwind demo database, a SET PROCEDURE...ADDITIVE is issued for this procedure library when the database is opened. When the database is closed, the procedure library is also closed.

**Example**

```
VFP/SQL> OPEN DATABASE hr EXCLUSIVE
VFP/SQL> SELECT staff_no, lastname from staff
VFP/SQL> CLOSE DATABASES
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## PACK DATABASE

---

### Class

Databases

### Purpose

Packs each table in the active database or packs the catalog and rebuilds the catalog index tags for a specified database

### Syntax

PACK DATABASE [<database name> | ?]

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INDEX, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DB\_MAXWKA

### Description

The PACK DATABASE command packs all the tables in the active database. Packing a table removes all records previously marked for deletion from the table.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the SQL OPEN DATABASE command.

If the <database name> is specified, the PACK DATABASE command will operate on the specified database's catalog file: the catalog file will be packed and its index tags rebuilt using INDEX ON. If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select the database. The dialog defaults to the DB\_DATADIR directory. This is only applicable for Recital Terminal Developer: for Recital Database and Mirage Servers, the <database name> must be specified.

### Example

```
VFP/SQL>open database southwind
VFP/SQL>pack database
VFP/SQL>close databases
VFP/SQL>pack database southwind
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## PREPARE

---

### Class

SQL Applications

### Purpose

Set up an SQL statement for subsequent execution

### Syntax

PREPARE <statement> FROM :<variable>

### See Also

EXECUTE, EXECUTE IMMEDIATE

### Description

The PREPARE command is used to set up an SQL statement for subsequent execution. The EXECUTE command is used to execute the statement. Variables used as parameters in the SQL statement must be declared before execution.

Keywords	Description
statement	An identifier for the SQL statement.
variable	The name of a variable containing the SQL statement as a string

### Example

```
stmtbuf = 'SELECT * FROM customers WHERE contactnam = ?'
```

```
exec sql
```

```
    prepare mystmt from :stmtbuf;
```

```
gcAuthor = 'Ann Devon'
```

```
exec sql
```

```
    execute mystmt using :gcAuthor;
```

```
gcAuthor = 'Yang Wang'
```

```
exec sql
```

```
    execute mystmt using :gcAuthor;
```

```
stmtbuf = 'INSERT INTO customers (customerid, companynam, contactnam) VALUES (?, ?, ?)'
```

```
exec sql
```

```
    prepare mystmt from :stmtbuf;
```

```
buf1 = '00101'
```

```
buf2 = 'Recital'
```

```
buf3 = 'US'
```

```
exec sql
```

```
    execute mystmt using :buf1, :buf2, :buf3;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## REBUILD DATABASE

---

### Class

Databases

### Purpose

Rebuilds a database catalog

### Syntax

REBUILD DATABASE [<database name> | ?]

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INDEX, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK, PACK DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DB\_MAXWKA

### Description

The REBUILD DATABASE command rebuilds the catalog for the active or specified database. This packs the catalog file and rebuilds its index tags. All valid programs, tables and known associated indexes from the database's directory are added into the catalog.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the SQL OPEN DATABASE command.

If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select the database. The dialog defaults to the DB\_DATADIR directory. This is only applicable for Recital Terminal Developer: for Recital Database and Mirage Servers, the <database name> must be specified.

### Example

```
VFP/SQL>open database southwind
VFP/SQL> rebuild database
VFP/SQL>close databases
VFP/SQL> rebuild database southwind
VFP/SQL> rebuild database ?
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## REINDEX DATABASE

---

### Class

Databases

### Purpose

Rebuilds the indexes for each table in the active database or rebuilds the catalog index tags for a specified database

### Syntax

REINDEX DATABASE [<database name> | ?]

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INDEX, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DB\_MAXWKA

### Description

The REINDEX DATABASE command rebuilds the indexes for each table in the active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the SQL OPEN DATABASE command.

If the <database name> is specified, the REINDEX DATABASE command will operate on the specified database's catalog file: the catalog's index tags will be rebuilt. If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select the database. The dialog defaults to the DB\_DATADIR directory. This is only applicable for Recital Terminal Developer: for Recital Database and Mirage Servers, the <database name> must be specified.

### Example

```
VFP/SQL>open database southwind
VFP/SQL>reindex database
VFP/SQL>close databases
VFP/SQL>reindex database southwind
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## RESTORE DATABASE

---

### Class

Databases

### Purpose

Imports bridge, table, and associated files into the current or specified database from ASCII format files created by the BACKUP DATABASE command

### Syntax

RESTORE DATABASE [<database name> | ?]

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, BUILD, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INSTALL, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV()

### Description

The RESTORE DATABASE command issues an INSTALL to import bridge files, tables and associated memo, dictionary and multiple index files into the currently open or specified database. The files must be in ASCII format previously created by the BACKUP DATABASE command and can be transferred from a binary incompatible platform.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

If the <database name> is omitted, the RESTORE DATABASE command will operate on the active database. If no database is currently open, an error will be returned. If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select a database. The dialog defaults to the DB\_DATADIR directory. This is only applicable for Recital Terminal Developer: for Recital Database and Mirage Servers, the <database name> must be specified if the required database is not already open.

The ASCII files must be in a sub-directory of the Recital backup directory. The environment variable / symbol DB\_BACKUPDIR points to the current Recital backup directory and can be queried using the GETENV() function. The sub-directory must have the same name as the database.



**Example**

// On Source machine:

Recital/SQL> backup database southwind;

// Transfer southwind sub-directory from DB\_BACKUPDIR on source machine

// to DB\_BACKUPDIR on target machine

// On Target Machine

Recital/SQL> restore database southwind;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## REVOKE

---

### Class

SQL Applications

### Purpose

Revoke access privileges for users to tables.

### Syntax

REVOKE

```
ALL | [SELECT [(<column> [...])]] [UPDATE [(<column> [...])]]
[INSERT] [DELETE] [ALTER] [READ ONLY [(<column> [...])]]
ON [<database>!]<table> FROM '<user>,<group>' [...] | PUBLIC
```

### See Also

GRANT, GETENV(), SET TCACHE

### Description

To revoke access privileges for users to tables. The REVOKE statement can only remove existing user privileges given with the GRANT statement. To revoke privileges you must be the owner of the table or have already been granted ALTER privileges.

Keywords	Description
ALL	All privileges are revoked.
SELECT	The ability to name any column in a SELECT statement. The privilege can be restricted to one or more columns by listing them.
UPDATE	The ability to name any column in an UPDATE statement. The privilege can be restricted to one or more columns by listing them.
INSERT	The ability to INSERT rows into the table.
DELETE	The ability to DELETE rows from the table.
ALTER	The data type to be stored in that column, and the applicable length or precision.
READ ONLY	The ability to read from any column in a SELECT statement. The privilege can be restricted to one or more columns by listing them.
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table from which to revoke the privileges.
user	The user access control string that will be revoked from the privilege. User access control strings are defined by the operating system.
group	The group access control string that will be revoked the privilege. Group access control strings are defined by the operating system.
PUBLIC	All users and groups will be revoked privileges

**Example**

// Revoke update privilege for columns lastname and firstname and insert on the table

EXEC SQL

```
    REVOKE UPDATE (lastname, firstname) INSERT  
    ON customer  
    FROM '[20,100]';
```

// Grant all privileges to all users

EXEC SQL

```
    REVOKE ALL  
    ON test  
    FROM PUBLIC;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ROLLBACK

---

### Class

SQL Applications

### Purpose

Ends the current transaction, undoing all changes

### Syntax

ROLLBACK [TRANSACTION <transaction> | <savepoint>] [WORK] [TO SAVEPOINT <savepoint>]

### See Also

BEGIN...END TRANSACTION, COMMIT, SAVE TRANSACTION, SAVEPOINT, SET TRANSACTION, TXNISOLATION(), TXNLEVEL()

### Description

The ROLLBACK statement ends the current or specified transaction and any transactions that are nested within it and discards all changes performed in the transaction or transactions.

TRANSACTION <transaction> | <savepoint>

The optional TRANSACTION <transaction> | <savepoint> is used to specify the name of the transaction to be rolled back. The <transaction> is the name of the transactions defined in the BEGIN TRANSACTION <transaction> statement. The <savepoint> is the name defined in the SAVE TRANSACTION <savepoint> statement.

### WORK

The optional WORK keyword is included for SQL ANSI 92 compatibility. ROLLBACK WORK and ROLLBACK operate in the same way.

TO SAVEPOINT <savepoint>

The optional TO SAVEPOINT <savepoint> clause causes a partial rollback of the transaction to be carried out. Changes made since the specified <savepoint> was declared are discarded and the transaction continues from the <savepoint>.

A transaction is a sequence of SQL statements that Recital treats as a single unit. A transaction begins with the first executable SQL statement after a BEGIN TRANSACTION. A transaction ends with a COMMIT, ROLLBACK or END TRANSACTION.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db

// Transactions
BEGIN TRANSACTION trans1;
INSERT INTO customer
  (TITLE, LAST_NAME, FIRST_NAME, INITIAL, STREET,
   CITY, STATE, ZIP, LIMIT, START_DATE)
VALUES
  ('Ms', 'Jones', 'Susan', 'B', '177 High Street', 'Beverly', 'MA', '01915', 2000, date());
INSERT INTO accounts (ORD_VALUE) VALUES (30);
// Rollback the trans1 transaction
```

```
ROLLBACK TRANSACTION trans1;  
END TRANSACTION;  
// End of program
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SAVE TRANSACTION

---

### Class

SQL Applications

### Purpose

Identifies a stage within a transaction which can subsequently be used as ROLLBACK point.

### Syntax

SAVE TRANSACTION <savepoint>

### See Also

BEGIN...END TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION, TXNISOLATION(), TXNLEVEL()

### Description

The SAVE TRANSACTION statement identifies a stage within a transaction which can subsequently be used as ROLLBACK point. The name of the savepoint is specified in <savepoint>.

Issuing the SAVE TRANSACTION <savepoint> statement causes the TXNLEVEL() to increase by 1. If the transaction is rolled back to the <savepoint> using the ROLLBACK command, the TXNLEVEL will decrease by 1 and a partial rollback of the transaction will be carried out. Changes made since the specified <savepoint> was declared are discarded and the transaction continues from the <savepoint>.

A transaction is a sequence of SQL statements that Recital treats as a single unit. A transaction begins with the first executable SQL statement after a BEGIN TRANSACTION. A transaction ends with a COMMIT, ROLLBACK or END TRANSACTION.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db

// SAVE TRANSACTION <savepoint>
BEGIN TRANSACTION parent_and_child;
INSERT INTO customer
  (TITLE, LAST_NAME, FIRST_NAME, INITIAL, STREET,
   CITY, STATE, ZIP,LIMIT, START_DATE)
VALUES
  ('Ms', 'Jones', 'Susan', 'B', '177 High Street', 'Beverly', 'MA', '01915', 2000, date());
SAVE TRANSACTION parent_added;
INSERT INTO accounts (ORD_VALUE) VALUES (30);
ROLLBACK TRANSACTION parent_added;
END TRANSACTION;
// End of program
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SAVEPOINT

---

### Class

SQL Applications

### Purpose

Identifies a stage within a transaction which can subsequently be used as ROLLBACK point.

### Syntax

SAVEPOINT <savepoint>

### See Also

BEGIN...END TRANSACTION, COMMIT, ROLLBACK, SAVE TRANSACTION, SET TRANSACTION, TXNISOLATION(), TXNLEVEL()

### Description

The SAVEPOINT statement identifies a stage within a transaction which can subsequently be used as ROLLBACK point. The name of the savepoint is specified in <savepoint>.

Issuing the SAVEPOINT <savepoint> statement causes the TXNLEVEL() to increase by 1. If the transaction is rolled back to the <savepoint> using the ROLLBACK command, the TXNLEVEL will decrease by 1 and a partial rollback of the transaction will be carried out. Changes made since the specified <savepoint> was declared are discarded and the transaction continues from the <savepoint>.

A transaction is a sequence of SQL statements that Recital treats as a single unit. A transaction begins with the first executable SQL statement after a BEGIN TRANSACTION. A transaction ends with a COMMIT, ROLLBACK or END TRANSACTION.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db

// SAVEPOINT <savepoint>
BEGIN TRANSACTION parent_and_child;
INSERT INTO customer
  (TITLE, LAST_NAME, FIRST_NAME, INITIAL, STREET,
   CITY, STATE, ZIP,LIMIT, START_DATE)
VALUES
  ('Ms', 'Jones', 'Susan', 'B', '177 High Street', 'Beverly', 'MA', '01915', 2000, date());
SAVEPOINT parent_added;
INSERT INTO accounts (ORD_VALUE) VALUES (30);
ROLLBACK TRANSACTION parent_added;
END TRANSACTION;
// End of program
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SELECT

---

### Class

SQL Applications

### Purpose

Retrieves data from one or more tables or views

### Syntax

```
SELECT [ALL | DISTINCT | DISTINCTROW | TOP <expN> [PERCENT]]
* |
[<t_alias>|<"t_alias">].<column>|<"column"> | <constant> | <function> | <aggregate>
[[ AS] <c_alias>] [,...] [, *]
FROM {OJ <"t_alias"> <table> [LEFT | RIGHT | FULL] OUTER JOIN
    <"t_alias"> <table2> ON <exp> = <exp>}
| [FORCE] [<database>!]<table> | <view> [AS <t_alias> [, ...]]
[INNER JOIN | OUTER JOIN | LEFT [OUTER] JOIN | RIGHT [OUTER] JOIN
    (<nested select>) | [<database>!]<table2>
    ON [<database>!]<table>.<column> = [<database>!]<table2>.<column>
| CROSS JOIN | FULL [OUTER] JOIN
    [<database>!]<table2>]
[WHERE <condition>]
[GROUP BY <expr> | <column> | <number> [ASC | DESC] [,...]]
[HAVING <condition>]
[ORDER BY <expr> | <column> | <number> [ASC | DESC] [,...]]
[FOR UPDATE]
[INTO <data variable> [,...] | ARRAY <array-name>
    | CURSOR <cursor-name> [NOFILTER | READWRITE]
    | DBF | TABLE <table-name> [DATABASE <database> [NAME <long table-name>]]
| SAVE AS [<database>!]<table-name>
    | XML <xml filename> [FORMAT <RECITAL | ADO>]]
| TO FILE <text filename> [DELIMITED]
    | PRINTER [PROMPT]
    | SCREEN
[PREFERENCE <preference>] [NOCONSOLE] [PLAIN] [NOWAIT]
[UNION [ALL] <nested select>]
```

### See Also

CREATE TABLE, FETCH, INSERT, UPDATE, SET TCACHE, SET XMLFORMAT, AGGREGATES, SYSTEM TABLES, OPERATORS, PREDICATES, PSEUDO COLUMNS

### Description

The SELECT statement is used to retrieve data from one or more tables or views. It creates a logical table from other tables. A logical table is a temporary collection of data that satisfy conditions specified in a SELECT statement. To select data you must be the owner of the table or have already been granted SELECT privileges.

If no destination is specified for the results (INTO, SAVE AS or TO), then they are saved to a cursor, a temporary table with the alias name 'cursor'. This table is automatically opened in the next empty workarea and a BROWSE is issued.



