SQL Reference

EnterpriseDB GridSQL

Version 2.0

February 2010

EnterpriseDB™

Table of Contents

1 Introduction

1.1 Overview

EnterpriseDB GridSQL's supported SQL is very similar to that of Postgres Plus Advanced Server and PostgreSQL. To be clear, using GridSQL in conjunction with the underlying database does not mean that you will have access to the full functionality of that particular database. Nonetheless, GridSQL allows for a lot of customization in the gridsql.config file to allow the DBA to define additional functions as well as map GridSQL commands to the underlying database.

This is not intended to be a comprehensive analysis of the SQL language, but is intended to provide information regarding the supported SQL and its syntax, and to point out noteworthy GridSQL implementation details to bear in mind.

2 Data Types

The following data types are supported:

Data Type	Comments
BIGINT, INT8	
BIT, VARBIT	
BLOB, BINARY, BYTE, IMAGE, RAW, LONG RAW, VARBINARY	Maps to BYTEA
BOOLEAN	(May not be supported by all underlying databases)
CHAR[ACTER] (length)	Width is fixed to length.
CIDR	IPv4 and IPv6 networks
DATE	Accepts format YYYY-MM-DD or YYYYMMDD
DATETIME	Combination of date and time
DOUBLE PRECISION, FLOAT8	8 byte floating point number
DEC[IMAL] (length, decimals)	Mapped to NUMERIC
FLOAT[(length, decimals)], SMALLFLOAT, FLOAT4	
INET	IPv4 and Ipv6 hosts and networks
INT[EGER], INT4	
INTERVAL [YEAR|MONTH|DAY|HOUR|MINUTE TO YEAR|MONTH|DAY|HOUR|MINUTE]	For time intervals
MACADDR	MAC addresses
NCHAR[ACTER] (length)	For multi-language support, like CHAR
NUMERIC[(length[,decimals])], MONEY, SMALLMONEY, YEAR	
NVARCHAR[ACTER] (length)	For multi-language support, like VARCHAR
REAL[(length, decimals)]	
SERIAL, BIGSERIAL	4 and 8 byte serial
SMALLINT, INT2, TINYINT	
TEXT, CLOB, LONG, LONG VARCHAR, LONGTEXT, LVARCHAR, MEDIUMTEXT	Acts as a CLOB
TIME	Accepts hh:mm:ss or hhmmss format
TIMESTAMP[(length)], SMALLDATETIME	Combination of date and time, with optional fractional second precision
VARCHAR[2] (length) or CHAR[ACTER] VARYING (length), TINYTEXT	Varying number of characters, with a maximum of specified length

3 Identifiers & Literals

Identifiers behave similar to Postgres Plus. They may be double-quoted, in which case they are case sensitive. If they are not quoted, they are treated as if they were typed in lower case. Please keep this in mind as you work with databases, tables and columns.

Literal string values should be enclosed by single quotes.

4 Expressions

4.1 Expressions

Logical expressions are typically found as part of the WHERE clause of various statements in determining the rows that will be effected by the statement. The operands of a logical expression to be evaluated by a logical operator may in turn be a logical expression, or an SQL expression of any of the supported data types.

Operators are discussed in the following section. In addition, available functions are covered in a later chapter.

4.2 Operators

The logical and mathematical operators that are used in expressions that GridSQL recognizes appear below, in ascending order of precedence by line.

Parentheses can also be used in expression to determine precedence.

4.2.1 Logical Operators

```
OR
AND
NOT
BETWEEN
CASE, WHEN, THEN, ELSE
=,  !=, <>, >=, >, <=, <,
IS, LIKE, ILIKE, SIMILAR TO, IN, ~
BETWEEN

~ matches
~* matches, case insensitive
!~ not matches
!~* not matches, case insensitive
```

The between operator allows for the comparison of a range of values.

Example:

```
SELECT *
FROM customer
WHERE est_income BETWEEN 100000 and 200000
```

CASE

```
CASE expression
  WHEN compare_expression THEN result_value
 [WHEN compare_expression THEN result_value ...]
```

```
 [ELSE result_value]
END

CASE
  WHEN condition THEN result_value
 [WHEN condition THEN result_value ...]
 [ELSE result_value]
END
```

There are two forms of CASE. The first form allows for comparing a single expression against a list of possible values, while the second form allows for various conditions to be evaluated.

In either case, an ELSE clause may appear to specify a default value.

Example:

```
SELECT CASE custtype
       WHEN 'P' THEN 'Platinum'
       WHEN 'G' THEN 'Gold'
       WHEN 'S' THEN 'Silver'
       ELSE 'Standard'
       END
FROM customer
WHERE state = 'CA'

SELECT CASE
       WHEN custtype = 'P' THEN 'Red Carpet'
       WHEN ordercount <= 1 THEN 'New'
       ELSE 'Standard'
       END
```

IS

Example:

```
SELECT *
FROM customer
WHERE lastname IS NULL
```

LIKE

LIKE is used to match part of a string. Wildcard characters are used as part of a string pattern.

Example: Select all customers whose names begin with the letter B.

```
SELECT *
FROM customer
WHERE lastname LIKE 'B%'
```

IN

IN is used to compare an expression to a set of expressions

Example: Select all customers in New England states

```
SELECT *
FROM customer
WHERE state IN ('NH', 'VT', 'CT', 'MA', 'ME')
```

4.2.2 Mathematical Operators

```
|/  square root
||/ cube root
! factorial
!! factorial (prefix operator)
@ absolute value
-, +
*, /, DIV, MOD
^
- (numeric negation)
```

Standard mathematical operators '-', '+', '*', and '/' are available. In addition the following are available: '^' (raise to a power), DIV (divisor—no remainder), and MOD (Modulo- remainder).

4.2.3 Bit Operators

```
& bitwise AND
| bitwise OR
# bitwise XOR
<< bitwise shift left
>> bitwise shift right
```

4.2.4 Date/Time Operators

The following describes the operators and their usage that are applicable to Date/Time expressions.

