Base Guide

Appendix B Comparison of HSQLDB and Firebird

Data Types and Functions

Copyright

Contributors

To this edition

Pulkit Krishna

To previous editions

Pulkit Krishna
Hazel Russman
Randolph GAMO
Robert Großkopf
Jean Hollis Weber
Jost Lange
Jochen Schiffers

Feedback

Please direct any comments or suggestions about this document to the Documentation Team's mailing list: documentation@global.libreoffice.org

Note

Everything you send to a mailing list, including your email address and any other personal information that is written in the message, is publicly archived and cannot be deleted.

Publication date and software version

Published September 2020. Based on LibreOffice 6.4.

Contents

Data types and functions in HSQLDB and Firebird

The tables in this Appendix are taken from the manuals for HSQLDB and Firebird.

- http://hsqldb.org/doc/guide/
- https://www.firebirdsql.org/en/documentation/

The information for internal HSQLDB is the same as in Appendix A of this book.

The additional internal database Firebird is classified as experimental.

The tables first provide a comparison of the functions, especially the functions that are popular in forums, such as:

- Add a certain number of days to a date (DATEADD)
- Values from multiple data lines grouped together in one data line (LIST)

are currently only available in the external Firebird database, but not in the internal version.

Built-in functions and stored procedures

The following functions are available in the built-in databases. Unfortunately one or two functions can only be used when **Run SQL command directly** is chosen. While in that mode, queries cannot be edited.

Functions that work with the graphical user interface are marked [works in the GUI]. Functions that work only in direct SQL commands are marked [does not work in the GUI].

Numeric

As we are dealing here with floating point numbers, be sure to take care with the settings of the fields in queries. Mostly the display of decimal places is restricted, which can result in unexpected behavior in some cases. For example, column 1 might show 0.00 but actually contain 0.001, and column 2, 1000. If column 3 is set to show Column 1 * Column 2, it would actually show 1.

HSQLDB		***Firebird***	
`ABS(d)`	Returns the absolute value of a number. [works in the GUI]	`ABS(d)`	
`ACOS(d)`	Returns the arccosine. [works in the GUI]	`ACOS(d)`	
`ASIN(d)`	Returns the arcsine. [works in the GUI]	`ASIN(d)`	
`ATAN(d)`	Returns the arctangent. [works in the GUI]	`ATAN(d)`	
`ATAN2(a,b)`	Returns the arctangent via coordinates. 'a' is the value of the x-axis, 'b' is the value of the y-axis. [works in the GUI]	`ATAN2(x,y)`	
`BITAND(a,b)`	Both the binary notation of 'a' and the binary notation of 'b.' Must have a '1' in the same position to yield '1' in the result. BITAND (3,5) returns 1; 0011 AND 0101 = 0001 [works in the GUI]	`BIN_AND(x,y [,z...])`	
`BITOR(a,b)`	Either the binary notation of 'a' or the binary notation of 'b.' Must have a '1' in the same position to yield '1' in the result. BITOR (3,5) returns 7; 0011 OR 0101 = 0111 [works in the GUI]	`BIN_OR(x,y [,z...])`	
		`BIN_SHL(n,exp)`	n · 2 exp [works in the GUI]
		`BIN_SHR(n,exp)`	n / 2 exp The result is shown as a rounded integer. [works in the GUI]