Keywords	Description
ALL	Returns all the selected rows including duplicates. This is the default.
DISTINCT	Only returns one copy of each set of duplicate rows selected. Duplicate rows are those with matching values of each expression in the select list.
DISTINCTROW	Only returns one copy of each set of duplicate rows selected. Duplicate rows are those with matching values of the entire row, not just the columns in the select list.
TOP <expN> [PERCENT]	The <expN> defines the TOP number of rows to be returned from the specified select statement. The optional PERCENT keyword causes the <expN> to be used as the percentage of rows to be returned.
*	Selects all columns from all tables listed in the FROM clause.
t_alias	Provides a different name for the table. Other references to table name throughout the query must refer to this alias name. The t_alias can be enclosed in double-quotes if required.
column	The name of the column you are selecting. The column can be enclosed in double-quotes if required.
AS	Used to specify an alternative name for a table or a column
c_alias	Provides a different name for the column and column heading.
constant	This specifies a constant expression. See expressions for more information.
function	A 4GL function that may or may not include column names. See the function references for more information.
aggregate	An aggregate expression uses an aggregate function to summarize selected data from a table.
FORCE	The FORCE keyword specifies that the tables are joined in the order in which they are listed in the SELECT statement.
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The table name from which to select data. When data is being selected from encrypted tables, the table reference can include the three-part encryption key, enclosed in angled brackets, appended to the table name. The SET ENCRYPTION command allows a default encryption key to be defined. If the key is not included in the <table>, this default key will be used. If the default key is not the correct key for the table, an error will be given. If no default key is active, a dialog box will be displayed in Recital Terminal Developer to allow the user to enter the key.
view	The name of a view defined with the CREATE VIEW statement
{OJ ... OUTER JOIN ...}	Specifies the join type as left outer. This will return all the rows from the left table and matching rows from the right or a null row if no match is found.
{OJ ... LEFT OUTER JOIN ...}	Specifies the join type as left outer. This will return all the rows from the left table and matching rows from the right or a null row if no match is found.
{OJ ... RIGHT OUTER JOIN ...}	Specifies the join type as right outer. This will return all the rows from the right table and the matching rows from the left or a null row if no match is found.

{OJ ... FULL OUTER JOIN ...}	Specifies the join type as full outer. This will return all the rows from both tables.
------------------------------	--

Keywords	Description
table2	The name of the joined table in the query. When data is being selected from encrypted tables, the table2 reference can include the three-part encryption key, enclosed in angled brackets, appended to the table name. The SET ENCRYPTION command allows a default encryption key to be defined. If the key is not included in the <table2>, this default key will be used. If the default key is not the correct key for the table, an error will be given. If no default key is active, a dialog box will be displayed in Recital Terminal Developer to allow the user to enter the key.
ON <exp> = <exp>	The expression used to JOIN the two tables together.
INNER JOIN	Specifies the join type as inner. An inner join names the linking criterion used to find matches between the two tables. Only rows for which a match is found in both tables are returned.
OUTER JOIN	Specifies the join type as outer. An outer join takes two tables and displays all the rows from one table and the matching rows from the other or a null row if no matches are found.
LEFT [OUTER] JOIN	Specifies the join type as left outer. This will return all the rows from the left table and matching rows from the right or a null row if no match is found.
RIGHT [OUTER] JOIN	Specifies the join type as right outer. This will return all the rows from the right table and the matching rows from the left or a null row if no match is found.
CROSS JOIN	Specifies the join type as cross join. This will return a Cartesian product: all combination of rows.
FULL [OUTER] JOIN	Specifies the join type as full outer. This will return all the rows from both tables.
<nested select>	An additional SELECT statement.
ON <table>.column = <table2>.column	The expression used to JOIN the two tables together.
WHERE	This restricts the rows selected to those for which the condition is TRUE. If this clause is omitted than all rows are returned. The condition can be a SELECT sub-query.
GROUP BY	Groups the selected rows based on the value of an expression, the column name or number for each row and returns a single row of summary information for each group. The default, ASC, returns the data in ascending order, specifying DESC returns the data in descending order.
HAVING	Restricts the groups of rows returned to those groups for which the specified condition is TRUE. If you omit this clause then all rows are returned.
ORDER BY	Orders rows based on the value returned by an expression, a column name or number. The default, ASC, returns the data in ascending order, specifying DESC returns the data in descending order.
FOR UPDATE	Locks the selected rows.
INTO <data variable> [...]	Specify data variables to receive the data retrieved by the select statement. The select statement can only be a singleton select. The data variables will be created if they do not exist and overwritten if they do.
INTO ARRAY <array-name>	Specify an array to receive the data retrieved by the select statement. The array is automatically created, so need not be pre-defined.

Keywords	Description
INTO CURSOR <cursor-name>	Specify a cursor to receive the data retrieved by the select statement. This saves the data into a temporary table in a workarea. The Recital/4GL SELECT command can be used to select the workarea for processing with Recital/4GL commands. The NOFILTER keyword is used for creating a cursor that can be used in subsequent queries. The READWRITE keyword is used to create a temporary modifiable cursor.
INTO DBF   TABLE <table-name>	Specify a table to receive the data retrieved by the select statement. The table's database and a long name can optionally be specified using the DATABASE and NAME clauses respectively.
SAVE AS <table-name>	Specify a table name to be created and populated with the result of the select statement.
SAVE AS XML <xml filename> [FORMAT <RECITAL   ADO>]]	Specify an XML filename to be created and populated with the result of the select statement. A Document Type Definition (DTD) file, used to validate the XML file, will also be created if the XML format is set to RECITAL. The default format for XML if not specified is Microsoft® ActiveX® Data Objects (ADO). This default can also be set with the command SET XMLFORMAT TO <RECITAL   ADO>.
TO FILE [DELIMITED]	Saves the results to the specified text file. If the DELIMITED keyword is included, the results are written out in delimited format.
TO PRINTER	Sends the results to the currently defined printer. The optional PROMPT keyword is used to display a print dialog before printing.
TO SCREEN	Sends the results to the main screen or active window.
PREFERENCE	The PREFERENCE clause is used to save BROWSE window preferences.
NOCONSOLE	The NOCONSOLE keyword is used to prevent results sent to a file or printer (TO FILE   PRINTER) also being displayed on the screen.
PLAIN	The PLAIN keyword is used to disable the display of column headings.
NOWAIT	The NOWAIT keyword is used to continue program execution immediately after BROWSE window display instead of when the BROWSE window is closed.
UNION [ALL]	Combines the end result of the main SELECT statement with a secondary <nested select> SELECT statement. The ALL keyword specifies that duplicates should be retained.

**Example**

```
// Select all rows, including duplicates, from an encrypted table
```

```
EXEC SQL
```

```
    SELECT ALL *
    FROM enctab<key_1,key2,key_3>;
```

```
// Select "last_name" column from rows with a unique "last_name"
```

```
EXEC SQL
```

```
    SELECT DISTINCT last_name
    FROM customer;
```

```
// Select "last_name" column from unique rows
```

```
EXEC SQL
```

```
    SELECT DISTINCTROW last_name
    FROM customer;
```

// Select first 10 rows

EXEC SQL

```
SELECT TOP 10 *
FROM accounts;
```

// Select first 50% of the rows

EXEC SQL

```
SELECT TOP 50 PERCENT *
FROM accounts;
```

// Crystal Reports / ODBC style JOINS: LEFT OUTER, RIGHT OUTER, FULL OUTER

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM
{OJ "customer" customer LEFT OUTER JOIN "accounts" accounts
ON customer.account_no = accounts.account_no};
```

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM
{OJ "customer" customer RIGHT OUTER JOIN "accounts" accounts
ON customer.account_no = accounts.account_no};
```

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM
{OJ "customer" customer FULL OUTER JOIN "accounts" accounts
ON customer.account_no = accounts.account_no};
```

// JOINS: INNER, LEFT OUTER, RIGHT OUTER, CROSS, FULL

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM customer
INNER JOIN accounts
ON customer.account_no = accounts.account_no;
```

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM customer
LEFT OUTER JOIN accounts
ON customer.account_no = accounts.account_no;
```

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM customer
RIGHT OUTER JOIN accounts
ON customer.account_no = accounts.account_no;
```

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM customer
CROSS JOIN accounts;
```

EXEC SQL

```
SELECT customer.account_no, customer.last_name, accounts.ord_value
FROM customer
FULL OUTER JOIN accounts;
```

// JOIN with nested select

EXEC SQL

```
SELECT last_name, SUM(ord_value)
FROM customer INNER JOIN
(SELECT account_no, SUM(ord_value) AS ord_value
FROM accounts
GROUP BY account_no)
ON customer.account_no = accounts.account_no
GROUP BY last_name;
```

// Multiple JOINS

EXEC SQL

```
SELECT customer.account_no, customer.state,
state.descript, accounts.ord_value
FROM
customer RIGHT OUTER JOIN accounts
ON customer.account_no = accounts.account_no,
customer INNER JOIN state
ON customer.state = state.state
ORDER BY account_no;
```

// Select account number and order value details for Massachusetts customers

EXEC SQL

```
SELECT account_no, ord_value
FROM accounts
WHERE account_no in (SELECT account_no FROM customer WHERE state = 'MA')
ORDER BY account_no;
```

// Select all overdue accounts with 15% commission in sorted "last\_name" order.

EXEC SQL

```
SELECT last_name, zip, balance, balance*1.15
FROM customer
WHERE balance > 0
ORDER BY last_name;
```

// Select total and average balance for all overdue accounts, grouped by "limit"

EXEC SQL

```
SELECT SUM(balance), AVG(balance)
FROM customer
WHERE balance > 0
GROUP BY limit;
```

// Select total and average balance for all overdue accounts, grouped by "limit" with column aliases

EXEC SQL

```
SELECT SUM(balance) AS Total, AVG(balance) AS Average
FROM customer
WHERE balance > 0
GROUP BY limit;
```

// Select total and average balance for all overdue accounts, grouped by "limit" with column aliases

```
// For Massachusetts customers only
EXEC SQL
    SELECT SUM(balance) AS Total, AVG(balance) AS Average
    FROM customer
    WHERE balance > 0
    GROUP BY limit
    HAVING state = "MA";
```

```
// Save into an array
EXEC SQL
    SELECT SUM(balance) AS Total, AVG(balance) AS Average
    FROM customer
    WHERE balance > 0
    INTO ARRAY temp;
```

```
// Create a cursor
EXEC SQL
    SELECT SUM(balance) AS Total, AVG(balance) AS Average
    FROM customer
    WHERE balance > 0
    INTO CURSOR temp;
```

```
// Save as a table
EXEC SQL
    SELECT SUM(balance) AS Total, AVG(balance) AS Average
    FROM customer
    WHERE balance > 0
    INTO TABLE temp DATABASE mydbc;
```

```
//or
```

```
EXEC SQL
    SELECT SUM(balance) AS Total, AVG(balance) AS Average
    FROM customer
    WHERE balance > 0
    SAVE AS temp;
```

```
// Save in Microsoft® ActiveX® Data Objects XML format
// Any XML files created in the ADO format can be loaded
// with the Open method of the ADO Recordset object.
```

```
EXEC SQL
    SELECT SUM(balance) AS Total, AVG(balance) AS Average
    FROM customer
    WHERE balance > 0
    SAVE AS XML temp.xml FORMAT ADO;
```

```
// In Visual Basic the file can then be loaded:
```

```
// Set adoPrimaryRS = New Recordset
```

```
// adoPrimaryRS.Open "temp.xml"
```

```
// Save in text file format
EXEC SQL
    SELECT SUM(balance) AS Total, AVG(balance) AS Average
    FROM customer
    WHERE balance > 0
    TO FILE temp.txt;

// Select all customer accounts that have an outstanding balance or are based in Massachusetts
EXEC SQL
    SELECT account_no
    FROM customer
    WHERE state = 'MA'
    UNION SELECT account_no
    FROM accounts
    WHERE balance > 0
    ORDER BY account_no;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## UPDATE

---

### Class

SQL Applications

### Purpose

Updates specified columns

### Syntax

```
UPDATE [<database>!]<table>
    [FROM XML <xml filename> | SET <column> = <expr> [...]]
    [WHERE <condition> | CURRENT OF <cursor>]]
```

### See Also

ALTER TABLE, CREATE TABLE, EXECUTE IMMEDIATE, PSEUDO COLUMNS, SELECT, INSERT, SET TCACHE, SET XMLFORMAT, GETENV()

### Description

The UPDATE statement updates columns in the specified <table>. To update data you must be the owner of the table or have already been granted UPDATE privileges.

Keywords	Description
database	The name of the database to which the table belongs. Databases in Recital are implemented as directories containing files that correspond to the and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. The '!' character must be included between the database name and the table name.
table	The name of the table on which to perform the update. When data is being updated in encrypted tables, the table reference can include the three-part encryption key, enclosed in angled brackets, appended to the table name. The SET ENCRYPTION command allows a default encryption key to be defined. If the key is not included in the <table>, this default key will be used. If the default key is not the correct key for the table, an error will be given. If no default key is active, a dialog box will be displayed in Recital Terminal Developer to allow the user to enter the key.
FROM XML <xml filename>	Specify an XML file to use as input for the UPDATE.
column	The name of a column of the table or view that is to be updated. If you omit a column from the table then the data will be unchanged.
expr	The new value to be assigned to the corresponding column. Date constants can be specified as valid dates in the current format (SET DATE, SET CENTURY, SET MARK) or as a character string in the format "DD-MMM-YYYY", e.g. "01-Sep-2002".
condition	Restricts the rows updated to those for which the specified condition is TRUE.
CURRENT OF	Updates only the row most recently fetched by the cursor.



**Example**

// Update all accounts that are now overdue by adding a 15% commission charge

EXEC SQL

```
    UPDATE accounts
    SET ord_value=ord_value*1.15, due_date = date()+30
    WHERE paid_date < date();
```

// Declare the cursor to select records from the accounts table

EXEC SQL

```
    DECLARE accounts CURSOR FOR
    SELECT name, address, ord_value, balance
    FROM accounts
    WHERE ord_date < date();
```

// Open the cursor

EXEC SQL

```
    OPEN accounts;
```

// Fetch records one at a time from the cursor and update them

EXEC SQL

```
    FETCH accounts
    INTO m_name, m_address, m_ord_value, m_balance;
DO WHILE sqlcode = 0
    IF .not. empty(m_name) .and. m_balance <> 0
        EXEC SQL
            UPDATE accounts
            SET ord_value = ord_value*1.15, due_date = date()+30
            WHERE CURRENT OF accounts;
    ENDIF
    EXEC SQL
        FETCH accounts INTO m_name, m_address, m_ord_value, m_balance;
ENDDO
```

// Close the cursor and free up any resources used for the cursor

EXEC SQL

```
    CLOSE accounts;
```

EXEC SQL

```
    DROP CURSOR accounts;
```

// Update an encrypted table

EXEC SQL

```
    UPDATE encacc<key_1,key_2,key_3>
    SET ord_value=ord_value*1.15, due_date = date()+30
    WHERE paid_date < date();
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## USE

---

### Class

SQL Applications

### Purpose

Sets the specified database as the default database for subsequent queries

### Syntax

USE <database name>

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DROP DATABASE, DROP INDEX, DROP TABLE, SET SQL, GETENV()

### Description

The USE command sets the specified database as the default database for subsequent queries. The database remains current until the end of the session or until another USE statement is issued. Tables from other databases can still be accessed, but must be indicated by including the database name in the table reference, database!table.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

SQL must be set to MySQL before using the USE command in this way.

### Example

```
Recital/SQL> set sql to mysql
Recital/SQL> USE hr;
Recital/SQL> SELECT staff_no, lastname from staff;
Recital/SQL> USE accounts;
Recital/SQL> SELECT salesid from customer;
Recital/SQL> USE hr;
Recital/SQL> SELECT staff_no, lastname, customerno from staff, accounts!customer
               where staff.staff_no = accounts!customer.salesid;
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DATABASE EVENTS

---

### Class

Databases

### Purpose

Database events are triggered by certain database operations

### See Also

ALTER TABLE, ADD TABLE, ALTER INDEX, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DB\_MAXWKA

### Description

Database events are triggered by certain database operations. They can have code associated with them that is run automatically when the event occurs.

Event	Description
DBC_CLOSE	Occurs when a database closes
DBC_OPEN	Occurs when a database is opened

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Programs associated with database events should reside in the DB\_DATADIR sub-directory for the relevant database.

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DBC\_CLOSEDATA

---

### Class

Databases

### Purpose

Database event triggered by the closure of a database

### Syntax

```
PROCEDURE dbc_CloseData(<expC>,<expL1>)
// Commands
ENDPROC | RETURN [<expL2>]
```

or

```
PROCEDURE dbc_CloseData
LPARAMETERS [<expC>,<expL1>]
// Commands
ENDPROC | RETURN [<expL2>]
```

### See Also

ALTER TABLE, ADD TABLE, ALTER INDEX, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DB\_MAXWKA, DATABASE EVENTS

### Description

The DBC\_CLOSEDATA database event is triggered by the closure of a database. The associated code is run before the database is actually closed. Databases are closed using the CLOSE DATABASES command.

