VisiBone MySQL Card
Statements A-G

ALTER ~~OFFLINE~~ / ONLINE IGNORE **TABLE** *t*
ADD ~~COLUMN~~ *c type◊ cops◊* *seq◊* ;
ADD ~~COLUMN~~ (*c type◊ cops◊*)
ADD *index-definition◊*
ALTER ~~COLUMN~~ *c* SET DEFAULT *:* / DROP DEFAULT
CHANGE ~~COLUMN~~ *c_OLD c_NEW type◊ cops◊* *seq◊*
CONVERT TO / ~~DEFAULT~~ CHARACTER SET *cs* COLLATE *cll*
DROP ~~COLUMN~~ *c*
DROP INDEX *i* / PRIMARY KEY / FOREIGN KEY *cn*
DISABLE KEYS
ENABLE KEYS
MODIFY ~~COLUMN~~ *c type◊ cops◊* *seq◊*
ORDER BY *c*
RENAME ~~TO~~ *t_NEW*
tops◊
partition-command◊

ALTER **DATABASE** / SCHEMA *d* *charset◊* ;

ALTER EVENT *evt* *event-definition◊* ;

ALTER **FUNCTION** *f* / **PROCEDURE** *p* *rops◊* ;

ALTER *vops◊* **VIEW** *vw* (*c*,,,) AS SELECT-*statement* *checkopt◊* ;

ANALYZE TABLE *t*,,, ; scan for errors or warnings

BEGIN ~~WORK~~ ; = START TRANSACTION

CACHE INDEX *t* KEY / INDEX (*i*) IN *cache* ;

CALL *p* (*:*,,,) ;

CASE *:* WHEN *:* *statement*;;; ELSE *statement*;;; **END CASE;** ℜ() ala switch in C

CASE WHEN *b* *statement*;;; ELSE *statement*;;; **END CASE;** ℜ() ala if-else... in C

CHECK TABLE *t*,,, FOR UPGRADE / ~~MEDIUM~~ QUICK / FAST EXTENDED / CHANGED ; diagnostic

CHECKSUM TABLE *t*,,, QUICK EXTENDED ; 32-bit checksum of entire table

CLOSE *cursor* ; ℜ()

COMMIT ~~WORK~~ AND NO CHAIN NO RELEASE ; makes permanent all since START

CREATE **DATABASE** / SCHEMA IF NOT EXISTS *d* *charset◊* ;

CREATE EVENT IF NOT EXISTS *evt* *event-definition◊* ;

CREATE ~~OFFLINE~~ / ONLINE *index-definition◊* ;

CREATE DEFINER = *user@host* **FUNCTION** *f* (*v type◊*) RETURNS *type◊* / **PROCEDURE** *p* (~~IN~~ / OUT / INOUT *v type◊*) *rops◊* BEGIN *statement*;;; END ; / RETURN *:* ;
For interactive mysql, use the "delimiter" command to avoid ; confusion.

DECLARE *v type◊* DEFAULT *:*
name CONDITION FOR *condition◊*
cursor CURSOR FOR SELECT-*statement*
h-before◊ HANDLER FOR *h-after◊* *BEGIN...END◊* ;

CREATE AGGREGATE **FUNCTION** *f* RETURNS STRING / INTEGER / REAL **SONAME** *library.so* ;

CREATE TEMPORARY **TABLE** IF NOT EXISTS *t* (*c type◊ cops◊* / *index-definition◊*) *tops◊* *as-select◊* ;
t_NEW (LIKE *t_OLD*)

CREATE DEFINER = *user@host* **TRIGGER** *triggername* BEFORE / AFTER INSERT / UPDATE / DELETE ON *t* FOR EACH ROW *BEGIN...END◊* ;

CREATE USER *user@host* IDENTIFIED BY ~~PASSWORD~~ '*password*' ; then GRANT privileges

CREATE OR REPLACE *vops◊* **VIEW** *vw* (*c*,,,) AS SELECT-*statement* *checkopt◊* ;

DEALLOCATE PREPARE *stmt* ;

DELETE LOW_PRIORITY / QUICK / IGNORE
FROM *t*
t .* FROM *table◊*
FROM *t* .* USING *table◊*
WHERE *expression* ORDER BY *c* ~~ASC~~ / DESC LIMIT *nrows* ;
Deletes every row, unless there's a WHERE or LIMIT

DESCRIBE ~~DESCRIBE~~ *t* *c* / '*%pattern%*' ; similar to SHOW COLUMNS

DO *:*,,, ; Like SELECT, except no output

DROP **DATABASE** / SCHEMA IF EXISTS *d* ;

DROP EVENT IF EXISTS *evt* ;

DROP FUNCTION IF EXISTS *f* ;

DROP ~~OFFLINE~~ / ONLINE **INDEX** *i* ON *t* ;

DROP PREPARE *stmt* ;

DROP PROCEDURE IF EXISTS *p* ;

DROP TEMPORARY **TABLE** IF EXISTS *t*,,, ;

DROP TRIGGER IF EXISTS *d*. *triggername* ;

DROP USER *user@host*,,, ;

DROP VIEW IF EXISTS *vw*,,, ~~CASCADE~~ / ~~RESTRICT~~ ;

EXECUTE *stmt* USING @*variable*,,, ; → PREPARE

EXPLAIN *t* ; = SHOW COLUMNS FROM *t*

EXPLAIN SELECT-*statement* ; optimizer clues

FETCH *cursor* INTO @*variable*,,, ; ℜ()

FLUSH HOSTS / DES_KEY_FILE / LOGS / PRIVILEGES / QUERY CACHE / STATUS / TABLES *t*,,, / TABLES WITH READ LOCK / USER_RESOURCES ;

REVOKE

GRANT *t-privilege◊* (*c*,,,) ON ~~TABLE~~ *t* / * / *.* / *d*.* TO *user@host* IDENTIFIED BY ~~PASSWORD~~ '*password*' REQUIRE *encryption◊* ;
r-privilege◊ ON **FUNCTION** / **PROCEDURE** *name* / *d.name*
WITH GRANT OPTION (delegate) / MAX_QUERIES_PER_HOUR *n* / MAX_UPDATES_PER_HOUR *n* / MAX_CONNECTIONS_PER_HOUR *n* / MAX_USER_CONNECTIONS *n*

◊*charset*
CHARSET / CHARACTER SET *cs* COLLATE *cll*
UNICODE = CHARACTER SET ucs2
ASCII = CHARACTER SET latin1

◊*checkopt*
WITH ~~CASCADED~~ / LOCAL CHECK OPTION

◊*BEGIN...END*
label: **BEGIN** *statement*;;; **END** *label*
statement

◊*seq*
FIRST
AFTER *c*

◊*as-select*
IGNORE / REPLACE ~~AS~~ SELECT-*statement*

◊*vops* (view options)
ALGORITHM = UNDEFINED / MERGE / TEMPTABLE
DEFINER = *user@host* / CURRENT_USER
SQL SECURITY DEFINER / INVOKER

◊*tops* (table options)
AUTO_INCREMENT = *n*
AVG_ROW_LENGTH = *n*
~~DEFAULT~~ CHARACTER SET *cs*
CHECKSUM = 0 / 1
COLLATE *cll*
COMMENT = '*string*'
CONNECTION = '*string*'
DATA DIRECTORY = '*path*'
DELAY_KEY_WRITE = 0 / 1
ENGINE = *engine◊*
INDEX DIRECTORY = '*path*'
INSERT_METHOD = NO / FIRST / LAST
KEY_BLOCK_SIZE = *n*
MAX_ROWS = *n*
MIN_ROWS = *n*
PACK_KEYS = 0 / 1 / DEFAULT
PASSWORD = '*string*'
ROW_FORMAT = DEFAULT / DYNAMIC / FIXED / COMPRESSED / REDUNDANT / COMPACT
UNION = *t*,,,
partition-definition◊

