Contents
Introduction ............................................................................................................................... 9
Environment .............................................................................................................................. 9
System Requirements for Jaguar Server .................................................................................. 9
System Requirements for Jaguar Client .................................................................................. 9
JaguarDB Installation ............................................................................................................ 9
Operating Systems ................................................................................................................ 10
Linux System ........................................................................................................................ 10
JaguarDB Server and Client Setup ......................................................................................... 10
Linux System ........................................................................................................................ 10
Configuration ........................................................................................................................ 11
Jaguar Server Startup ........................................................................................................... 13
Linux System ........................................................................................................................ 13
Jaguar Architecture ............................................................................................................... 14
Server Distribution ................................................................................................................ 15
Client Distribution ................................................................................................................ 15
System Configuration .......................................................................................................... 15
Mount noatime ..................................................................................................................... 16
Resource limits ..................................................................................................................... 16
  Maximum Number of Open Files ....................................................................................... 16
  Maximum Number of Threads or Processes Per User ......................................................... 16
  Maximum Kernel Threads ............................................................................................... 16
  Maximum Number of Process IDs .................................................................................... 16
Installation Verification ........................................................................................................ 17
Test Run ................................................................................................................................. 17
Test Approaches .................................................................................................................. 17
Baseline Performance .......................................................................................................... 17
Insert performance .............................................................................................................. 18
Preparation ............................................................................................................................ 18
Programming Guide ............................................................................................................ 19
Shell ....................................................................................................................................... 19
C++/C ...................................................................................................................................... 20
Jaguar Database Security ........................................................................................................ 64
Network Protection ............................................................................................................. 64
Server System Protection .................................................................................................... 64
User Privilege and File Permission ....................................................................................... 64
Database User Authentication .............................................................................................. 64
User Level Control ............................................................................................................. 65
Server Communication Control .......................................................................................... 65
Access Control List ............................................................................................................ 65
Log Monitoring .................................................................................................................. 65
Data Import and Synchronization .......................................................................................... 66
Step One: Create Tables on Jaguar ..................................................................................... 66
Step Two: Create Changelog Triggers ................................................................................ 66
Step Three: Importing Data .................................................................................................. 66
Step Four: Updating Jaguar Tables ........................................................................................ 67
Spark Data Analysis ............................................................................................................ 68
SparkR with Jaguar ............................................................................................................. 74
Spatial Data Management .................................................................................................... 76
Spatial Data Types .............................................................................................................. 76
Spatial Data Storage ............................................................................................................ 83
Creating Table Containing Spatial Data ............................................................................. 83
Inserting Spatial Data .......................................................................................................... 84
Loading Spatial Data .......................................................................................................... 85
Spatial Data Query ................................................................................................................ 85
Coordinate .......................................................................................................................... 85
Within .................................................................................................................................. 85
NearBy ............................................................................................................................... 86
Intersect .............................................................................................................................. 87
CoveredBy ........................................................................................................................... 88
Cover .................................................................................................................................. 88
Contain ............................................................................................................................... 88
Disjoint ................................................................................................................................ 88
Distance .............................................................................................................................. 88
Shapes for Location Relation .............................................................................................. 89
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>InnerRingN</td>
<td>108</td>
</tr>
<tr>
<td>PolygonN</td>
<td>108</td>
</tr>
<tr>
<td>Unique</td>
<td>108</td>
</tr>
<tr>
<td>Union</td>
<td>108</td>
</tr>
<tr>
<td>Collect</td>
<td>108</td>
</tr>
<tr>
<td>ToPolygon</td>
<td>108</td>
</tr>
<tr>
<td>Text</td>
<td>108</td>
</tr>
<tr>
<td>Difference</td>
<td>109</td>
</tr>
<tr>
<td>SymDifference</td>
<td>109</td>
</tr>
<tr>
<td>IsConvex</td>
<td>109</td>
</tr>
<tr>
<td>Interpolate</td>
<td>109</td>
</tr>
<tr>
<td>LineSubstring</td>
<td>109</td>
</tr>
<tr>
<td>LocatePoint</td>
<td>109</td>
</tr>
<tr>
<td>AddPoint</td>
<td>110</td>
</tr>
<tr>
<td>SetPoint</td>
<td>110</td>
</tr>
<tr>
<td>RemovePoint</td>
<td>110</td>
</tr>
<tr>
<td>Reverse</td>
<td>110</td>
</tr>
<tr>
<td>Scale</td>
<td>111</td>
</tr>
<tr>
<td>ScaleAt</td>
<td>111</td>
</tr>
<tr>
<td>ScaleSize</td>
<td>111</td>
</tr>
<tr>
<td>Translate</td>
<td>111</td>
</tr>
<tr>
<td>TransScale</td>
<td>112</td>
</tr>
<tr>
<td>Rotate</td>
<td>112</td>
</tr>
<tr>
<td>RotateSelf</td>
<td>112</td>
</tr>
<tr>
<td>RotateAt</td>
<td>113</td>
</tr>
<tr>
<td>Affine</td>
<td>113</td>
</tr>
<tr>
<td>Voronoi Polygons</td>
<td>113</td>
</tr>
<tr>
<td>Voronoi Lines</td>
<td>114</td>
</tr>
<tr>
<td>Delaunay Triangles</td>
<td>114</td>
</tr>
<tr>
<td>GeoJson</td>
<td>114</td>
</tr>
<tr>
<td>ToMultipoint</td>
<td>115</td>
</tr>
<tr>
<td>WKT (Well Known Text)</td>
<td>115</td>
</tr>
<tr>
<td>MinimumBoundingCircle</td>
<td>115</td>
</tr>
</tbody>
</table>

7
Introduction

The user manual is an introduction to the highly scalable and fast NoSQL database JaguarDB which is not like any other NoSQL database. JaguarDB uses a brand-new data storage model which facilitates fast point query as well as range query from big datasets. JaguarDB also provides strong support for time series data, location data, and geometric shape data.

Environment

JaguarDB consists of both Server and Client packages. You may install Server and Client packages either on the same host or on multiple hosts.

System Requirements for Jaguar Server

Hardware: CPU 8 Core, 2GHz, 32GB RAM, 1000GB HD
Software: Linux, CentOS 7, RedHat 7, Ubuntu, x86_64
File systems: ext4, XFS, ZFS

System Requirements for Jaguar Client

Hardware: CPU 4 Core, 2GHz, 16GB RAM, 512GB HD
Software: Linux CentOS 7, RedHat 7, Ubuntu, x86_64
File systems: ext4, XFS, ZFS

JaguarDB Installation

You can download all binary packages for 64 bit machines of Jaguar software and then you can install Jaguar in just one step. You may install the binaries either in the same machine or different servers. Jaguar Server will listen on TCP/IP port 8888, and the process name is “jaguar.bin”. The process can be started by any user, each having a different listening port. Jaguar provides shell scripts to start and stop the Jaguar server.
Operating Systems

Jaguar supports Linux 64 bit operating systems. There are Linux jaguar server and client libraries in the downloaded files.

Linux System

In a Linux 64 bit system (such as Centos 7 or Redhat 7), you can open a terminal (or putty, xterm, etc.) and saved the downloaded file in any directory.

JaguarDB Server and Client Setup

Linux System

If you use Linux hosts, on any server in your cluster, you can execute the following script:

```bash
$ tar -zxf jaguar-n.n.n.tar.gz
```

Then related files will be unzipped into jaguar-n.n.n directory:

```bash
$ cd jaguar-n.n
```

If you are a Linux system, which has sshd server running and ssh client, you can install jaguar on all servers in the cluster with one command

```bash
$ ./install_jaguar_database_on_all_hosts.sh -f HOSTFILE
```

You must create the HOSTFILE which must contain the hostname of all hosts in the Jaguar database cluster (including the current host).

Example of HOSTFILE:
```
nodel
node2
node3
node4
node5
```

The node names should be properly setup in system /etc/hosts file. Prior to installing Jaguar on all hosts, please make sure you have a user account on all the hosts that have the same password for the user. The HOSTFILE file is very important for setting up your
database cluster. If you have messed up in the installation process, you can execute the following command to uninstall JaguarDB on all the hosts you have prepared in the HOSTFILE:

$ ./uninstall_jaguar_database_on_all_hosts.sh

The script `install_jaguar_database_on_all_hosts.sh` also takes “-d <TARGETDIRECTORY>” command option to have Jaguar installed on a different directory other than $HOME/. If a different directory is used to install Jaguar, then the JAGUAR_HOME environment variable in the bin/jaguarstart, bin/jaguarstatus, bin/jaguarstop scripts is set to this directory. By default, the JAGUAR_HOME directory as an environment variable is set to $HOME of the user who is installing Jaguar. Once it is set, $HOME/.jaguarhome file will contain the path value of $JAGUAR_HOME.

If you have installed jaguar in the past, later when you are upgrading jaguar with new releases, the “-f HOSTFILE” will not be required.

File $HOME/.jagsetupssh is created when user’s public keys have been setup on all hosts in the cluster. If this file does not exist, “setupsshkeys -f CONFFILE” command is executed to set up the public keys. The setupsshkeys program can be executed anytime to have the public keys installed in the cluster.

Configuration

The above scripts will copy config file server.conf to $JAGUAR_HOME/conf/ and jaguar programs to $JAGUAR_HOME/bin/. You should setup your $PATH environment variable to include the directory $JAGUAR_HOME/bin.

Configuration file $JAGUAR_HOME/conf/server.conf includes the following parameters:

- PORT is the listening port number of Jaguar server.
- LISTEN_IP is the IP address that the server will use if there are multiple network interfaces on the same server host. If there is only one IP address on the server host, this parameter should be commented out and ignored.
- MEMORY_MODE specifies whether more or less memory will be used by jaguar server. If high is specified, then a little more memory is used by Jaguar. If low is given, then less memory is used by Jaguar. Default value is high.
- REPLICATION is the number of copies for each data record. For every data record, it is replicated to multiple hosts. The default value is 3. If the number of
servers is less than 3, then the replication number is equal to the number of servers. Once the system is up and running, the parameter cannot be changed. For free-trial version, this parameter is always one, i.e., data is not replicated.

- **BUFF_READER_BLOCKS**  When scanning a table, blocks of underlying file are loaded into a buffer which size is specified by this number. Higher number can boost performance during join or any scan operations. Default value is 4096.