Event	Description
<expC>	The database name
<expL1>	Specifies whether the ALL keyword was included on the CLOSE DATABASES, .T. or not, .F.
<expL2>	If the dbc_closedata procedure returns .F., the database is not closed, if .T., the close operation continues

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Programs associated with database events should reside in the DB\_DATADIR sub-directory for the relevant database.

**Example**

```
procedure dbc_closedata  
lparameters cName, lAll  
close procedures  
return
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DBC\_OPENDATA

---

### Class

Databases

### Purpose

Database events are triggered by certain database operations

### Syntax

```
PROCEDURE dbc_OpenData(<expC>,<expL1>,<expL2>,<expL3>)
// Commands
ENDPROC | RETURN [<expL4>]
```

or

```
PROCEDURE dbc_OpenData
LPARAMETERS [<expC>,<expL>,<expL2>,<expL3>]
// Commands
ENDPROC | RETURN [<expL4>]
```

### See Also

ALTER TABLE, ADD TABLE, ALTER INDEX, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DATABASE(), DBUSED(), GETENV(), DB\_MAXWKA

### Description

The DBC\_OPENDATA database event is triggered when a database is opened. The associated code is run before the database is actually opened. Databases are opened using the OPEN DATABASE <database-name> command.

Event	Description
<expC>	The database name
<expL1>	Specifies whether the database is being opened exclusively, .T. or shared, .F.
<expL2>	Specifies whether the NOPUPDATE keyword was included on the OPEN DATABASE command, .T. or not, .F.
<expL3>	Specifies whether the VALIDATE keyword was included on the OPEN DATABASE command, .T. or not, .F.
<expL4>	If the dbc_opendata procedure returns .F., the database is not opened, if .T., the open operation continues

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Programs associated with database events should reside in the DB\_DATADIR sub-directory for the relevant database.

**Example**

```
procedure dbc_opendata  
lparameters cName, lExcl, lRo, lValidate  
set procedure to proclib1, proclib2 additive  
return
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## AGGREGATE FUNCTIONS

---

**Class**

SQL Applications

**Purpose**

Aggregate Functions

**See Also**

SELECT, AVG(), COUNT(), MAX(), MIN(), SUM()

**Description**

Aggregate Functions, also known as Group Functions, operate on a group of rows rather than individual rows. They return a single result row.

Aggregate Function	Description
AVG()	The AVG() Aggregate Function returns the average value for the specified numeric column or expression.
COUNT()	The COUNT() Aggregate Function returns a row count.
MAX()	The MAX() Aggregate Function returns the maximum value for the specified numeric or date column or expression.
MIN()	The MIN() Aggregate Function returns the minimum value for the specified numeric or date column or expression.
SUM()	The SUM() Aggregate Function returns the sum of the specified numeric column or expression.



## AVG()

---

**Class**

SQL Applications

**Purpose**

Returns an average value in a SELECT statement

**Syntax**

AVG(<expN>)

**See Also**

SELECT, COUNT(), MAX(), MIN(), SUM(), AGGREGATES

**Description**

Returns the average value of <expN>.

**Example**

```
SELECT AVG(sal) Average FROM accounts;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## COUNT()

---

**Class**

SQL Applications

**Purpose**

Returns the number of rows in a SELECT statement query

**Syntax**

COUNT(\* | [DISTINCT] <expr>)

**See Also**

SELECT, AVG(), MAX(), MIN(), SUM(), AGGREGATES

**Description**

Returns the number of rows in the query. If you specify DISTINCT, then only the rows which are unique in <expr> are counted. Specifying <expr> on its own returns the number of rows from the query for which <expr> is not NULL. If you specify the asterisk (\*), then all rows are counted.

**Example**

// Get a count of all rows in the accounts table.

```
SELECT COUNT(*) Total FROM accounts;
```

// Get a count of jobs

```
SELECT COUNT(jobs) Jobs FROM employee;
```

// Get a count of distinct rows for jobs

```
SELECT COUNT(DISTINCT jobs) Jobs FROM employee;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## MAX()

---

**Class**

SQL Applications

**Purpose**

Returns a maximum value in a SELECT statement

**Syntax**

MAX(<expr>)

**See Also**

SELECT, AVG(), COUNT(), MIN(), SUM(), AGGREGATES

**Description**

Returns the maximum value of <expr>. The <expr> can be a numeric or date expression.

**Example**

```
SELECT MAX(sal) Maximum FROM accounts;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## MIN()

---

**Class**

SQL Applications

**Purpose**

Returns a minimum value in a SELECT statement

**Syntax**

MIN(<expr>)

**See Also**

SELECT, AVG(), COUNT(), MAX(), SUM(), AGGREGATES

**Description**

Returns the minimum value of <expr>. The <expr> can be a numeric or date expression.

**Example**

```
SELECT MIN(sal) Minimum FROM accounts;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SUM()

---

**Class**

SQL Applications

**Purpose**

Returns a summed value in a SELECT statement

**Syntax**

SUM(<expN>)

**See Also**

SELECT, AVG(), COUNT(), MAX(), MIN(), SUM(), AGGREGATES

**Description**

Returns the sum of values of <expN>.

**Example**

```
SELECT SUM(sal) Total FROM accounts;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## REMOTE DATA CONNECTIVITY FUNCTIONS

---

### Class

SQL Applications

### Purpose

Remote data connectivity functions

### See Also

CREATE CONNECTION, INSERT, SELECT, UPDATE

### Description

The Visual FoxPro compatible remote data connectivity functions are used to handle gateway connections to third party data sources. The following functions are available:

Function	Description
SQLCANCEL()	Request that an executing SQL statement be cancelled
SQLCOLUMNS()	Store column information to a cursor
SQLCOMMIT()	Commit a transaction
SQLCONNECT()	Connect to a data source
SQLDISCONNECT()	Disconnect from a data source
SQLEXEC()	Send an SQL statement to a data source
SQLGETPROP()	Query property settings for a connection or the environment
SQLMORERESULTS()	Check if more results sets are available and if so, copy next results set to a cursor
SQLPREPARE()	Prepare an SQL statement that will be executed by the SQLEXEC() function
SQLROLLBACK()	Rollback a transaction
SQLSETPROP()	Set property settings for a connection
SQLSTRINGCONNECT()	Connect to a data source using a gateway connection string
SQLTABLES()	Store data source table names to a table

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLCANCEL()

---

### Class

SQL Applications

### Purpose

Request that an executing SQL statement be cancelled

### Syntax

SQLCANCEL(<nStatementHandle>)

### See Also

CREATE CONNECTION, SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLCANCEL() function is used to request that an executing SQL statement be cancelled. It can be used to cancel any of the following functions when they are running in asynchronous mode:

- SQLCOLUMNS()
- SQLEXEC()
- SQLMORERESULTS()
- SQLTABLES()

The SQLCANCEL() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open

Return values:

Return Value	Description
1	SQL statement was cancelled successfully
-1	Connection error
-2	Environment error

The SQLCANCEL() function is included for compatibility only. Recital gateway functions do not operate in asynchronous mode.

### Example

```
nStatHand=SQLSTRINGCONNECT("rec@rec1:user1/pass1/usr/recital/uas/data/southwind.tcpip",.T.)
if nStatHand < 1
  dialog box [Could not connect]
else
  SQLSETPROP(nStatHand, "Asynchronous", .T.)
  SQLEXEC(nStatHand, "SELECT * from example")
  SQLCANCEL(nStatHand)
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer





## SQLCOLUMNS()

---

### Class

SQL Applications

### Purpose

Store column information to a cursor

### Syntax

SQLCOLUMNS(<nStatementHandle>, <cTableName> [, "FOXPRO" | "NATIVE"] [, <cCursorName>])

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLCOLUMNS() function is used to store column information for a specified data source table to a cursor.

The SQLCOLUMNS() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open
cTableName	The table from which the column information should be returned
"FOXPRO"   "NATIVE"	Used to specify the format for the column information. "FOXPRO" is the default. "NATIVE" uses the data source format. The cursor column information is shown in the table below. "NATIVE" many include additional columns.
cCursorName	The name of the cursor to use. If cCursorName is not specified, the default name <b>SQLRESULT</b> is used.

"FOXPRO" Cursor Columns:

Column	Description
Field_name	Column name
Field_type	Column data type
Field_len	Column length
Field_dec	Number of decimal places

"NATIVE" Cursor Columns:

Column	Description
Table_qualifier	Table qualifier id
Table_owner	Table owner id
Table_name	Table name
Table_type	Table type
Column_name	Column identifier
Data_type	Column data type
Type_name	Column data type name
Precision	Column precision
Length	Data transfer size

Scale	Column scale
Radix	Base for Numeric type
Nullable	Null value support
Remarks	Table description

Return values:

Return Value	Description
.T.	Format is "NATIVE" and cTableName does not exist
.F.	Format is "FOXPRO" and cTableName does not exist
1	The table was created successfully
0	SQLCOLUMNS() still executing
-1	Connection error
-2	Environment error

### Example

```
nStatHand=SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",.T.)
if nStatHand < 1
    dialog box [Could not connect]
else
    nColEnd = SQLCOLUMNS(nStatHand, "accounts", "NATIVE", "tabinfo")
    if nColEnd = 1
        select tabinfo
        browse
    else
        dialog box [Table of Table Information could not be created]
    endif
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLCOMMIT()

---

### Class

SQL Applications

### Purpose

Commit a transaction

### Syntax

SQLCOMMIT(<nStatementHandle>)

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLCOMMIT() function is used to commit a transaction. The SQLCOMMIT() function is only required when Manual Transactions are in effect. The Transactions setting, set using SQLSETPROP(), can be either Automatic (1) or Manual (2). Manual Transactions can be rolled back using the SQLROLLBACK() function.

The SQLCOMMIT() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open

Return values:

Return Value	Description
1	SQL statement was committed successfully
-1	Error occurred

**Example**

```

nStatHand=SQLSTRINGCONNECT("rec@rec1:user1/pass1/usr/recital/uas/data/southwind.tcpip",.T.)
if nStatHand < 1
    messagebox("Cannot make connection", 16, "SQL Connect Error")
else
    messagebox("Connection made", 48, "SQL Connect Message")
    nSetEnd = SQLSETPROP(nStatHand,"Transactions",2)
    if nSetEnd = 1
        dialog box [Manual Transactions enabled]
    else
        dialog message [Unable to enable Manual Transactions. Continue?]
        if lastkey() <> 89
            SQLDISCONNECT(nStatHand)
            return
        endif
    endif
    nRET=SQLEXP(nStatHand,"INSERT INTO example (ACCOUNT_NO, TITLE, LAST_NAME,
FIRST_NAME, INITIAL, STREET, CITY, STATE, ZIP, LIMIT, START_DATE) VALUES
('00200','Mr','Doe','John','L','1 High Street','Beverly','MA','01916', 12000, {05/12/2003})")
    if SQLGETPROP(nStatHand, "Transactions") = 2
        dialog message [Commit Insert?]
        if lastkey() = 89
            dialog box "SQLCOMMIT() returned " + etos(SQLCOMMIT(nStatHand))
        else
            dialog box "SQLROLLBACK() returned " + etos(SQLROLLBACK(nStatHand))
        endif
    endif
    nSetEnd = SQLSETPROP(nStatHand, "Transactions",1)
    if nSetEnd = 1
        dialog box [Automatic Transactions enabled]
    else
        dialog message [Unable to enable Automatic Transactions.]
    endif
endif
SQLDISCONNECT(nStatHand)

```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLCONNECT()

---

### Class

SQL Applications

### Purpose

Connect to a data source

### Syntax

SQLCONNECT([<nStatementHandle>])

SQLCONNECT([<cConnectionName> | <cDataSourceName> [, <cUserID> [, <cPassword>]]  
[, <lShared>]])

SQLCONNECT([<cServerDataSource>])

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLDISCONNECT(),  
SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(),  
SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLCONNECT() function is used to connect to a data source and return a statement handle for use by other SQL functions.

If SQLCONNECT() is called without specifying any data source information, the **Select a Gateway** dialog is displayed allowing an existing gateway definition file (.gtw) to be selected. Gateway files can be created in Recital Terminal Developer using the CREATE GATEWAY | MODIFY GATEWAY tool.

Keywords	Description
nStatementHandle	The workarea in which a gateway data source is open and to which a new connection should be made.

Or

Keywords	Description
cConnectionName	Existing gateway (.gtw) file
cDataSourceName	ODBC data source name (ODBC supported platforms only)
cUserID	Data source login user id
cPassword	Data source login password
lShared	False (.F.) Connection created is not shared (default) True (.T.) Connection created is shared

Or

Keywords	Description
cServerDataSource	ODBC, OLEDB or JDBC server side data source in the format: <b>odbc:</b> odbc_data_source_name_on_server <b>oledb:</b> oledb_connection_string_on_server <b>jdbc:</b> dbc_driver_path_on_server;jdbc:Recital:args

Return values:

Return Value	Description
> 0	Statement handle to connection
-2	Connection creation error

### Example

```
// SQLCONNECT() using the 'Select a Gateway' dialog
nStatHand=SQLCONNECT()
```

```
// SQLCONNECT(<nStatementHandle>) to an active gateway opened with SQLSTRINGCONNECT()
nStatHand=SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",.T.)
if nStatHand < 1
    messagebox([Could not connect])
else
    messagebox([Connected in workarea ]+str(nStatHand,3))
    nStatHand2=SQLCONNECT(nStatHand)
    if nStatHand2 < 1
        messagebox([Could not connect])
    else
        messagebox([Connected in workarea ]+str(nStatHand2,3))
    endif
endif
```

```
// SQLCONNECT() to an existing Gateway definition file
nStatHand = SQLCONNECT("gateway1.gtw",.T.)
```

```
// SQLCONNECT() to an ODBC data source
nStatHand = SQLCONNECT("SouthWind", "recital", "recital", .T.)
```

```
//SQLCONNECT() alternative syntax to an ODBC data source
nStatHand = SQLCONNECT("odbc:southwind")
```

```
//SQLCONNECT() to an OLEDB data source
nStatHand = SQLCONNECT ("oledb:Provider=vfpoledb.1;" + ;
    "Data Source=C:\Data;Collating Sequence=general")
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLDISCONNECT()

---

### Class

SQL Applications

### Purpose

Disconnect from a data source

### Syntax

SQLDISCONNECT(<nStatementHandle>)

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLDISCONNECT() function is used to disconnect from a data source.

The SQLDISCONNECT() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open. Specifying 0 causes all active connections to be disconnected.

Return values:

Return Value	Description
1	Disconnection completed successfully
-1	Connection error
-2	Environment error

### Example

```
nStatHand=SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",.T.)
if nStatHand < 1
    dialog box [Could not connect]
else
    dialog box [Connected]
    SQLDISCONNECT(nStatHand)
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLEEXEC()

---

### Class

SQL Applications

### Purpose

Send an SQL statement to a data source

### Syntax

SQLEEXEC(<nStatementHandle>, [<cSQLCommand>, [<cCursorName>]])

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLEEXEC() function is used to send an SQL statement to the specified data source.

The SQLEEXEC() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open.
cSQLCommand	The SQL statement to be passed to the data source. The cSQLCommand can be omitted if the SQL statement has already been set up using SQLPREPARE().
cCursorName	The name of the temporary table to use. If cCursorName is not specified, the default name SQLRESULT is used. If the SQLEEXEC() is running a pre-prepared statement, the cCursorName is taken from the SQLPREPARE() setting.

Return values:

Return Value	Description
<n>	Number of results sets if more than 1
0	SQLEEXEC() is still executing
1	SQLEEXEC() finished executing
-1	Connection error



**Example**

```
nStatHand=SQLSTRINGCONNECT("rec@rec1:user1/pass1/usr/recital/uas/data/southwind.tcpip",.T.)
if nStatHand < 1
    messagebox('Cannot make connection', 16, 'SQL Connect Error')
else
    messagebox('Connection made', 48, 'SQL Connect Message')
    store "00010" to myVar
    SQLEXEC(nStatHand, "SELECT * FROM example WHERE account_no = ?myVar", "restab")
    browse
    SQLDISCONNECT(nStatHand)
endif
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLGETPROP()

---

### Class

SQL Applications

### Purpose

Query property settings for a connection or the environment

### Syntax

SQLGETPROP(<nStatementHandle>, <cSetting>)

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLGETPROP() function is used to query the property settings for a specified connection or for the current environment.