◊*rops* (routine options)
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER / INVOKER
COMMENT '*stg*'
CONTAINS SQL / NO SQL / READS SQL DATA / MODIFIES SQL DATA

◊*r-privilege* (routines)
ALL ~~PRIVILEGES~~
ALTER ROUTINE
EXECUTE
GRANT OPTION

◊*t-privilege* (table privilege)
ALL ~~PRIVILEGES~~
ALTER
CREATE
CREATE ROUTINE
CREATE TEMPORARY TABLES
CREATE USER
CREATE VIEW
DELETE
DROP
EVENT
FILE
GRANT OPTION
LOCK TABLES
INDEX
INSERT
PROCESSS
REFERENCES
RELOAD
REPLICATION CLIENT / SLAVE
SELECT
SHOW DATABASES
SHOW VIEW
SHUTDOWN
SUPER
TRIGGER
UPDATE
USAGE

◊*engine*
MyISAM
InnoDB
Memory
Merge
Archive
Federated
CSV
Blackhole
Example
Falcon

◊*cops* (column options)
NOT NULL
COMMENT '*string*'
DEFAULT *value*
AUTO_INCREMENT
UNIQUE ~~KEY~~
~~PRIMARY~~ KEY
COLUMN_FORMAT FIXED / DYNAMIC / ~~DEFAULT~~
STORAGE DISK / ~~MEMORY~~
references◊

CHAR(*n*) stores *n* characters space-padded, stripped on read
BINARY(*n*) stores *n* bytes, NUL-padded, but not stripped

VARCHAR(*n*) stores 0-*n* characters plus a length
VARBINARY(*n*) stores 0-*n* bytes plus a length

CHAR, VARCHAR and TEXT are CcIi
BINARY, VARBINARY and BLOB are CScs

◊*type*
CHAR (*nchar*) BINARY *charset◊*
BINARY (*nbyte*)
VARCHAR (*ncharmax*) BINARY *charset◊*
VARBINARY (*nbytemax*)
64K chars **TEXT** BINARY *charset◊*
TINYTEXT 255 characters
MEDIUMTEXT 16M characters
LONGTEXT 4G characters
64K bytes **BLOB**
TINYBLOB 255 bytes
MEDIUMBLOB 16M bytes
LONGBLOB 4G bytes
± 2e9 **INTEGER** (*nDigits*) UNSIGNED ZEROFILL
TINYINT ± 127
SMALLINT ± 32e3
MEDIUMINT ± 8e6
BIGINT ± 9e18
e.g. (5,2) for 999.99
DECIMAL (*nDigits, nRight*) UNSIGNED ZEROFILL
FLOAT
DOUBLE ~~**PRECISION**~~
some DECIMAL math may actually be done in DOUBLE PRECISION
aka **DEC**
aka **FIXED**
65 digits, aka **NUMERIC**
7 digits
15 digits, aka **REAL**
DATE e.g. '1999-12-31' 1000AD-9999AD
TIME e.g. '23:59:59'
DATETIME 1000AD-9999AD
TIMESTAMP 1970AD-2037AD
YEAR ~~(4)~~ 1901AD-2155AD
YEAR (2) 1970AD-2069AD (Y2K bugaboo)
BOOLEAN = TINYINT(1)
BIT (*nbits*)
SET ('*member*',,,) *charset◊*
ENUM ('*value*',,,) *charset◊*

◊*event-definition*
ON SCHEDULE AT *time* + INTERVAL *n interval◊*
EVERY *n interval◊* STARTS *dt* ENDS *dt*
RENAME TO *evt* CREATE EVENT can't RENAME
ON COMPLETION NOT PRESERVE
~~ENABLE~~ / DISABLE
COMMENT '*string*'
DO *BEGIN...END◊*
ALTER EVENT may omit ON or DO clause

◊*index-definition*
CONSTRAINT *cn* INDEX / KEY / PRIMARY KEY / FOREIGN KEY / UNIQUE / FULLTEXT / SPATIAL INDEX / KEY *i* USING BTREE / USING HASH (*c* (*n*)) KEY_BLOCK_SIZE *n* / USING BTREE / USING HASH / WITH PARSER *name* / COMMENT '*string*' *references◊*
CONSTRAINT *cn* is only for FOREIGN keys, and is not allowed in CREATE INDEX

◊*references*
REFERENCES *t* (*c*,,,) MATCH FULL / PARTIAL / SIMPLE
ON DELETE / ON UPDATE RESTRICT / CASCADE / SET NULL / NO ACTION
only for FOREIGN keys

HANDLER *t* OPEN AS *handle* ;

HANDLER *handle* CLOSE ;

HANDLER *handle* READ *i* = >= <= < > (*z,,,*) FIRST PREV NEXT LAST WHERE *expr-ssion* LIMIT *nrow* | *nskip, nrow* | *nrow* OFFSET *nskip* ;

HANDLER is a fast primitive read, SELECT is more advanced

IF *b* THEN *statement;;;* ELSEIF *b* THEN *statement;;;* ELSE *statement;;;* END IF; ℜ()

INSERT *insert-options◊* ~~INTO~~ *t* SET *c* = *z* | (*c,,,*) VALUES (*z,,,*) | (*c,,,*) SELECT-*statement* ON DUPLICATE KEY UPDATE *c* = *z* ;

INSERT duplicates are usually errors. REPLACE pre-deletes to avoid duplicates.

◊*insert-options*: LOW_PRIORITY | HIGH_PRIORITY | DELAYED | IGNORE

ITERATE *label* ; ℜ() ala continue, in C

KILL ~~CONNECTION~~ | QUERY *process-id* ; → SHOW PROCESSLIST

LEAVE *label* ; ℜ() ala break, in C

LOAD DATA | XML *load-options◊* INFILE '*path*' REPLACE | IGNORE INTO TABLE *t* *fields◊* *lines◊* IGNORE *n* LINES (*c,,,*) SET *c* = *z* ; ROWS IDENTIFIED BY <*tagname*> ← XML only, defaults to <ROW>

◊*load-options*: LOW PRIORITY | CONCURRENT | LOCAL

LOAD INDEX INTO CACHE *t* KEY | INDEX (*i*) IGNORE LEAVES IN *cache* ;

LOCK TABLES *t* AS *alias* READ LOCAL | LOW_PRIORITY WRITE | IN SHARE MODE | IN EXCLUSIVE MODE NOWAIT ;

label: LOOP *statement;;;* END LOOP *label* ; ℜ() infinite loop, → LEAVE

OPEN *cursor* ; ℜ()

OPTIMIZE TABLE *t,,,* ; recover deleted space, defrag

PREPARE *stmt* FROM '*statement*' ; ? in *statement* will be replaced by EXECUTE's arguments

RENAME DATABASE | SCHEMA *d* TO *d* ;

RENAME TABLE *t* TO *t* ;

RENAME USER *user@host* TO *user@host* ;

REPAIR TABLE *t,,,* QUICK EXTENDED USE_FRM ;

label: REPEAT *statement;;;* UNTIL *b* END REPEAT *label* ; ℜ()

REPLACE LOW_PRIORITY | DELAYED ~~INTO~~ *t* SET *c* = *z* | (*c,,,*) VALUES (*z,,,*) | (*c,,,*) SELECT-*statement* ;

RESET QUERY CACHE ;

REVOKE *t-privilege◊* (*c,,,*) ON ~~TABLE~~ *t* | * | *.* | *d*.* FROM *user@host* ;
r-privilege◊ ON FUNCTION | PROCEDURE *name* | *d.name*