- **JAG_LOG_LEVEL**  Lower number (min is 0) makes the server generate less logging messages. A higher number (max is 9) makes the server generate more debugging information.

- **LOCAL_BACKUP_PLAN**  Specifies when and how data is backed up. There are five types of intervals when duplicate data is saved: 15MIN, HOURLY, DAILY, WEEKLY, and MONTHLY. When data is saved, it can be either SNAPSHOT or OVERWRITE mode. SNAPSHOT means each and separate copy of data is saved with a timestamp (uses more storage space as times goes on). OVERWRITE means only one copy of data is saved. The format for LOCAL_BACKUP_PLAN is frequency:policy|frequency:policy|… where frequency is one of 15MIN, HOURLY, DAILY, WEEKLY, and MONTHLY, and policy is one of SNAPSHOT or OVERWRITE. If no value is provided for BACKUP_PLAN, then no data is saved as backup.

- **REMOTE_BACKUP_SERVER** and **REMOTE_BACKUP_INTERVAL**: These parameters specify remote backup server IP address and backup interval in seconds. The remote backup server can be a SAN storage server and must have enough capacity. If these parameters are provided, all servers in the cluster will periodically send local data to the remote server for backup.

Configuration file $JAGUAR_HOME/conf/cluster.conf is created from the HOSTFILE when executing the `install_jaguar_database_on_all_hosts.sh` script and it includes the following parameters:

- Host name of server 1
- Host name of server 2
- Host name of server 3
- Host name of server 4
- ...

For example (conf/cluster.conf):

- Host1
- Host2
- Host2
- Host4
Make sure that cluster.conf is the same on all server hosts. Please note that you cannot modify cluster.conf. If you want to add more hosts in cluster.conf, you need to use the “addcluster” command. The file server.conf should be the same on all hosts too (except that LISTEN_IP is different in case it is used).

Configuration file $JAGUAR_HOME/conf/datacenter.conf is configuration file for supporting multiple data centers and it includes the following parameters:

- IP address or host name of any server in data center 1:<PortNumber>:<Type>
- IP address or host name of any server in data center 2:<PortNumber>:<Type>
- IP address or host name of any server in data center 3:<PortNumber>:<Type>

……

Where PortNumber is the port a server listens (default is 8888), and Type is the type of server, which takes value: HOST, GATE, or PGATE.

For example (conf/datacenter.conf):

- 192.168.1.100:8888:HOST
- 221.108.3.211:8888:GATE
- 192.168.1.200:8888:PGATE

Normally, for multiple datacenters, the data flow follows the path: HOST→GATE→GATE→HOST. A GATE server protects the HOSTs and forward traffic from/to HOSTs.

**Jaguar Server Startup**

**Linux System**

On a Linux system, you may start Jaguar server on all hosts with this command:

```
$ $JAGUAR_HOME/bin/jaguarstart_on_all_hosts.sh
```

Then Jaguar server will listen on port 8888. After Server is started up, you can login using the "admin" account and “jaguarjaguarjaguar” as password. It is recommended that you change the password for admin account. You may create more Databases and User Accounts. The server log file will be in $JAGUAR_HOME/log/ directory.
You may also check the status of Jaguar on all hosts:

$ $JAGUAR_HOME/bin/jaguarstatus_on_all_hosts.sh

All Jaguar server processes can be stopped with:

$ $JAGUAR_HOME/bin/jaguarstop_on_all_hosts.sh

Jaguar Architecture

Distributed Jaguar Database system is massively scalable with flat master-master architecture. Any Jaguar client can connect to any Jaguar server. Data is updated in real time among all Jaguar servers. Jaguar system is linearly scalable. If more server hosts are deployed, cluster storage capacity and performance are increased nearly linearly.

Jaguar also supports multiple data centers. In each data center, a number of Jaguar servers can be deployed. If data in any server in a data center is updated, data in all other data centers are also automatically updated.
Server Distribution

All Jaguar server hosts are specified in the conf/cluster.conf file which contains all the IP addresses of the server hosts. Each Jaguar server maintains socket communication channels to other servers for update of schema changes of tables. Among the servers there are messages communicated about synchronization of server status information. Each server manages table data locally for data writes and reads.

Client Distribution

The Jaguar clients are directly connected to all Jaguar servers. When a client connects to a Jaguar server, it makes connections to other servers as well. When a data record is sent out by the client to Jaguar cluster, the clients calculates a hash value of the key of the data record, and it sends the insert request to the server from the hash value. Multiple records are sent to different servers simultaneously and achieve high concurrent write speed. Read requests are implemented in a similar fashion to write requests.

System Configuration
Mount noatime

File Input and Output (IO) is one of the most important performance indicator for Database. We suggest that you turn off the access time option for your file system. You may disable this in /etc/fstab as root:

```
defaults,noatime
```

Resource limits

Maximum Number of Open Files
Small number of maximum open files and number of processes and threads is a common problem in Linux systems. We suggest you increase the parameters by adding the following lines with root account to /etc/security/limits.conf

```
*       hard    nofile     1000000
*       soft    nofile     1000000
```

Maximum Number of Threads or Processes Per User
/etc/security/limits.conf:

```
*       hard    nproc     500000
*       soft    nproc     500000
```

Maximum Kernel Threads
/etc/sysctl.conf:

```
kernel.threads-max = 1000000
```

Maximum Number of Process IDs
/etc/sysctl.conf:

```
kernel.pid_max = 1000000
```

Please note the if there config files in /etc/security/limits.d/ directory. The settings in the config files under this directory will override the settings in the /etc/security/limits.conf file. So make sure you make the changes in the files under /etc/security/limits.d/. Please do not set nproc to an extremely high number or to “unlimited” which could cause users unable to login to the system.

Once you save the file, please reboot the system. The parameters will take effect after reboot. (If you wish not to reboot the system, please execute sysctl -p and close the old terminal and open a new TICK terminal)
Installation Verification

After you install the Jaguar Server, please make sure:
1) No other service or processes use port 8888
2) Directory $JAGUAR_HOME/ was created
3) Following files exist and is executable:

   $JAGUAR_HOME/bin/jaguar.bin
   $JAGUAR_HOME/bin/jaguarstart (start local jaguar)
   $JAGUAR_HOME/bin/jaguarstop (stop local jaguar)

Test Run

Test Approaches

There are two ways to interact with Jaguar servers:

1. Interaction between Jaguar Client and Server side:

   Test by running the $JAGUAR_HOME/bin/jag client, typing SQL-like commands. Then the Server will respond when receiving queries.

2. APIs calls

   Test by writing programs which calls Jaguar APIs to perform related data Select, Insert operations. Client API bindings include Java, C++, Python, PHP, Go, NodeJS, and Ruby languages.

Baseline Performance

Following benchmarks can demonstrate the performance advantages of Jaguar:

- Data Load/Insert
- Date Query
- Indexing performance
- Memory usage
Insert performance

There are 2 ways to test Insert performance:
1) Perform batch load where data records are read from a file;
2) Insert single record from Client side;

Please make sure Server, Client are correctly installed and configured.

Preparation:

1. Create the user when logging in as ‘admin’ using the jag command:
   $ jaguar> createuser test;

2. Create the table

   $ jag -u test -p test -d test -h 127.0.0.1:8888 
   (or jag -u test -d test -h 127.0.0.1:8888 which will prompt for password)
   jaguar> create table test ( key: uid char(16), value: addr char(16) );

(A) Batch Load
You can test Jaguar Server by loading 3 million records. And the sample 3 million
records can be generated by program genrand which comes with Client bin directory.

   $ genrand 3000000 71 (Do this outside of the jag program)
   $ mv genrand.out /tmp/3M.txt

Then in jag client (any user can run jag client side) use the following command to
load 3 million records to test table: (in jag program)
   jaguar> load /tmp/3M.txt into test;

Expected behavior: After about 2 minutes, jag will tell how long it takes to load data in
milliseconds.

You can also write all your SQL commands in a file and feed the file to jag program:

   $ vi mycommands.sql

   create table test1 ( key: uid char(16), value: addr char(16) );
   load file /tmp/1000.txt into test1;
quit;

Then execute shell command:

```
$ jag -u test -p test -d test -h 127.0.0.1:8888 < mycommands.sql
```

(B) Client single record Insert

`jbench` program in Client package will help insert, modify and query records on Server. The following command will generate 10000 numbers randomly and insert record to `jbench` table in the Server.

```
$ jbench -u test -p test -r "10000:0:0:0" -k 0 | tee -a test.log
```

In “10000:0:0:0”, the first number “10000” is the number of times for Insertion, the second “0” for Update, the third “0” for select, and the last number for delete.

The database used in jbench is ‘test’, and the table in the jbench program is ‘jbench’. The table ‘jbench’ has a key named ‘uid’ of 16 bytes, a value named ‘addr’ of 32 bytes.

Programming Guide

There are example programs in $JAGUAR_HOME/doc directory which can be used as a guide.