The SQLGETPROP() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open. Specifying 0 causes the SQLGETPROP() function to return the property setting for the environment.
cSetting	The property setting to query. For available property settings, please see SQLSETPROP().

Return values:

Return Value	Description
<expression>	The queried property setting
-1	Connection error
-2	Environment error

### Example

```
nStatHand = SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",T.)
if nStatHand < 1
    dialog box [Could not connect]
else
    eGetProp = SQLGETPROP(nStatHand,"ConnectionString")
    if type("eGetProp") = "N" and eGetProp < 0
        dialog box [Error Occurred]
    else
        dialog box etos(eGetProp)
    endif
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLMORERESULTS()

---

### Class

SQL Applications

### Purpose

Check if more results sets are available and if so, copy next results set to a cursor

### Syntax

SQLMORERESULTS(<nStatementHandle>)

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLMORERESULTS() function is used in non-BatchMode to check whether more results sets are available and if so, to copy the next results set to a cursor. BatchMode is set using the SQLSETPROP() function. BatchMode is always True and cannot be used to return results sets individually. It is included for compatibility reasons only.

The SQLMORERESULTS() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open

Valid Return values:

Return Value	Description
2	No more data found
1	SQL statement has finished executing
0	SQL statement is still executing
-1	Connection error
-2	Environment error

SQLMORERESULTS() always returns 2.

### Example

```
nStatHand=SQLSTRINGCONNECT("rec@rec1:user1/pass1/usr/recital/uas/data/southwind.tcpip",.T.)
if nStatHand < 1
    messagebox('Cannot make connection', 16, 'SQL Connect Error')
else
    messagebox('Connection made', 48, 'SQL Connect Message')
    store "00010" to myVar
    SQLEXEC(nStatHand, "SELECT * FROM example WHERE account_no = ?myVar", "restab")
    SQLMORERESULTS(nStatHand)
    browse
    SQLDISCONNECT(nStatHand)
endif
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLPREPARE()

---

### Class

SQL Applications

### Purpose

Prepare an SQL statement that will be executed by the SQLEXEC() function

### Syntax

SQLPREPARE(<nStatementHandle>, <cSQLCommand>, [<cCursorName>])

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLPREPARE() function is used to prepare an SQL statement which will subsequently be executed by the SQLEXEC() function on the specified data source.

The SQLPREPARE() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open.
cSQLCommand	The SQL statement to be passed to the data source.
cCursorName	The name of the cursor to use. If cCursorName is not specified, the default name SQLRESULT is used.

Return values:

Return Value	Description
1	SQLPREPARE() successful
-1	Connection error

### Example

```
nStatHand=SQLSTRINGCONNECT("rec@rec1:user1/pass1/usr/recital/uas/data/southwind.tcpip",.T.)
if nStatHand < 1
  messagebox('Cannot make connection', 16, 'SQL Connect Error')
else
  messagebox('Connection made', 48, 'SQL Connect Message')
  store "00010" to myVar
  SQLPREPARE(nStatHand, "SELECT * FROM example WHERE account_no = ?myVar", "restab")
  SQLEXEC(nStatHand)
  browse
  SQLDISCONNECT(nStatHand)
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLROLLBACK()

---

**Class**

SQL Applications

**Purpose**

Rollback a transaction

**Syntax**

SQLROLLBACK(&lt;nStatementHandle&gt;)

**See Also**

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLSETPROP(), SQLSTRINGCONNECT(), SQLTABLES()

**Description**

The SQLROLLBACK() function is used to roll back a transaction. The SQLROLLBACK() function is only required when Manual Transactions are in effect. The Transactions setting, set using SQLSETPROP(), can be either Automatic (1) or Manual (2). Manual Transactions can be committed using the SQLCOMMIT() function.

The SQLROLLBACK() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open

Return values:

Return Value	Description
1	SQL statement was rolled back successfully
-1	Error occurred

**Example**

```

nStatHand=SQLSTRINGCONNECT("rec@rec1:user1/pass1/usr/recital/uas/data/southwind.tcpip",.T.)
if nStatHand < 1
    messagebox("Cannot make connection", 16, "SQL Connect Error")
else
    messagebox("Connection made", 48, "SQL Connect Message")
    nSetEnd = SQLSETPROP(nStatHand,"Transactions",2)
    if nSetEnd = 1
        dialog box [Manual Transactions enabled]
    else
        dialog message [Unable to enable Manual Transactions. Continue?]
        if lastkey() <> 89
            SQLDISCONNECT(nStatHand)
            return
        endif
    endif
    nRET=SQLEXP(nStatHand,"INSERT INTO example (ACCOUNT_NO, TITLE, LAST_NAME,
FIRST_NAME, INITIAL, STREET, CITY, STATE, ZIP, LIMIT, START_DATE) VALUES
('00200','Mr','Doe','John','L','1 High Street','Beverly','MA','01916', 12000, {05/12/2003})")
    if SQLGETPROP(nStatHand, "Transactions") = 2
        dialog message [Commit Insert?]
        if lastkey() = 89
            dialog box "SQLCOMMIT() returned " + etos(SQLCOMMIT(nStatHand))
        else
            dialog box "SQLROLLBACK() returned " + etos(SQLROLLBACK(nStatHand))
        endif
    endif
    nSetEnd = SQLSETPROP(nStatHand, "Transactions",1)
    if nSetEnd = 1
        dialog box [Automatic Transactions enabled]
    else
        dialog message [Unable to enable Automatic Transactions.]
    endif
endif
SQLDISCONNECT(nStatHand)

```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLSETPROP()

---

### Class

SQL Applications

### Purpose

Set property settings for a connection

### Syntax

SQLSETPROP(<nStatementHandle>, <cSetting> [, <eExpression>])

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSTRINGCONNECT(), SQLTABLES()

### Description

The SQLSETPROP() function is used to set the property settings for a specified connection.

The SQLSETPROP() function operates on the gateway data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open
cSetting	The property setting to set. Please see table below for available property settings
eExpression	The value to be set. If this optional parameter is not specified, the property is set to its default value. Please see table below for available property settings values

### Property Settings:

Setting	Default	Read-only	Description
Asynchronous	False (.F.)	No	Determines whether results sets are returned synchronously (.F.) or asynchronously (.T.)
BatchMode	True (.T.)	No	Determines whether results sets are returned all at once (.T.) or one at a time using SQLMORERESULTS() (.F.)
ConnectBusy		Yes	.T. = Shared connection is busy .F. = Shared connection is not busy
ConnectString		Yes	Login connection string
ConnectTimeout	15	No	Determines the time to wait before generating a connection time-out error. (0 – 600 seconds)
DataSource		No	Data source name as specified in ODBC.INI
DispLogin	1	No	1 = The Login dialog box is displayed if any required information is missing 2 = The Login dialog box is always displayed 3 = The Login dialog box is never displayed and an error is generated if any required information is missing
DispWarnings	False (.F.)	No	.T. = Error messages are displayed .F. = Error messages are not displayed
IdleTimeout	0 (Wait indefinitely)	No	Determines the time to wait before terminating an idle connection (seconds)
ODBChdbc		Yes	Internal ODBC connection handle



## REMOTE DATA CONNECTIVITY FUNCTIONS

ODBCstmt		Yes	Internal ODBC connection handle
PacketSize	4K	No	Determines the network packet size used
Password		Yes	Login connection password
QueryTimeout	0 (Wait indefinitely)	No	Determines the time to wait before generating a time-out error. (0 – 600 seconds)
Shared		Yes	.T. = Connection is shared .F. = Connection is not shared
Transactions	1	No	1 = Automatic transaction handling 2 = Manual transaction handling using SQLCOMMIT() and SQLROLLBACK()
UserId		Yes	Login connection user id
WaitTime	100 milliseconds	No	Determines the time to wait before checking that an SQL statement has completed

Return values:

Return Value	Description
1	Property setting completed successfully
-1	Connection error
-2	Environment error

### Example

```
nStatHand = SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",.T.)
if nStatHand < 1
    dialog box [Could not connect]
else
    nSetEnd = SQLSETPROP(nStatHand,"Transactions",2)
    if nSetEnd = 1
        dialog box [Manual Transactions Enabled]
    else
        dialog box [Unable to enable Manual Transactions]
    endif
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLSTRINGCONNECT()

---

### Class

SQL Applications

### Purpose

Connect to a data source using a gateway connection string

### Syntax

SQLSTRINGCONNECT([<IShared>] | [<cConnectionString> [, <IShared>]])

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLTABLES()

### Description

The SQLSTRINGCONNECT() function is used to connect to a data source using a gateway connection string and return a statement handle for use by other SQL functions.

Keywords	Description
IShared	False (.F.) Connection created is not shared (default) True (.T.) Connection created is shared
cConnectionString	The data source gateway connection string. The cConnectionString gateway connection string format is: server@machinename:username/password-database.protocol. The shortened driver:datasource format can also be used for ODBC, OLEDB or JDBC data sources on the server. If cConnectionString is not specified, the 'Select a Gateway' dialog is displayed allowing an existing gateway definition file (.gtw) to be selected. Gateway files can be created in Recital Terminal Developer using the CREATE GATEWAY   MODIFY GATEWAY tool or via the CREATE CONNECTION command.

Gateway connection string format for cConnectionString:

Parameter	Description
Server	Database server name or abbreviation, e.g. mys or mysql.
Nodename	IP Address or hostname of the machine on which the database resides.
Username	Username to log in to the external database.
Password	Password for username above.
Database	Database to connect to.
Protocol	The network protocol. TCP/IP is assumed.
Driver:DataSource	ODBC, OLEDB or JDBC server side data source in the format: <b>odbc:</b> odbc_data_source_name_on_server <b>oledb:</b> oledb_connection_string_on_server <b>jdbc:</b> jdbc_driver_path_on_server;jdbc:Recital:args

Return values:

Return Value	Description
> 0	The workarea in which the gateway data source has been opened
-1	Connection creation error

**Example**

```
// When cConnectionString is omitted, 'Select a Gateway' dialog is displayed
// Specify connection should be shared
nStatHand = SQLSTRINGCONNECT(.T.)

// Including cConnectionString makes the connection
// Store the return value to use as the nStatementHandle for subsequent function calls
// Specify connection should be shared
nStatHand = SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",.T.)

// OLEDB DataSource
nStatHand = SQLSTRINGCONNECT ("oledb:Provider=vfpoledb.1;" +;
    "Data Source=C:\Data\;Collating Sequence=general",.T.)
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SQLTABLES()

---

### Class

SQL Applications

### Purpose

Store data source table names to a table

### Syntax

SQLTABLES(<nStatementHandle>[, <cTableTypes>] [, <cCursorName>])

### See Also

CREATE CONNECTION, SQLCANCEL(), SQLCOLUMNS(), SQLCOMMIT(), SQLCONNECT(), SQLDISCONNECT(), SQLEXEC(), SQLGETPROP(), SQLMORERESULTS(), SQLPREPARE(), SQLROLLBACK(), SQLSETPROP(), SQLSTRINGCONNECT()

### Description

The SQLTABLES() function is used to store table name information for a specified connection to a table.

The SQLTABLES() function operates on the data source specified by <nStatementHandle>.

Keywords	Description
nStatementHandle	The workarea in which the gateway data source is open
cTableType	The table type or types to include. 'TABLE', 'VIEW' and 'SYSTEM_TABLE' are all valid table types. If cTableType is omitted or empty, all types of table are included
cCursorName	The name of the table to create. If cCursorName is not specified, the default name SQLRESULT is used. The table column information is shown in the table below

cCursorName Table Columns:

Column	Description
TABLE_QUALIFIER	Table qualifier id
TABLE_OWNER	Table owner id
TABLE_NAME	Table name
TABLE_TYPE	Table type
REMARKS	Table description

Return values:

Return Value	Description
1	The cursor was created successfully
0	SQLTABLES() still executing
-1	Connection error
-2	Environment error

**Example**

```

nStatHand=SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",.T.)
if nStatHand < 1
    dialog box [Could not connect]
else
    // Store names for all table types to default results table
    nTabEnd = SQLTABLES(nStatHand)
    if nTabEnd = 1
        select sqlresult
        browse
    endif
endif

nStatHand= SQLSTRINGCONNECT("mysql@linux1:user1/pass1-database1.tcpip",.T.)
if nStatHand < 1
    dialog box [Could not connect]
else
    // Store names for specified table types to specified results table
    nTabEnd = SQLTABLES(nStatHand, "SYSTEM TABLE', 'VIEW'", "myresults")
    if nTabEnd = 1
        select myresults
        browse
    endif
endif

```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ADATABASES()

---

### Class

SQL Applications

### Purpose

Function to place the names of all open databases and their paths into a variable array.

### Syntax

ADATABASES(<array>)

### See Also

CLOSE DATABASES, DISPLAY STATUS, LIST STATUS, OPEN DATABASE, USE, ADIR(), ALIAS(), DBF(), DBUSED(), GETENV(), USED(), SET SQL

### Description

The ADATABASES() function is used to place the names of all open databases and their paths into a variable array. The name if the array is specified in <array>. If the array does not exist, it is created. If the array is smaller or larger than required, it is resized. The array is two-dimensional with two columns. The first column contains the name of an open database, the second the path for that database.

The ADATABASES() function returns the number of database names added to the array. If no databases are open or the array cannot be created, the ADATABASES() function returns 0.

NOTE: The ADATABASES() function operates on databases, not tables.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Databases can be opened using the SQL USE command, with SQL set to MYSQL, or using the SQL OPEN DATABASE command.

### Example

```
VFP/SQL> OPEN DATABASE hr EXCLUSIVE
VFP /SQL> nDatabases = adatabases(aDBCNames)
VFP /SQL> CLOSE DATABASES
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BETWEEN()

---

**Class**

Expressions and Type Conversion

**Purpose**

Function to determine whether a specified expression is between two other specified expressions

**Syntax**

BETWEEN(<exp1>,<exp2>,<exp3>)

**See Also**

BETWEEN PREDICATE

**Description**

The BETWEEN() function returns true (.T.) if the value of a specified expression lies in the range delimited by two other expressions all of the same data type. The BETWEEN() function operates on Character, Date and Numeric expressions.

Parameters	Description
<exp1>	The expression to be checked.
<exp2>	Expression representing the range minimum.
<exp3>	Expression representing the range maximum.

The <exp1> must be equal to or more than <exp2> and equal to or less than <exp3> for BETWEEN() to return true (.T.). If <exp1> is not within this range, BETWEEN() returns false (.F.).

**Example**

? between(date(),date()-10,date()+10)

**.T.**

? between("A","A","Z")

**.T.**

? between(10,1,9)

**.F.**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BITAND()

---

### Class

Bitwise Operations

### Purpose

Function to perform bitwise AND operation

### Syntax

BITAND(<expN1>,<expN2>[,...<expN26>])

### See Also

BITCLEAR(), BITLSHIFT(), BITNOT(), BITOR(), BITRSHIFT(), BITSET(), BITTEST(), BITXOR()

### Description

The BITAND() function performs a bitwise AND operation on the specified numeric parameters. Up to 26 parameters can be specified. These parameters, if not integers, will be converted to integer values before the operation takes place.

BITAND() compares each bit in turn of <expN1> and <expN2>. If both bits are 1, the corresponding bit in the result is set to 1, otherwise the result bit is 0. If <expN3> is specified, the initial result is compared bit by bit with <expN3> and a new result evaluated. This new result is then compared with <expN4>, if specified, and so on.