ROLLBACK ~~WORK~~ AND NO CHAIN NO RELEASE TO SAVEPOINT *name* ; undo since START

SAVEPOINT *name* ;

SELECT *sops◊* *cx* ~~AS~~ *alias* | * | *t*.* | *d.t*.* FROM *table◊* | ~~FROM DUAL~~ WHERE *expr-ssion* GROUP BY *cAX* ~~ASC~~ | DESC WITH ROLLUP HAVING *expr-ssion* ORDER BY *cAX* ~~ASC~~ | DESC LIMIT *nrow* | *nskip, nrow* | *nrow* OFFSET *nskip* *into◊* ; *into◊*

no column aliases in WHERE *expr-ssion*

ROLLUP: super-group totals

column aliases ok in HAVING *expr-ssion*

GROUP BY merges duplicates, see ᚔ group functions like SUM()

FOR UPDATE | LOCK IN SHARE MODE | PROCEDURE *pExternal* (*z,,,*) | PROCEDURE analyse(*n*, *n*)

e.g. SELECT * FROM t;

e.g. SELECT * FROM ta LEFT JOIN tb ON ta.id = tb.id WHERE ta.date >= 2009;

◊*sops* (SELECT options): ~~ALL~~ | DISTINCT | DISTINCTROW; HIGH_PRIORITY; STRAIGHT_JOIN; SQL_SMALL_RESULT; SQL_BIG_RESULT; SQL_BUFFER_RESULT; SQL_CACHE; SQL_NO_CACHE; SQL_CALC_FOUND_ROWS

◊*subquery-expression*: *b* — NOT EXISTS | *z* | ROW (*z,,,*) NOT IN | = | < | > | <= | >= | <> ALL | ANY | SOME (SELECT-*statement*)

A subquery makes for a very powerful WHERE *expr-ssion*

ANY ≡ SOME

◊*into*: INTO @*variable,,,* | INTO OUTFILE '*path*' *fields◊* *lines◊* | INTO DUMPFILE '*path*'

◊*fields*: FIELDS TERMINATED BY '*separator*' OPTIONALLY ENCLOSED BY '*quoter*' ESCAPED BY '*slasher*'

default field *separator* is '\t' (TAB)

◊*lines*: LINES STARTING BY '*prefix*' TERMINATED BY '*suffix*'

default line *suffix* is '\n' (LF)

SET GLOBAL | SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE ; fiddles with START, COMMIT

SET PASSWORD FOR *user@host* = PASSWORD ('*password*') ;

SET @*globalApplicationVariable* | *localApplicationVariable* | GLOBAL *@setting* | @@global . *setting* | SESSION *@setting* | @@session . @*setting* | LOCAL *@setting* | @@local . @*setting* | @@*setting* = | := *z* ; → SHOW VARIABLES

SHOW DATABASES | SCHEMAS LIKE '*%pattern%*' ;
CHARACTER SET | COLLATION | FUNCTION STATUS | PROCEDURE STATUS
CREATE DATABASE | SCHEMA *d* | EVENT *evt* | FUNCTION *f* | PROCEDURE *p* | TABLE *t* | VIEW *vw*
EVENTS | TABLE STATUS | TRIGGERS | FULL TABLES | OPEN FROM *d* LIKE '*%pattern%*'
ERRORS | WARNINGS LIMIT *nrow* | *nskip, nrow* | *nrow* OFFSET *nskip*
FULL COLUMNS FROM *t* | FROM *d.t* LIKE '*%pattern%*'
INDEX | KEYS FROM *t* | FROM *d.t*
~~SESSION~~ | GLOBAL STATUS | VARIABLES LIKE '*%pattern%*'
ENGINE *engine◊* LOGS | STATUS
~~STORAGE~~ ENGINES
FULL PROCESSLIST
PRIVILEGES
PLUGINS
GRANTS FOR '*user@host*' @'*client-host*'

, is archaic shorthand for INNER JOIN, except the relational *expr-ssion* goes in the WHERE-clause

OUTER includes orphans (unrelated rows that never satisfy the ON *expr-ssion*) and LEFT or RIGHT says from which table.

◊*table* (and the venerable JOIN): *t* ~~AS~~ *table-alias* USE | FORCE | IGNORE INDEX (*i,,,*)
table◊ , *table◊*
table◊ ~~INNER~~ JOIN *table◊*
table◊ LEFT ~~OUTER~~ JOIN *table◊*
table◊ RIGHT ~~OUTER~~ JOIN *table◊* ON *expr-ssion* | USING (*c,,,*)
table◊ NATURAL JOIN *table◊*
table◊ NATURAL LEFT ~~OUTER~~ JOIN *table◊*
table◊ NATURAL RIGHT ~~OUTER~~ JOIN *table◊*

USING (col) means ON t1.col = t2.col
NATURAL automagically relates columns

INNER JOIN ≡ CROSS JOIN

The heart and soul of SQL: inventing new, imaginary tables by pretending to JOIN real ones. This hinges ON expressing how they're related, (which is why we call them relational databases) usually by having certain column values in common.

To understand JOIN well, you must imagine the preposterous: that joining **m** rows with **n** rows generates all **m×n** combinations! If you distill that down with a sensible ON *expr-ssion*, the optimizer avoids overload, and you get to work with a reasonable number of joined rows.

Terminology

cluster, partition, replication: multi-machine server
column aka **field**, specified type, e.g. TEXT or INTEGER
cursor row pointer, looped with OPEN, FETCH, CLOSE
database aka schema, collection of tables
event time-scheduled statement(s), ☯ trigger
index aka **key**, a column prepared for quick search
row aka record, cohesive data (e.g. name, rank, serial)
~~**stored**~~ **routine:** a ~~stored~~ **procedure** or **function**
table data structured into columns and rows, or a view
transaction all-or-nothing series between START, COMMIT
trigger data-change-initiated statement(s), ☯ event
view a stored SELECT statement, behaves like a table

START TRANSACTION WITH CONSISTENT SNAPSHOT ; COMMIT completes. ROLLBACK undoes.

TRUNCATE ~~TABLE~~ *t* ; ☛ Empties table.

(SELECT-*statement*) UNION ~~DISTINCT~~ | ALL (SELECT-*statement*) ... ORDER BY *cx* ~~ASC~~ | DESC LIMIT *nrow* | *nskip, nrow* | *nrow* OFFSET *nskip* ;

JOIN glues columns.
UNION glues rows.

UNLOCK TABLES ;

UPDATE LOW_PRIORITY IGNORE *t* | *table◊* SET *c* = *z* WHERE *expr-ssion* ORDER BY *cx* ~~ASC~~ | DESC LIMIT *nrow* | *nskip, nrow* | *nrow* OFFSET *nskip* ;

USE *d* ; sets default, so you can SELECT * FROM *t* instead of FROM *d.t*

label: WHILE *b* DO *statement;;;* END WHILE *label* ; ℜ()

☛ UPDATE without a WHERE or LIMIT clause affects every row.