Operator	Example	Result
+	date '2001-09-28' + integer '7'	date '2001-10-05'
+	date '2001-09-28' + time '03:00'	timestamp '2001-09-28 03:00'
-	date '2001-10-01' - date '2001-09-28'	integer '3'
-	date '2001-10-01' - integer '7'	date '2001-09-24'

5 Functions

The following functions are supported as part of SQL Expressions. You may also map functions from one to another or create templates that are substituted at runtime, which is useful for user-defined functions. Please see the Administrator's Guide for more details on how to set that up in the gridsql.config file.

5.1 Aggregate Functions

Aggregate functions differ from regular functions in that regular functions are applied on a row by row basis, where as aggregate functions apply to a group of rows, either the entire set of rows being returned based on the specified conditions, or a subset of that, as determined by the GROUP BY clause.

AVG(n)

Calculates the average or mean of a numeric expression.

BIT_AND(expression)

Returns the bitwise AND of all non-null input values, or null if none.

BIT_OR(expression)

Returns the bitwise OR of all non-null input values, or null if none.

BOOL_AND(expression)

Returns true if all input values are true, otherwise false.

BOOL_OR(expression)

Returns true if at least one input value is true, otherwise false.

COUNT(*)
COUNT([DISTINCT] expr)

COUNT(*) counts the number of rows that make up a group of rows.
COUNT(DISTNCT expr) counts the number of unique appearances of the expression in the projected results.

EVERY(expression)

Returns equivalent to BOOL_AND.

MAX(expr)

Calculates the maximum value for a group of rows, whether the expression is of a numeric or string type.

MIN(expr)

Calculates the minimum value for a group of rows, whether the expression is of a numeric or string type.

SUM(n)

Calculates the SUM of an expression for a group of rows.

5.2 Statistical aggregate functions

CORR(*Y*, *X*)

Calculates the correlation-coefficient of the two numbers.

COVAR_POP(Y, X)

Calculates the population covariance of the two numbers.

COVAR_SAMP(Y, X)

Calculates the sample covariance of the two numbers.

REGR_AVGX(Y, X)

Calculates the average of the independent variable(X) of the regression line.

REGR_AVGY(Y, X)

Calculates the average of the dependent variable(Y) of the regression line.

REGR_COUNT(Y, X)

Calculates the number of non-null number pairs to fit the regression line.

REGR_INTERCEPT(Y, X)

Calculates the y-intercept of the least-squares-fit linear equation determined by the (X, Y) pairs.

REGR_R2(Y, X)

Calculates the square of the correlation coefficient.

REGR_SLOPE(Y, X)

Calculates the slope of the least-squares-fit linear equation determined by the (*X*, *Y*) pairs.

REGR_SXX(Y, X)

Calculates the sum of squares of the independent variable.

REGR_SXY(Y, X)

Calculates the sum of products of independent times dependent variable.

REGR_SYY(*Y*, *X*)

Calculates the sum of squares of the dependent variable.

STDDEV(n)

Calculates the standard deviation of a group of rows.

STDDEV_POP(expression)

Calculates the population standard deviation of the input values.

STDDEV_SAMP(expression)

Calculates the sample standard deviation of the input values.

VARIANCE(n)

Calculates the statistical variance of a group of rows.

VAR_POP(expression)

Calculates the population variance of the input values.

VAR_SAMP(expression)

Calculates the sample variance of the input values.

5.3 Mathematical Functions

ABS(n)

Calculates the absolute value of the numeric expression.

CBRT(DOUBLE PRECISION)

Returns cube root of the given number.

CEIL[ING](n)

Calculates the ceiling of the numeric expression, that is, the next highest integer. CEIL(1.2) is equal to 2, and CEIL(-1.2) is equal to –1.

EXP(DOUBLE PRECISION or NUMERIC)

Returns exponential of the given number.

FLOOR(DOUBLE PRECISION or NUMERIC)

Returns largest integer not greater than argument.

LN(n)

Calculates the natural logarithm of numeric expression n.

LOG(n1)
LOG(n1, n2)

In the first form with one parameter, it is equivalent of LN(), the natural logarithm function.
In the second form, it calculates the logarithm of n1 for base n2.

MOD(n1, n2)

Modulo. Calculates the remainder of n1 divided by n2.
This is equivalent to n1 mod n2 or n1 % n2.

PI()

Returns the value of Pi.

POWER(n1, n2)

Calculates n1 to the power of n2.

RANDOM()

Returns random value between 0.0 and 1.0.

ROUND(n)

Calculates n rounded to the nearest integer.

SETSEED(DOUBLE PRECISION)

Sets seed for subsequent random() calls (value between 0 and 1.0).

SIGN(n)

Calculates the sign of numeric expression n. The return value is –1 if n is negative, 0, if n is 0, or 1 is n is positive.

SQRT(n)

Calculates the square root of n.

TRUNC(n1)
TRUNC(n1,n2)

In the first form, with a single parameter, the decimal places are truncated.

In the second form, with n1 > 0, the number n1 is truncated to n2 decimal places. If n2 < 0 then n2 places in front of the decimal point are set to 0.

WIDTH_BUCKET(op NUMERIC, b1 NUMERIC, b2 NUMERIC, count INTEGER)

Returns the bucket to which operand would be assigned in an equidepth histogram with count buckets, in the range b1 to b2.

Trigonometric Functions

ACOS(n)

Returns the arc cosine of n.

ASIN(n)

Returns the arc sin of n.

ATAN(n)

Returns the arc tangent of n.

COS(n)

Returns the cosine of n.

COT(n)

Returns the cotangent of n.

DEGREES(n)

Returns the radians value of n converted to degrees.

RADIANS(n)

Returns the degrees value of n converted to radians.

SIN(n)

Returns the sine of n.

TAN(n)

Returns the tangent of n

5.4 Date & Time Functions

AGE (timestamp [, timestamp)

Returns the age or age difference of the timestamp.

CLOCK_TIMESTAMP()

Returns current date and time.

CURRENT_DATE

Returns the current date.

CURRENT_TIME

Returns the current time.

CURRENT_TIMESTAMP

Returns the current date and time as a timestamp (datetime) type.

DATE_PART(s, timestamp)

Returns the part element specified in the first argument from the timestamp.