<expN1> bit	<expN2> bit	Result bit
0	0	0
0	1	0
1	1	1
1	0	0

### Example

```
x = 3      && 0011
y = 6      && 0110
z = 7      && 0111
? bitand(x,y)
    2      && 0010
? bitand(x,y,z)
    2      && 0010
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## BITCLEAR()

---

### Class

Bitwise Operations

### Purpose

Function to clear a specified bit in a numeric value

### Syntax

BITCLEAR(<expN1>,<expN2>)

### See Also

BITAND(), BITLSHIFT(), BITNOT(), BITOR(), BITRSHIFT(), BITSET(), BITTEST(), BITXOR()

### Description

The BITCLEAR() function clears the specified bit <expN2> in a numeric value <expN1> and returns the new value. If <expN1> and <expN2> are not integers, they will be converted to integer values before the clear takes place. The bit position, <expN2>, can range from 0 (rightmost bit) to 31.

### Example

```
x = 6          && 0110
y = 1
? bitclear(x,y)
   4    && 0100
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **BITLSHIFT()**

---

### **Class**

Bitwise Operations

### **Purpose**

Function to shift the bits in a numeric value a specified number of places to the left

### **Syntax**

BITLSHIFT(<expN1>,<expN2>)

### **See Also**

BITAND(), BITCLEAR(), BITNOT(), BITOR(), BITRSHIFT(), BITSET(), BITTEST(), BITXOR()

### **Description**

The BITLSHIFT() function shifts the bits in the numeric value <expN1> the specified number of places to the left <expN2> and returns the new value. If <expN1> and <expN2> are not integers, they will be converted to integer values before the shift takes place.

### **Example**

```
x = 6      && 0110
y = 1
? bitlshift(x,y)
    12      && 1100
```

### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BITNOT()

---

### Class

Bitwise Operations

### Purpose

Function to perform bitwise NOT operation

### Syntax

BITNOT(<expN>)

### See Also

BITAND(), BITCLEAR(), BITLSHIFT(), BITOR(), BITRSHIFT(), BITSET(), BITTEST(), BITXOR()

### Description

The BITNOT() function performs a bitwise NOT operation on the specified numeric parameter. If <expN> is not an integer, it will be converted to an integer value before the operation takes place.

BITNOT() switches each bit in turn of <expN1>. Each 1 becomes a 0, and each 0 becomes a 1.

<expN> bit	Result bit
0	1
1	0

### Example

```
x = 3      && 0...0011
? bitnot(x)
-4    && 1...1100
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BITOR()

---

### Class

Bitwise Operations

### Purpose

Function to perform bitwise OR operation

### Syntax

BITOR(<expN1>,<expN2>[,...<expN26>])

### See Also

BITAND(), BITCLEAR(), BITLSHIFT(), BITNOT(), BITRSHIFT(), BITSET(), BITTEST(), BITXOR()

### Description

The BITOR() function performs a bitwise OR operation on the specified numeric parameters. Up to 26 parameters can be specified. These parameters, if not integers, will be converted to integer values before the operation takes place.

BITOR() compares each bit in turn of <expN1> and <expN2>. If either bit is 1, the corresponding bit in the result is set to 1: if both bits are 0, the result bit is 0. If <expN3> is specified, the initial result is compared bit by bit with <expN3> and a new result evaluated. This new result is then compared with <expN4>, if specified, and so on.

<expN1> bit	<expN2> bit	Result bit
0	0	0
0	1	1
1	1	1
1	0	1

### Example

```
x = 3      && 0011
y = 6      && 0110
z = 7      && 0111
? bitor(x,y)
    7      && 0111
? bitor(x,y,z)
    7      && 0111
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BITRSHIFT()

---

### Class

Bitwise Operations

### Purpose

Function to shift the bits in a numeric value a specified number of places to the right

### Syntax

BITRSHIFT(<expN1>,<expN2>)

### See Also

BITAND(), BITCLEAR(), BITLSHIFT(), BITNOT(), BITOR(), BITSET(), BITTEST(), BITXOR()

### Description

The BITRSHIFT() function shifts the bits in the numeric value <expN1> the specified number of places to the right <expN2> and returns the new value. If <expN1> and <expN2> are not integers, they will be converted to integer values before the shift takes place.

### Example

```
x = 6      && 0110
y = 1
? bitrshift(x,y)
      3      && 0011
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BITSET()

---

### Class

Bitwise Operations

### Purpose

Function to set a specified bit in a numeric value

### Syntax

BITSET(<expN1>,<expN2>)

### See Also

BITAND(), BITCLEAR(), BITLSHIFT(), BITNOT(), BITOR(), BITRSHIFT(), BITTEST(), BITXOR()

### Description

The BITSET() function sets the specified bit <expN2> in a numeric value <expN1> to 1 and returns the new value. If <expN1> and <expN2> are not integers, they will be converted to integer values before the clear takes place. The bit position, <expN2>, can range from 0 (rightmost bit) to 31.

### Example

```
x = 6      && 0110
y = 3
? bitset (x,y)
    14      && 1110
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## **BITTEST()**

---

**Class**

Bitwise Operations

**Purpose**

Function to test the value of a specified bit in a numeric value

**Syntax**

BITTEST(&lt;expN1&gt;,&lt;expN2&gt;)

**See Also**

BITAND(), BITCLEAR(), BITLSHIFT(), BITNOT(), BITOR(), BITRSHIFT(), BITSET(), BITXOR()

**Description**

The BITTEST() function tests the value of the specified bit <expN2> in a numeric value <expN1>. If the bit is 1, BITTEST() returns True (.T.), otherwise it returns False (.F.). If <expN1> and <expN2> are not integers, they will be converted to integer values before the clear takes place. The bit position, <expN2>, can range from 0 (rightmost bit) to 31.

**Example**

```
x = 6      && 0110
```

```
y = 3
```

```
z = 2
```

```
? bittest (x,y)
```

```
.F.
```

```
? bittest (x,z)
```

```
.T.
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## BITXOR()

---

### Class

Bitwise Operations

### Purpose

Function to perform bitwise XOR (exclusive OR) operation

### Syntax

BITXOR(<expN1>,<expN2>[,...<expN26>])

### See Also

BITAND(), BITCLEAR(), BITLSHIFT(), BITNOT(), BITRSHIFT(), BITSET(), BITTEST(), BITXOR()

### Description

The BITXOR() function performs a bitwise XOR (exclusive OR) operation on the specified numeric parameters. Up to 26 parameters can be specified. These parameters, if not integers, will be converted to integer values before the operation takes place.

BITXOR() compares each bit in turn of <expN1> and <expN2>. If only one of the bits is 1, the corresponding bit in the result is set to 1, otherwise the result bit is 0. If <expN3> is specified, the initial result is compared bit by bit with <expN3> and a new result evaluated. This new result is then compared with <expN4>, if specified, and so on.

<expN1> bit	<expN2> bit	Result bit
0	0	0
0	1	1
1	1	0
1	0	1

### Example

```
x = 3      && 0011
y = 6      && 0110
z = 7      && 0111
? bitxor(x,y)
    5      && 0101
? bitxor(x,y,z)
    2      && 0010
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## CAST()

---

### Class

Expressions and Type Conversion

### Purpose

Function to convert the data type of an expression

### Syntax

CAST(<exp> AS <expC>[(<expN1>[, <expN2>]]) [NULL | NOT NULL])

### See Also

DTOC(), DTOS(), ETOS(), LTOS(), MTOS(), STR()

### Description

The CAST() function converts the expression in <exp> to the data type specified in <expC> and returns the result. The <expC> can be the full data type name, e.g. Character or supported abbreviation, e.g. C or Char. For data types requiring width and precision information, these are specified in <expN1> and <expN2> respectively.

Data Type Abbreviations:

Abbreviations	Data Type	Width Required	Precision Required
B	TINYINT/DOUBLE	No	No
C, Char	CHARACTER	Yes	No
D	DATE	No	No
F	FLOAT	Yes	No, defaults to 0
G	LONG VARBINARY/GENERAL	No	No
I, Int	INTEGER	Yes	No, defaults to 0
L	LOGICAL/BIT	No	No
M	LONG VARCHAR/MEMO	No	No
N, Num	NUMERIC	Yes	No, defaults to 0
P	PACKED DECIMAL	Yes	No, defaults to 0
Q	QUAD	Yes	No, defaults to 0
R	REAL	Yes	No, defaults to 0
S	SHORT	Yes	No, defaults to 0
T	DATETIME	No	No
V	VAXDATE	No	No
Y	CURRENCY	No	No
Z	ZONED	Yes	No, defaults to 0

If the specified width, <expN1> is less than that of <exp>, the result will be truncated. If greater, the result will be padded. Precision may be lost if the precision specified in <expN2> is less than that of <exp>.

The optional NULL | NOT NULL determines whether null values are permitted or not.

**Example**

```
> m_var = "date"  
> ? cast("12/12/2005" as (m_var))  
12/12/2005
```

```
> open database southwind  
> set sql on  
Recital/SQL> select cast(limit-balance as numeric(10,2)) from example;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CDOW()

---

### Class

Date and Time Data

### Purpose

Function to return the character day of week

### Syntax

CDOW(<expD> | <expT>)

### See Also

AMPM(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EPOCH(), GOMONTH(), HOUR(), HOURS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

CDOW() returns the name of the day of the week from the specified date expression <expD> or datetime expression <expT> as a character string.

### Example

store cdown(date()) to dayofweek

? dayofweek

#### Sunday

dayofweek = cdown(date())

? dayofweek

#### Sunday

dayofweek = cdown(datetime())

? dayofweek

#### Sunday

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CLEARRESULTSET()

---

### Class

Data Connectivity

### Purpose

Function to clear the marker from an SQL cursor previously marked as a resultset

### Syntax

CLEARRESULTSET()

### See Also

GETRESULTSET(), SETRESULTSET(), SQL SELECT

### Description

The CLEARRESULTSET() function clears the marker from an SQL cursor previously marked as a resultset by the SETRESULTSET() function. The SETRESULTSET() function is particularly used in returning a resultset from a stored procedure in SQL client/server applications.

CLEARRESULTSET() will return the workarea number of the SQL cursor from which the marker was cleared, or 0 (zero) if no SQL cursor was marked as a resultset.

### Example

```
function GetExampleCursor
lparameters lcAccountNo
select * from example where account_no = lcAccountNo into cursor curExample
return setresultset("curExample")

open database southwind
GetExampleCursor("00050")
select * from curexample
? "Cleared resultset marker in work area #" + ltrim(str(clearresultset()))
? iif(getresultset() > 0, "Resultset available in work area #" + ltrim(str(getresultset())));
? "No resultsets available"
?
close databases
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CMONTH()

---

### Class

Date and Time Data

### Purpose

Function to return the character month from a specified date or datetime

### Syntax

CMONTH(<expD> | <expT>)

### See Also

AMPM(), CDOW(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EPOCH(), GOMONTH(), HOUR(), HOURS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

CMONTH() is the character month function. It returns the name of the month from the specified date <expD> or the specified datetime <expT> as a character string.

### Example

```
? cmonth(datetime())
```

#### October

```
? cmonth(date())
```

#### October

```
store cmonth(date()) to month
```

```
? month
```

#### October

```
? type("month")
```

C

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CONNECTED()

---

**Class**

Data Connectivity

**Purpose**

Determine whether there is a gateway connection active in a workarea

**Syntax**

CONNECTED([<workarea | alias>])

**See Also**

GATEWAY(), SET GATEWAY

**Description**

The CONNECTED() function returns .T. if the workarea specified by <workarea | alias> is connected to a Recital Database Gateway. If there is no gateway connection in the specified workarea then .F. is returned. If no <workarea | alias> is specified then the CONNECTED() function defaults to the current workarea.

**Example**

set gateway to "ora@sales:scott/tiger.tcpip"

? connected()

.T.

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CTOT()

---

### Class

Expressions and Type Conversion

### Purpose

Function to convert character to datetime

### Syntax

CTOT(<expC>)

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The CTOT() function is the character to datetime conversion function. It converts the character expression specified to a datetime variable. The character expression must be correctly formatted based on the current SET DATE, SET MARK, SET SECONDS and SET CENTURY settings. For example, the default settings, SET DATE AMERICAN, SET SECONDS ON and SET CENTURY ON, require a datetime in the format “MM/DD/YYYY HH:MM:SS AM/PM”. If <expC> is an invalid datetime format the CTOT() function will return an empty datetime.

### Example

```
mdate = ctot("01/21/2004 01:44:44 PM")
```

```
? mdate
```

```
01/21/2004 01:44:44 PM
```

```
? type("mdate")
```

```
T
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## CURSORNAME()

---

### Class

Data Connectivity

### Purpose

Returns the name of the cursor open in a workarea where that workarea has an active gateway

### Syntax

CURSORNAME([<workarea | alias>])

### See Also

CLOSE, DECLARE CURSOR, FINDCURSOR(), DROP CURSOR, FETCH, OPEN, GATEWAY()

### Description

The CURSORNAME() function returns the cursor name corresponding to the workarea specified by <workarea | alias>. If there is no cursor open, or no gateway active in the specified workarea then an empty string is returned. If no <workarea | alias> is specified then the CURSORNAME() function defaults to the current workarea.

### Example

```
set gateway to "ora@sales:scott/tiger"
```

```
EXEC SQL
```

```
    DECLARE employees CURSOR FOR
        SELECT empno, ename, job
        FROM emp
        ORDER BY deptno;
```

```
EXEC SQL
```

```
    OPEN employees;
```

```
? cursorname()
```

```
EMPLOYEES
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## DATABASE()

---

**Class**

Databases

**Purpose**

Function to return the name of the currently open database

**Syntax**

DATABASE()

**See Also**

CLOSE DATABASES, DISPLAY STATUS, LIST STATUS, OPEN DATABASE, USE, ADATABASES(), ALIAS(), DBF(), DBUSED(), USED(), SET FILECASE, SET SQL

**Description**

The DATABASE() function returns the name of the currently open database or a null string if none is open.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Databases can be opened using the SQL USE command, with SQL set to MYSQL, or using the SQL OPEN DATABASE command.

**Example**

```
VFP/SQL> OPEN DATABASE hr
```

```
VFP/SQL> ? database()
```

```
hr
```

```
VFP/SQL> CLOSE DATABASES
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DATE()

---

### Class

Date And Time Data

### Purpose

Function to return current system date or a date from specified year, month and day values

### Syntax

DATE([<expN1>, <expN2>, <expN3>])

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The DATE() function returns the current system date as a date type. The display format of dates is affected by the SET CENTURY, SET DATE, SET EPOCH and SET MARK set commands.

The optional <expN1>,<expN2>,<expN3> can be used to specify numeric years, months and days and return a valid corresponding date. If any parameter is invalid, an empty date is returned.

<expN1>	A valid number of years, -100 (1900) to 900 (2900)
<expN2>	A valid number of months, 1 to 12
<expN3>	A valid number of days, 1 to 31

### Example

```
? date()
02/02/2000
mdate = date()
? mdate
02/02/2000
? type("mdate")
D
? date(4,4,4)
04/04/2004
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DATETIME()

---

### Class

Date And Time Data

### Purpose

Function to return current system date and time or a datetime from specified date and time values

### Syntax

DATETIME([<expN1>, <expN2>, <expN3> [, <expN4> [, <expN5> [, <expN6>]]]])

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TROC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The DATETIME() function returns the current system date and time as a datetime type. The display format of the date part of datetimes is affected by the SET CENTURY, SET DATE, SET EPOCH and SET MARK set commands. The SET SECONDS set command determines whether seconds are displayed in the time part of datetimes. The SET HOURS set command determines whether hours are shown in 24 hour format or in 12 hour format with AM | PM postfix.

The optional parameters can be used to specify numeric years, months, days, hours, minutes and seconds and return a valid corresponding datetime. If any date parameter is invalid, an empty datetime is returned. If any time parameter is invalid, an out of range error is generated.

<expN1>	A valid number of years, -100 (1900) to 900 (2900)
<expN2>	A valid number of months, 1 to 12
<expN3>	A valid number of days, 1 to 31
<expN4>	A valid number of hours, 0 to 23
<expN5>	A valid number of minutes, 0 to 59
<expN6>	A valid number of seconds, 0 to 59

### Example

? datetime()

**04/04/2004 11:54:45 AM**

mdatetime = datetime()

? mdatetime

**04/04/2004 11:54:45 AM**

? type("mdatetime")

**T**

? datetime(4,4,4,4,4,4)

**04/04/2004 04:04:04 AM**

? datetime(4,4,4,14)

**04/04/2004 02:00:00 PM**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DAY()

---

### Class

Date and Time Data

### Purpose

Function to return the day of the month from a specified date or datetime

### Syntax

DAY(<expD> | <expT>)

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EPOCH(), GOMONTH(), HOUR(), HOURS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The DAY() function returns the day of the month from the specified date expression <expD> or datetime expression <expT> as a numeric value.