XA *distributed-transaction-commands◊* ;

-- *comment* one line
# *comment* one line
/* *comments* */ multi-line
/*! *mysql-only statements* */ other SQLs treat as a comment
/*!50206 *mysql-version-5.2.6 statements* */ earlier MySQL treats as a comment

VisiBone MySQL Card
Statements H-Z

Most **identifiers** consist of:
abc...xyz 123...890 $ _
Use backticks otherwise, e.g.
SELECT * FROM `a-game`;
Also for *reserved-words◊*, e.g.
SELECT * FROM `order`;

Legend

VERBATIM MySQL reserved word, CcIi
symbol abstraction | optional
optional | optional | optional
~~cosmetic~~ has no effect
( ) ; decorative punctuation
one of | these must choose exactly one
any of | these choose any, all or none
x or *x,,,* comma-separated list
x;;; ;-separated list, ending in a ;
x space-separated list
reference◊ or *reference-online◊*
◊*definition* or visibone.com/sql/tech
expr-ssion often an equality test
nonstandard MySQL, not ISO/ANSI
MySQL version 6 nonstanard v6
CScs case sensitive CcIi insensitive
CScs'nix CcIi'doze – OS-dependent
0 zero-based counting (0,1,2,...)
1 one-based counting (1,2,3,...)
-R negative=right (1,2,3,...,-3,-2,-1)
0=NF zero means not-found
nKMG e.g. 3K, 6M, 9G=9 gigabytes

X stuff you'll use a lot
ᚔ for GROUP BY columns
☛ gotcha (be careful)
≡ identical to
☯ contrast with
ℜ() only in a stored routine
→ X see also X
x ← results in type *x*
☚ MySQL recommends

Names

(Manual calls them identifiers)

alias for column or table CcIi
c (or ***t.c*** or ***d.t.c***) column name CcIi
cx column name or expression
cAX column name or alias or expression
cs character set name (language alphabet)
cll collation name (text sort order)
cn constraint name (data consistency)
cursor cursor name, ℜ() row pointer
d database name CScs'nix CcIi'doze
evt event name CcIi
f function name CcIi
i index name CcIi
label e.g. label: BEGIN ... END label
p procedure name CcIi
stmt statement name: PREPARE, EXECUTE
t (or ***d.t***) table name CScs'nix CcIi'doze
triggername CScs
user@host e.g. 'jim'@'%.server.com'
v *localApplicationVariable* (no at-sign)
@variable @*globalApplicationVariable*
vw (or ***d.vw***) view CScs'nix CcIi'doze

VisiBone MySQL Card

Functions A-L

n ← ABS(n) absolute value

x ← ACOS(x) arccosine, in radians e.g. 0.25 = ACOS(SQRT(0.5))/PI()

d ← ADDDATE(d , ndays)

d ← ADDDATE(d , INTERVAL n interval∅) = DATE_ADD()

t ← ADDTIME(t , t) e.g. '23:59:59' = ADDTIME('12:00:00', '11:59:59')

s ← AES_DECRYPT(z , skey) e.g. 'a' = AES_DECRYPT(AES_ENCRYPT('a', 'key'), 'key')

z ← AES_ENCRYPT(s , skey) e.g. 16 = LENGTH(AES_ENCRYPT('a', 'key'))

b ← b AND b e.g. FALSE = (TRUE AND FALSE)

n ← ASCII(s) ASCII value of 1st character e.g. 32 = ASCII(' ') ☯ CHAR()

x ← ASIN(x) arcsine, in radians e.g. 0.25 = ROUND(ASIN(SQRT(0.5))/PI(), 15)

x ← ATAN(x) arctangent, in radians e.g. 0.25 = ATAN(1)/PI()

x ← ATAN2(x , x) 4-quadrant arctangent, in radians e.g. -0.75 = ATAN2(-1,-1)/PI()

x ← AVG(DISTINCT c_X) average of non-NULLs ⫼

0 ← BENCHMARK(nTimes , zExpression) e.g. BENCHMARK(1e3, COMPRESS(''))

b ← z NOT BETWEEN zlo AND zhi inclusive range e.g. TRUE = (55 BETWEEN 11 and 99)

s ← BIN(n) binary digit encode e.g. '101010' = BIN(42)

s ← BINARY s CcIi = CAST(s AS BINARY) e.g. (BINARY 'ABC') != (BINARY 'abc')

x ← BIT_AND(c_X) bitwise AND of non-NULLs ⫼

n ← BIT_COUNT(n) count the 1 bits e.g. 4 = BIT_COUNT(b'11011')

n ← BIT_LENGTH(s) usually 8*LENGTH(s) unless Unicode

x ← BIT_OR(c_X) bitwise OR of non-NULLs ⫼

x ← BIT_XOR(c_X) bitwise XOR of non-NULLs ⫼

z ← CASE WHEN b THEN z ... ELSE z END ala if(b){z} else if(b){z} ... else{z} in C

z ← CASE z WHEN z THEN z ... ELSE z END ala switch(z){case z: z; case z: z; ... default:z;} in C

z ← CAST(z AS cast-type∅) similar to CONVERT()

n ← CEILING(x) nearest integer greater e.g. -7 = CEIL(-7.9) ☯ FLOOR()

s ← CHAR($n_{...}$ USING cs) make ASCII e.g. '123' = CHAR(49,50,51) ☯ ASCII()

n ← CHAR_LENGTH(s) or CHARACTER_LENGTH(s) string length in characters, ☯ OCTET_LENGTH(), LENGTH()

s ← CHARSET(s) e.g. 'latin1' = CHARSET('a')

z ← COALESCE($z_{...}$) first non-NULL e.g. 3 = COALESCE(NULL,NULL,3,4,5)

0-6 ← COERCIBILITY(s) 0=rigid, 6=loose e.g. 4 = COERCIBILITY('literal')

s ← s COLLATE cll e.g. 'latin1_bin' = COLLATION('a' COLLATE 'latin1_bin')

s ← COLLATION(s) e.g. 'latin1_swedish_ci' = COLLATION('a')

z ← COMPRESS(s) a long string into a short blob e.g. 'a lot to say' = UNCOMPRESS(COMPRESS('a lot to say'))

s ← CONCAT($s_{...}$) string concatenation, e.g. 'abc' = CONCAT('a','b','c') There's a shorter way with literals, e.g. 'xyz' = 'x' 'y' 'z'

s ← CONCAT_WS(separator , $s_{...}$) e.g. '1.2.3' = CONCAT_WS('.',1,2,3)

n ← CONNECTION_ID( ) unique among current connections

s ← CONV(s , nbasefrom , nbaseto) e.g. 'FF' = CONV('255',10,16)

z ← CONVERT(z , cast-type∅)

s ← CONVERT(s USING cs) e.g. 'greek' = CHARSET(CONVERT('a' USING 'greek'))

t ← CONVERT_TZ(t , tzfrom , tzto) change timezone -E +W

x ← COS(x) cosine, in radians e.g. 0.5 = ROUND(COS(PI()/3), 15)

x ← COT(x) cotangent In radians e.g. 1 = ROUND(COT(PI()/4), 15)

n ← COUNT(DISTINCT c_X , c_X , ...) non-NULL rows (DISTINCT ignores duplicates) ⫼

n ← COUNT(*) rows ⫼

n ← CRC32(s) e.g. 3904355907 = CRC32('a')

d ← CURRENT_DATE( ) or CURDATE( ) e.g. '2019-12-31' = CURDATE()

t ← CURRENT_TIME( ) or CURTIME( ) similar to NOW() e.g. '23:59:59' = CURTIME()

dt ← CURRENT_TIMESTAMP( ) current date and time = NOW()

s ← CURRENT_USER( ) similar to USER()

s ← DATABASE( ) from last USE command e.g. 'databasename' = DATABASE()')

d ← DATE(dt) date only e.g. '1999-12-31' = DATE('1999-12-31 00:00:00')

d ← DATE_ADD(d , INTERVAL n interval∅) = ADDDATE()

n ← DATEDIFF(dLate , dEarly) days difference e.g. 1 = DATEDIFF('2000-01-01', '1999-12-31')

s ← DATE_FORMAT(d , sFormat∅) custom date and time formatting

d ← DATE_SUB(d , INTERVAL n interval∅) e.g. '1999-12-31 23:59:59' = DATE_SUB('2000-01-01 00:00:00', INTERVAL 1 SECOND)