Example:

```
select DATE_PART('month', '2001-12-31') from table1
```

Returns

12

DATE_TRUNC(s, timestamp)

Returns the timestamp truncated to the specified precision.

EXTRACT (field from [timestamp|interval])

Extracts the specified field from the timestamp

select EXTRACT (day from timestamp '1999-12-31')

Returns

31

ISFINITE (timestamp/interval)

Tests for finite timestamp

JUSTIFY_DAYS(interval)

Adjusts interval so 30-day time periods are represented as months.

JUSTIFY_HOURS(interval)

Adjusts interval so 24-hour time periods are represented as days.

JUSTIFY_INTERVAL(interval)

Adjusts interval using justify_days and justify_hours, with additional sign adjustments.

LAST_DAY(timestamp)

Returns the last day of the month represented by the given date.

LOCALTIME [(int)]

Returns time of day

LOCALTIMESTAMP [(int)]

Returns current timestamp

MONTHS_BETWEEN(TIMESTAMP1,TIMESTAMP2)

Returns the number of months between two dates.

NEXT_DAY(TIMESTAMP,TEXT)

Returns the first occurrence of the given weekday strictly greater than the given date.

NOW()

Returns current timestamp with time zone.

STATEMENT_TIMESTAMP()

Returns current date and time (at the start of current statement).

SYSDATE

Returns current date and time.

TIMEOFDAY()

Returns the current date and time as a timestamp.

5.5 String and Character Functions

ASCII(s)

Returns the ASCII value of string s.

BIT_LENGTH(s)

Returns the number of bits in the string.

BTRIM(string TEXT [,characters TEXT])

Removes the longest string consisting only of characters in characters (a space by default) from the start and end of string.

CHAR_LENGTH(s)
CHARACTER_LENGTH(s)

Returns the number of characters in the string.

CHR(INTEGER)

Returns the character with the given ASCII code.

CONCAT(str1, str2)

Perfoms string concatenation.

CONVERT(str using conversion_name)

Change encoding using specified conversion name.

DECODE(expr, expr1a,expr1b [,expr2a,expr2b]...[,default])

Finds first match of expr with expr1a, expr2a, etc. When match found, returns corresponding parameter pair, expr1b, expr2b, etc. If no match found, returns default. If no match found and default not specified, returns null.

DECODE(string TEXT,type TEXT)

Decode binary data from string previously encoded with encode. Parameter type is same as in encode.

ENCODE(data BYTEA,type TEXT)

Encode binary data to ASCII-only representation. Supported types are: base64, hex, escape.

INITCAP(s)

Returns the string passed in transformed such that the first letter in upper case, and the other letters are in lower case.

INSTR(*string*, *set*, [*start*, [*occurrence*]])

Finds the location of a set of characters in a string, starting at position start in the string string, and looking for the first, second, third and so on occurrences of the set.

LENGTH(s)

Returns the number of characters in string s.

LOWER(s)

Returns string s with all characters converted to lower case.

LPAD(s1, n[, s2])

Returns string based on string s1 set to length n, with any extra padding needed taken from s2. The s2 is optional and by default a space is used.

LTRIM(s1 [, s2])

Remove the longest string containing only characters from s2 (a space by default) from the start of string s1.

MD5(string)

Calculates the MD5 hash of string, returning the result in hexadecimal.

NVL(expr1, expr2)

If expr1 is not null, then nvl returns expr2.

NVL2(expr1,expr2,expr3)

Returns expr3 if expr1 is null, otherwise returns expr2.

OCTET_LENGTH(s)

Returns the number of bytes in the string

OVERLAY(s1 PLACING s2 FROM int [FOR int])

Replaces characters in string s1 with those from s2 starting at the position specified.

PG_CLIENT_ENCODING()

Current client encoding name.

POSITION(s1 IN s2)

Returns the character position in which string s1 is found in string s2.

QUOTE_IDENT(s)

Returns the given string in double quoted form. Quotes are added only if necessary (i.e., if the string contains non-identifier characters or would be case-folded).

QUOTE_LITERAL(s)

Returns the given string suitably quoted to be used as a string literal in an SQL statement string. Embedded quotes and backslashes are properly doubled.

REPEAT(string TEXT,number INTEGER)

Repeats string the specified number of times.

REPLACE(string TEXT,search_string TEXT,[replace_string]TEXT)

Replaces one value in a string with another. If replace_string is not specified, the search_string value when found, is removed.

RPAD(s1,n,s2)

Returns string based on string s set to length n, with any extra padding needed taken from s2.

RTRIM(s1)

Returns string s1 with trailing spaces removed.

SPLIT_PART(string TEXT, delimiter TEXT, field INTEGER)

Splits string on delimiter and return the given field (counting from one).

STRPOS(string,substring)

Location of specified substring in the string.

SUBSTR(s1, n1, n2)
SUBSTRING(s1 FROM n1 [FOR n2])

Returns the substring of string s1, starting at position n1, continuing for n2 characters.

TO_ASCII(string text[, encoding text])

Converts string to ASCII from another encoding (only supports conversion from LATIN1, LATIN2, LATIN9, and WIN1250 encodings).

TO_HEX(number)

Converts number to its equivalent hexadecimal representation.

TRANSLATE(string TEXT, from TEXT, to TEXT)

Any character in string that matches a character in the from set is replaced by the corresponding character in the to set.

TRIM(leading | trailing | both c from s1)

Remove the longest string containing only the characters c (a space by default) from the start, end, or both ends of the string s1.

UPPER(s)

Returns the string s converted into all upper case letters.

5.6 Other Functions

CURRENT_USER()

Returns the current user.

CAST(expr1 as datatype)

Casts expression to datatype.

DATABASE()

Returns the current database being used.

VERSION()

Returns the version of the database server being used.

USER()

Returns the current user.

6 SQL Commands

6.1 Data Definition Statements

6.1.1 CREATE TABLESPACE

Syntax:

```
CREATE TABLESPACE tablespace
    LOCATION 'path' ON NODE[S] nodenum [,nodenum...]
  [,LOCATION 'path' ON NODE[S] nodenum [,nodenum...]]
```

EnterpriseDB supports the creation of tablespaces. This gives database administrators more flexibility in specifying locations of tables on physical disks. For example, a DBA may want to put a particularly large and often-used fact table on its own set of RAID hard drives, while keeping other tables in another location. Or, the DBA may want to put indexes on a table in a different location than the table itself.