### Example

```
? day({01/01/2004})
1
store day({01/01/2004}) to m_Day
? m_Day
1
m_Day = day(date())
? type("m_Day")
N

? day({10/10/2004 10:15:43 AM})
10
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DBUSED()

---

### Class

SQL Applications

### Purpose

Function to check whether a database is open.

### Syntax

DBUSED(<expC>)

### See Also

CLOSE DATABASES, DISPLAY STATUS, LIST STATUS, OPEN DATABASE, USE, ADATABASES(), ADIR(), ALIAS(), DBF(), GETENV(), USED(), SET SQL

### Description

The DBUSED() function is used to check whether the database whose name is specified in <expC> is open. If the database is open, DBUSED() returns True (.T.), if not it returns False (.F.).

NOTE: The DBUSED() function operates on databases, not tables.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Databases can be opened using the SQL USE command, with SQL set to MYSQL, or using the SQL OPEN DATABASE command.

### Example

```
Recital/SQL> set sql to vfp
Recital/SQL> OPEN DATABASE hr EXCLUSIVE
Recital/SQL> ? dbused("hr")
.T.
Recital/SQL> CLOSE DATABASES
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DMY()

---

### Class

Date and Time Data

### Purpose

Function to return a date or datetime as a character string

### Syntax

DMY(<expD> | <expT>)

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The DMY() function returns the specified date expression <expD> or date part of the datetime expression <expT> as a character string in the format: day, month name and year. If CENTURY is OFF, then a 2-digit year is returned.

### Example

```
? dmy({04/04/2005})
```

**4 April 2005**

```
? dmy({04/04/2005 06:12:45 PM})
```

**4 April 2005**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DOW()

---

### Class

Date and Time Data

### Purpose

Function to return numeric day of the week from a specified date

### Syntax

DOW(<expD> | <expT> [,<expN>])

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EPOCH(), GOMONTH(), HOUR(), HOURS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The DOW() function returns a number representing the day of the week from the given date expression <expD> or datetime expression <expT>.

Return Value	Day of Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

The optional <expN> is used to specify an alternative first day of the week.

### Example

```
set date american
```

```
? dow({02/29/2004})
```

```
1
```

```
? dow({02/29/2004},2)
```

```
7
```

```
store dow({02/29/2004}) to dayofweek
```

```
? dayofweek
```

```
1
```

```
dayofweek = dow({02/29/2004 11:35:27 A.M.})
```

```
? dayofweek
```

```
1
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## DTOC()

---

### Class

Expressions and Type Conversion

### Purpose

Function to perform date to character conversion

### Syntax

DTOC(<expD> | <expT> [,1])

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The DTOC() function converts the date expression <expD> or date part of the datetime expression <expT> to a character string in the format of the current SET DATE, SET MARK and SET CENTURY settings. For example, the default settings, SET DATE AMERICAN, and SET CENTURY ON, will return a date in the format “MM/DD/YYYY”.

If the optional 1 is specified, the date will be returned in DTOS() format, suitable for index key purposes.

### Example

```
? dtoc({02/29/2004})
02/29/2004
? dtoc({02/29/2004 10:34:21 A.M.})
02/29/2004
? dtoc({02/29/2004},1)
20040229
store dtoc({02/29/2004}) to m_date
? m_date
02/29/2004
? type("m_date")
C
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## DTOS()

---

### Class

Expressions and Type Conversion

### Purpose

Function to perform date to string conversion

### Syntax

DTOS(<expD> | <expT>)

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The DTOS() function converts the date expression <expD> or date part of the datetime expression <expT> to a character string in the format “YYYYMMDD”. This function is particularly useful when creating indexes with mixed data types. The SET DATE, SET MARK and SET CENTURY settings have no effect on this function.

### Example

```
? dtos({02/29/2004})
```

```
20040229
```

```
? dtos({02/29/2004 11:18:54 PM})
```

```
20040229
```

```
store dtos({02/29/2004})to m_date
```

```
? m_date
```

```
20040229
```

```
? type("m_date")
```

```
C
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## EMPTY()

---

### Class

Expressions and Type Conversion

### Purpose

Function to check for empty value

### Syntax

EMPTY(<expC> | <expD> | <expT> | <expL> | <memofield> | <expN>)

### See Also

CTOD(), CTOT(), IIF(), SPACE(), TYPE(), SET CENTURY, SET DATE, SET MARK, SET SECONDS

### Description

The EMPTY() function returns .T. if the specified expression is 'empty'.

Data Type	Empty
Character	Space(0), "", [], '', no text entered
Date	CTOD(""), {}, {00/00/0000}, { / / } and other SET CENTURY, SET MARK and SET DATE variations
Datetime	CTOT(""), { / / : : AM} and other SET CENTURY, SET MARK, SET SECONDS and SET DATE variations
Logical	.F.
Memo	No data entered
Numeric	0

### Example

```
if empty(name + address)
    dialog box "Name and address not specified."
endif
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## FINDCURSOR()

---

**Class**

Data Connectivity

**Purpose**

Returns the workarea number corresponding to the specified gateway cursor

**Syntax**

FINDCURSOR(<expC>)

**See Also**

CLOSE, DECLARE CURSOR, CURSORNAME(), DROP CURSOR, FETCH, OPEN, GATEWAY()

**Description**

The FINDCURSOR() function returns a workarea number corresponding to the open gateway cursor specified by <expC>. If the cursor name specified is not open, or no gateway is active, then a value of -1 is returned.

**Example**

```
select 3
set gateway to "ora@sales:scott/tiger"
EXEC SQL
    DECLARE employees CURSOR FOR
    SELECT empno, ename, job
    FROM emp
    ORDER BY deptno;
EXEC SQL
    OPEN employees;
```

```
? findcursor(cursorname())
```

**3**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## GATEWAY()

---

### Class

Data Connectivity

### Purpose

Returns information about a gateway connection

### Syntax

GATEWAY([<expN>])

### See Also

CONNECTED(), SET GATEWAY

### Description

The GATEWAY() function returns specified information about the connection to the Recital Database Gateway in the current workarea. The following table lists the optional information that may be requested.

Value	Description
	Returns the server type, Recital, Oracle, ODBC, Informix, DB2, etc
0	Returns the server type, Recital, Oracle, ODBC, Informix, DB2, etc
1	Returns the node name that the gateway is connected to.
2	Returns the user name that was used to connect to the gateway.
3	Returns the password associated with the user name.
4	Returns the database connected to on the server.

### Example

```
set gateway to ora@sales:scott/tiger.tcpip
```

```
? gateway()
```

**ORA**

```
? gateway(1)
```

**sales**

```
? gateway(2)
```

**scott**

```
? gateway(3)
```

**tiger**

```
? gateway(4)
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## GETRESULTSET()

---

### Class

Data Connectivity

### Purpose

Function to return the workarea number of an SQL cursor previously marked as a resultset

### Syntax

GETRESULTSET()

### See Also

CLEARRESULTSET(), SETRESULTSET(), SQL SELECT

### Description

The GETRESULTSET() function returns the workarea number of an SQL cursor previously marked as a resultset by the SETRESULTSET() function. The SETRESULTSET() function is particularly used in returning a resultset from a stored procedure in SQL client/server applications.

GETRESULTSET() will return 0 (zero) if no SQL cursor is currently marked as a resultset. The marker can be cleared from an SQL cursor using the CLEARRESULTSET() function.

### Example

```
function GetExampleCursor
```

```
lparameters lcAccountNo
```

```
select * from example where account_no = lcAccountNo into cursor curExample
```

```
return setresultset("curExample")
```

```
open database southwind
```

```
GetExampleCursor("00050")
```

```
? "Returned resultset is in work area #" + ltrim(str(getresultset()))
```

```
set sql off
```

```
select getresultset()
```

```
display all
```

```
? "Cleared resultset marker in work area #" + ltrim(str(clearresultset()))
```

```
? iif(getresultset() > 0, "Resultset available in work area #" + ltrim(str(getresultset()));
```

```
    "No resultsets available")
```

```
?
```

```
close databases
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## GOMONTH()

---

### Class

Date and Time Data

### Purpose

Function to return a date that is a specified number of months before or after a particular date or datetime

### Syntax

GOMONTH(<expD> | <expT>, <expN>)

### See Also

CDOW(), CMONTH(), CTOD(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), EPOCH(), MDY(), MONTH(), MTOD(), QUARTER(), STOD(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK

### Description

The GOMONTH() function returns a date which is <expN> months before or after the date expression <expD> or datetime expression <expT>. If <expN> is a negative number, the GOMONTH() function returns a date that is before <expD> or <expT>. If <expN> is a positive number, the GOMONTH() function returns a date that is after <expD> or <expT>.

### Example

```
?gomonth({04/14/2004}, 4)
```

**08/14/2004**

```
? gomonth({01/21/2004 03:18:33 PM}, -12)
```

**01/21/2003**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## HOUR()

---

### Class

Date and Time Data

### Purpose

Function to return the numeric hours from a specified datetime

### Syntax

Hour(<expT>)

### See Also

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOURS(), MINUTE(), MINUTES(), SEC(), SECONDS(), SECS(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

### Description

The HOUR() function returns the hours from the specified datetime expression <expT> as a numeric value.

### Example

```
? hour({ 10/10/2004 10:15:43 AM})
```

**10**

```
m_Hour = hour(datetime())
```

```
? type("m_Hour")
```

**N**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## HOURS()

---

**Class**

Date and Time Data

**Purpose**

Function to extract number of hours from a time string

**Syntax**

HOURS(<time-string>)

**See Also**

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOUR(), MINUTE(), MINUTES(), SEC(), SECONDS(), SECS(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

**Description**

The HOURS() function extracts the hours from the specified time-string and returns the number of hours as a numeric value.

**Example**

```
? hours("10:00:00")
```

**10**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## ICASE()

---

**Class**

Applications

**Purpose**

Function to execute an immediate case statement

**Syntax**

ICASE(<expL1>, <exp1>[, <expL2>, <exp2>] [...] [, <exp-otherwise>)

**See Also**

DO CASE, IF, IF()

**Description**

The ICASE() function operates as an immediate DO CASE statement. It evaluates the specified conditions <expL1> to <expLn> and returns the matching expression <exp1> to <expn> for the first condition that evaluates to True (.T.). If none of the conditions evaluates to True, the 'otherwise' expression, <exp-otherwise>, is returned. If there is no otherwise expression specified, and none of the conditions evaluates to True, ICASE() returns a null (.NULL.).

**Example**

```
accept "Enter a command: " to command
&(icase(upper(command) = "BROWSE", "browse", upper(command) = "DIR", "dir",;
  "Set message to [Unknown command.]"))
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## IF()

---

**Class**

Applications

**Purpose**

Function to execute an immediate if

**Syntax**

IF(<expL>, <exp1>, <exp2>)

**See Also**

ACC(), CALC(), ICASE(), IIF(), IF

**Description**

The IF() function returns the value of the expression <exp1> or the value of the expression <exp2>, depending on the result of the expression <expL>. If the expression <expL> returns .T., the IF() function returns the value of <exp1>. If the expression <expL> returns .F., the IF() function returns the value of <exp2>. The expressions <exp1> and <exp2> may themselves contain IF() functions. The IF() function may be used anywhere that a normal expression can be used (e.g. in SQL SELECT column expressions).

The IF() function is synonymous with the IIF() function.

**Example**

event = "drama"

? if(event = "drama", "It's for Drama", "It's not for Drama")

**It's for Drama**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## IIF()

---

**Class**

Applications

**Purpose**

Function to execute an immediate if

**Syntax**

IIF(<expL>, <exp1>, <exp2>)

**See Also**

ACC(), CALC(), ICASE(), IF(), IF

**Description**

The IIF() function returns the value of the expression <exp1> or the value of the expression <exp2>, depending on the result of the expression <expL>. If the expression <expL> returns .T., the IIF() function returns the value of <exp1>. If the expression <expL> returns .F., the IIF() function returns the value of <exp2>. The expressions <exp1> and <exp2> may themselves contain IIF() functions. The IIF() function may be used anywhere that a normal expression can be used (e.g. as part of expressions in REPORT, DICTIONARY and LABEL formats).

**Example**

```
event = "drama"
```

```
? iif(event = "drama", "It's for Drama", "It's not for Drama")
```

**It's for Drama**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## ISNULL()

---

**Class**

Expressions and Type Conversion

**Purpose**

Function to test if an expression evaluates to NULL

**Syntax**

ISNULL(<expr>)

**See Also**

EMPTY(), ISALPHA(), ISBLANK(), ISDIGIT()

**Description**

The ISNULL() function will return True (.T.) if the specified expression, <expr> is NULL otherwise False (.F.).

**Example**

```
select account_no, isnull(last_name) from customer;
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## LIKE()

---

### Class

String Data

### Purpose

Function to compare two strings for similarities

### Syntax

LIKE(<expC1>, <expC2>)

### See Also

LIKE PREDICATE

### Description

The LIKE() function is used to compare two character strings for matching qualities. The <expC1> specifies the matching pattern to look for. The <expC2> specifies the string in which to look. The <expC1> accepts “wildcard” characters with which to construct the pattern. Wildcard characters may represent upper case or lower case characters, but specific characters are case sensitive. The Recital/4GL supports the following wildcard characters:

Characters	Description
?	Matches any one character
%	Synonymous with ?
*	Matches zero or more characters

If the LIKE() function is used in SQL mode (SET SQL ON or embedded EXEC SQL statements), the following wildcard characters are available:

Characters	Description
_	Matches any one character
%	Matches zero or more characters

The LIKE() function returns .T. if a matching pattern is found, or .F. otherwise.

### Example

```
?like(".*Recital*", "This is Recital")
```

.T.

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## MINUTE()

---

### Class

Date and Time Data

### Purpose

Function to return the numeric minutes from a specified datetime

### Syntax

MINUTE(<expT>)

### See Also

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOUR(), HOURS(), MINUTES(), SEC(), SECONDS(), SECS(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

### Description

The MINUTE() function returns the minutes from the specified datetime expression <expT> as a numeric value.

### Example

```
? minute({ 10/10/2004 10:15:43 AM})
```

**15**

```
m_Min = minute(datetime())
```

```
? type("m_Min")
```

**N**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## MINUTES()

---

**Class**

Date and Time Data

**Purpose**

Function to extract minutes from a specified time string

**Syntax**

MINUTES(<time-string>)

**See Also**

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOUR(), HOURS(), MINUTE(), SEC(), SECONDS(), SECS(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

**Description**

The MINUTES() function extracts the number of minutes from a specified time string and returns it as a numeric value.

**Example**

? minutes("10:21:00")

21

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## MONTH()

---

**Class**

Date and Time Data

**Purpose**

Function to return month from a specified date or datetime

**Syntax**

MONTH(<expD> | <expT>)

**See Also**

CDOW(), CMONTH(), CTOD(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), EPOCH(), GOMONTH(), MDY(), MTOD(), QUARTER(), STOD(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK

**Description**

The MONTH() function returns the numeric month of the year from the given date expression <expD> or datetime expression <expT>.

**Example**

? month(date())

**10**

? month(datetime())

**10**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## NVL()

---

**Class**

Expressions and Type Conversion

**Purpose**

Function to specify an alternative value for a null expression

**Syntax**

NVL(<expr1>,<expr2>)

**See Also**

ETOS(), ISNULL(), SET NULL

**Description**

The NVL() function evaluates the expression in <expr1>, and if the expression does not evaluate to NULL, the evaluated result is returned. If the expression in <expr1> does evaluate to NULL, the expression in <expr2> is evaluated. If <expr2> does not evaluate to NULL, the evaluated result is returned. If both <expr1> and <expr2> evaluate to NULL, the NVL() function returns NULL.