n ← DAY(d) day of the month, 1-31 e.g. 9 = DAY('2011-11-09') = DAYOFMONTH(d)

s ← DAYNAME(d) day of the week e.g. 'Friday' = DAYNAME('1999-12-31')

n ← DAYOFMONTH(d) 1-31 e.g. 8 = DAYOFMONTH('2011-11-08') = DAY(d)

n ← DAYOFWEEK(d) 1=Sun, 7=Sat 1 e.g. 7 = DAYOFWEEK('2000-01-01') ☯ WEEKDAY(d)

n ← DAYOFYEAR(d) Julian day, 1-366 1 e.g. 365 = DAYOFYEAR('1999-12-31') = DAY(d)

s ← DECODE(z , spassword) e.g. 'zebra' = DECODE(ENCODE('zebra', 'key'), 'key')

x ← DEFAULT(c) corresponds to the DEFAULT value in cops∅

x ← DEGREES(xradians) e.g. 45 = DEGREES(PI()/4)

z ← DES_ENCRYPT(s , skey nkey) e.g. 9 = LENGTH(DES_ENCRYPT('a', 'key'))

s ← DES_DECRYPT(z , skey) e.g. 'a' = DES_DECRYPT(DES_ENCRYPT('a', 'key'), 'key')

n ← n DIV n e.g. 3 = (10 DIV 3) = FLOOR(n/n)

z ← ELT(n , z1 , z2 , ...) ennumerated lookup table, NULL=not found, ☯ FIELD()

z ← ENCODE(s , spassword) same-length binary hash e.g. x'6A68D16091' = ENCODE('zebra', 'x')

s ← ENCRYPT(s , salt) 13-character hash Unix: "man crypt" e.g. 'saZAsH0KCIhEE' = ENCRYPT('a', 'sa')

x ← EXP(x) e^x e.g. 2.7182818284590 = ROUND(EXP(1.0), 13)

s ← EXPORT_SET(n , son , soff , sep , nbits) bitmask to string, e.g. '0:0:0:1' = EXPORT_SET(8,'1','0',':',4)

s ← EXTRACT(interval∅ FROM d) e.g. '12' = EXTRACT(MONTH FROM '1999-12-31')

n ← FIELD(z , z1 , z2 , ...) table search, 0=NF e.g. '0' = FIELD('z', 'A', 'B') ☯ ELT()

n ← FIND_IN_SET(single , set) e.g. 2 = FIND_IN_SET('two', 'one,two,three')

n ← FLOOR(x) nearest integer less e.g. -8 = FLOOR(-7.2) ☯ CEILING()

s ← FORMAT(x , ndecimals) big numbers with commas e.g. '10,000.00' = FORMAT(1E4,2)

n ← FOUND_ROWS( ) e.g. SELECT SQL_CALC_FOUND_ROWS * FROM t LIMIT 1; SET @total=FOUND_ROWS();

d ← FROM_DAYS(ndays_after_0_AD) encode date e.g. '2000-01-01' = FROM_DAYS(730485)

dt ← FROM_UNIXTIME(nseconds_after_1970) decode unix-epoch, ☯ UNIX_TIMESTAMP(d)

s ← FROM_UNIXTIME(nseconds_after_1970 , sFormat∅) unix-epoch to formatted-string

s ← GET_FORMAT(DATETIME TIMESTAMP DATE TIME , 'EUR' 'INTERNAL' 'ISO' 'JIS' 'USA') e.g. '%Y-%m-%d %H.%i.%s', readymade sFormat∅

z ← GET_LOCK(sLockName , nSeconds) mutex, ☯ RELEASE_LOCK()

z ← GREATEST($z_{...}$) ☯ MAX(z)

s ← GROUP_CONCAT(DISTINCT $s_{...}$ ORDER BY c_X ASC DESC SEPARATOR sep) ⫼

s ← HEX(n) base-16 encode, e.g. 'F' = HEX(15)

s ← HEX(s) ASCII encode ☯ UNHEX(s) e.g. '6162630A' = HEX('abc\n')

n ← HOUR(t) hour of the day, 0-23 e.g. 23 = HOUR('23:59:59')

z ← IF(b , ztrue , zfalse) chooser, ala (b ? ztrue : zfalse) in C e.g. 'f' = IF(FALSE,'t','f')

z ← IFNULL(z , zNullSymbol) = IF(ISNULL(z), zNullSymbol, z) encodes NULL → zNullSymbol

b ← z NOT IN($z_{...}$) set membership CScs e.g. TRUE = (99 IN (77,88,99))

n ← INET_ATON(sIPaddress) e.g. 0x01020304 = INET_ATON('1.2.3.4')

s ← INET_NTOA(nIPaddress) e.g. '1.2.3.4' = INET_NTOA(0x01020304)

s ← INSERT(s , npos 1 , nremove , sinsert) string insertion e.g. 'sATs' = INSERT('s@s', 2, 1, 'AT')

n ← INSTR(sLooong , sShort) 1 search 0=NF e.g. 3 = INSTR('abcde', 'c') ☞ POSITION() swapped

n ← INTERVAL(z , zA , zB , ...) 0=below all, 1=above zA , 2=above zB , ...

b ← ISNULL(z) ☞ bad: 'z = NULL' good: 'ISNULL(z)' or: 'z IS NULL'

b ← z IS NOT TRUE FALSE UNKNOWN NULL

b ← IS_FREE_LOCK(sLockName) see GET_LOCK(), RELEASE_LOCK()

n ← IS_USED_LOCK(sLockName) yes: CONNECTION_ID(), no: NULL

z ← LAST_DAY(d) of the month e.g. '2000-02-29' = LAST_DAY('2000-02-01')

z ← LAST_INSERT_ID(nfaked) cops∅ AUTO_INCREMENT, e.g. 8791 = LAST_INSERT_ID()

z ← LEAST($z_{...}$) row-wise ☯ MIN(z)

s ← LEFT(s , nchar) left portion = SUBSTRING(s,1,nchar) e.g. 'abc' = LEFT('abcd',3)

n ← LENGTH(s) bytes, e.g. 3 = LENGTH('abc') = OCTET_LENGTH() ☯ CHAR_LENGTH()

b ← s NOT LIKE spattern ESCAPE 'slash' CcIi compare, % is any string, _ any character, ☯ REGEXP

e.g. TRUE = ('springtime' LIKE '%ing%')

x ← LN(x) $\log_e(x)$ e.g. 1 = ROUND(LN(2.7182818284591), 13) = LOG(x)

s ← LOAD_FILE(serverpath)

dt ← LOCALTIMESTAMP( ) = NOW()

n ← LOCATE(sShort , sLooong , n nstart 1) 0=NF string search, similar to POSITION()

e.g. 5 = LOCATE('a', 'alpha', 2)

x ← LOG(xbase , x) e.g. 3 = ROUND(LOG(5,125), 15)

x ← LOG10(x) $\log_{10}(x)$ e.g. 1 = ROUND(LOG10(10), 15)

x ← LOG2(x) $\log_2(x)$ e.g. 5 = LOG2(32)

s ← LOWER(s) or LCASE(s) all lower case

s ← LPAD(s , ntarget , sfiller) pad left e.g. '--lefty' = LPAD('lefty', 7, '-')

s ← LTRIM(s) remove leading spaces e.g. 'lefty' = LTRIM(' lefty')

∅interval (☞ never plural)

YEAR
MONTH
DAY
HOUR
MINUTE
SECOND
MICROSECOND
YEAR_MONTH
DAY_HOUR
DAY_MINUTE
DAY_SECOND
DAY_MICROSECOND
HOUR_MINUTE
HOUR_SECOND
HOUR_MICROSECOND
MINUTE_SECOND
MINUTE_MICROSECOND
SECOND_MICROSECOND