		`BIN_XOR(x,y [,z...])`	Either the binary notation of 'a' or the binary notation of 'b.' Must have a '1' in the same position to yield '1' in the result. `BIN_XOR(3,5)` returns 6; 0011 X OR 0101 = 011 0 [works in the GUI]
`CEILING(d)`	Specifies the smallest integer that is not less than d. [works in the GUI]	`CEIL(d)` `CEILING(d)`	
`COS(d)`	Returns the cosine. [works in the GUI]	`COS(radians)`	The radians can also be represented using the angle (here for the unit circle): radians = (2 * PI () * angles / 360)
		`COSH(d)`	
`COT(d)`	Returns the cotangent. [works in the GUI]	`COT(d)`	
`DEGREES(d)`	Converts radians to degrees. [works in the GUI]		
`EXP(d)`	Returns e^{d} (e: (2.718 ...)). [works in the GUI]	`EXP(d)`	
`FLOOR(d)`	Returns the largest integer that is not greater than d. [works in the GUI]	`FLOOR(d)`	
`LOG(d)`	Returns the natural logarithm of base 'e'. [works in the GUI]	`LN(d)`	
`LOG10(d)`	Returns the base 10 logarithm. [works in the GUI]	`LOG10(d)`	
		`LOG(base,d)`	Returns the log at any Basis again.
`MOD(a,b)`	Returns the remainder as an integer that results from dividing 2 integers. MOD (11,3) returns 2 because 3 * 3 + 2 = 11 [works in the GUI]	`MOD(a,b)`	
`PI()`	Returns π (3.1415...) [works in the GUI]	`PI()`	
`POWER(a,b)`	a^{b} , POWER (2,3) = 8, because $2^{3} = 8$ [works in the GUI]	`POWER(x,y)`	

RADIANS(d)	Converts degrees to radians. [works in the GUI]		
EDGE()	Returns a random number x greater than or equal to 0.0 and less than 1.0. [works in the GUI]	EDGE( )	
ROUND(a,b)	Rounds a to b digits after the decimal point. [works in the GUI]	ROUND(d [, places])	Rounds after the specified number of digits from the decimal point. ROUND (123.45, 1) returns 123.50 ROUND (123.45, –2) returns 100.00 [works in the GUI]
ROUNDMAGIC (d)	Solves rounding problems caused by floating point numbers. 3.11−3.1−0.01 may not be exactly 0, but is displayed as 0 in the GUI. ROUNDMAGIC turns it into an actual 0 value. [works in the GUI]		
SIGN(d)	Returns −1 if 'd' is less than 0, 0 if 'd' is equal to 0, and 1 if 'd' is greater than 0. [works in the GUI]	SIGN(d)	
SIN(A)	Returns the sine of an angle in radians. [works in the GUI]	SIN(radians)	
		Sinh(d)	
SQRT(d)	Returns the square root. [works in the GUI]	SQRT(d)	
TAN(A)	Returns the tangent of an angle in radians. [works in the GUI]	TAN(radians)	
		TANH(d)	
TRUNCATE (a,b)	Cuts 'a' to 'b' characters after the decimal point. TRUNCATE(2.37456.2) = 2.37 [works in the GUI]	TRUNC(d[,jobs])	Sets to 0 after the specified number of digits from the decimal point . TRUNC (123.45, 1) returns 123. 4 0 ROUND (123.45, –2) returns 100.00 [works in the GUI]

Text

HSQLDB		*Firebird*	
`ASCII(s)`	Returns the ASCII code of the first letter of the string. [works in the GUI]	`ASCII_VAL ('s')`	Returns the numerical value that matches the character entered. [works in the GUI]
`BIT_LENGTH (str)`	Returns the length of the text str in bits. [works in the GUI]	`BIT_LENGTH ('s')`	
`CHAR(c)`	Returns the letter that belongs to the ASCII code. It is not just about letters, but also about control characters. CHAR (13) creates a line break in a query, which is visible in multi-line fields of a form or in reports. [works in the GUI]	`ASCII_CHAR (n)`	Returns the letter that belongs to the ASCII code. It is not just about letters, but also about control characters. [works in the GUI]
`CHAR_LENGTH (str)`	Returns the length of the text in letters. [works in the GUI]	`CHAR_LENGTH ('s')` `CHARACTER_LEN GTH('s')`	
		`CHAR_TO_UUID`	Converts a 36-character Universally Unique Identifier (UUID) to a 16-byte UUID (outputs unreadable characters).
`CONCAT (STR1,STR2)`	Joins str1 + str2. [works in the GUI]		
`DIFFERENCE (s1,s2)`	Shows the sound difference between s1 and s2. Only an integer is output. 0 means the same sound. So 'for' and 'four' with 0 appear equal, shortening and seasoning is set to 1, mouth and moon back to 0. [works in the GUI]		
`HEXTORAW (s1)`	Translates hexadecimal code into other characters. [works in the GUI]		