Tablespaces are not required. If not used, all tables will be created in the default tablespace on the underlying database on each node.

GridSQL allows you to conveniently assign a logical tablespace name that groups together and corresponds to tablespaces on the individual EnterpriseDB instances on the individual nodes. Ideally, nodes should be equal in processing capability and configuration, so you could specify the same file system path location for all nodes. There is also the capability to specify different locations on different nodes, if desired.

Once a tablespace has been defined, it can be referenced when creating tables or indexes.

A table can be moved to a separate tablespace by using the ALTER TABLE SET TABLESPACE command.

Note: the paths specified for the tablespaces must be empty directories that have already been created, and the EnterpriseDB postmaster process must have permission to write to this directory.

6.1.2 CREATE TABLE

Syntax:

```
CREATE [TEMP] TABLE table_name (create_definition, ...)
       [partitioning_options]
       [INHERITS (table_name)]
       [WITH XROWID | WITHOUT XROWID]
       [TABLESPACE tablespace]
```

CREATE TABLE AS:

```
CREATE [TEMP] TABLE table_name [(create_definition, ...)]
       [partitioning_options]
       [INHERITS (table_name)]
       [WITH XROWID | WITHOUT XROWID]
       [TABLESPACE tablespace]
       AS select


create_definition:
  column_name data_type [NOT NULL | NULL] [DEFAULT default_value]
            [PRIMARY KEY]
  or
  constraint_definition

constraint_definition: [CONSTRAINT constraint_name]
        PRIMARY KEY (index_column_name,...)
  or    FOREIGN KEY [CONSTRAINT symbol]  (index_column_name,...)
            [reference_definition]
  or    CHECK (expression)

data_type: (one of the data types that appear in the Data Type section

:
reference_definition:
        REFERENCES table_name [(index_column_name,...)]

partitioning_options:
    [PARTITIONING KEY column_name] ON ALL
  | [PARTITIONING KEY column_name] ON [NODE[S]] node_num[,node_num …]
  | REPLICATED
  | ROUND ROBIN ON (ALL | NODES node_num[,node_num …])
```

The CREATE TABLE statement is used to create tables in databases. It can be executed only by a user of type DBA or RESOURCE.

Table names may not start with the temporary table character sequence defined in `xdb.tempTablePrefix` in the gridsql.config file.

Tablespaces

The optional TABLESPACE clause allows a tablespace to be specified, to determine the location of the table data. See the CREATE TABLESPACE command for more details.

Internode Partitioning

The partitioning_options allow the DBA to specify a partitioning strategy, which is very important to the performance of the system. **Additional information appears in the GridSQL Planning Guide, and should be read carefully.**

Partitioning allows the DBA to distribute the data amongst multiple nodes, either based on round robin partitioning, or a partitioning column. The value in this column is used to calculate a hash value, which is mapped to a destination node. (GridSQL does not support range partitioning or round-robin, just hash partitioning.)

This distribution of data allows EnterpriseDB GridSQL to parallelize queries. In choosing a partitioning key, it is important to take into consideration what other tables this table will likely join with. That way, if these other tables are partitioned on the corresponding join column, the GridSQL Optimizer will recognize that local joins can occur without having to resort to any row shipping.

It is recommended to use all available nodes when choosing which nodes to use for the partitioned table.

Normally, one would probably want to select a column to designate as the partitioning column. Designating a partitioning column allows local joins to occur for other tables that are similarly partitioned, in cases the tables have a parent-child relationship. ROUND ROBIN partitioning may be useful, too, however, in such cases where a table typically does not join with any other tables (or just replicated ones), and there is no natural column to select as a partitioning column.

Another important partitioning_option is REPLICATED. This is appropriate for "lookup" tables, such as a state code table. Replicated tables appear on all nodes, with each node containing the exact same data. This also allows joins to occur on all nodes without having to ship any data. Depending on your database schema and queries, a DBA may even consider making other larger tables replicated, but caution is urged.

A table may also simply appear on a single node or subset of the available nodes, via the ON NODE clause, but it is recommended to use all nodes.

****Note that if no partitioning table options are specified, the table is partitioned on the on the first element of the primary key across all nodes. If none such key exists, the first column in the table is chosen as the partitioning key automatically.**

Examples:

```
CREATE TABLE part  ( p_partkey     INTEGER NOT NULL,
                     p_name        VARCHAR(55) NOT NULL,
                     p_mfgr        CHAR(25) NOT NULL,
                     p_brand       CHAR(10) NOT NULL,
                     p_type        VARCHAR(25) NOT NULL,
                     p_size        INTEGER NOT NULL,
                     p_container   CHAR(10) NOT NULL,
                     p_retailprice DECIMAL(15,2) NOT NULL,
```

```
                              p_comment     VARCHAR(23) NOT NULL )
PARTITIONING KEY p_partkey ON ALL;

CREATE TABLE nation  ( n_nationkey  INTEGER NOT NULL,
                              n_name       CHAR(25) NOT NULl,
                              n_regionkey  INTEGER NOT NULL,
                              n_comment    VARCHAR(152)) REPLICATED;
```

Constraint Exclusion Partitioning

EnterpriseDB has the ability to partition within a database instance via check constraints. This allows the DBA to create segments for a table that contain ranges of values, for example. A table named orders could be partitioned into monthly subtables, allowing queries that include a condition based on order date to scan with a smaller set of data, and therefore have a faster query time.

This is a powerful feature that should be taken advantage of. Constraint exclusion partitioning coupled with GridSQL's partitioning across multiple nodes will result in significantly faster query response times; a large table can be broken into multiple subtables, each of which is partitioned across multiple nodes in the cluster.

Note that the postgresql.conf parameter `constraint_exclusion` is off by default and must be set to on in order to take advantage of this feature.