Operators

loose ↑ binding

z ← av := z assignment e.g. @now := NOW()

b ← b || b = OR e.g. TRUE = (TRUE || FALSE) ☞ no concat

b ← b && b = AND e.g. FALSE = (TRUE && FALSE)

b ← z = z equality test e.g. 4=2+2 'a' = 'A' ☞ never: (z = NULL)

b ← z <=> z NULL-safe equality test e.g. NULL <=> NULL

b ← z <> z not equal e.g. 44 <> 55 'yin' <> 'yang'

b ← z != z not equal e.g. 44 != 55 'yin' != 'yang'

b ← z < z e.g. 44 < 55 'angst' < 'courage'

b ← z <= z e.g. 44 <= 55 'manqué' <= 'munificent'

b ← z > z e.g. 55 > 44 'time' > 'money'

b ← z >= z e.g. 55 >= 44 'persistence' >= 'passion'

n ← n | n e.g. 0xFF = (0x55 | 0xAA)

n ← n & n e.g. 0x50 = (0xF0 & 0x55)

n ← n << n shift bits left (MSB) e.g. 32 = (1 << 5)

n ← n >> n shift bits right (LSB) e.g. 1 = (32 >> 5)

n ← n - n e.g. 3 = (10 - 7)

n ← n + n e.g. 4 = (2 + 2)

n ← n * n e.g. 6 = (2 * 3)

n ← n / n e.g. 4 = (12 / 3) similar to DIV

n ← n % n e.g. 3 = (123 % 10) = MOD

n ← n ^ n bitwise XOR e.g. 0xAA = (0x55 ^ 0xFF)

n ← ~ n bitwise complement e.g. 7 = ~(-8)

n ← - n negative e.g. 2 = (-3 + 5)

b ← ! b = NOT e.g. FALSE = (! TRUE)

tight ↓ binding

Habitat Web Browser:
www.phpMyAdmin.net

Habitat Shell: (command prompt or console window)

```
mysql -u user -p password -e"SELECT * FROM d.t ;"
```

Habitat PHP:

```
$db = mysql_connect("host", "user", "password");
mysql_select_db("d", $db);
$result = mysql_query("SELECT * FROM t ;", $db);
if ($result !== FALSE) {
    while (($row = mysql_fetch_assoc($result)) != NULL) {
        foreach($row as $colname => $value) {
            echo "$colname='$value'\n";
        }
        echo "-----\n";
    }
}
mysql_close($db);
```

∅cast-type

BINARY (n) charset∅
CHAR
DATE
DATETIME
TIME
DECIMAL
SIGNED INTEGER
UNSIGNED INTEGER

syntax error: **CAST**(z AS INT)
ok: **CAST**(z AS SIGNED INT)

GROUP_CONCAT() is 2-dimensional: For each row of a collapsed group, it makes this string: CONCAT(s, s,...) Then those row strings are appended together, with SEPARATOR between.

SEPARATOR defaults to comma, or turn off with SEPARATOR ' '

∅sFormat contains time and date codes, e.g. "%Y-%m-%d %H.%i.%s"

code	example 1999-12-31 23:59:58.999	example 2000-01-02 03:04:05	equivalent to
%a	Fri	Sun	LEFT(DAYNAME(d),3)
%b	Dec	Jan	LEFT(MONTHNAME(d),3)
%c	12	1	MONTH(d)
%D	31st	2nd	
%d	31	02	LPAD(DAYOFMONTH(d),0,2)
%e	31	2	DAYOFMONTH(d)
%f	999000	000000	LPAD(MICROSECOND(t),6,0)
%H	23	03	LPAD(HOUR(t),2,0)
%h %I	11	03	
%i	59	04	LPAD(MINUTE(t),2,0)
%j 1	365	002	LPAD(DAYOFYEAR(d),3,0)
%k	23	3	HOUR(t)
%l	11	3	(HOUR(t)+11)%12+1
%M	December	January	MONTHNAME(t)
%m	12	01	LPAD(MONTH(d),2,0)
%p	PM	AM	
%r	11:59:58 PM	03:04:05 AM	DATE_FORMAT(d,'%h:%i:%s %p')
%S %s	58	05	LPAD(SECOND(t),2,0)
%T	23:59:58	03:04:05	DATE_FORMAT(d,'%H:%i:%s')
%U	52	01	LPAD(WEEK(d,0),2,0)
%u	52	01	LPAD(WEEK(d,1),2,0)
%W	Friday	Sunday	DAYNAME(d)
%w 0	5	0	DAYOFWEEK(d)-1
%X%V	199952	200001	YEARWEEK(d,2)
%x%v	199952	199952	YEARWEEK(d,3)
%Y	1999	2000	YEAR(d)
%y	99	00	RIGHT(YEAR(d),2)
%%	%	%	"%"

VisiBone **MySQL Card**

Functions M-Z

d ← MAKEDATE(*myear* , *nday*) e.g. '1999-12-31' = MAKEDATE(1999,365)

s ← MAKE_SET(*n* , *s*...) bitmask to comma-list e.g. 'b,c' = MAKE_SET(6,'a','b','c') ◐ EXPORT_SET()

t ← MAKETIME(*nhour* , *nmin* , *nsec*) e.g. '23:59:59' = MAKETIME(23,59,59)

b ← MATCH(*c*...) AGAINST(*s*
IN BOOLEAN MODE
IN NATURAL LANGUAGE MODE
WITH QUERY EXPANSION
) used on FULLTEXT column

z ← MAX(*z*) column-wise ⧉ ◐ GREATEST(z,z,...)

s ← MD5(*s*) 128-bit hash e.g. '0cc175b9c0f1b6a831c399e269772661'

n ← MICROSECOND(*t*) e.g. 123 = MICROSECOND('23:59:59.000123')

s ← MID(*s* , *±nstart* , *nchar*) = SUBSTRING() e.g. 'bcd' = MID('abcde',2,3)

z ← MIN(*z*) column-wise ⧉ ◐ LEAST(z,z,...)

n ← MINUTE(*t*) 0-59 e.g. 34 = MINUTE('12:34:56')

n ← MOD(*n* , *n*) or *n* MOD *n* or *n* % *n* e.g. 3 = MOD(43,10) 3 = 43 MOD 10

s ← MONTH(*d*) month number, 1-12 e.g. 1 = MONTH('2000-01-31')

s ← MONTHNAME(*d*) month spelled out e.g. 'December' = MONTHNAME('1999-12-31')

b ← NOT *b* e.g. FALSE = (NOT TRUE)

dt ← NOW( ) current date and time e.g. '2009-07-22 02:35:21' = NOW()

z ← NULLIF(*z* , *zNullSymbol*) ≡ IF(*z*=*zNullSymbol*, NULL, *z*) decodes *zNullSymbol* → NULL

s ← OCT(*n*) octal digit encode e.g. '400' = OCT(256)

n ← OCTET_LENGTH(*s*) string length in bytes, ≡ LENGTH(s), ◐ CHAR_LENGTH()

s ← OLD_PASSWORD(*s*) 16-hex-digit hash e.g. '60671c896665c3fa' = OLD_PASSWORD('a')

b ← *b* OR *b* e.g. TRUE = (TRUE OR FALSE)

n ← ORD(*s*) value of 1st character (works for multi-byte) e.g. 65 = ORD('A') similar to ASCII()

s ← PASSWORD(*s*) 40-digit hash e.g. '*667F407DE7C6AD07358FA38DAED7828A72014B4E' = PASSWORD('a')

n ← PERIOD_ADD(*nYYYYMM* , *nmonths*) e.g. 200101 = PERIOD_ADD(199912, 13)