`INSERT(s, start, len,s2)`	Returns a text string, with part of the text replaced. Beginning with ‘start,’ a length ‘len’ is cut out of the text ‘s’ and replaced by the text ‘s2.’ INSERT(‘Bundesbahn’, 3, 4, ‘mmel’) converts Bundesbahn into Bummelbahn, where the length of the inserted text can be greater than that of the deleted text without causing any problems. So INSERT(‘Bundesbahn’, 3, 5, ‘s und B’) yields 'Bus und Bahn'. [works in the GUI]	`OVERLAY ('s' PLACING 's2' FROM pos [FOR length])`	If the start position is set so that it is greater than or equal to the actual text 's', then 's2' is directly appended to 's'. OVERLAY 'Bundesbahn' PLACING 'mmel' FROM 3 FOR 4) turns 'Bundesbahn' into 'Bummelbahn', where the length of the inserted text can be longer than that of the cut text. So OVERLAY ('Bundesbahn' PLACING ' s and B ' FROM 3 FOR 5) results in 'Bus und Bahn'. [does not work in the GUI]
`LCASE(s)`	Converts the string to lowercase. [works in the GUI]		
`LEFT(s, count)`	Returns the number of characters specified with count from the beginning of the string s. [works in the GUI]	`LEFT('s', length)`	
`LENGTH(s)`	Returns the length of a string. [works in the GUI]		
`LOCATE (search,s [,start])`	Returns the first match for the term from search in the string s. The match is given numerically: (1 = left, 0 = not found). Specifying a starting point within the string is optional. [works in the GUI]	`POSITION ('s2' IN 's')` `POSITION ('s2', 's' [, S tartpos ition ])`	
`POSITION (<string expression> IN <string expression>)`	If the first string is contained in the second, the position of the first string will be returned, otherwise 0 will be displayed. This could be used instead of a search option with LIKE. [works in the GUI]		
`LTRIM(s)`	Removes leading spaces and non-printable characters from text. [works in the GUI]		

`OCTET_LENGTH (str)`	Returns the length of a text string in bytes. In principle, this corresponds to twice the length in characters. [works in the GUI]	`OCTET_LENGTH (str)`	Returns the actual number of characters, taking spaces into account. The number depends on the character set. UTF-8 needs two bytes for special characters.
`RAWTOHEX(s1)`	Converts to hexadecimal notation; reverse of HEXTORAW(). [works in the GUI]		
`REPEAT(s, count)`	Repeats the text string ‘s’ count times. [works in the GUI]		
`REPLACE(s, replace,s2)`	Replaces all existing occurrences of ‘replace’ in the string ‘s’ with the text ‘s2’. [works in the GUI]	`REPLACE('s', 's2', replacement)`	REPLACE ('Schule', 'ul', 'eib') converts 'Schule' to 'Schieibe’. If 's2' does not appear in 's', nothing will be replaced. If N2 appears in 's2' or replacement, the result is NULL.
		`LPAD('s', total length [,characters] )`	LPAD ('Hello', 8 , '+') = +++ Hello Places any characters in front of a string until the total length is reached. If the total length is less than the length of the string, the string is cut off on the right. If the third parameter remains empty, spaces are placed in front of it.
		`RPAD('s', total length [,characters] )`	RPAD ('Hello', 8 , '+') = Hello +++ Places any characters behind a string until the total length is reached. If the total length is less than the length of the string, the string is cut off on the right. If the third parameter remains empty, spaces are placed behind it.
		`REVERSE('s')`	Inverts the string completely. This can be useful, for example, if you want to sort by character endings (e.g. domain endings).
`RIGHT(s, count)`	Reverse to LEFT; returns the number of characters specified with count from the end of the string. [works in the GUI]	`RIGHT('s', length)`	