Shell

```
$ $JAGUAR_HOME/bin/jag -u USERNAME -p PASSWORD -h HOST:PORT -d DATABASE
```

Example: $ $JAGUAR_HOME/bin/jag -u test -p mysecret -h hostip:8888 -d mydb

```
jaguar> insert into t1 (uid, addr) values ('Joe', '123Street, CA');
jaguar> select * from t1;
antb> select * from t1 where uid like 'jen%' and phone like '925%';
jaguar> select * from mytable where uid in ('tom', 'jack');
```
C++/C

```c
#include <JaguarAPI.h>
JaguarAPI jdb;
jdb.connect( host, port, username, passwd, dbname );
jdb.execute( "insert into mytable ( uid, addr, age ) values ( 'Joe', '123 A Street, CA', 35 ) " );
jdb.query( "select * from t1;" );
while ( jdb.reply() ) {
    jdb.printRow();
    char *p = jdb.getValue( "uid" );
    printf("uid=%s\n", p );
    free( p );
    p = jdb.jsonString();
    printf("JSON string=\[%s\n", p );
}
```

Java

```java
System.loadLibrary("JaguarClient");
Jaguar jdb = new Jaguar();
boolean rc = jdb.connect( "127.0.0.1", 8888, "test", "test", "test" );
jdb.execute("insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' );");
jdb.query("select * from tab;" );
while( jdb.reply() ) {
    val = jdb.getValue("uid");
    m1 = jdb.getValue("m1");
    System.out.println( "uid: " + val + " m1: " + m1 );
}
```
jdb.close();

Java JDBC

DataSource ds = new JaguarDataSource("127.0.0.1", 8888, "mydb");
Connection connection = ds.getConnection("testuser", "testpasswd");
Statement statement = connection.createStatement();
statement.executeUpdate("insert into tab (uid, addr) values ('Jill', '333 B Ave, CA');");
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery("select * from tab;");
String val;
String m1;
while(rs.next()) {
    val = rs.getString("uid");
    m1 = rs.getString("m1");
    System.out.println( "uid: " + val + " m1: " + m1 );
}
rs.close();
statement.close();

Scala

import com.jaguar.jdbc.internal.jaguar_
System.loadLibrary("JaguarClient");
val jdb = new Jaguar();
val rc = jdb.connect( "127.0.0.1", 8888, "test", "test", "test", ",", 0 );

jdb.execute("insert into tab (uid, addr) values ('Jill', '333 B Ave, CA');")

jdb.query("select * from tab;")

while( jdb.reply() ) {
    val u = jdb.getValue("uid");
    val m1 = jdb.getValue("m1");
    println("uid: "+u+" m1: "+m1);
}

jdb.close();

Python

Make sure the environment variable PYTHONPATH points to the directory where jaguarpy.so library file exists:

    export PYTHONPATH=$JAGUAR_HOME/lib
    export LD_LIBRARY_PATH=$JAGUAR_HOME/lib

Then in your python program:

import jaguarpy

jdb = jaguarpy.Jaguar()

rc = jdb.connect( "192.168.2.200", 8888, "userid", "password", "dbname" )

jdb.execute("insert into tab (uid, addr) values ('Jill', '333 B Ave, CA');")

jdb.query("select * from t1;")

while jdb.reply():
    jag.printRow();
    u = jdb.getValue( "uid" );
    a = jdb.getValue("addr");
    ds = 'uid is ' + repr(u) + ' addr is ' + repr(a)
    print( ds );
To program PHP with Jaguar, please use root or sudo and copy conf/jaguar.ini to /etc/php.d directory (Centos), or to /etc/php5/mods-available (Ubuntu), or to other PHP required directory. Also copy lib/jaguarphp.so and lib/libJaguarClient.so to proper directory.

`$ php -i | grep additional` Gives directory where jaguar.ini should be copied to.

`$ php -i |grep extension_dir` Gives directory where jaguarphp.so should be copied to.

**Example:**

Centos   # cp -f conf/jaguar.ini /etc/php.d/
Centos   # cp -f lib/jaguarphp.so /usr/lib64/php/modules
Ubuntu   # cp -f conf/jaguar.ini /etc/php5/mods-available/
Ubuntu   # cp -f lib/jaguarphp.so /usr/lib/php5/20121212
# cp -f lib/libJaguarClient.so /usr/lib

```php
<?php

$jdb = new Jaguar();

$jdb->connect( "192.168.2.200", 8888, "userid", "password", "dbname" );

$jdb->execute( "insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' );" );

$jdb->query( "select * from t1;" );

While ( $jdb->reply() ){
    $jag->printRow();
    $u = $jdb->getValue( "uid" );
    $a = $jdb->getValue("addr");
    print( "uid=$u addr=$a\n" );
```
Ruby

To use Jaguar Ruby client API, make sure lib/jaguarrb.so exist in the $JAGUAR_HOME/lib directory and export the RUBYLIB environment variable:

```bash
export RUBYLIB=$JAGUAR_HOME/lib
```

```ruby
require 'jaguarrb'

jdb = Jaguar.new()

jdb.connect("192.168.2.200", 8888, "userid", "password", "test");
jdb.execute("insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' );");
jdb.query("select * from t1;" );
While jdb.reply()
    jdb.printRow();
    u = jdb.getValue("uid");
    a = jdb.getValue("addr");
    print("uid=#{u} addr=#{a}\n");
end
```

NodeJS

To use Ruby client API, make sure lib/jaguarnode.node exist in the $JAGUAR_HOME/lib directory:

```javascript
var homedir = process.env.JAGUAR_HOME;
var libname = homedir + "/jaguar/lib/jaguarnode";
```
const jaguarnode = require('libname')
var jaguar = new jaguarnode.JagAPI();
jdb.connect("127.0.0.1", 8888, "admin", "jaguarjaguarjaguar", "test");
jdb.execute("insert into tab (uid, addr) values ( 'Jill', '333 B Ave, CA' );");
jdb.query("select * from t1;")
while (jdb.reply()) {
    jdb.printRow();
    var u = jdb.getValue("uid");
    var a = jdb.getValue("addr");
    process.stdout.write("uid: " + uid + " addr: " + addr + "\n");
}

Go

To use Go language client API, please go to the src/golang directory in github.com/datajaguar/jaguardb and read the readme file. The jaguargo directory contains the interface files between C++ and golang. The script compile.sh is a program to compile jaguargo package imported by the main.go program as an example.

main.go file:

package main

import(
    "jaguargo/jaguargo"
    "strconv"
    "flag"
    "fmt"
    "time"
    "os"
)

func main() {
    flag.Parse()
    ports := flag.Arg(0)
jdb := jaguargo.New()

fmt.Printf("connecting to jaguardb 127.0.0.1 port=%s\n", ports )

port, err := strconv.ParseUint(ports, 0, 64 )

if err != nil {
    fmt.Printf("error\n")
    os.Exit(1)
}

jdb.Connect("127.0.0.1", uint(port), "admin", "jaguarjaguarjaguar", "test" )
jdb.Execute("drop table if exists gotab123")
jdb.Execute("create table gotab123 (key: uid char(32), value: addr char(128))")
jdb.Execute("insert into gotab123 values ('id1001', '123 W. Washington Blvd')")
jdb.Execute("insert into gotab123 values ('id1002', '225 E. Sunshine St')")

jdb.Query("show databases")

fmt.Printf("List of databases:\n")
for {
    rc := jdb.Reply()
    if rc > 0 {
        jdb.PrintRow()
    } else {
        break
    }
}

jdb.Query("show tables")

fmt.Printf("List of tables:\n")
for {
    rc := jdb.Reply()
    if rc > 0 {
        jdb.PrintRow()
    } else {
        break
    }
}
time.Sleep(1*time.Second)
jdb.Query("select * from gotab123")
fmt.Printf("Data in table gotab123:
")
for {
    rc := jdb.Reply()
    if rc > 0 {
        jdb.PrintRow()
    } else {
        break
    }
}
jdb.Close()

To execute the main.go program:

```
export LD_LIBRARY_PATH=$HOME/jaguar/lib:/home/jaguar/jaguar/lib:/usr/local/gcc-7.1.0/lib64
unset GOPATH
export GO111MODULE=on
go run main.go 8888
```

Query with Index

Suppose table mytable contains key: uid and value: v1, v2, v3. If you need to query data in
mytable according to a non-key column (or several columns), then you can create an index on
the column(s) and query mytable by using the index. For example:

```
create index mytable_idx23 on mytable ( v2, v3 );
```

Shell
jaguar> select * from mytable_idx23 where v2='somevalue' and v3='somevalue';

C++/C

jdb.query( “select * from mytable_idx23 where v2 >= 'somevalue' ; ” );
while ( jdb.reply( ) ) {
    jdb.printRow();
    char *p = jdb.getValue( “uid” );
    printf(“uid=\%s\\n”, p ); free( p );
    p = jdb.jsonString();
    printf(“JSON string=[\%s]\\n”, p );
}

Java JDBC

Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery(“select * from mytable_idx23 where v2 >= ‘myvalue’;”);
String val;
String m1;
while(rs.next()) {
    val = rs.getString("uid");
    m1 = rs.getString("m1");
    System.out.println( "uid: " + val + " m1: " + m1 );
}
rs.close();
statement.close();
Client API Reference

The following methods are supported for C++, Java, Scala, Python, PHP, Ruby and other API calls:

1. bool connect( String host, int port, String uid, String pass, String db )
   Connects to server. Returns True for success, False for failure.

2. bool execute( String command )
   Execute a data modification command string such as create table, drop table, insert commands. The command must not be “select” query string where multiple rows are expected.

3. bool query( String query )
   Select data from server. This is where the “select” statement should be used.

4. bool reply()
   Return result data to the client. With a while loop around this call, you can obtain the selected result data row by row. When there is no more data, the reply() call returns false.

5. void printRow()
   Print out row data on standard output.

6. void close()
   Closes the connection to server and frees up relevant memory resources.

7. String getDatabase()
   Returns the database name of current client session.

8. bool hasError()
   Tests if there is error from the query command.

9. String error()
   If hasError() is true, this call returns the error string.

10. String getNthValue( int col )
    Returns the value of the N-th column (starting from 1) in the current row (inside the reply while loop).

11. String getValue( String columnName )
    Returns the value of a column of name columnName. For example, is “uid” is the column name of a table, then getValue(“uid”) returns the value of uid column in the current row.
12. String getMessage()
   Return the output data in the current row. Sometimes the current row data does not have
   any column structure, with only a raw message. For example, “desc table;” will output a
   text message describing the format of a table. In such cases, getMessage() should be
called.

13. long getLong(String columnName)
    If the column is known to be long integer type, this method returns the long integer
    value.

14. double getFloat(String columnName)
    If the column is known to be numerical double type, this method returns the double
    value.

15. int getColumnCount()
    Returns the number of columns in current row.

16. String getColumnName(int col)
    Returns the string name of the col-th column (starting from 1).

17. int getColumnType(int col)
    Return the numeric column type of col-th column (per JDBC definition)

18. String getColumnTypeName(int col)
    Return the string column type of col-th column (per JDBC definition)

19. String getTableName(int col)
    Return the table name of col-th column

Operation

Jaguar Admin

Jaguar Admin package is included for administration and operation of Jaguar cluster. A Jaguar
administrator can open a web browser and monitor the operation of Jaguar cluster. The
administrator just needs to install the jaguar-admin-nnn.tar.gz package and:
1) Execute `install_jaguar_admin_on_all_hosts.sh` so that Jaguar admin is installed on all hosts.