n ← PERIOD_DIFF(*nYYYYMMlate* , *nYYYYMMearly*) e.g. 12 = PERIOD_DIFF(200011, 199911)

x ← PI( ) 3.141592653589793

n ← POSITION(*sShort* IN *sLooong*) search ◦◦*n* e.g. 3 = POSITION('p' IN 'alpha') similar to LOCATE()

x ← POWER(*x* , *xponent*) $x^{xponent}$ e.g. 0.125 = POWER(2,-3)

s ← QUOTE(*s*) prep (e.g. user entry) as SQL codesnippet: slash-escape special characters, single quotes around

n ← QUARTER(*d*) of the year, 1-4 e.g. 4 = QUARTER('1999-12-31')

x ← RADIANS(*xdegrees*) e.g. 0.25 = RADIANS(45)/PI()

x ← RAND(*nseed*) pseudorandom unpredictable 0 to 0.9999999

b ← *s* NOT REGEXP *sRegularExpression*∅ CcIi ≡ RLIKE e.g. TRUE = ('springtime' REGEXP 'spr.ng')

z ← RELEASE_LOCK(*sLockName*) ◐ GET_LOCK()

s ← REPEAT(*s* , *n*) e.g. '000' = REPEAT('0',3)

s ← REPLACE(*s* , *sold* , *snew*) e.g. 'AlphA' = REPLACE('alpha', 'a', 'A')

s ← REVERSE(*s*) character order e.g. 'cba' = REVERSE('abc')

s ← RIGHT(*s* , *nchar*) right portion ≡ SUBSTRING(s,-nchar) e.g. 'xyz' = RIGHT('wxyz',3)

b ← *s* NOT RLIKE *sRegularExpression*∅ CcIi ≡ REGEXP e.g. TRUE = ('wellspring' RLIKE 'spr.ng')

s ← ROUND(*x* , *ndigits*) decimal formatter e.g. "2" = ROUND(1.899) e.g. "1.90" = ROUND(1.899,2)

n ← ROW_COUNT( ) affected by recent INSERT, DELETE, UPDATE command

s ← RPAD(*s* , *ntarget* , *sfiller*) pad right e.g. 'righty--' = RPAD('righty', 8, '-')

s ← RTRIM(*s*) remove trailing spaces e.g. 'righty' = RTRIM('righty ')

s ← SCHEMA( ) = DATABASE()

n ← SECOND(*t*) 0-59 e.g. 56 = SECOND('12:34:56')

t ← SEC_TO_TIME(*nSecondsSinceMidnight*) e.g. '23:59:59' = SEC_TO_TIME(86399) ◐ TIME_TO_SEC(t)

s ← SESSION_USER( ) = USER()

s ← SHA1(*s*) 40-hex-digit RFC-3174 checksum e.g. '86f7e437faa5a7fce15d1ddcb9eaeaea377667b8' = SHA('a')

n ← SIGN(*n*) -1, 0, +1 signifying <0, =0, >0 e.g. -1 = SIGN(-9.9)

x ← SIN(*x*) sine, in radians e.g. 0.5 = ROUND(SIN(PI()/6), 15)

n ← SLEEP(*nSeconds*)

b ← *s* SOUNDS LIKE *s* e.g. TRUE = ('Joannie' SOUNDS LIKE 'Jon') similar to SOUNDEX()

s ← SOUNDEX(*s*) spelling homogenizer e.g. SOUNDEX('Smith') = SOUNDEX('Schmidtty') similar to SOUNDS LIKE

s ← SPACE(*n*) multiple spaces

x ← SQRT(*x*) e.g. 5 = SQRT(25)

x ← STD(*cx*) or STDDEV(*cx*) or STDDEV_POP(*cx*) standard population deviation of non-NULLs ⧉

x ← STDDEV_SAMP(*cx*) standard sample deviation of non-NULLs ⧉

d ← STR_TO_DATE(*s* , *sFormat*∅) e.g. '1999-12-31' = STR_TO_DATE('31-Dec-99', '%d-%b-%y') ◐ DATE_FORMAT()

n ← STRCMP(*s* , *s*) comparson: -1,0,1 means <, =, > respectively ala C e.g. -1 = STRCMP('a', 'z') CcIi since v4 tip: CAST (either AS BINARY) for CScs e.g. -1 = STRCMP(CAST('Z' AS BINARY), 'a')

d ← SUBDATE(*d* , *ndays*)

d ← SUBDATE(*d* , INTERVAL *n interval*∅) e.g. '1999-12-31 23:59:59' = SUBDATE('2000-01-01 00:00:00', INTERVAL 1 SECOND) ≡ DATE_SUB()

s ← SUBSTRING(*s* FROM *±nstart* FOR *nchar*) or SUBSTRING(*s* , *±nstart* , *nchar*) e.g. 'ro' = SUBSTRING('heroic',3,2) ≡ MID(s,n,nchar)

s ← SUBSTRING_INDEX(*s* , *smark* , *±nwords*) portion before mark e.g. 'a/b' = SUBSTRING_INDEX('a/b/c', '/', 2)

t ← SUBTIME(*t* , *t*) subtraction, times-of-day e.g. '00:00:01' = SUBTIME('12:00:00', '11:59:59')

x ← SUM(DISTINCT *cx*) total of non-NULLs ⧉

s or n ← SYSDATE( ) e.g. 20091231235959 = SYSDATE()+0 e.g. '2009-12-31 23:59:59' = CONCAT(SYSDATE()) similar to NOW()

s ← SYSTEM_USER( ) = USER()

x ← TAN(*x*) radians e.g. 1 = ROUND(TAN(PI()/4), 15)

t ← TIME(*s*) time only e.g. '23:59:30' = TIME('2000-01-01 23:59:30')

s ← TIMEDIFF(*tLate* , *tEarly*) time difference e.g. '00:00:01.000002' = TIMEDIFF('2000-01-01 00:00:01.000001', '1999-12-31 23:59:59.999999')

dt ← TIMESTAMP(*dt* , *tadd*) e.g. '2000-01-01 00:00:00.000001' = TIMESTAMP('1999-12-31 23:59:59.999999', '00:00:00.000002')

d ← TIMESTAMPADD(*interval*∅ , *n* , *d*) e.g. '2000-01-07 23:59:59' = TIMESTAMPADD(WEEK, 1, '1999-12-31 23:59:59')

n ← TIMESTAMPDIFF(*interval*∅ , *dEarly* , *dLate*) e.g. 2 = TIMESTAMPDIFF(SECOND, '1999-12-31 23:59:59', '2000-01-01 00:00:01')

s ← TIME_FORMAT(*t* , *sFormat*∅) custom time formatting e.g. '11PM' = TIME_FORMAT('23:59:59', '%h%p') ◐ DATE_FORMAT(d, sFormat) does dates and times

n ← TIME_TO_SEC(*t*) Seconds since midnight e.g. 86399 = TIME_TO_SEC('23:59:59') ◐ SEC_TO_TIME(n)

n ← TO_DAYS(*d*) count days since 0 A.D. e.g. 730485 = TO_DAYS('2000-01-01') ◐ FROM_DAYS(ndays)

s ← TRIM(*s*) remove leading and trailing spaces (not \n), ≡ LTRIM(RTRIM(*s*))

s ← TRIM(
LEADING
TRAILING
BOTH
scharacter
spattern
FROM *s*) e.g. 'ror ref' = TRIM('er' FROM 'error referer')

s ← TRUNCATE(*x* , *ndigits*) toward zero e.g. "-7.00" = TRUNCATE(-7.009, 2)

s ← UNCOMPRESS(*z*) reverses COMPRESS(s) e.g. 'a lot to say' = UNCOMPRESS(COMPRESS('a lot to say'))