`RTRIM(s)`	Removes all spaces and non-printable characters at the end of the string. [works in the GUI]		
`SOUNDEX(s)`	Returns a code of 4 characters, which should correspond to the sound of s - fits the function DIFFERENCE (). [works in the GUI]		
`SPACE(count)`	Returns the number of spaces specified in count. [works in the GUI]		
`SUBSTR(s, start [,len])`	Abbreviation for SUBSTRING. [works in the GUI]		
`SUBSTRING (s,start [,len])`	Returns the string s from the start position. (1 = left). If the length is omitted, the entire string is returned. [works in the GUI]		
`SUBSTRING (<string expression> FROM <numeric expression> [FOR <numeric expression>] )`	Returns the part of a string from the start position specified in FROM, optionally in the length specified in FOR. If, for example, "Roberta" appears in the "Name" field, SUBSTRING ("Name" FROM 3 FOR 3) results in the substring 'bert'. [works in the GUI]	`SUBSTRING ('s' FROM start position [FOR length])`	
`TRIM ([{LEADING | TRAILING | BOTH}] FROM <string expression>)`	Special characters and spaces that cannot be printed are removed. [works in the GUI]	`TRIM ([Where 's2' FROM] 's')`	Where: BOTH | LEADING | TRAILING standard here is BOTH for both sides of 's'. s2: The character to be removed. By default, these are spaces (' '), but can also be other characters. TRIM (TRAILING '!' FROM 'Hallo!') returns 'Hallo'
`UCASE(s)`	Converts the string to uppercase. [works in the GUI]		
`LOWER(s)`	As LCASE(s) [works in the GUI]	`LOWER('s')`	

`UPPER(s)`	As UCASE(s). [works in the GUI]	`UPPER('s')`	
		`UUID_TO_CHAR ('s')`	Converts a 16-byte UUID to a 36-character ASCII format.

Date/Time

HSQLDB		***Firebird***	
		`DATEADD ( n DAY TO date )` `DATEADD (DAY, n, date)`	n is an integer and can also be negative for subtraction. YEAR | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND | MILLISECOND are to be used as terms for the time interval. Use either a date / date field, a time / time field, or a timestamp as the date term.
`DATEDIFF (string, datetime1, datetime2)`	Date difference between two dates or times. The entry in string decides in which unit the difference is shown: 'ms' = 'millisecond', 'ss' = 'second', 'mi' = 'minute', 'hh' = 'hour', 'dd' = 'day', 'mm' = 'month', 'yy' = 'year'. Both the long version and the short version of the string can be used. [works in the GUI]	`DATEDIFF ( DAY FROM date TO date )` `DATEDIFF (DAY, date, date)`	See DATEADD.
`EXTRACT ({YEAR|MONTH |DAY|HOUR| MINUTE| SECOND} FROM <date or time value>)`	Can replace many of the date and time functions. Returns the year, month, day, etc. from a date or time of day value. EXTRACT (DAY FROM "date") shows the day of the month. [works in the GUI]	`EXTRACT ({YEAR| MONTH | WEEK | DAY | WEEKDAY | YEARDAY | HOUR | MINUTE | SECOND | MILLISECOND } FROM <date or time value>)`	WEEKDAY Sunday = 0 YEARDAY January 1st = 0 WEEK 1st week: min. 4 days a year
`DAY(date)`	Returns the day of the month (1-31). [works in the GUI]		