2) Execute `$JAGUAR_HOME/jagadmin/bin/jaguarallstart` so that jagadmin runs on all hosts.

3) Copy index.cgi to `/var/www/cgi-bin/` and html/* to `/var/www/html/`.

4) Start Apache web server, `# systemctl start httpd`.

5) Open the URL `http://<IP>` in a browser, where IP is the IP address of the server host.

After entering admin user name and password, a web interface is displayed, showing cluster status, data request statistics, resource usage of all servers, and menus for creating new databases and user accounts.

Remote Backup

Setup on the first Jaguar server host

The data stored in all the servers of Jaguar cluster can be backed up in a remote server (such as a high-capacity storage server) frequently. In `conf/server.conf`, you can assign values to the `REMOTE_BACKUP_SERVER` and `REMOTE_BACKUP_INTERVAL` parameters to enable this feature. `REMOTE_BACKUP_SERVER` should point to the IP address of the remote backup server, and `REMOTE_BACKUP_INTERVAL` is the time interval (in seconds) specifying how often the backup is performed. This configuration needs to be completed on the first Jaguar server host only. It is not necessary to set it up on other Jaguar server hosts. The file `conf/syncpass.txt` (chmod 600 as user jaguar) should just contain the password (single word) of jaguar user to connect to the remote backup server host. This password can be different from jaguar’s system account password.

File `conf/syncpass.txt`:

`mypassword888`

Setup on the remote backup server host

On the remote backup server, rsync daemon should be setup correctly. The config file `/etc/rsyncd.conf` should have the following information:

```
uid = jaguar
```
gid = jaguar
use chroot = yes
max connections = 1000
pid file = /var/run/rsyncd.pid
log file = /var/log/rsyncd.log
exclude = lost+found/
transfer logging = yes
timeout = 900
ignore nonreadable = yes
read only = false
write only = false

[jaguardata]
path = /home/jaguar/jaguarbackup
comment = Jaguar repository (requires authentication)
auth users = jaguar
strict modes = false
secrets file = /etc/rsyncd.secrets

where “[jaguardata]” must be kept exactly as it is shown above but “path = /home/jaguar/jaguarbackup” can be any directory you wish to use. This directory should be owned by ‘jaguar’ user.

# mkdir -p /home/jaguar/jaguarbackup
# chown -R jaguar.jaguar /home/jaguar/jaguarbackup

In the file /etc/rsyncd.secrets (chmod 600 as root), you should have the password for jaguar user (username:password format):

    jaguar:mypassword888

In /etc/rsyncd.secrets there can be many lines specifying username and password for rsync daemon server to authenticate. If someuid takes the value of jaguar, i.e., rsync daemon will be
run as user jaguar. The password ‘mypassword888’ is just an example. You should use your own password and is the same as the one in Jaguar server’s conf/syncpass.txt.

Restart the rsync daemon server on this host after you have made the changes. On CentOS/Redhat systems, the command to restart rsync daemon server as root is:

```
# systemctl restart rsyncd
```

Restart Crashed Nodes

If a node in the Jaguar cluster had accident, please follow the rules below:

1. In case of hardware failure, such as disk crash, battery end of life, mother board damage, please fix the hardware or replace the node with a new node. Make sure its IP address and configuration are the same as the old node. Then execute this command to restart the server: `$JAGUAR_HOME/bin/jaguarstart_dorecover`

2. If Jaguar server program accidently exited or has been stopped, just run the `jaguarstart_dorecover` to restart jaguar server program, which will recover data automatically.

3. If it is just a network problem, simply fix the network problem and no other operation is needed. Once the network is restored, Jaguar will recover by itself automatically.

Data Types

Currently Jaguar supports these data types:

1. Character string
   char(length) – it is a fixed length character string in key columns. It is a variable length string in the value columns. It is same as varchar(length).

2. Boolean
   boolean – one byte integer field containing a single digit

3. Integer
   int or integer - integer between -9999999999 and +9999999999
4. Big integer
   bigint -- integer between -9999999999999999999 and +9999999999999999999

5. Small integer
   smallint -- integer between -99999 and +99999

6. Tiny integer
   tinyint -- integer between -999 and +999

7. Medium integer
   mediumint -- integer between -9999999 and +9999999

8. Float
   float(L,d) -- a float decimal number with total of L digits and d number of digits after the
decimal point.

9. Double
   double(L,d) -- similar to float except in internal representation and calculation, it is
treated as double precision float number.
   
   numeric(L,d) -- same as double(L,d)
   decimal(L,d) -- same as double(L,d)

10. LongDouble
    longdouble(L,d) -- similar to double except in internal representation and calculation, it is
treated as a long double column.

11. Date Time
    datet ime -- a 16 digits time value in terms of microseconds. When a time data is loaded or
    inserted into Jaguar, the following format must be used:

        YYYY-MM-DD hh:mm:ss[.uuuuuu][+HH:MM]
        YYYY-MM-DD hh:mm:ss[.uuuuuu][-HH:MM]

    Where YYYY is the 4-digit year symbol, such as 2025
    MM is the month (1-12), such as 10
    DD is the date in 1-31, such 04
    hh:mm:ss is hour:minute:seconds such as 02:23:21
    .uuuuuu is optional fractional seconds (or microseconds)

    +HH:MM and -HH:MM are optional time zone difference from GMT standard time.
    If no time zone information is given, then the input string is taken as local time of the
    client. If the client just wants to insert local time string, then the time zone string is not
necessary. The time zone info is only used when the client wants to insert time string of another time zone.

Example:

From California, USA:

```sql
insert into sa (uid, sttime) values (12, '2014-11-23 16:32:21 -08:00');
insert into sd (devid, ltime) values (1232, '2015-10-23 13:32:21.234019');
select * from sales where sdate > '2014-12-10 03:12:23';
```

12. DateTimeNano

datetimenano – similar to datetime except this has granularity of nanoseconds. When a datetimenano column is loaded or inserted into Jaguar, the following format must be used:

```sql
YYYY-MM-DD hh:mm:ss[.nnnnnnnnnn] [+HH:MM]
YYYY-MM-DD hh:mm:ss[.nnnnnnnnnn] [-HH:MM]
```

13. DateTimeSec

datetimesec – similar to datetime except this has granularity of seconds. When a datetimesec column is loaded or inserted into Jaguar, the following format must be used:

```sql
YYYY-MM-DD hh:mm:ss [+HH:MM]
YYYY-MM-DD hh:mm:ss [-HH:MM]
```

insert into sd (devid, dtcol) values (1232, '2022-10-23 13:32:21');
select * from sales where dtcol > '2014-12-10 03:12:23';

14. DateTimeMill

datetimemill – similar to datetime except this has granularity of milliseconds. When a datetimemill column is loaded or inserted into Jaguar, the following format must be used:

```sql
YYYY-MM-DD hh:mm:ss[.nnn] [+HH:MM]
YYYY-MM-DD hh:mm:ss[.nnn] [-HH:MM]
```

insert into sd (devid, dtcol) values (1232, '2022-10-23 13:32:21.123');
select * from sales where dtcol > '2014-12-10 03:12:23.123';

15. Date
The date type has input and output format: YYYY-MM-DD
Example:

```sql
insert into sales (uid, datecol) values (1234, '2015-03-12');
select * from sales where datecol='2015-12-23';
```

16. Time

time – type for tracking hour, minute, second, and microsecond. The input format of time column is:

```
HH:MM:SS.[uuuuuu] -- where uuuuuu represents microseconds
```

17. TimeNano

timenano -- type for tracking hour, minute, second, and nanosecond. The input format of timenano column is:

```
HH:MM:SS.[nnnnnnnn] -- where nnnnnnnn represents nanoseconds
```

18. Timestamp

timestamp -- same as datetimestamp with precision of microseconds. Both can take input in 'yyyy-mm-dd HH:MM:SS.123456 HH:MM' format or simply a number representing microseconds since the epoch (1 January 1970 00:00:00), for example 1482000884000000. The difference between timestamp and datetime is that during insertion of this column data, if no value is provided by the user, the current local time with precision of microseconds is automatically generated and inserted into the table for the timestamp column.

19. TimestampNano

timestampnano -- same as datetimestampnano with precision of nanoseconds. Both can take input in 'yyyy-mm-dd HH:MM:SS.123456789 HH:MM' format. The difference is that during insertion of this column data, if no value is provided by the user, the current local time with precision of nanoseconds is automatically generated and inserted into the table for the timestampnano column.

20. TimestampSec

timestampsec -- same as datetimestampsec with precision of seconds. Both can take input in 'yyyy-mm-dd HH:MM:SS HH:MM' format. The difference is that during insertion of this column data, if no value is provided by the user, the current local time with precision of seconds is automatically generated and inserted into the table for the timestampsec column.
21. **TimestampMill**
   Timestampmill -- same as datetimestampmill with precision of milliseconds. Both can take input in 'yyyy-mm-dd HH:MM:SS[.nnn] HH:MM' format. The difference is that during insertion of this column data, if no value is provided by the user, the current local time with precision of milliseconds is automatically generated and inserted into the table for the timestampmill column.

22. **Real**
   Real – the data type is same as a double(38,8), double of total 38 digits and 8 digits after the decimal point.

23. **Text**
   Text -- is the same as char(1024)

24. **TinyText**
   TinyText -- is the same as char(256)

25. **MediumText**
   MediumText -- is the same as char(2048)

26. **LongText**
   LongText -- is the same as char(10240)

27. **Blob**
   Blob -- is the same as char(1024)

28. **TinyBlob**
   TinyBlob -- is the same as char(256)

29. **MediumBlob**
   MediumBlob -- is the same as char(2048)

30. **LongBlob**
   LongBlob -- is the same as char(10240)

31. **String**
   String -- is the same as char(64)

32. **Varchar**
   Varchar(N) -- is same as char(N)

33. **Bit**
   Bit -- is one byte column, with value of 1 or 0
34. Eum

COLUMN enum (‘val1’, ‘val2’, ‘val3’, … ) – A column can take certain values only

35. File

File – is used to store any file (photo, image, audio, video, doc, ppt, pdf, etc). It has no limit in size.

36. Spatial Data Types

Please refer the Spatial Data Management chapter in this manual.