**Example**

```
set sql to vfp
set null on
CREATE TABLE nullon (firstname c(20), lastname c(20))
INSERT INTO nullon (lastname) VALUES ("Smith")
SELECT lastname, nvl(firstname,"Unknown") from nullon
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## QUARTER()

---

### Class

Date and Time Data

### Purpose

Function to return year quarter for the specified date or datetime

### Syntax

QUARTER(<expD> | <expT>[, <expN>])

### See Also

CDOW(), CMONTH(), CTOD(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), EPOCH(), GOMONTH(), MDY(), MONTH(), MTOD(), STOD(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK

### Description

The QUARTER() function returns the year quarter for the specified date expression <expD> or datetime expression <expT>. The optional <expN> is used to specify the number of an alternative starting month for the year; the default is 1 (January).

### Example

```
? quarter({01/22/2004})
```

**1**

```
? quarter({^20040822 12:34:29 PM})
```

**3**

```
? quarter({01/22/2004},2)
```

**4**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## REFERENCES()

---

### Class

Fields and Records

### Purpose

Function to perform a cross-table lookup for a specified key expression

### Syntax

REFERENCES(<key expression>, <workarea | alias> [,<tag name>])

### See Also

SET RELATION, LOOKUP(), RLOOKUP(), SEEK()

### Description

The REFERENCES() function looks up the specified <key expression> in the master tag index of the specified <workarea | alias>. The <workarea | alias> is the workarea or alias name of an open table. To search in a tag index which is not the current master index, the optional <tag name> parameter can be used. The tag name must be specified as a string.

The REFERENCES() function returns True (.T.) or False (.F.), depending on the success of the lookup operation.

Please see the RLOOKUP() function for cross-table lookups using single indexes (.ndx).

### Example

```
use customer.rdb
index on account_no tag account_no
index on upper(last_name) tag uplast
index on zip tag zip
? references("STEREK",customer,"uplast")
.T.
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SEC()

---

### Class

Date and Time Data

### Purpose

Function to return the numeric seconds from a specified datetime

### Syntax

SEC(<expT>)

### See Also

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOUR(), HOURS(), MINUTE(), MINUTES(), SECONDS(), SECS(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

### Description

The SEC() function returns the seconds from the specified datetime expression <expT> as a numeric value.

### Example

```
? sec({ 10/10/2004 10:15:43 AM})
```

**43**

```
m_Sec = sec(datetime())
```

```
? type("m_Sec")
```

**N**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SECONDS()

---

**Class**

Date And Time Data

**Purpose**

Function to extract seconds from a time string

**Syntax**

SECONDS([<time-string>])

**See Also**

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOUR(), HOURS(), MINUTE(), MINUTES(), SEC(), SECS(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLIPPER, SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

**Description**

The SECONDS() function returns the seconds from the current time as a number. The SECONDS() function will also return the seconds from an optionally specified <time-string>. If the command SET CLIPPER is ON, the SECONDS() function operates in the same way as the SECS() function.

**Example**

? seconds("10:33:21")

21

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SECS()

---

### Class

Date And Time Data

### Purpose

Function to return the number of seconds since midnight

### Syntax

SECS([<time-string>])

### See Also

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOUR(), HOURS(), MINUTE(), MINUTES(), SEC(), SECONDS(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLIPPER, SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

### Description

The SECS() function returns the number of seconds since midnight. The optional <time-string>, in the format HH:MM:SS, may be used to specify a time. If no <time-string> is passed, the SECS() function uses the current time. This function can be used to store time as seconds in a numeric field. The TSTRING() function is used to convert the seconds back to a time-string. If SET CLIPPER is ON, the SECS() function behaves like the SECONDS() function.

### Example

```
? secs("10:10:10")
```

**36610**

```
// Another Example
```

```
use accounts
```

```
replace seconds with secs(time())
```

```
? seconds
```

**39306**

```
? tstring(seconds)
```

**10:55:00**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SEQNO()

---

**Class**

Table Basics

**Purpose**

Function to return a unique sequence number

**Syntax**

SEQNO([<workarea | alias>])

**See Also**

ALTER TABLE, CREATE TABLE, SET SEQNO

**Description**

The SEQNO() function returns the next unique sequence number for the current table. Automatic locking is performed during the operation of this function if the specified table is opened shareable. The optional <workarea | alias> will return the next unique sequence number from the specified table. If there is no active table the SEQNO() function will return 0.

The sequence number of a table can be reset with the command SET SEQNO TO <expN>.

**Example**

append blank

replace custno with seqno()

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SETRESULTSET()

---

### Class

Data Connectivity

### Purpose

Function to mark an SQL cursor as a resultset

### Syntax

SETRESULTSET(<expN>|<expC>)

### See Also

CLEARRESULTSET(), GETRESULTSET(), SQL SELECT

### Description

The SETRESULTSET() function marks an SQL cursor as a resultset. Any previous SQL cursor marker is cleared. The workarea number or alias name of the cursor should be specified in <expN> or <expC> respectively. The SETRESULTSET() function is particularly used in returning a resultset from a stored procedure in SQL client/server applications.

The GETRESULTSET() function can be used to return the workarea number of an SQL cursor marked as a resultset by SETRESULTSET(). The resultset marker can be cleared from an SQL cursor using the CLEARRESULTSET() function.

### Example

```
function GetExampleCursor
lparameters lcAccountNo
select * from example where account_no = lcAccountNo into cursor curExample
return setresultset("curExample")

open database southwind
GetExampleCursor("00050")
select * from curexample
? "Cleared resultset marker in work area #" + ltrim(str(clearresultset()))
? iif(getresultset() > 0, "Resultset available in work area #" + ltrim(str(getresultset())));
? "No resultsets available")
?
close databases
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## SQLVALUES()

---

**Class**

Data Connectivity

**Purpose**

Function to return the single row result of an SQL statement

**Syntax**

SQLVALUES(<SQL statement>)

**See Also**

SELECT

**Description**

The SQLVALUES() function executes the <SQL statement> and returns the result as a string. The required <SQL statement> cannot return multiple rows. If a SELECT statement is specified it must be a singleton select.

The SQLVALUES() function can only be used with an active gateway connection.

**Example**

```
login "Oracle","node","user","Password"  
// Count the number of records in the employee table  
cTotal = sqlvalues("SELECT COUNT(*) FROM emp")  
total=val(cTotal)  
// Count the number of records in the employee table matching the key "Mr"  
cTotal = sqlvalues("SELECT COUNT(*) FROM emp WHERE title = 'Mr'")
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## TIME()

---

**Class**

Date and Time Data

**Purpose**

Function to return the current system time

**Syntax**

TIME([<expN>])

**See Also**

AMPM(), CTOT(), DATE(), DATETIME(), ELAPTIME(), HOUR(), HOURS(), MINUTE(), MINUTES(), SEC(), SECONDS(), SECS(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), VALIDTIME(), SET CLOCK, SET CLOCKRATE, SET SECONDS, SET VAXTIME

**Description**

The TIME() function returns the current system time as a character string in the format HH:MM:SS. The TIME() function always returns the time in 24 hour format, and is not affected by the SET HOURS TO [12/24] command. The optional numeric expression <expN> must result in a non-zero value, and when specified, the current time including hundredths of seconds is returned. This is provided for Xbase language compatibility. The TIME() function will always return hundredths of seconds as 00.

**Example**

? time()

**17:47:24**

? time(1)

**17:47:31.00**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## TTOC()

---

### Class

Expressions and Type Conversion

### Purpose

Function to convert a datetime expression to a string value in an optionally specified format

### Syntax

TTOC(<expT> [, <expN>])

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The TTOC() function converts the datetime expression <expT> to a string value. By default, the date part of the string returned will conform to the current SET DATE, SET MARK and SET CENTURY settings, in the same format as DTOC(). The time will be returned in the format hh:mm:ss AM | PM. If the expression to be converted contains no time information, 12:00:00 AM will be assumed. If SET SECONDS is OFF (ON by default), no seconds will be displayed. The SET HOURS set command determines whether hours are shown in 24 hour format or in 12 hour format with AM | PM postfix.

<expN>

The optional <expN> can be used to specify the format of the return value:

<expN>	Format
0	As defaults
1	YYYYMMDDhhmmss
2	Time only: hh:mm:ss AM PM (SET SECONDS ON) or hh:mm AM PM (SET SECONDS OFF)

### Example

```
set date american
```

```
? ttoc({^2004-03-29 10:15:43 AM})
```

**03/29/2004 10:15:43 AM**

```
? ttoc({^2004-03-29 10:15:43 AM},1)
```

**20040329101543**

```
? ttoc({^2004-03-29 10:15:43 AM},2)
```

**10:15:43 AM**

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## TTOD()

---

### Class

Expressions and Type Conversion

### Purpose

Function to convert datetime to date

### Syntax

TTOD(<expT>)

### See Also

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK, SET SECONDS, SET VAXTIME

### Description

The TTOD() function is the datetime to date conversion function. It converts the <expT> datetime expression specified to a date. The <expT> must be a valid datetime, or the TTOD() function will return an empty date. The date returned will conform to the current SET DATE, SET MARK and SET CENTURY settings. For example, the default settings, SET DATE AMERICAN and SET CENTURY ON, will return a date in the format “MM/DD/YYYY”.

### Example

```
set date american
set century on
mdate = ttod({^2004-03-29 10:15:43 AM})
? mdate
03/29/2004
? type("mdate")
D
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## TXNISOLATION()

---

**Class**

Transaction Processing

**Purpose**

Function to return the current Transaction Isolation Level setting

**Syntax**

TXNISOLATION()

**See Also**

BEGIN...END TRANSACTION, COMMIT, ROLLBACK, SAVE TRANSACTION, SAVEPOINT, SET TRANSACTION, TXNLEVEL()

**Description**

The TXNISOLATION() function returns the current Transaction Isolation setting. The Transaction Isolation Level is set using the SET TRANSACTION [ISOLATION LEVEL <level>] command.

The Transaction Isolation Level can be any of the following, please see the SET TRANSACTION Set Command for full details:

- **SERIALIZABLE**
- **REPEATABLE READ**
- **READ COMMITTED**
- **READ UNCOMMITTED**

**Example**

```
set transaction isolation level read uncommitted;  
cTrans = txnisolation()
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## TXNLEVEL()

---

### Class

Transaction Processing

### Purpose

Function to return the current Transaction Level number

### Syntax

TXNLEVEL()

### See Also

BEGIN...END TRANSACTION, COMMIT, ROLLBACK, SAVE TRANSACTION, SAVEPOINT, SET TRANSACTION, TXNISOLATION()

### Description

The TXNLEVEL() function returns the current Transaction level as a number. Transactions can be nested by issuing a further BEGIN TRANSACTION when a transaction is already active. If no transaction is active, the TXNLEVEL() function returns 0. A transaction and any transactions nested within it are closed when a COMMIT, ROLLBACK or END TRANSACTION is issued.

### Example

```
// config.db
set sql to recital
set sql on
// end of config.db

// Nested Transactions
? txnlevel() && 0
BEGIN TRANSACTION trans1;
? txnlevel() && 1
INSERT INTO customer
  (TITLE, LAST_NAME, FIRST_NAME, INITIAL, STREET,
   CITY, STATE, ZIP,LIMIT, START_DATE)
  VALUES
  ('Ms', 'Jones', 'Susan', 'B', '177 High Street','Beverly', 'MA', '01915', 2000, date());
INSERT INTO accounts (ORD_VALUE) VALUES (30);
BEGIN TRANSACTION trans2;
? txnlevel() && 2
INSERT INTO accounts (ORD_VALUE) VALUES (60);
// Commit the trans1 transaction and any transactions
// nested in trans1
COMMIT TRANSACTION trans1;
? txnlevel() && 0
END TRANSACTION;
? txnlevel() && 0
// End of program
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## TYPE()

---

### Class

Expressions and Type Conversion

### Purpose

Function to return a letter code that represents a data type

### Syntax

TYPE(<expC>)

### See Also

ERROR(), ERRNO(), MESSAGE(), ON ERROR

### Description

The TYPE() function returns a letter code which represents the data type of the expression in <expC>. The return value of the letter code is a character string from the table below.

Data type	Return Value
ARRAY (STATIC)	A
ARRAY (DYNAMIC)	O
BIGINT	N
BIT	L
BYTE	N
CHARACTER	C
CURRENCY	Y
DATE	D
DATETIME	T
DECIMAL	N
DOUBLE	N
FLOAT	N
GENERAL	G
INTEGER	N
LOGICAL	L
LONG VARCHAR	M
LONG VARBINARY	G
MEDIUMINT	N
MEMO	M
NUMERIC	N
OBJECT	O
PACKED	N
QUAD	N
REAL	N
SHORT	N
SMALLINT	N
Syntax error	U
TEXT	M
TIME	C
TIMESTAMP	T
TINYINT	N
Undefined	U

VAXDATE	C
VARCHAR	C
ZONED NUMERIC	C

If <expC> contains a syntax error, or an undeclared variable, then TYPE() returns 'U'. TYPE() will also return a 'U' for an undefined variable if SET CLIPPER is ON. TYPE() is primarily used to check for the existence of a variable, or the syntax of an expression.

**Example**

```
i = 10
? type("i")
N
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer



## VARTYPE()

---

### Class

Expressions and Type Conversion

### Purpose

Function to return a letter code that represents a data type

### Syntax

VARTYPE(<exp>[,<expL>])

### See Also

ERROR(), ERRNO(), MESSAGE(), TYPE(), ON ERROR

### Description

The VARTYPE() function returns a letter code which represents the data type of the expression in <exp>. The return value of the letter code is a character string from the table below.

Data type	Return Value
ARRAY (STATIC)	A
ARRAY (DYNAMIC)	O
BIGINT	N
BIT	L
BYTE	N
CHARACTER	C
CURRENCY	Y
DATE	D
DATETIME	T
DECIMAL	N
DOUBLE	N
FLOAT	N
GENERAL	G
INTEGER	N
LOGICAL	L
LONG VARCHAR	M
LONG VARBINARY	G
MEDIUMINT	N
MEMO	M
NULL	X
NUMERIC	N
OBJECT	O
PACKED	N
QUAD	N
REAL	N
SHORT	N
SMALLINT	N
Syntax error	U
TEXT	M
TIME	C
TIMESTAMP	T
TINYINT	N

Undefined	U
VAXDATE	C
VARCHAR	C
ZONED NUMERIC	C

If <exp> contains a syntax error, or an undeclared variable, then VARTYPE() returns 'U'. VARTYPE() will also return a 'U' for an undefined variable if SET CLIPPER is ON. Unlike the TYPE() function, VARTYPE() does not require the expression for evaluation to be enclosed in quotes.

The optional <expL> is used to determine whether VARTYPE() returns the data type for expressions which evaluate to null (.NULL.). If <expL> is True (.T.) the data type is returned for <exp>. If <expL> is False (.F.), then VARTYPE() returns "X".

#### **Example**

i = 10

? vartype(i)

N

#### **Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## YEAR()

---

**Class**

Date and Time Data

**Purpose**

Function to extract year from date or datetime

**Syntax**

YEAR(<expD> | <expT>)

**See Also**

CDOW(), CMONTH(), CTOD(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), EPOCH(), GOMONTH(), MDY(), MONTH(), MTOD(), QUARTER(), STOD(), VTOD(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK

**Description**

The YEAR() function returns the numeric year value from a date expression, <expD> or a datetime expression <expT>. If SET CENTURY is ON (default), the century will be displayed with normal date displays.

**Example**

? date()

**04/04/2004**

? year(date())

**2004**

? year(datetime())

**2004**

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET...

---

**Class**

SQL Applications

**Purpose**

Used to issue a SET COMMAND

**Syntax**

SET <SET COMMAND>

**See Also**

EXEC, SET EXCLUSIVE, SET DELETED,

**Description**

Any Recital/4GL SET COMMAND can be issued in an SQL application by prefixing it with the EXEC SQL statement prefix.