n ← UNCOMPRESSED_LENGTH(*z*) room for UNCOMPRESS() e.g. 1 = UNCOMPRESSED_LENGTH(COMPRESS('a'))

s ← UNHEX(*s*) hexadecimal string decode, e.g. 'abc\n' = UNHEX('6162630A') ◐ HEX()

n ← UNIX_TIMESTAMP(*d*) seconds since 1970 ◐ FROM_UNIXTIME(nsec)

s ← UPPER(*s*) or UCASE(*s*) all upper case

s ← USER( ) for the connection

s or n ← UTC_DATE( ) e.g. 20091231 = UTC_DATE+0 e.g. '2009-12-31' = CONCAT(UTC_DATE)

s or n ← UTC_TIME( ) e.g. 235959.000000 = UTC_TIME+0 e.g. '23:59:59' = CONCAT(UTC_TIME)

s or n ← UTC_TIMESTAMP( ) e.g. 20091231235959.000000 = UTC_TIMESTAMP+0 e.g. '2009-12-31 23:59:59' = CONCAT(UTC_TIMESTAMP) similar to SYSDATE()

s ← UUID( ) e.g. '7ddc891d-cbcb-2137-a675-115105411d3a' = UUID()

x ← VALUES(*c*) insertion candidates, used in ON DUPLICATE KEY UPDATE clause

x ← VAR_POP(*cx*) or VARIANCE(*cx*) population standard variance of non-NULLs, ⧉

x ← VAR_SAMP(*cx*) sample variance of non-NULLs ⧉

s ← VERSION( ) e.g. '5.2.3-falcon-alpha' = VERSION()

n ← WEEK(*d* , *nscheme*∅) 2-digit week number. e.g. 52 = WEEK('1999-12-31', 0)

n ← WEEKDAY(*d*) 0=Mon, 6=Sun ◦ e.g. 0 = WEEKDAY('2000-01-03') ◐ DAYOFWEEK(d)

n ← WEEKOFYEAR(*d*) = WEEK(*d*, 3)

s ← WEIGHT_STRING(*s* AS CHAR / BINARY (*nPad*) LEVEL *n* / *n-n* ASC / DESC / REVERSE)

e.g. WEIGHT_STRING('aeiou')
= WEIGHT_STRING('áéíóú')

b ← *b* XOR *b* boolean XOR e.g. TRUE = (FALSE XOR TRUE)

s ← YEAR(*d*) 4-digit year e.g. 1999 = YEAR('1999-12-31')

s ← YEARWEEK(*d* , *nscheme*∅) 6-digit year/week number. e.g. '199952' = YEARWEEK('1999-12-31', 2)

♠ **NULL** is like a black hole. Almost any part of an expression being NULL makes it all NULL.
Some exceptions: *z* <=> *z* *z* IS NULL

Data Types symbols and literals

b	boolean	TRUE FALSE
n	integer	0 –999 0xFFF x'3c' b'11010'
x	floating point	3.14159 1.5e2 (150) 5.0e–1 (½)
s	string CcIi or binary blob CScs	"double" 'single' '\r\n\t\b\0\Z' '\'' "\"" '\\' 0x650D0A x'1B5B' '''' """" _utf8'unicode'
d	date or datetime	'1999–12–31'
t	time or datetime	'23:59:59'
dt	datetime	'1999–12–31 23:59:59'
z	any type, value, or expression	

∅sRegularExpression (only used in REGEXP or RLIKE comparison)

^	starts with	'enjoy' REGEXP '^en'
$	ends with	'enjoy' REGEXP 'joy$'
.	any character	'gestalt' REGEXP 'g.st'
*	any number (0 or more)	'haah' REGEXP 'ha*h'
?	optional (0 or 1)	'regardless' REGEXP '(ir)?regardless'
+	etc. (1 or more)	'toot' REGEXP 'to+t'
{ }	exactly (n)	'boot' REGEXP 'bo{2}t'
{ ,}	at least (n or more)	'shoooed' REGEXP 'sho{2,}ed'
{ , }	range (n to n)	'referrer' REGEXP 'refer{1,2}er'
(|)	either or	'image.gif' REGEXP '(jpg|gif|png)$'
[ ]	one of these characters	'mad' REGEXP 'm[aeiou]d'
[-]	a character from X to X	'm6d' REGEXP 'm[a-z0-9]d'
[^]	any character except	'max' REGEXP 'ma[^aeiou]'
[[.newline.]] [[.LF.]] [[.\n.]]	*named-characters∅* (gratuitously many)	'---\r\n---' REGEXP '[[.CR.]][[.LF.]]'
[[:alpha:]]	letter	'ABCxyz' REGEXP '^[[:alpha:]]+$'
[[:lower:]]	lowercase letter	'abcxyz' REGEXP '^[[:lower:]]+$'
[[:upper:]]	uppercase letter	'ABCXYZ' REGEXP '^[[:upper:]]+$'
[[:digit:]]	digit	'0123456789' REGEXP '^[[:digit:]]+$'
[[:xdigit:]]	hexadecimal digit	'0123ABCdef' REGEXP '^[[:xdigit:]]+$'
[[:alnum:]]	letter or digit	'123ABCxyz' REGEXP '^[[:alnum:]]+$'
[[:blank:]]	space only	' ' REGEXP '^[[:blank:]]+$'
[[:space:]]	space or \r\n\t	' \r\n\t' REGEXP '^[[:space:]]+$'
[[:cntrl:]]	x'01' thru x'1F' or x'7F'	'\b\Z' REGEXP '^[[:cntrl:]]+$'
[[:punct:]]	no control, letter, number, nor space	'!\"#$%&'() *+,-./:;?@ <=>[\\] ^_`{\|}~' REGEXP '^[[:punct:]]+$'
[[:graph:]]	no control, no space	'@#12AByz' REGEXP '^[[:graph:]]+$'
[[:print:]]	no control, yes space	' @#12AByz' REGEXP '^[[:print:]]+$'
[[:<:]]	word start	'bold begin' REGEXP '[[:<:]]b'
[[:>:]]	word end	'solid end' REGEXP 'd[[:>:]]'

Data Conversion

n← *s*+ 0	*n*← CAST(*s* AS INTEGER)
x← *s*+ 0.0	*x*← CAST(*s* AS DOUBLE)
s← CONCAT(*n*)	*s*← CAST(*n* AS CHAR)
x← *n*+ 0.0	*x*← CAST(*n* AS DOUBLE)
n← CEILING(*x*) *n*← FLOOR(*x*) *n*← ROUND(*x*) *n*← TRUNCATE(*x*)	*n*← CAST(*x* AS INTEGER)

SUBSTRING_INDEX() demystified:
Marks separate the string into 'words'.
Returns *n* words, with *n*-1 marks.
Use -*n* to count from the right.

∅nscheme

8 ways to number weeks: make 3 decisions		Decision 3: Sun-Sat	Decision 3: Mon-Sun
Decision 1:	Decision 2:	S M T **W** T F S	M T W **T** F S S
Year-Dominate: **Natural Years** fragmented weeks	Week 1 is the 1st **full week**	YEAR(d), WEEK(d,**0**)	YEAR(d), WEEK(d,**5**)
	Week 1 has the 1st **hump day**	YEAR(d), WEEK(d,**4**)	YEAR(d), WEEK(d,**1**)
Week-Dominate: **Full Weeks** fudged years	Year starts **Dec 29 - Jan 4**	YEARWEEK(d,6)	YEARWEEK(d,3)
	Year starts **Jan 1 - Jan 7**	YEARWEEK(d,2)	YEARWEEK(d,7)

♠ YEARWEEK() is broken for *nscheme* 0,1,4,5 (mimics 2,3,6,7)