`DAYNAME (date)`	Returns the English name of the day. [works in the GUI]		
`DAYOFMONTH (date)`	Returns the day of the month (1-31), synonym for DAY(). [works in the GUI]		
`DAYOFWEEK (date)`	Returns the day of the week as a number (1 means Sunday.) [works in the GUI]		
`DAYOFYEAR (date)`	Returns the day of the year (1-366). [works in the GUI]		
`HOUR(time)`	Returns the hour (0-23). [works in the GUI]		
`MINUTE(time)`	Returns the minute (0-59). [works in the GUI]		
`MONTH(date)`	Returns the month (1-12). [works in the GUI]		
`MONTHNAME (date)`	Returns the English name of the month. [works in the GUI]		
`QUARTER (date)`	Returns the quarter of the year (1-4). [works in the GUI]		
`SECOND(time)`	Returns the seconds of a time (0-59). [works in the GUI]		
`WEEK(date)`	Returns the week of the year (1-53). [works in the GUI]		
`YEAR(date)`	Returns the year from a date input. [works in the GUI]		

Database connection

HSQLDB		*Firebird*	
`DATABASE()`	Returns the path and name of the database belonging to this connection. [works in the GUI]		
		`CURRENT_TRANSACTION`	SELECT CURRENT_T RANSACTION FROM RDB $ DATABASE returns the unique identifier of the transaction as an integer.
		`CURRENT_CONNECTION`	SELECT CURRENT_CONNECTION FROM RDB $ DATABASE returns an integer value for the current connection.
		`CURRENT_ROLE`	SELECT CURRENT_ROLE FROM RDB $ DATABASE reflects the role of the current user. If no role is defined, the result is NONE.
		`RDB $ SET_CONTEXT ('<namespace>', '<variable name>', value | NULL )`	Namespace: USER_SESSION | USER_TRANSACTION The variable name can have a maximum of 80 characters, and the value can have a maximum of 255 characters.
`CURRENT_USER`	SQL standard function, synonym for USER(). It should be noted that there are no parentheses here. [works in the GUI]	`CURRENT_USER`	
`USER()`	Returns the username of this connection. The username is important if the database is to be converted into an external database. [SQL direct - does not work with the GUI]	`USER`	
`IDENTITY()`	Returns the last value for an autovalue field that was created in the current connection. This is used in macro programming to create a foreign key for another table from a primary key created for one table. [works in the GUI]	`GEN_ID (generator name, <step>)`	Autovalues are created with a generator. The step size should be given here as 1. In principle, any integer value is possible. `new.rec_id = gen_id (gen_recnum, 1);`

System

HSQLDB		*Firebird*		
`IFNULL (exp, value)`	If exp is NULL, value is returned, otherwise exp. Instead, COALESCE () can also be used as an extension. Exp and value must have the same data type. IFNULL is an important function when fields are linked with each other by invoice or CONCAT. The content of the result would be NULL, even if only one value is NULL. "Last name" || ',' || "first name" would result in an empty field for people who, for example, lack the entry for "first name", ie NULL. "Last Name" || IFNULL (',' || "First Name", '') would just print "Last Name" instead. [works in the GUI]			
`CASE WHEN (exp, v1, v2)`	If exp is true v1 is returned, otherwise v2. CASE WHEN can also be used instead. CASEWHEN ("a" > 10, 'goal reached', 'still practice') returns 'goal reached' if the content of the field "a" is greater than 10. [works in the GUI]	`IIF (<condition>, v1, v2)`		
`CONVERT (term, type)`	Converts term to another data type. CONVERT ("a", DECIMAL (5,2)) makes the field "a" a field with 5 digits, including 2 decimal places . If the number is too large, an error is output. [works in the GUI]			
`CAST (term AS type)`	Synonym to CONVERT () [works in the GUI]	`CAST (term AS type)`	***From***	***To***
			Numeric types	Numeric types [VAR] CHAR BLOB