**Example**

```
exec sql
    set exclusive on
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET AUTOCATALOG

---

### Class

Databases

### Purpose

Enable files to be automatically added to a database catalog

### Syntax

```
SET AUTOCATALOG ON | OFF | (<expL>)
SET AUTOCATALOG TO <database>
```

### See Also

ADD TABLE, ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, REINDEX DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DBUSED(), DB\_MAXWKA

### Description

The SET AUTOCATALOG commands allow tables and their index files to be automatically added to a database catalog. The database itself should be closed when using the auto catalog commands. The SET AUTOCATALOG TO <database> command specifies the name of the database for which the catalog should be updated. SET AUTOCATALOG ON | OFF allows updates to the catalog to be toggled on and off. Once the auto catalog commands are active, tables and their indexes can be opened from the interactive prompt or from an application and the database catalog will automatically be updated. For additional details on the information stored in the database catalog, please see the OPEN DATABASE command.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB\_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Databases are opened using the SQL OPEN DATABASE command.

### Example

```
close databases
set autocatalog to southwind
set autocatalog on
do myapp
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET CENTURY

---

**Class**

Date and Time Data

**Purpose**

Enable century in dates

**Syntax**

SET CENTURY ON | OFF | (<expL>)

**See Also**

CDOW(), CMONTH(), CTOD(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), EPOCH(), GOMONTH(), MDY(), MONTH(), MTOD(), QUARTER(), STOD(), VTOD(), YEAR(), SET DATE, SET EPOCH, SET HOURS, SET MARK

**Description**

If SET CENTURY is ON, then dates are displayed, and can be input with the century prefix specified. If CENTURY is OFF then the year part of dates is only two digits, and the 20th century is assumed. By default, CENTURY is ON.

**Example**

```
set century on
use patrons index dates
list all for date = ctod("01/01/2003")
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET GATEWAY

---

**Class**

Data Connectivity

**Purpose**

Establishes a connection to a Recital Database Gateway

**Syntax**

SET GATEWAY TO [<expC1>] [IN <workarea/alias>] [ALIAS <expC2>]

**See Also**

CONNECTED(), GATEWAY()

**Description**

The SET GATEWAY command is used to establish a connection to a Recital Database Gateway via the Recital Database Server. Each workarea can have a separate gateway established.

<expC1> is a character string that must be formatted in the following way:  
servername@machinename:username/password-database.protocol

If <expC1> is not included with the SET GATEWAY command, the connection in that workarea will be detached.

An optional ALIAS <expC2> keyword can be used to specify an alias name for the workarea.

**Example**

```
set gateway to ora@sales:scott/tiger.tcpip
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET HOURS

---

**Class**

Recital Terminal Developer Environment

**Purpose**

Change time display to 12 or 24 hours

**Syntax**

SET HOURS TO [12 | 24]

**See Also**

AMPM(), CDOW(), CMONTH(), CTOD(), CTOT(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), ELAPTIME(), EMPTY(), EPOCH(), GOMONTH(), HOUR(), HOURS(), LTOS(), MDY(), MINUTE(), MINUTES(), MONTH(), MTOD(), MTOS(), QUARTER(), SEC(), SECONDS(), SECS(), STOD(), STR(), TIME(), TIMESTAMP(), TSTRING(), TTOC(), TTOD(), TYPE(), VAL(), VALIDTIME(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET MARK, SET SECONDS, SET VAXTIME

**Description**

The SET HOURS TO command changes the system clock to a 12 hour or a 24 hour display. If the optional qualifier 12 | 24 is not specified, then the clock is set to the default. The default setting is 12 hours. This command also affects the display of datetime values, determining whether hours are shown in 24 hour format or in 12 hour format with AM | PM postfix.

**Example**

```
set hours to 24
```

**Products**

Recital Terminal Developer



## SET NULL

---

### Class

SQL Applications

### Purpose

To determine NULL value support

### Syntax

SET NULL ON | OFF

### See Also

ALTER TABLE, CREATE TABLE, INSERT, EMPTY(), ISBLANK(), ISNULL(), SET NULLDISPLAY

### Description

The SET NULL ON | OFF command is used to determine whether columns in a table support NULL values. With SET NULL ON, table columns will support NULL values by default. INSERT will insert a NULL into any column that does not have a value specified. With SET NULL off, NULL values are not supported by default. INSERT will insert a NULL into any column that does not have a value specified.. This default can be overridden by specifying the NULL or NOT NULL column constraint on an individual column.

SET NULL is OFF by default.

### Example

```
set sql to vfp
set null on
CREATE TABLE nullon (firstname c(20), lastname c(20))
INSERT INTO nullon (lastname) VALUES ("Smith")
? [SET NULL ON]
? [ISNULL() ], isnull(firstname)
? [EMPTY() ], empty(firstname)
wait
SET NULL ON
ISNULL() .T.
EMPTY() .F.
Press any key to continue...
```

```
set null off
CREATE TABLE nulloff (firstname c(20), lastname c(20))
INSERT INTO nulloff (lastname) VALUES ("Smith")
? [SET NULL OFF]
? [ISNULL() ], isnull(firstname)
? [EMPTY() ], empty(firstname)
wait
SET NULL OFF
ISNULL() .F.
EMPTY() .T.
Press any key to continue...
```

```
set null off
CREATE TABLE nulloff2 (firstname c(20) NULL, lastname c(20))
INSERT INTO nulloff2 (firstname,lastname) VALUES (NULL,"Smith")
? [SET NULL OFF, NULL Column Constraint]
? [ISNULL() ], isnull(firstname)
? [EMPTY() ], empty(firstname)
wait
  SET NULL OFF, NULL Column Constraint
  ISNULL() .T.
  EMPTY() .F.
  Press any key to continue...
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET NULLDISPLAY

---

### Class

SQL Applications

### Purpose

To specify the text displayed for NULL values

### Syntax

SET NULLDISPLAY TO [<expC>]

### See Also

ALTER TABLE, CREATE TABLE, INSERT, EMPTY(), ISBLANK(), ISNULL(), SET NULL

### Description

The SET NULLDISPLAY command is used to specify the text displayed for NULL values. By default NULL values are displayed as .NULL.. The optional <expC> is used to specify alternative display text. If <expC> is omitted, then the display is reset to the default.

### Example

```
set sql to vfp
set null on
set heading off
CREATE TABLE nullon (firstname c(20), lastname c(20))
INSERT INTO nullon (lastname) VALUES ("Smith")
list off
      .NULL.      Smith
```

```
set nulldisplay to "<null>"
list off
      <null>      Smith
```

```
set nulldisplay to
list off
      .NULL.      Smith
```

### Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET SECONDS

---

**Class**

Date and Time Data

**Purpose**

Specify whether the display of datetime values includes seconds

**Syntax**

SET SECONDS ON | OFF | (<expL>)

**See Also**

CDOW(), CMONTH(), CTOD(), DATE(), DATETIME(), DAY(), DAYS(), DMY(), DOW(), DTOC(), DTOM(), DTOS(), DTOV(), EPOCH(), GOMONTH(), MDY(), MONTH(), MTOD(), QUARTER(), STOD(), VTOD(), YEAR(), SET CENTURY, SET DATE, SET EPOCH, SET HOURS, SET MARK

**Description**

If SET SECONDS is ON, then the display of datetime values includes seconds. If SECONDS is OFF then the time part of datetime values only includes hours, minutes and AM | PM. By default, SECONDS is ON. The SET HOURS set command determines whether hours are shown in 24 hour format or in 12 hour format with AM | PM postfix.

**Example**

```
set seconds off
```

```
? datetime()
```

```
01/23/2004 01:18 PM
```

```
set seconds on
```

```
? datetime()
```

```
01/23/2004 01:18:22 PM
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET SEQNO

---

**Class**

Table Organization

**Purpose**

Reset sequence numbering to specified number

**Syntax**

SET SEQNO TO <expN>

**See Also**

ALTER TABLE, CREATE TABLE, SEQNO

**Description**

The SET SEQNO TO <expN> command resets the sequence number of the currently active table to the specified <expN> value. The next time the SEQNO() function is called, the value returned will be <expN> + 1.

**Example**

set seqno to 2000

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET SQL

---

### Class

Environment

### Purpose

Enable or disable use of SQL statements or configure SQL statement syntax

### Syntax

SET SQL ON | OFF | (<expL>)

SET SQL[DIALECT] TO RECITAL | VFP | MYSQL

### See Also

EXEC SQL

### Description

When SQL is set ON, commands that follow are assumed to be SQL, not Recital/4GL. In the development environment of Recital Terminal Developer, the interactive SQL prompt will be displayed and valid Recital/ SQL commands can be executed. Most of the Recital non-SQL commands can also be executed, but commands that conflict with SQL are not permitted. When SQL is set OFF, the normal Recital prompt is displayed. By default, SQL is OFF.

The SET SQL ON command can also be used in config.db configuration files for session, application or system wide setting. Since it affects the compilation of programs, it should be set prior to compilation rather than in a program itself. Program files with a '.sql' file extension are automatically compiled and run with SET SQL ON.

Where Recital, MySQL and VFP differ in their SQL syntax, the SET SQL TO <dialect> command can be used to select the syntax to be used. By default, SQL is set to RECITAL (see exceptions in notes below). Since the SQL setting affects program compilation, it should be set prior to compilation rather than in a program itself, for example in a config.db configuration file.

### NOTES:

;

The semi-colon, ';', is used to terminate SQL statements when SQL is set to RECITAL or MYSQL. It is used as a line continuation character when SQL is set to VFP.

### EXEC SQL

When SQL is set to RECITAL, SQL statements embedded in programs must be preceded by the EXEC SQL statement unless SET SQL is ON.

### .sql programs

When a program with a '.sql' file extension is run, SQL is automatically set to MYSQL and SET SQL is set ON.

### Recital Command Prompt

At the Recital/4GL Command Prompt in Recital Terminal Developer and Recital Visual Developer, SQL is always set to VFP.

**Example**

```
> set sql on
```

```
Recital/SQL> select * from accounts;
```

```
set sql to vfp
```

```
set sql to mysql
```

```
set sql on
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET SQLPROMPT

---

**Class**

Recital Terminal Developer Environment

**Purpose**

Specify the Recital/SQL interactive prompt

**Syntax**

SET SQLPROMPT TO <expC> | DEFAULT

**See Also**

SET SQL

**Description**

The SET SQLPROMPT TO <expC> command allows you to specify the interactive SQL prompt. The <expC> can be a maximum of 10 characters long. The SET SQLPROMPT TO DEFAULT command sets the SQL prompt to 'SQL>'.

**Example**

```
Recital/SQL> set sqlprompt to "R-SQL> "  
R-SQL> set sqlprompt to default  
SQL>
```

**Products**

Recital Terminal Developer



## SET SQLROWID

---

**Class**

SQL Applications

**Purpose**

To include a unique row identifier in SELECT \* statements

**Syntax**

SET SQLROWID ON | OFF

**See Also**

SELECT, UNIQUEROWID()

**Description**

The SET SQLROWID ON | OFF command is used to determine whether a unique row identifier in should be included in SELECT \* statements. If SET SQLROWID is ON, the unique row identifier will be included, if SET SQLROWID is OFF, only the fields from the table will be included.

**Example**

```
set sql on
set sqlrowid on
select * from state.rdb where state = "M";
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET TCACHE

---

### Class

SQL Applications

### Purpose

Enable, disable and configure table caching during SQL operations

### Syntax

SET TCACHE ON | OFF | TO <expN>

### See Also

ALTER INDEX, ALTER TABLE, CREATE INDEX, CREATE TRIGGER, DELETE, DROP INDEX, GRANT, INSERT, REVOKE, SELECT, UPDATE

### Description

The SET TCACHE command is used to enable or disable table caching and configure the number of tables that can be cached during SQL operations. If SET TCACHE is ON, a table accessed by an SQL statement is left open until the session or connection is closed. With SET TCACHE OFF, tables are opened and closed with every SQL statement.

With SET TCACHE ON, the default number of tables that can be cached corresponds to the number of available workareas (DB\_MAXWKA environment variable / symbol). This number can be reduced using the SET TCACHE TO <expN> command. The <expN> specifies the maximum number of tables that can be cached.

Enabling TCACHE can give significant performance benefits where multiple operations are being carried out on the same table or tables. With TCACHE ON, individual tables can still be closed if required using the USE command in the relevant workarea or the CLOSE <alias> command.

### Example

// Up to 4 tables will remain open after a completed SQL statement

```
set tcache on
set tcache to 4
set sql to recital
```

### EXEC SQL

```
OPEN DATABASE southwind;
```

### EXEC SQL

```
INSERT INTO shippers
VALUES (4, "Recital Corporation", "(978) 921-5594");
```

### EXEC SQL

```
UPDATE employees
SET extension="256"
WHERE employeeid=4;
```

### EXEC SQL

```
DELETE FROM suppliers
WHERE supplierid=1;
```

```
EXEC SQL  
  ALTER TABLE example  
  ADD (email char(40));
```

```
EXEC SQL  
SELECT * from products;
```

```
// Next free workarea is workarea 5  
// as tables in workareas 1 to 4 remain open  
? workarea()  
?  
close databases
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET TRANSACTION

---

**Class**

SQL Applications

**Purpose**

To set the current transaction state

**Syntax**

SET TRANSACTION [READ ONLY | READ WRITE]

SET TRANSACTION ISOLATION LEVEL <level>

**See Also**

CLOSE, DECLARE CURSOR, DROP CURSOR, FETCH, OPEN, SELECT

**Description**

A current transaction state can be either read-only or read-write. Three further aspects of transaction behavior are configurable: dirty reads, non-repeatable reads and phantom reads.

Dirty reads occur when a transaction updates a row, then a second transaction reads that row before the first transaction commits. If the first transaction rolls back the change, the information read by the second transaction becomes invalid.

Non-repeatable reads occur when a transaction reads a row then another transaction updates the same row. If the second transaction commits, subsequent reads by the first transaction get different values than the original read.

Phantoms occur when a transaction reads a set of rows that satisfy a search condition and then another transaction updates, inserts, or deletes one or more rows that satisfy the first transaction's search condition. In this case, if the first transaction performs subsequent reads with the same search condition, it reads a different set of rows.

The <level> can be any one of the following:-

- **SERIALIZABLE**
- **REPEATABLE READ**
- **READ COMMITTED**
- **READ UNCOMMITTED**

If you use a SET TRANSACTION statement, it must be the first statement in your transaction.

NOTE: This command can also be used as a standard SET COMMAND in the config.db file, to set the transaction state on a system or application wide basis.

<b>Keywords</b>	<b>Description</b>
READ ONLY	Set the default transaction type to read-only.
READ WRITE	Set the default transaction type to read-write.
ISOLATION LEVEL	Specify how the transaction will perform.
SERIALIZABLE	This will disable dirty reads, non-repeatable reads and phantom reads. This is the default isolation level.
REPEATABLE READ	This will disable dirty reads, non-repeatable reads and enable only phantom reads.
READ UNCOMMITTED	This will enable dirty reads, non-repeatable reads and phantom reads.
READ COMMITTED	This will disable dirty reads and enable non-repeatable reads and phantom reads.

**Example**

set transaction isolation level read uncommitted;

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

## SET XMLFORMAT

---

**Class**

SQL Applications

**Purpose**

Specify the default format for XML files created by SELECT...SAVE AS XML

**Syntax**

SET XMLFORMAT TO <RECITAL | ADO>

**See Also**

SELECT

**Description**

The SET XMLFORMAT TO <RECITAL | ADO> command allows you to specify the default format for XML files created by SELECT...SAVE AS XML. The XMLFORMAT can be either RECITAL or ADO (Microsoft® ActiveX® Data Objects). Any XML files created in the ADO format can be loaded with the Open method of an ADO Recordset object.

The default XMLFORMAT setting is ADO. The default XMLFORMAT setting can also be overridden using the FORMAT clause on the SELECT statement.

**Example**

```
set xmlformat to ADO
EXEC SQL
SELECT * FROM example
SAVE AS XML example;
// In Visual Basic the file can then be loaded:
// Set adoPrimaryRS = New Recordset
// adoPrimaryRS.Open "example.xml"
```

**Products**

Recital Database Server, Recital Mirage Server, Recital Terminal Developer