An example appears below.

```
CREATE TABLE orders  (
 o_orderkey      INTEGER NOT NULL,
 o_custkey        INTEGER NOT NULL,
 o_orderstatus    CHAR(1) NOT NULL,
 o_totalprice     DECIMAL(15,2) NOT NULL,
 o_orderdate      DATE NOT NULL,
 o_orderpriority  CHAR(15) NOT NULL,
 o_clerk          CHAR(15) NOT NULL,
 o_shippriority   INTEGER NOT NULL,
 o_comment        VARCHAR(79) NOT NULL)
PARTITIONING KEY o_orderkey ON ALL;

CREATE TABLE orders_199201
( CHECK (o_orderdate BETWEEN '19920101'::DATE AND '19920131'::DATE) )
INHERITS (orders);

CREATE TABLE orders_199202
( CHECK (o_orderdate BETWEEN '19920201'::DATE AND '19920228'::DATE) )
INHERITS (orders);
:
```

A query like "SELECT o_orderdate, count(*) from orders where o_orderdate between '1992-01-01' and '1992-01-15' GROUP BY o_orderdate" will only use tuples found from the orders_199201 subtable (and the orders table, which should be left empty).

Note that when loading data, you must insert data into the proper subtable. Using the above example, in the current implementation, loading into orders directly will not automatically just insert the data into the correct subtable.

Another important consideration when creating subtables is to be aware that the current implementation is a bit datatype sensitive, and you might find that the underlying EnterpriseDB executor is not taking full advantage of eliminating subtables.

In the case of dates, we recommend using the above syntax to cast the date type, as in

```
CHECK (o_orderdate BETWEEN '19920101'::DATE AND '19920131'::DATE)
```

Leaving it as either just a date, or as a quoted string may cause queries in Postgres Plus Advanced Server (or PostgreSQL) to not be executed optimally. This depends on how the CHECK constraints are formulated and how the WHERE conditions are formulated. The above check constraint syntax appears to handle various date constructs (quoted, cast) in SELECT WHERE clauses properly.

Temporary Tables

Temporary tables (or temp tables) may be created using the CREATE TEMP TABLE command. A temporary table is accessible only for the session that created it. After the session ends, the temp table will be dropped automatically.

When a temporary table is created in GridSQL, one or more temp tables will also be created on the underlying database. The gridsql.config configuration related properties appear below.

gridsql.config parameter	Default	Description
xdb.tempTablePrefix	TMP	Temporary table prefix to use in underlying database. Various databases have different conventions, like "TEMP." or "#".
xdb.sqlcommand.createTempTable .start	CREATE TABLE	Start of command for CREATE TABLE statement for creating temp table on the underlying database.
xdb.sqlcommand.createTempTable .suffix	WITHOUT OIDS	Suffix to add at the end of CREATE statements for temp tables.

Note that GridSQL currently uses non-temporary tables in the temporary table implementation by default.

Column Definitions

Columns may be of any of the data types listed in the data type chapter. In addition, the user may specify whether or not nulls are allowed with NULL or NOT NULL, and include a DEFAULT clause to specify any default value for the column if none is specified as part of an insert. A column may also be designated as a primary key, or reference a foreign key.

Constraint Definitions

The CREATE TABLE statement may also include primary key or foreign key constraint definitions. Alternatively, these can also be specified as separate statements as part of ALTER TABLE. Please see ALTER TABLE for a more detailed discussion about constraints.

Xrowid

In previous versions of GridSQL, an internal row identifier was always created for each row to aid with distributed constraint checking. This is no longer required if you have a primary key or unique index on a table. If you do not use distributed constraints (e.g., a tuple in one table references a tuple in another), you do not need to worry about a unique key for each row. For compatibility, one can still create XROWID by including the WITH XROWID clause. The implicit default is WITHOUT XROWID.

6.1.3 DROP TABLE

Syntax:

```
DROP TABLE table_name [,table_name...]
```

The DROP TABLE command is used to drop tables from the database.

If any foreign key constraints exist and other tables or views reference a table being dropped, the DROP will not succeed. The foreign key constraint from the other table must be dropped first.

6.1.4 TRUNCATE TABLE

Syntax:

```
TRUNCATE table_name
```

The TRUNCATE command effectively deletes all the rows in a table. It is much faster than DELETE without any WHERE condition since it will not scan the entire table.

6.1.5 ALTER TABLE … ADD COLUMN

Syntax:

```
ALTER TABLE table_name
        ADD [COLUMN] create_definition
```

This command is used to add columns to an existing table.

6.1.6 ALTER TABLE … DROP COLUMN

Syntax:

```
ALTER TABLE table_name
        DROP COLUMN column_name
```

This command is used to drop a column from a table in the database.

6.1.7 ALTER TABLE … ADD PRIMARY KEY

Syntax:

```
ALTER TABLE table_name
      ADD PRIMARY KEY (column_name [,column_name...])
```

A primary key may be added either as part of the CREATE TABLE statement, or as part of ALTER TABLE.

EnterpriseDB GridSQL will try and have constraints be enforced locally on the nodes if possible. That means, if the first column in the primary key is also the partitioning column for a table, GridSQL will not enforce the primary key itself, and will allow the underlying database to do it, since we know that no two rows with the same key can appear on different nodes. In addition, if the table is a lookup table and replicated to all nodes, or it just appears on a single node, GridSQL will also leave it to the underlying database to be enforced.

If, however, the table is partitioned, and the first column in the primary key is not the partitioning column, GridSQL will enforce the primary key in case of INSERTs or UPDATEs. **Distributed constraints like this are expensive to enforce, so if your database does a considerable amount of such write operations, you may not want to create any distributed constraints.** If just doing periodic loads, it should not be an issue.

If a unique index already exists on the primary key columns, it will be used. Otherwise, the command is issued on the underlying nodes.

6.1.8 ALTER TABLE … ADD FOREIGN KEY

Syntax:

```
ALTER TABLE table_name
 ADD [CONSTRAINT constraint_name]
 FOREIGN KEY  (column_name [, column_name...])
 REFERENCES table_name (column_name [, column_name… ])
```

Foreign key constraints may be added by either the CREATE TABLE or ALTER TABLE command.

Foreign keys help to guarantee referential integrity in your database. The referring table's corresponding values must exist as a primary key or unique index in the referenced table.