37. Range

Range(datetime) -- a range of datetime with begin and end data
Format: “YYYY-MM-DD HH:MM:SS[.nnnnnn]”

Range(datetime) -- a range of datetime with begin and end data
Format: “YYYY-MM-DD HH:MM:SS”

Range(datetime) -- a range of datetime with begin and end data
Format: “YYYY-MM-DD HH:MM:SS[.nnn]”

Range(datetime) -- a range of datetime with begin and end data
Format: “YYYY-MM-DD HH:MM:SS[.nnnnnnn]”

Range(date) -- a range of date with begin and end data
Format: “YYYY-MM-DD”, example “2018-09-12”

Range(time) -- a range of time with begin and end

Range(double) -- a range of double numbers with begin and end

Range(longdouble) -- a range of longdouble numbers with begin and end

Range(float) -- a range of float numbers with begin and end

Range(bigint) -- a range of bigint numbers with begin and end

Range(int) -- a range of integer numbers with begin and end

Range(medint) -- a range of medint numbers with begin and end

Range(smallint) -- a range of smallint numbers with begin and end

Range(tinyint) -- a range of tinyint numbers with begin and end

Examples:

Insert into a values ( 123, range('2010-01-01 00:00:00', '2020-12-31 23:59:59') );

Insert into b values ( 123, range('2010-01-01', '2020-12-31') );

Insert into c values ( 123, range('00:00:00', '13:00:00') );
Insert into d values ('abc', range(100, 500));
Insert into e values ('abc', range(2.34, 100.918));
Select * from e where within(r, range(10, 500));
Select a, b, r:begin, r:end from e where r:begin >= 300 and r:end <= 1000;

Functions within(), contain(), cover(), converedby(), intersect(), disjoint() are supported for the range data types.

Default Values

Any column in a table can take a one-byte default value. The timestamp and datetime columns can have default value of CURRENT_TIMESTAMP. Also upon update of a row, its timestamp column can be automatically updated by entering "ON UPDATE CURRENT_TIMESTAMP". For example:

Create table tab123 {
  Key: id uuid,
  Value:
    a int default '0',
    b char(16) default 'b',
    bitv bit default b'1',
    bitm bit default b'0',
    tm1 timestamp DEFAULT CURRENT_TIMESTAMP,
    tm2 timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    tm3 timestamp ON UPDATE CURRENT_TIMESTAMP,
    speed enum ('low', 'med', 'high') default 'med'
};

Data Type Mapping Between Jaguar and Java
The following table specifies the mapping between Jaguar data types and Java data types:

<table>
<thead>
<tr>
<th>Jaguar Type</th>
<th>Format</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>bool</td>
<td>boolean</td>
</tr>
<tr>
<td>char</td>
<td>char(length)</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>smallint</td>
<td>smallint</td>
<td>int</td>
</tr>
<tr>
<td>tinyint</td>
<td>tinyint</td>
<td>int</td>
</tr>
<tr>
<td>mediumint</td>
<td>mediumint</td>
<td>int</td>
</tr>
<tr>
<td>bigint</td>
<td>bigint</td>
<td>long</td>
</tr>
<tr>
<td>double</td>
<td>double(m,n)</td>
<td>double</td>
</tr>
<tr>
<td>float</td>
<td>float(m,n)</td>
<td>float</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp (in milliseconds)</td>
<td>java.util.Date</td>
</tr>
<tr>
<td>datetime</td>
<td>datetime (in milliseconds)</td>
<td>java.util.Date</td>
</tr>
<tr>
<td>datetimenano</td>
<td>datetimenano (in microseconds)</td>
<td>java.sql.Timestamp</td>
</tr>
<tr>
<td>time</td>
<td>time</td>
<td>SimpleDateFormat</td>
</tr>
<tr>
<td>timenano</td>
<td>timenano</td>
<td>SimpleDateFormat</td>
</tr>
</tbody>
</table>

**Jaguar Functions**

Jaguar supports a number of built-in functions, which can be operated on one or multiple columns from select statements or join statements. The following description illustrates how to use Jaguar functions.

**Syntax:**

```
SELECT FUNC( EXPR(COL) ) from TABLE [WHERE] [LIMIT];
```

**EXPR(COL):**

**Numeric columns:** columns with arithmetic operation

```
+  addition
-  subtraction
*  multiplication
/  division
%  modulo
```
\(^\text{power (exponential)}\)

String columns: Concatenation of columns or string constants

- string column + string column
- string column + string constant
- string constant + string column
- string constant + string constant
- string constant: ‘some string’

\text{FUNC(EXPR(COL))}: \\
\text{min(Expr(COL))} -- minimum value of column expression \\
\text{max(Expr(COL))} -- maximum value of column expression \\
\text{avg(Expr(COL))} -- average value of column expression \\
\text{sum(Expr(COL))} -- sum of column expression \\
\text{count(1)} -- count number of rows \\
\text{stddev(Expr(COL))} -- standard deviation of column expression \\
\text{first(Expr(COL))} -- first value of column expression \\
\text{last(Expr(COL))} -- last value of column expression \\
\text{abs(Expr(COL))} -- absolute value of column expression \\
\text{acos(Expr(COL))} -- arc cosine function of column expression \\
\text{asin(Expr(COL))} -- arc sine function of column expression \\
\text{ceil(Expr(COL))} -- smallest integral value not less than column expression \\
\text{cos(Expr(COL))} -- cosine value of column expression \\
\text{cot(Expr(COL))} -- inverse of tangent value of column expression \\
\text{floor(Expr(COL))} -- largest integral value not greater than column expression \\
\text{log2(Expr(COL))} -- base-2 logarithmic function of column expression \\
\text{log10(Expr(COL))} -- base-10 logarithmic function of column expression \\
\text{log(Expr(COL))} -- natural logarithmic function of column expression \\
\text{ln(Expr(COL))} -- natural logarithmic function of column expression \\
\text{mod(Expr(COL),Expr(COL))} -- modulo value of first over second column expression
pow(EXPR(COL), EXPR(COL)) -- power function of first to second column expression
radians(EXPR(COL)) -- convert degrees to radians
degrees(EXPR(COL)) -- convert radians to degrees
sin(EXPR(COL)) -- sine function of column expression
sqrt(EXPR(COL)) -- square root function of column expression
tan(EXPR(COL)) -- tangent function of column expression
substr(EXPR(COL), start, length) -- sub string of column expression
substr(EXPR(COL), start, length, 'UTF8') -- sub string of UTF8 encoded string
substring(EXPR(COL), start, length) same as substr() above
diff(EXPR(COL), EXPR(COL)) Levenshtein distance (edit distance) between two strings
diff(COL, 'stringconstant') Levenshtein distance (edit distance) between two strings
upper(EXPR(COL)) -- upper case string of column expression
lower(EXPR(COL)) -- lower case string of column expression
ltrim(EXPR(COL)) -- remove leading white spaces of string column expression
rtrim(EXPR(COL)) -- remove trailing white spaces of string column expression
trim(EXPR(COL)) -- remove leading and trailing white spaces of string column expression
length(EXPR(COL)) -- length of string column expression
second(TIMECOL) -- value of second in a datetime column
minute(TIMECOL) -- value of minute in a datetime column
hour(TIMECOL) -- value of hour in a datetime column
date(TIMECOL) -- value of date in a datetime column
month(TIMECOL) -- value of month in a datetime column
year(TIMECOL) -- value of year in a datetime column
datediff(type, BEGIN_TIMECOL, END_TIMECOL) -- difference of two datetime columns
   type: second (difference in seconds)
   type: minute (difference in minutes)
   type: hour (difference in hours)
   type: day (difference in days)
type: month  (difference in months)

type: year   (difference in years)

The result is the \text{END\_TIMECOL} – BEGIN\_TIMECOL.

dayofmonth(\ \text{TIMECOL}) -- the day of the month in a datetime column (1-31)
dayofweek(\ \text{TIMECOL}) -- the day of the week in a datetime column (0-6)
dayofyear(\ \text{TIMECOL}) -- day of the year in a datetime column (1-366)
curdate() -- current date (yyyy-mm-dd) in client’s local time

curtime() -- current time (hh:mm:ss) in client’s local time

now() -- current date and time (yyyy-dd-dd hh:mm:ss) in client’s local time

Example:

\begin{verbatim}
select sum(amt) as amt_sum from sales limit 3;
select cos(lat), sin(lon) from map limit 3;
select tan(lat+sin(lon)) as t, cot(lat^2+lon^2) as c from map;
select uid, uid+addr, length(uid+addr) from user limit 3;
select price/2.0 + 1.25 as newp, lead*1.25 - 0.3 as newd from plan;
\end{verbatim}

Jaguar SQL Statements

The commands and SQL statements supported by Jaguar can be shown by the help command in the interactive shell jql program:

\begin{verbatim}
jaguar:test> help;
You can enter the following commands (ending with semicolon):

help admin       (how to for admin account)
help use         (how to use databases)
help desc        (how to describe tables)
help show        (how to show tables)
help create      (how to create tables)
\end{verbatim}
help insert (how to insert data)
help load (how to load data from client host)
help copy (how to copy data from server host)
help select (how to select data)
help update (how to update data)
help delete (how to delete data)
help drop (how to drop a table completely)
help alter (how to alter a table and rename a key column)
help truncate (how to truncate a table)
help func (how to call functions in select)
help spool (how to write output data to a file)
help password (how to change the password of current user)

Please note that in a query command, keywords (such as create, table, select, where ) can only be separated by blank spaces, \t', \r', \n' characters. Other non-printable characters are not allowed and may cause parsing errors when executing the query.

Admin commands

These commands should be executed by the “admin” account to manage user accounts and databases.

createdb DBNAME;
dropdb DBNAME;
createuser UID; -- The command will prompt for password of the new user
createuser UID:PASSWORD; -- New account is created with clear-text password
dropuser UID;
showusers;

Example:

createdb mydb;
Grant command

After admin has created a user account, permissions of the user should be granted by the admin. The grant command can be used in the following manner:

jaguar:test> help grant;
jaguar> grant all on all to user;
jaguar> grant PERM1, PERM2, ... PERM on DB.TAB.COL to user;
jaguar> grant PERM on DB.TAB.* to user;
jaguar> grant PERM on DB.TAB to user;
jaguar> grant PERM on DB to user;
jaguar> grant PERM on all to user;
jaguar> grant select on DB.TAB.COL to user [where TAB.COL1 > NNN and TAB.COL2 < MMM;

Only the admin account can issue this command.