			[VAR] CHAR BLOB	[VAR] CHAR BLOB Numeric types DATE TIME TIMESTAMP
			DATE TIME	[VAR] CHAR BLOB TIMESTAMP
			TIMESTAMP	[VAR] CHAR BLOB DATE TIME
`COALESCE (expr1, expr2, expr3, ...)`	If expr1 is not NULL, expr1 is displayed, otherwise expr2 is checked, then expr3, etc. All expressions must have at least a similar data type. This is the alternative representation of integers and floating point numbers, but not also of a date or time value. [works in the GUI]	`COALESCE (expr1, expr2 [, expr3 ...]`		
`NULLIF (v1, v2)`	If v1 is equal to v2, null is returned, otherwise value of v1 is returned. The data must be comparable in type. [works in the GUI]	`NULLIF (v1, v2)`		
`CASE v1 WHEN v2 THEN v3 [ELSE v4] END`	If v1 is equal to v2, v3 is executed. Otherwise v4 is executed or NULL if no ELSE is formulated. [SQL direct - does not work with the GUI]	`DECODE ( test expression , expression , result [, expression2 , Earnings2 ...] [, default expression ] )`	DECODE (UPPER ("gender"), 'M', 'male', 'F', 'female', 'unknown')	

`CASE WHEN expr1 THEN v1 [WHEN expr2 THEN v2] [ELSE v4] END`	If expr1 is true, v1 is returned [Optionally, further cases can be specified]. Otherwise v4 is reproduced or NULL if no ELSE is formulated. CASE WHEN DAYOFWEEK ("date") = 1 THEN 'Sunday' WHEN DAYOFWEEK ("date") = 2 THEN 'Monday'… END could output the day of the week via SQL, which is otherwise only available in English in the function. [works in the GUI]		DECODE (EXTRACT (WEEK DAY FROM "date"), 0 , ' Sunday ', 1 , ' Monday ', ' etc. ')
		`GEN_UUID ()`	Returns a unique ID as a 16-byte character set.
		`HASH (s)`	Returns the hash value of an arbitrarily long string. Hash values of the same character strings must be the same.
		`MAXVALUE (expr [, expr ...])`	Returns the maximum value of a list of values. Works with strings, numeric values, date or time values.
		`MINVALUE (expr [, expr ...])`	Returns the minimum value of a list of values. Works with strings, numeric values, date or time values.

Aggregate functions (especially with GROUP BY)

HSQLDB		***Firebird***	
		MAX (expr)	Maximum value of a field in a table.
		MIN (expr)	Minimum value of a field in a table.
		LIST ([ALL | DISTINCT] 's' , [' s2'])	Connects fields of several data records to one field with the corresponding connection term 's2'. [works in the GUI]

Variables (depending on the computer)

HSQLDB		*Firebird*	
`CURRENT_TIME`	Synonym for CURTIME (), SQL standard. [works in the GUI]	`CURRENT_TIME`	Time in hours, minutes and seconds. CURRENT_TIME (3) also specifies milliseconds.
`CURTIME ()`	Returns the current time. [works in the GUI]		
`CURRENT_TIMESTAMP`	Synonym for NOW (), SQL standard. [works in the GUI]	`CURRENT_TIMESTAMP [(accuracy)]`	Time specification with date and milliseconds. The accuracy can be set with [0 | 1 | 2 | 3]. The standard is 3 decimal places. SELECT CURRENT_TIMESTAMP (2) FROM RDB $ DATABASE returns the time stamp with tenths and hundredths of a second (2 decimal places).
`NOW ()`	Returns the current date and time together as a timestamp. CURRENT_TIMESTAMP can also be used instead. [works in the GUI]	`CAST ('NOW' AS DATE | TIME | TIMESTAMP)` `or DATE 'NOW'`	'NOW', written alone, is understood as a string. With the appropriate conversion, it becomes a date, a time, or a time stamp (each with 1/1000 s). The short form does not work in the GUI.
`CURRENT_DATE`	Synonym for CURDATE (), SQL standard. [works in the GUI]	`CURRENT_DATE`	
`CURDATE ()`	Returns the current date. [works in the GUI]		