EnterpriseDB GridSQL will try and have constraints be enforced locally on the nodes if possible. If we are referencing a replicated lookup table for example, GridSQL will leave it to the underlying nodes to enforce, and will create a foreign key constraint on each node locally.

In addition, if the table is partitioned and we are referencing another partitioned table, if the first column in the referenced and referring key in each of the tables is also the partitioning key for each, we will also rely on the underlying databases to enforce the constraint.

In other cases, local enforcement is not possible. In those cases, GridSQL will provide enforcement of foreign keys for INSERT, UPDATE and DELETE operations, as a distributed constraint.

Distributed constraints like this are expensive to enforce, so if your database does a considerable amount of such write operations, you may not want to create any distributed constraints. If just doing periodic loads, it should not be an issue.

6.1.9 ALTER TABLE … DROP CONSTRAINT

Syntax:

```
ALTER TABLE table_name
      DROP CONSTRAINT constraint_name
```

This command is used to drop an existing primary key or foreign key constraint. If the constraint had been created after an index on which it was based, only the constraint definition is dropped, and not the previously created index. If an internal index needed to be created for the constraint, it will be dropped as well, however.

If no constraint name was specified at the time of its creation, the internally generated name may be determined by issuing a SHOW CONSTRAINTS command.

6.1.10 ALTER TABLE … DROP PRIMARY KEY

Syntax:

```
ALTER TABLE table_name
     DROP PRIMARY KEY
```

The DROP PRIMARY KEY clause of the ALTER TABLE command is used to remove the primary key for a table. If the primary key constraint was created after a unique index on which it was based, the index will not be dropped. If an internally generated index was created, it will be dropped along with the primary key.

6.1.11 ALTER TABLE … ALTER COLUMN

Syntax:

```
ALTER [ COLUMN ] column TYPE type [ USING expression ]
ALTER [ COLUMN ] column SET DEFAULT expression
ALTER [ COLUMN ] column DROP DEFAULT
ALTER [ COLUMN ] column { SET | DROP } NOT NULL
```

This command is used to modify the definition of a column, changing its type, its default, and whether or not in can be null. These can also be combined into a single ALTER command.

6.1.12 ALTER TABLE … OWNER

Syntax:

```
ALTER TABLE table OWNER TO new_owner
```

This variant of ALTER TABLE is used modify the owner of the table.

6.1.13 ALTER TABLE … SET TABLESPACE

Syntax:

```
ALTER TABLE table_name SET TABLESPACE tablespace
```

This variant of ALTER TABLE is used to move a table from one tablespace to another. More information about tablespaces can be found in the topic of CREATE TABLESPACE.

6.1.14 RENAME TABLE

Syntax:

```
RENAME TABLE table_name TO new_table_name
```

This command allows the DBA or owner of a table to rename a table.

6.1.15 CREATE INDEX

Syntax:

```
CREATE [UNIQUE] INDEX index_name
   ON table_name [USING index_type]
   (column_name [[ASC|DESC|operator_class],... )
   [TABLESPACE tablespace]
   [WHERE predicate]
```

This command is used to create indexes for tables on the specified columns. The optional TABLESPACE clause allows a tablespace to be specified.

If the UNIQUE modifier is specified, the set of values in the columns must be unique. If the table is partitioned and the first column in the index is the partitioning column, GridSQL will rely on the underlying nodes to enforce the unique index. If that is not the case, GridSQL will enforce the unique index itself. **Note that enforcing a unique distributed index will significantly slow down INSERT and UPDATE operations.**

The USING clause allows for other types of indexes to be used, if the underlying database supports it.

The WHERE predicate allows for the support of partial indexes.

6.1.16 DROP INDEX

Syntax:

```
DROP INDEX index_name [ON table_name]
```

This command is used to drop indexes on tables. It can be executed by the DBA or owner of the table.

6.1.17 CLUSTER

Syntax:

```
CLUSTER index_name ON table_name
CLUSTER table_name
CLUSTER
```

Clustering a table on an existing index causes the table to be reordered based on the specified index, to allow for faster access.

Note that when new rows are added to the table they will not be in clustered order. Issuing a CLUSTER command with just the table_name parameter will recluster the table on the specified index. Issuing a CLUSTER command without any parameters will recluster all clustered tables.

As a practical matter, if your tables get quite large, this command may execute for a long time. Using CLUSTER may just be practical if you have a loading strategy where you create subtables (see CREATE TABLE) based on a window of time of data, and where this data is fairly static. Then, you can CLUSTER the particular subtable just once and leave it alone after that.

6.1.18 CREATE VIEW

Syntax:

```
CREATE VIEW view_name [(column1[,column…])] AS select_statement
```

This statement allows views to be defined from one or more tables as SELECT statements.

GridSQL views are not updatable; one cannot update data in a view, or insert data into a view.

Views are not materialized, and are evaluated as part of a query at execution time.

6.1.19 DROP VIEW

Syntax:

```
DROP VIEW view_name
```

This statement is used to drop a view.

6.2 Data Manipulation Statements

6.2.1 INSERT

Syntax:

```
INSERT [INTO] table_name [(column_name,...)]
        VALUES ((expression),...)

INSERT [INTO] table_name [(column_name,...)]
        SELECT ...
```

The INSERT command is used to insert data into the table. There are two forms of the command, one for single row inserts, and one for multiple row inserts by taking the results of a SELECT statement.

If the table is replicated, inserted rows will be inserted into the replicated table on all nodes.

If the table is on a single node, the rows will just be inserted into the table on that node.

If the table is partitioned, a hash value is calculated based on the partitioning column of the table, and the row to be inserted will be inserted at the appropriate target node.

For the INSERT to succeed, no unique index, primary key or referenced foreign key constraint violations may occur.

6.2.2 UPDATE

Syntax:

```
UPDATE table_name
    SET [table_name.]column_name1=expression1
        [, [table_name.]column_name2=expression2 ...]
    [WHERE where_definition]
```

The UPDATE command is used to update data within a table.

If a table is replicated, the table will be updated the same on all of the nodes.

If the table is on a single node, the rows will just be update in the table on that node.

For the UPDATE to succeed, no unique index, primary key or referenced foreign key constraint violations may occur.