PERM is one of: all/create/insert/select/update/delete/alter/truncate

All means all permissions.

The where statement, if provided, will be used to filter rows in select and join.

Example:

jaguar> grant all on all to user123;
jaguar> grant all on mydb.tab123 to user123;
jaguar> grant select on mydb.tab123.* to user123;
jaguar> grant select on mydb.tab123.col2 to user3 where tab123.col4>100;
jaguar> grant delete, update on mydb.tab123.col4 to user1;
Revoke command

Permissions of a user can be revoked with the following commands:

jaguar:test> help revoke;

jaguar> revoke al on all from user;

jaguar> revoke PERM1, PERM2, ... PERM on DB.TAB.COL from user;

jaguar> revoke PERM on DB.TAB.* from user;

jaguar> revoke PERM on DB.TAB from user;

jaguar> revoke PERM on DB from user;

jaguar> revoke PERM on all from user;

Only the admin account can issue this command.

PERM is one of: all/create/insert/select/update/delete/alter/truncate

All means all permissions. The permission to be revoked must exist already.

Example:

jaguar> revoke all on all from user123;

jaguar> revoke all on mydb.tab123 from user123;

jaguar> revoke select on mydb.tab123.* from user123;

jaguar> revoke select, update on mydb.tab123.col2 from user3;

jaguar> revoke update, delete on mydb.tab123.col4 from user1;

Use command

Change the database in a client session:

    use DATABASE;

Example:
```
use myuserdb;

Describe command

Describe a table or index:

desc TABLE;
desc INDEX;

Example:
desc usertab;
desc addr_index;

Show command

Show information about database system:
show databases  (display all databases in the system)
show tables     (display all tables in current database)
show indexes    (display all indexes in current database)
show currentdb  (display current database being used)
show task       (display all active tasks)
show indexes from/in table (display all indexes of a table in currently selected database)
show server version (display Jaguar server version)
show client version (display Jaguar client version)
show user       (display username of current session)

Example:
  show databases;
  show tables;
  show indexes from mytable;
  show indexes;
```
Create command

Commands for creating table and index:

```
create table TABLE ( key: KEY TYPE(size), ..., value: VALUE TYPE(size), ... );

create table TABLE ( COL1 TYPE(size), COL2 TYPE(srid:ID,metrics:M), ... );

create index INDEXNAEME on TABLE(COL1, COL2, ...[, value: COL,COL]);
create index INDEXNAEME on TABLE(key: COL1, COL2, ...[, value: COL,COL]);

Example:
create table user ( key: name char(32),
                   value: age int, address char(128), rdate date );
create table sales ( key: name char(32), stime datetime,
                     value: author char(32) );
create table sales ( key asc: id bigint, stime datetime,
                     value: member char(32) );
create table users ( name char(32), age int, address char(128) );

create index addr_index on user(address);
create index addr_index on user( address, value: zipcode );
create index addr_index on user( key: address, value: zipcode, city );
create table media ( key: uid int, value: audio file, video file );
create table ls( key: id int, value: s linestring(srid:4326,metrics:5) );
create table cirm ( key: a int, value: c circle(metrics:2), d int );
create table if mmetrics ( key: a int, value: pt point(srid:4326,
                          metrics:3), b int );
```
In creating table, if there is no key specified, an UUID column is automatically added as a unique key to the table with the name "_id".

When creating an index, you can add several value columns which will not be used as a key column in the index. It is purely for easy data access without going to the main table for reading the value columns. Creating an index from a table which has data already may take some time to complete, but it will be faster than the initial time spent on inserting the table data.

If the column type is geometric or geologic, the default value of SRID is zero. Metrics specifies the number of metrics associated with each point of raster shape or with a vector shape. There can be multiple metrics corresponding to each location point in a raster geometry.

**Insert SQL Commands**

```
insert into TABLE (col1, col2, col3, ...) values ( 'val1', 'val2', intval, ... );
insert into TABLE values ( k1, k2, 'val1', 'val2', intval, ... );
insert into TAB1 select TAB2.col1, TAB2.col2, ... from TAB2 [WHERE] [LIMIT];
insert into TAB1 (TAB1.col1, TAB1.col2, ...) select TAB2.col1, TAB2.col2, ...
from TAB2 [WHERE] [LIMIT];
insert into TABLE values ( k1, k2, load_file(/path/to/file), 'vvv4');
```

Example:

```
insert into user ( fname, lname ) values ( 'John S.', 'Doe' );
insert into user ( fname, lname, age ) values ( 'David', 'Doe', 30 );
insert into user ( fname, lname, age, addr ) values ( 'Larry', 'Lee', 40,
'123 North Ave., CA' );
insert into member ( name, datecol ) values ( 'LarryK', '2015-03-21' );
insert into member ( name, timecol ) values ( 'DennyC', '2015-12-23
12:32:30.022012 +08:30' );
insert into t1 select * from t2 where t2.key1=1000;
insert into t1 (t1.k1, t1.k2, t1.c2) select t2.k1, t2.c2, t2.c4 from t2
where t2.k1=1000;
insert into t1 values ( k1, load_file(/tmp/a.jpg), 'vvv4' );
insert into t1 values ( k1, load_file($HOME/img/a.jpg), 'vvv4' );
insert into media values ( 100, '/tmp/myaudio.aud', '/tmp/muvideo.mov' );
insert into mmetrics values ( 110, point( 0.2 0.3 'A' 'B' 'C' ), 234 );
```
insert into mmetrics values ( 220, point( 0.2 0.3 '10' '00' '30' ), 43 );
insert into cirm values ( 100, circle( 22 33 100 'PARK' 'tower' ), 209 );
insert into cirm values ( 200, circle( 24 31 100 'SCHL' 'bank' ), 258 );

Metrics data must be enclosed with single quotes or double quotes. The number of metrics data must be less than or equal to the number of metrics defined when creating the table with the columns that have metrics fields. Each metric is a string that has a length less than or equal to 8 characters or numbers. Metrics normally are used as tags or attributes describing a location or a shape. During table creation, the number of metrics can be as large as desired.

If there is a column of type uuid, then its value must not be listed in the insert command. The database server will automatically generate a unique string (40 bytes) for the column and insert the whole record into database.

For datetime, datetimenano, timestamp fields, if no time zone information is provided, the input is considered from the client's local time zone.

If there is load_file( FILEPATH) command in a column value, the file data is encoded with base64 encoding and loaded into the corresponding column. The file indicated by FILEPATH can contain client's environment variable, e.g., $HOME/img/a.jpg where $HOME will be expanded to full path of a user's home directory.

Load command

Loading data in a file into database:

load /path/input.txt into TABLE [columns terminated by C] [lines terminated by N] [quote terminated by Q];
(Instructions inside [ ] are optional. /path/input.txt is located on client host.)
Default values:
  columns terminated by: ','
  lines terminated by: '\n'
  column values can be quoted by singe quote (') character.

Example:
  load /tmp/input.txt into user columns terminated by '>';
The above load command can load a CSV file into the database.

**Select SQL command**

Data can be selected in various ways from the database:

```
(_SELECT) from TABLE [WHERE] [GROUP BY] [ORDER BY] [LIMIT] [TIMEOUT N];
(_SELECT) from INDEX [WHERE] [GROUP BY] [ORDER BY] [LIMIT] [TIMEOUT N];
select * from TABLE;
select * from TABLE limit N;
select * from TABLE limit S,N;
select COL1, COL2, ... from TABLE;
select COL1, COL2, ... from TABLE limit N;
select COL1, COL2, ... from TABLE limit N;
select COL1, COL2, ... from TABLE where KEY='...' or KEY='...' and ( ... ) ;
select COL1, COL2, ... from TABLE where (. . .) or ( ... and ... );
select COL1, COL2, ... from TABLE where KEY='abc' and KEY2 like 'abc%';
selct * from TABLE where KEY='abd' and KEY2 match 'abc.*z';
select COL1, COL2, ... from TABLE where KEY='key88' and VAL1 between m and n;
select COL1 as col1label, COL2 col2label, ... from TABLE;
select count(*) from TABLE;
select min(COL1), avg(COL3) as avg3, sum(COL4) sum4, count(1) from TABLE;
select FUNC(COL1) fc1, FUNC(COL2) as x from TABLE timeout 100;
```

If no limit is provided, a default of 10000 records is displayed on screen. Timeout parameter is optional and specifies the number of seconds for the server to timeout for the select operation. If no timeout is provided, server processing will timeout in 60 seconds for the select. Please be warned that in certain select operations, it will take a long time if your dataset is large. It is prudent to first try a timeout and check how long a query can take.

The match operation takes a regular expression enclosed with two single quotes. If the selected column matches to the regular expression, then the test evaluates to true.

Examples:
select * from user;
select * from user limit 100;
select * from user limit 1000,100;
select fname, lname, address from user;
select fname, lname, address, age from user limit 10;
select fname, lname, address from user where fname='Sam' and lname='Walter';
select * from user where fname='Sam' and lname='Walter';
select * from user where fname='Sam' or (fname='Ted' and lname like 'Ben%');
select * from user where fname >= 'Sam';
select * from user where fname >= 'Sam' and fname < 'Zack';
select * from user where fname >= 'Sam' and lname like 'Ben%';
select * from user where fname = 'Sam' or lname like 'Ben%';
select * from sales where stime between '2014-12-01 00:00:00 -08:00' and '2014-12-31 23:59:59 -08:00';
select avg(amt) as amt_avg from sales;
select sum(amt) amt_sum from sales where ...
select sum(amt) amt_sum from sales group by key1, key2 limit 10;
select sum(amt+fee) as amt_sum from sales timeout 300;
select * from metrics1;
select a, pt:x, pt:y, pt:m1, pt:m2, pt:m3 from metrics;
select * from cirm;
select c:x, c:y, c:m1, c:m2 from cirm;

The c:m1 and c:m2 are the values of the metrics associated with the column that has metrics.

Getfile command

If there are some columns that are of type ‘file’, you can download the file data and save into client side local file. The syntax is:

Getfile COL into localfilapath from table where key=...;
Where localfilepath is file on client’s computer and please make sure the “where” condition must specify the unique row that contain the file.