Operators and statements

HSQLDB		*Firebird*	
`'str1' || 'str2' || 'str3'` or `'str1' + 'str2' + 'str3'`	Connects str1 + str2 + str3; simpler alternative to CONCAT. [works in the GUI]	`'s1' || ' s 2 ' [ ||' s3 '... ]`	Connects s1, s2 etc. to a new string [works in the GUI]
		`ALL`	

		`ANY / SOME`	
		`IN ( )`	
		`IS [NOT] DISTINCT FROM`	Result is 'yes' or 'no'.
		`NEXT VALUE FOR sequence name`	See GEN_ID (), but does not allow any steps other than 1.
		`SOME`	

Data types for the table editor

Integers

Type	*Option*	*HSQLDB*	*Firebird*	*Range*	*Storage Space*
Tiny integer	TINYINT	TINYINT		2^8 = 256 | – 128 to + 127	1 byte
Small integer	SMALLINT	SMALLINT	SMALLINT	2^{16} = 65536 | – 32768 to + 32767	2 bytes
integer	INTEGER	INTEGER | INT	INTEGER	2^{32} = 4294967296 | – 2147483648 to + 2147483647	4 bytes
BigInt	BIGINT	BIGINT	BIGINT	2^{64} (-2^{63} to $+2^{63}$)	8 bytes

Floating point numbers

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Decimal	DECIMAL	DECIMAL	DECIMAL (n, m)	Unlimited, through GUI to 50 digits, adjustable, fixed decimal places, exact accuracy	2, 4 or 8 Byte
Number	NUMERIC	NUMERIC	NUMERIC (n, m)	Unlimited, through GUI to 50 digits, adjustable, fixed decimal places, exact accuracy	2, 4 or 8 Byte

Float	FLOAT	(DOUBLE is used instead)	FLOAT	$3.4 * 10^{-38}$ to $3.4 * 10^{38}$ adjustable, not exact, 7 decimal places maximum	4 Byte
Real	REAL	REAL			
Double	DOUBLE	DOUBLE [PRECISION] | FLOAT	DOUBLE PRECISION	$1 , 7 * 10^{-308}$ to $1 , 7 * 10^{308}$ adjustable, not exact, 15 decimal places maximum	8 bytes

Text

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Text	VARCHAR	VARCHAR	VARCHAR (n)	Adjustable	Variable Firebird: 1 to 32767 bytes
Text	VARCHAR_IGNORECASE	VARCHAR_IGNORECASE		Adjustable, affect sorting, ignores differences between upper and lower case	variable
Text (fixed)	CHAR	CHAR | CHARACTER		Adjustable, rest of the actual text is filled with spaces	fixed
Memo	LONGVARCHAR	LONGVARCHAR	BLOB (BLOB SUB_TYPE 1)		variable Firebird: <32 GB

Time

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Date	DATE	DATE	DATE		4 bytes
Time	TIME	TIME	TIME	Firebird: 0:00 to 23:59,9999	4 bytes
Date/Time	TIME STAMP	TIMESTAMP | DATE TIME	TIME STAMP	Adjustable (HSQLDB: 0, 6 - 6 means with milliseconds)	8 bytes

Other

Type	*Option*	*HSQLDB*	*Firebird*	*Scope*	*Memory requirements*
Yes No	BOOLEAN	BOOLEAN | BIT			
Binary field (fixed)	BINARY	BINARY		Like integer	fixed
Binary field	VARBINARY	VARBINARY		Like integer	variable
Image	LONGVARBINARY	LONGVARBINARY	BLOB SUB_TYPE 0 BLOB SUB_TYPE binary	Like integer	variable, intended for larger images Firebird: <32 GB
OTHER	OTHER	OTHER | OBJECT			