If any distributed foreign key or primary key constraints exist, the execution of UPDATE may slow down considerably while these are checked cross-node. If the UPDATE affects the partitioning column of the table, the updated row may physically move from one node to another on the underlying database.

6.2.3 DELETE

Syntax:

```
DELETE FROM table_name
    [WHERE where_definition]
```

The DELETE command is used to delete data within a table.

If a table is replicated, the rows from the table will be deleted in the same manner on all of the nodes.

If the table is on a single node, the rows will just be deleted in the table on that node.

For the DELETE to succeed, no foreign key constraint references to this table may be violated.

If any distributed foreign key references exist, the execution of DELETE may slow down considerably while these are checked cross-node.

6.2.4 SELECT

Syntax:

```
SELECT
       [DISTINCT | UNIQUE | ALL]
    select_expression,...
    [INTO [ TEMPORARY | TEMP ] [ TABLE ] new_table]
    [FROM table_references
      [WHERE condition]
      [GROUP BY {unsigned_integer | column_name | expression}...]
      [HAVING where_definition]
[UNION select…...]
      [ORDER BY {unsigned_integer | column_name | expression} [ASC |
DESC] ,...]
      [LIMIT n] [OFFSET m]


table_references may be
:
table_reference, table_reference
table_reference [CROSS] JOIN table_reference
table_reference [INNER] JOIN table_reference join_condition
table_reference NATURAL JOIN table_reference ON (column[,column_name...])
table_reference LEFT [OUTER] JOIN table_reference join_condition


Where table_reference is defined as:

table_name [[AS] alias]
and join_condition is defined as:

ON conditional_expr | USING (column_list)

When using USING, column_list must be column names that exist in both
tables.


WHERE condition_expr:

[NOT] condition_expr [ {AND | OR} condition_expr]


condition_expr:

expression operator [ANY|ALL] expression
expression [NOT] BETWEEN expression AND expression
expression [NOT] {IN|EXISTS} ({select_statement|expression_list})
[{table|alias].]column IS [NOT] NULL
[{table|alias].]column [NOT] {LIKE] } string
```

An expression itself can be a string literal, mathematical expression, the result of a SELECT, etc. More information on expressions and operators can be found in the Expressions chapter earlier in this document.

6.2.5 EXPLAIN

Syntax:

```
EXPLAIN [VERBOSE] select_statement
```

EXPLAIN is used to view the output of the execution plan for the specified SELECT statement.

By default it will just display a summary. When the `VERBOSE` clause is included, more details including the node involvement are displayed.

6.3 Importing and Exporting via COPY

Syntax:

```
COPY tablename [ ( column [, ...] ) ]
    FROM { 'filename' | STDIN }
    [ [ WITH ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ]

COPY { tablename [ ( column [, ...] ) ] | ( query ) }
    TO { 'filename' | STDOUT }
    [ [ WITH ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ]
```

GridSQL's COPY command is similar to PostgreSQL's command.

The first form, COPY FROM, is used for importing data into a table from a file or STDIN. The second form, COPY TO, is used for exporting data from a table out to a file or STDOUT.

Note that you can also use the gs-loader utility for better error handling, which is described in the Import and Export Utilities guide.

6.4 Users and Privileges

This section contains the SQL commands for creating and manipulating users and granting and revoking privileges.

A valid user must be specified when connecting to the GridSQL database. When a database is first created with createdb, the user is required to assign a DBA username and password.

A user must be granted access to a table before being able to access it. By default, a user who creates a table has all privileges on that table.

There are 3 classes of users: DBA, RESOURCE, and STANDARD. DBA users have Database Administration privileges. RESOURCE users can create tables. STANDARD users cannot create tables, but can access the database.

6.4.1 CREATE USER

Syntax:

```
CREATE USER username PASSWORD password [user_class]

      user_class = [DBA|RESOURCE|STANDARD]
```

By default, user_class is RESOURCE.

This command may only be executed by a DBA user.

STANDARD users may only access tables for which they have been granted permission (or via PUBLIC, of course). They may also create temp tables.

RESOURCE users are the same as STANDARD, only they also have the ability to create tables.

DBA users have all of the permissions of a RESOURCE user, but can additionally perform other tasks: starting and stopping a database, and executing ANALYZE, CREATE USER, DROP USER, and ALTER USER.

6.4.2 ALTER USER

Syntax:

```
ALTER USER username PASSWORD password
ALTER USER username user_class
ALTER USER username PASSWORD password user_class
```

This command may only be executed by a DBA user, or by the same user wanting to change his or her own password.

A user cannot be demoted to the user class STANDARD if they own one or more tables. In such a case, this will fail. The DBA should instead first either drop the tables or reassign ownership via ALTER TABLE using the OWNER clause.

6.4.3 DROP USER

Syntax:

```
DROP USER username
```

This command may only be executed by a DBA user, and removes the user from the database.

6.4.4 GRANT

Syntax:

```
GRANT privilege_list ON [TABLE] table_list TO grantee_list

      privilege_list ::= ALL | privilege[,privilege…]
      privilege ::= SELECT | INSERT | UPDATE | DELETE |
                             REFERENCES | INDEX | ALTER

      table_list ::= * | table[,table…]

      grantee_list ::= grantee[,grantee]
      grantee ::= PUBLIC | user_name
```

The GRANT command is used to grant privileges on a table, and may only be executed by a DBA user, or the owner of a table.

The privileges for SELECT, INSERT, UPDATE and DELETE are for those respective commands.

The REFERENCES privilege refers to the ability to create foreign key constraints, which requires REFERENCES privilege in both the referrer and referee tables.

The INDEX privilege allows the user to create and drop indexes.

The ALTER privilege allows the user to execute the ALTER table command.

6.4.5 REVOKE

Syntax:

```
REVOKE privilege_list ON [TABLE] table_list FROM grantee_list


privilege_list ::= ALL | privilege[,privilege…]
privilege ::= SELECT | INSERT | UPDATE | DELETE |
                       REFERENCES | INDEX | ALTER

table_list ::= * | *.* | table[,table…]

grantee_list ::= grantee[,grantee]
grantee ::= PUBLIC | user_name
```

REVOKE is used to revoke privileges on a table.