You can also download multiple files from server into files on client side.

\[
\text{Getfile COL1 into fpath2, COL2 into fpath2 from table where key=...;}
\]

You can get the file size, file time, md5sum of files in a table:

\[
\text{Getfile col1 size, col2 time, col2 md5 from t123 where ...;}
\]

Output is: col1.size:[38393] col2.time:[...] col2.md5:[IEdjJDDKKDnxnE]

To get the file type of a file, use the following command:

\[
\text{Getfile col1 type from t123 where ...;}
\]

To get the hostname where a file is stored, use the following command:

\[
\text{Getfile col1 host from t123 where ...;}
\]

To get the full path of a file, use the following command:

\[
\text{Getfile col1 path from t123 where ...;}
\]

To get the host and full path of a file, use the following command:

\[
\text{Getfile col1 hostpath from t123 where ...;}
\]

To get the size of a file in MB (megabytes), use the following command:

\[
\text{Getfile col1 sizemb from t123 where ...;}
\]

To get the size of a file in GB (gigabytes), use the following command:

\[
\text{Getfile col1 sizegb from t123 where ...;}
\]
SQL Join Support

Jaguar provides inner join or simple join operation for two tables. Any two tables can be joined by any column, either key or value.

(SELECT ) from TABLE1 [inner] join TABLE2 on TABLE1.COL1=TABLE2.COL2 [WHERE CLAUSE] [GROUPBY] [ORDERBY] [LIMIT] [TIMEOUT];

(SELECT ) from TABLE1, TABLE2 where TABLE1.COL1=TABLE2.COL2 [MORE WHERE CLAUSE] [GROUPBY] [ORDERBY] [LIMIT] [TIMEOUT];

The Join statement is only support on single host systems.

Example:

select tab1.name, tab2.id from tab1 join tab2 on tab1.id=tab2.uid where tab2.zip=230210;

select tab1.name, tab2.id from tab1, tab2 where tab1.id=tab2.uid and tab2.city=123 order by tab1.indate;

Remember that Join operation has the same timeout as select (60 seconds if no timeout is provided at the end of the select statement).

Update SQL Command

update TABLE set VALUE='...', VALUE='...', ... where KEY1='...' and KEY2='...', ... ;

update TABLE set VALUE='...', VALUE='...', ... where KEY1>='...' and KEY2>='...', ... ;

update TABLE set KEY='...', VALUE='...', ... where KEY='...' and VALUE='...', ... ;

Example:

update user set address='200 Main St., SR, CA 94506' where fname='Sam' and lname='Walter';

update user set fname='Tim', address='201 Main St., SR, CA 94506' where fname='Sam' and lname='Walter';
Delete SQL Command

delete from TABLE;
delete from TABLE where KEY='...' and KEY='...' and ...;
delete from TABLE where KEY>='...' and KEY<='...' and ...;

Example:
   delete from junktable;
   delete from user where fname='Sam' and lname='Walter';

Drop command

Tables or indexes can be dropped with the drop command:

drop table [if exists|force] TABLE;
drop index INDEX on TABLE;

Example:
   drop table user;
   drop index user_idx1 on user;

Truncate command

Data in a table can be deleted with the truncate command (schema is left untouched):

   truncate table TABLE;

Data in table will be deleted, but the table tableschema still exists.

Example:
   truncate table user;
Alter command

The name of a key column can be changed to a different name:

```sql
alter table TABLE rename OLDKEY to NEWKEY;
```

Rename a key name in table TABLE.

Example:

```sql
alter table mytable rename mykey1 to userid;
```

Spool command

Send the output of a command to a file on client host:

```sql
spool LOCALFILE;
spool off;
```

Example:

```sql
spool /tmp/myout.txt;
(sp The above command will make the output data to be written to file
 /tmp/myout.txt)
spool off;
(sp The above command will stop writing output data to any file)
```

Change password

A user can change the login password:

```sql
changepass;
```

Example:
jaguar > changepass;
jaguar > New password: ********
      New password again: *****

The user can also change the password with clear-text password (less secure):

jaguar > changepass mypassword888;

Group By Statement

Aggregation operation can be performed on numerical columns of a table or index with group
by clause. The elements in the group by columns can be any column or columns. If they are all
the keys or the left-subset of keys in the table or index, no sorting operation is performed so it
would be faster than non-key group by.

If a non-numerical column is selected in the select clause without the “lastvalue”, the value of
any record is used and displayed.

Select [aggregation(COL)] from TABLE/INDEX group by c1, c2, c3, ... order by ... limit ...;

Group By LastValue Statement

The last records of certain groups in a table or index can be selected with “group by lastvalue”
statement.

Select [COL1, COL2, ...] from TABLE/INDEX group by lastvalue k1, k2, k3;

As a result of the above statement, the records are grouped according to the keys k1, k2, and k3,
and the very last record of each such group is displayed.

Order By Statement

From the select results (which may contain group by statement), data can be further ordered by
one or more columns:
order by COL1, COL2, COL3 [ASC/DESC]...

The default sorting order is ASC (meaning ascending). Descending order can be represented by DESC. The ordered columns have to be either all in ASC or all in DESC. Mixed ordering (one in ASC another in DESC) is not supported.

Aggregation Statement

Aggregation functions can be applied to one more columns in a table or index in combination with other aggregation functions.

Examples include:

Select sum(col1 + col2 ) + 2* avg(col3) from tab123 where ...;
select sum(x_coord + y_coord) as ss, 2*avg(minute1) as min2 from t123;
select sum(x_coord + y_coord) as ss, 2*stddev(minute1) as std2 from t123;

System Limits

Limits of Table Columns

A table can have a maximum of 4096 columns.

Limits on Length of A Database Name

The name of a database can have a maximum of 64 characters.

Limits on Length of A Column Name

The name of a column can have a maximum of 32 characters.
Limits on Number of Bytes of A Row

Each record or row in a table can have a maximum of 2 Billion bytes.

Data Export and Import

Data in a table of a database can be exported into insert format SQL files, which can imported back to the table later.

Export

Export table data to all server hosts

$ $JAGUAR_HOME/bin/jagexport -d <DATABASE> -t <TABLE>

For example:

$JAGUAR_HOME/bin/jagexport -d mydb -t salestab

Export table data to a SQL file on client side

$ $JAGUAR_HOME/bin/jagexportsql -d <DATABASE> -t <TABLE>

The export data is saved into DATABASE.TABLE.sql file.

For example:

$JAGUAR_HOME/bin/jagexportsql -d mydb -t salestab

The export data is saved into mydb.salestab.sql file on the client computer. Please make sure you have enough disk space on your client host when the table is large.

Export table data to a CSV file on client side

$ $JAGUAR_HOME/bin/jagexportcsv -d <DATABASE> -t <TABLE>

The export data is saved into DATABASE.TABLE.csv file.
For example:

$\text{JAGUAR\_HOME/bin/jagexportcsv -d mydb -t salestab}$

The export data is saved into mydb.salestab.csv file on the client computer.

**Import**

Import table data from all server hosts

$ $\text{JAGUAR\_HOME/bin/jagimport -d <DATABASE> -t <TABLE>}$

For example:

$\text{JAGUAR\_HOME/bin/jagimport -d mydb -t salestab}$

Import table data from a SQL file on client side

$ $\text{JAGUAR\_HOME/bin/jagimportsql DATABASE.TABLE.sql}$

The data in file DATABASE.TABLE.sql is created from the jagexportsql command.

For example:

$\text{JAGUAR\_HOME/bin/jagimportsql mydb.salestab.sql}$

Import table data from a CSV file on client side

$ $\text{JAGUAR\_HOME/bin/jagimportcsv -d DB -t TAB -f DATABASE.TABLE.csv}$

The data in file DATABASE.TABLE.csv is in CSV format.

For example:

$\text{JAGUAR\_HOME/bin/jagimportcsv -d mydb -t t1 -f mydb.t1.csv}$

**Schema Change**
Use spare_ Column

When a table is created, a spare_ column with 30% extra space is allocated (which can be configurable in server.conf file). Users can add more columns to a table, using the extra spare_ column. If the spare_ column still has space, then the following command can be used to add a new column:

```
alter table  TABLE add COLUMN TYPE;
```

Example: alter table  tab123 add spacex char(4);

Table Change

When the schema of a table does need a major change (in early stage recommended), the following procedures are recommended:

1) Execute the jagexport command to export the table data
2) Drop the table
3) Re-create the table with new columns by following these rules:
   a) Some new columns can be added
   b) Some old columns can be dropped
   c) Smaller size columns can be changed to bigger size columns (int->bigint, wider char)
   d) Remaining column names should be kept the same
4) Execute the jagimport command
5) After SUCCESSFUL import, run the jag client program to cleanup the exported data:
   ```
   $ jag -u admin -p DB -h :8888
   jaguar> import into DB.TABLE complete;
   ```

Fault Tolerance

In an operational Jaguar cluster, one or more Jaguar nodes can go down but the cluster will still function. Data records are replicated to nodes that are alive. When the down-node is up again, data is restored from the live nodes. Keep in mind that you should always have one or more spare servers ready to be commissioned. The spare servers should be installed with the same version of Jaguar software and its $JAGUAR_HOME/data directory is empty. If one Jaguar node is completely broken (such as damaged hard drive, etc.), the spare server should be configured with the same IP address as the broken server and conf/cluster.conf file is updated. Then the spare server can just be connected to the Jaguar cluster network. Data will flow from other live nodes into this new server and everything will work normally. If a Jaguar server is temporarily
disconnected from the rest of the nodes in the cluster, nothing needs to be done. When the network connection comes back up, data will be automatically restored to the node.

Expanding Jaguar Cluster

With the growth of data size, a Jaguar cluster may need to expand in order to store more data or improve the performance of the cluster. A Jaguar cluster can be expanded or scaled-out by only a few simple steps. Unlike other NoSQL databases where data needs to be migrated from old servers to new servers and the process may take hours or days, scaling process in Jaguar is instant and requires no data migration among servers. Jaguar cluster operates normally before and after the scaling process.