Note that we only revoke what has previously been granted. If a table has been granted permission to PUBLIC, revoking for user1 will still allow user1 to access the table because the table is still accessible to the PUBLIC.

6.5 Other Commands

GridSQL also includes some administrative commands for administering the database and finding out information.

Command	Description
SHOW DATABASES	Lists all of the user-created GridSQL databases
SHOW TABLES	Lists all of the tables that exist in the current database
SHOW VIEWS	Lists all of the views that exist in the current database
DESCRIBE <table>	Lists the columns and their definitions of the specified table
DESCRIBE <view>	Displays the view definition for the specified view
SHOW INDEXES ON <table>	Lists all indexes for <table>
SHOW CONSTRAINTS ON <table>	Lists the following types of constraints for <table>: primary keys, foreign keys, foreign key references
SHOW USERS	Lists all defined users and their class
SHOW STATEMENTS	Lists all of the currently executing SQL statements
KILL <request_id>	Kills execution of the request id specified. Request ids can be obtained by executing the SHOW STATEMENTS command
ANALYZE	Updates the internal statistics in the database for creating better execution plans
VACUUM	Potentially frees up space in the database
EXECUTE DIRECT	Bypasses GridSQL and executes SQL commands on the underlying database directly.

6.5.1 SHOW DATABASES

Lists all of the user-created GridSQL databases. Columns:

Column	Description
Database	Database name
Status	The status of the database, can be either Started or Down.
Nodes	A comma separated list of all of the node id numbers that the database is on.

6.5.2 SHOW TABLES

Lists the columns and their definitions of the specified table.

Column	Description
Table_name	The name of the table
Table_partitioning_column	If the table is partitioned, the column used in partitioning, otherwise it is null for single-node or replicated tables.
Table_nodes	A comma separated list of all of the node id numbers that the table is on.

6.5.3 SHOW VIEWS

Lists the views defined in the database.

Column	Description
View_name	The name of the view

6.5.4 SHOW TABLE <table>

Lists the columns and their definitions of the specified table.

Column	Description
Column_name	The name of the column
Sql_data_type	Data type of column
Type_name	The data type name
Is_nullable	If YES, allows nulls, otherwise NO
Key	If when the table was defined, this single column was denoted as the primary key
Default	The default value of the column

6.5.5 SHOW VIEW <view>

Displays the view definition and any named view columns.

Column	Description
View_text	The text defining the view
View_column	The comma separated list of the column names defined by the creator of the view, if any.

6.5.6 SHOW INDEXES ON <table>

Lists all indexes for specified table.

Column	Description
Indexname	The name of the index
Isunique	If YES, the index only allows unique values.

Columns	A comma-spearated list of the columns that make up the index.

6.5.7 SHOW CONSTRAINTS ON <table>

Lists all constraints for specified table, including primary keys, foreign keys, foreign key references.

Column	**Description**
Constname	Constraint name
Type	The constraint type
Sourcetable	If a foreign key constraint, the referencing table
Sourcecolumns	If a foreign key constraint, the referencing columns
Desttable	If a foreign key constraint, the referenced table.
Destcolumns	If a foreign key constraint, the referenced columns.

6.5.8 SHOW USERS

Lists all defined users.

Column	**Description**
User_name	User name
User_class	Class of user

6.5.9 SHOW STATEMENTS

Shows all of the currently executing statements.

Column	Description
Request_id	The id of the request
Session_id	The session id number
Submit_time	The time at which the request was submitted
Status	P=processing, Q=queued
Statement	The actual request submitted
Nodes	For future use
Current_step	For future use

6.5.10 KILL

Syntax:

```
KILL statement_number
```

Kill is used to abort a currently executing or queued request. Kill is followed by the request id number, which can be determined from the SHOW STATEMENTS command.

6.5.11 ANALYZE

Syntax:

```
ANALYZE [ table [ (column [, ...] ) ] ]
```

The ANALYZE command is a synonym for UPDATE STATISTICS. Both commands are supported to make administration easier for DBAs accustomed to one or the other command from other DBMS systems.

ANALYZE updates internal statistics within node databases level as well as within GridSQL's metadata database, to allow better query plans to be created when processing queries.

ANALYZE by itself will analyze all tables. If a table name is included, it will analyze just the particular table. For finer granularity, individual column names may be specified as well.

See also: VACUUM.

6.5.12 VACUUM

Syntax:

```
VACUUM [ FULL | FREEZE ] [ANALYZE] [ table [ (column [, ...] ) ]]
```

The VACUUM command is used to clean up the database internally, freeing up space due to deleted and updated rows. It is also a good idea to run it periodically due to transaction id wrap-around (more information can be found in the Postgres Plus Advanced Server or PostgreSQL documentation).

It is a good idea to run the variant VACUUM ANALYZE, achieving both the results of VACUUM and ANALYZE at the same time.

VACUUM can optionally accept a single table name, as well as a subset of columns to update.

The FULL parameter does a more thorough VACUUM, which may free up more space, but it takes longer to run.

FREEZE is used to mark tuples so that they will not be subject to the transaction id wrap-around issue. It is recommended to read more about FREEZE in the Postgres Plus Advanced Server documentation before using.

See also: ANALYZE

6.5.13 EXECUTE DIRECT

Syntax:

```
EXEC[UTE] DIRECT ON [ALL | NODE[S] <node_list>]
      '<native_command>'
```

The EXECUTE DIRECT command can be used to bypass GridSQL's processing and execute a SQL statement directly on the specified nodes. This can be useful for maintenance and troubleshooting. Instead of using GridSQL's syntax, the user uses the native syntax of the underlying database.

Please exercise caution when using EXECUTE DIRECT. If you perform DDL commands like CREATE TABLE, GridSQL's metadata database will not be updated and it will have no knowledge of the table.

If the command is a query, one ResultSet is returned from each of the nodes. The results are displayed in the order in which the node numbers were listed. If ALL was specified, the results are returned in their natural order.

If using the cmdline utility, the results from each node will be displayed one after another.

Example:

```
EXECUTE DIRECT ON ALL 'select count(*) from customer';
```