Here are the three simple steps to expand your current cluster:

1. Set up your new cluster like when you setup your existing cluster. The file conf/cluster.conf contains only the IP addresses of the hosts in the new cluster. (one IP address per line). Start all Jaguar servers of the new cluster.
2. Copy conf/cluster.conf in the new cluster to one of the hosts in the old cluster and name it as conf/newcluster.conf.
3. On the host which has the conf/newcluster.conf file, connect to Jaguar cluster with admin account and in exclusive mode:

   ```bash
   $JAGUAR_HOME/bin/jag -u admin -p -x yes -h 127.0.0.1:8888
   jaguar> addcluster;
   ```

After the command "addcluster" is executed, the new server hosts are accepted by the current cluster and will start to take read and write requests. When needed in the future, each new cluster of servers can be added with the same method.

The following example demonstrates how you can add a new cluster of hosts:

Suppose you have 192.168.1.10, 192.168.1.11, 192.168.1.12, 192.168.1.13 on your current cluster. You want to add a new cluster with new hosts: 192.168.1.14, 192.168.1.15, 192.168.1.16, 192.168.1.17 to the system. The following steps demonstrate the process to add the new cluster into the system:

Step 1. Provision the new hosts 192.168.1.14, 192.168.1.15, 192.168.1.16, 192.168.1.17 and install jaguardb on these hosts (cluster.conf can be empty)

Step 2. The new cluster is a blank cluster, with no database schema and table data
Step 3. Admin user should log in (or ssh) to a host in EXISTING cluster, e.g., 192.168.1.10

Step 4. On host 192.168.1.10, prepare the newcluster.conf file, with the IP addresses of the hosts on each line separately:

In $JAGUAR_HOME/conf/newcluster.conf file:

192.168.1.14
192.168.1.15
192.168.1.16
192.168.1.17

Step 5. Connect to local jaguardb server from the host that contains the newcluster.conf file

$JAGUAR_HOME/bin/jag -u admin -p <adminpassword> -h 192.168.1.10:8888 -x yes

Step 6. While connected to the jaguardb, execute the addcluster command:

jaguardb> addcluster;

The addcluster will take a few seconds to finish. All the new hosts are instantly added to current system which will function normally.

Note:

1. Never directly add new hosts in the file $JAGUAR_HOME/conf/cluster.conf manually
2. Any plan to add a new cluster of hosts must implement the addcluster process described here.
3. Execute addcluster command in the existing cluster, NOT in the new cluster.
4. It is recommended that existing clusters and new cluster contain large number of hosts. (dozens or hundreds).
   For example, the existing cluster can have 30 hosts, and the new cluster can have 100 hosts.
5. Make sure jaguardb is installed on all the hosts of the new cluster, and connectivity is good among all the hosts.
6. The server and client software must have the same version, on all the hosts of both existing clusters and the new cluster.
7. After adding a new cluster, all hosts will have the same cluster.conf file.

8. Make sure REPLICATION factor is the same on all the hosts.

Jaguar Database Security

User data is considered extremely import in Jaguar database. Several measures can be taken to protect user data in Jaguar database system.

Network Protection

In the network or subnet where Jaguar is in operation, firewall or Security Policy can be setup for protecting the system against malicious attempts. In an on-premise environment, router firewall can be configured to allow only Jaguar database traffic. In a cloud environment, security policy can be configured to allow only Jaguar data communication. Even a database firewall can be employed to allow only legitimate SQL commands to be passed through, thus any threats such as SQL-injection or other attacks can be prevented.

Server System Protection

SELinux is a hardened Linux kernel that provides strong system security to Linux systems. With SELinux installed and enabled, user permission, process control, file control are better managed to achieve higher-level security.

User Privilege and File Permission

All files and data in Jaguar are owned by only one user (jaguar). Other users do not have the permission to read and write data in Jaguar database. The authorized user has password in the Linux system and we strongly recommend a strong-security password for the user. The credentials should be securely saved and protected. File permission should be strictly enforced and maintained across Jaguar database servers.

Database User Authentication
User accounts in Jaguar database are also required to have a password that is minimum of 16 characters long. Any shorter passwords are rejected by the system. The username and password should be kept properly by all users and developers of the system and they should be frequently updated with string-security content.

User Level Control

Users of Jaguar are classified into two categories: 1) administrator; 2) regular user. Only the administrator has the privilege to create and delete databases, regular user accounts. The regular users can only create and drop tables, indexes, insert and modify data records.

Server Communication Control

In a cluster of Jaguar database servers, messages between servers are frequently passed and processed. The servers use tokens (SERVER_TOKEN) to identify and authorized themselves to obtain permission to send request to other servers. The tokens are created during initial database installation process and are unique among Jaguar customers. This ensures the integrity of a Jaguar database cluster.

Access Control List

There are whitelist (conf/whitelist.conf) and blacklist (conf/blacklist.conf) control files that are used to limit client access to Jaguar servers. Only the clients whose IP address or IP segment is included in the whitelist are authorized to connect to Jaguar servers. For certain IP addresses in the whitelist, access can be denied if they belong to a blacklist. If no whitelist and blacklist are provided, then all client access is granted. We strongly recommend that the access control lists be used in Jaguar cluster for maximum system security.

Log Monitoring

Jaguar servers generate log entries for client connection and table management. Database administrator is recommended to regularly monitor the log information, and check for illegal access to the database or database table modifications.
Data Import and Synchronization

In Jaguar github web site there are programs to import and synchronize data between other databases and Jaguar database. There are also example programs on how to import data and synch data from Oracle, MySQL and other databases. The mechanism to synchronize data is: 1) Jaguar database must create same table as in other databases; 2) Other databases create changelog and triggers to capture changes in an original table or tables; 3) import all data from other databases to Jaguar; 4) start java sync server on a Jaguar server to monitor the records in the changelog tables and update the Jaguar tables.

Step One: Create Tables on Jaguar

Suppose you have some tables on other database, you must first create the corresponding tables on Jaguar. **This must be performed on a Jaguar host.**

Example: Use github.com/datajaguar/jaguardb: importsync/databaseimport/from_oracle/create_jaguar_table.sh to create tables on Jaguar from any Jaguar host. In example1 directory you can use create_jaguar_table_example1.sh as a reference. Note: please make sure you first compile the JDBC programs: cd importsync/jdbc; ./compile.sh

Step Two: Create Changelog Triggers

On other database system you must create changelog and triggers for the tables. **This step must be performed on the other database system.**

Please go to github and the following program to create changelog and triggers:
importsync/databasesync/oracle/oracleToJaguar/oracle_create_changelog_trigger.sh

Result: The changelog for table234 is created. If table234 has any insert, update, or delete, a new record in changelog is added.

Step Three: Importing Data
Importing data from other database to Jaguar database. This step must be executed on a Jaguar server.

Please goto github and find this program:
importsync/databaseimport/from_oracle/example1/ import_from_oracle.sh
Please note that in appconf.oracle you need to have correct source_jdbcurl, dest_jdbcurl, and other parameters.

Step Four: Updating Jaguar Tables

On a Jaguar host, a java sync server needs to be started to monitor the changelog tables on the other database system. **This step must be performed on Jaguar host.**

Please use the following example program in jaguar github:
importsync/databasesync/oracle/OracleToJaguar/example1/start_sync_oracle_to_jaguar.sh
You need to change appconf.oracle to suite your own environment.

appconf.oracle:
source_jdbcurl=jdbc:oracle:thin:@//192.168.7.120:1522/test
(192.18.7.120 is IP address of Oracle server)
source_table=table234|table345 (separate tables with vertical line)
source_user=test
source_password=test
sleep_in_millis=3000 (scan changelog table every 3 seconds)
keep_rows=10000 (keeping some records in changelog)

dest_jdbcurl=jdbc:jaguar://localhost:8888/test (port of jaguar server)
dest_user=test
dest_password=test

### set true to stop java server anytime when java is running
# stop=true
If you are just importing data from other database to Jaguar, then you need to execute only step one (creating jaguar table) and step three (importing data to jaguar). The java sync server can be stopped any time and restarted without affecting the synchronization. However, for real-time updates, it is recommended that the sync server be running all the time and a smaller sleep interval is desired.

Spark Data Analysis

Since Jaguar provide JDBC connectivity, developers can use Apache Spark to load data from Jaguar and perform data analytics and machine learning. The advantages provided by Jaguar is that Spark can load data faster, especially for loading data satisfying complex conditions, from Jaguar than from other data sources. The following code is based on two tables that have the following structure:

```sql
create table int10k ( key: uid int(16), score float(16.3), value: city char(32) );
create table int10k_2 ( key: uid int(16), score float(16.3), value: city char(32) );
```

Scala program:

```scala
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import scala.collection._
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.log4j.Logger
```
import org.apache.log4j.Level
import com.jaguar.jdbc.internal.jaguar_
import com.jaguar.jdbc.JaguarDataSource

object TestScalaJDBC {
  def main(args: Array[String]) {
    sparkfunc()
  }

  def sparkfunc() {
    Class.forName("com.jaguar.jdbc.JaguarDriver");
    val sparkConf = new SparkConf().setAppName("TestScalaJDBC")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    import sqlContext.implicits._

    Logger.getLogger("org").setLevel(Level.OFF)
    Logger.getLogger("akka").setLevel(Level.OFF)

    val people = sqlContext.read.format("jdbc")
                .options(
                    Map("url" -> "jdbc:jaguar://127.0.0.1:8888/test",
                        "dbtable" -> "int10k",
                        "user" -> "test",
                        "password" -> "test",
                        "partitionColumn" -> "uid",
                        "lowerBound" -> "2",
                        "upperBound" -> "100000")
                )
  
}
"upperBound" -> "2000000",
"numPartitions" -> "4",
"driver" -> "com.jaguar.jdbc.JaguarDriver"
}).load()

// work fine
people.registerTempTable("int10k")
people.printSchema()

val people2 = sqlContext.read.format("jdbc")
  .options(
    Map( "url" -> "jdbc:jaguar://127.0.0.1:8888/test",
        "dbtable" -> "int10k_2",
        "user" -> "test",
        "password" -> "test",
        "partitionColumn" -> "uid",
        "lowerBound" -> "2",
        "upperBound" -> "2000000",
        "numPartitions" -> "4",
        "driver" -> "com.jaguar.jdbc.JaguarDriver"
      )
  ).load()
people2.registerTempTable("int10k_2")

// sort by columns

people.sort("score").show()
people.sort($"score".desc).show()
people.sort($"score".desc, $"uid".asc).show()
people.orderBy("score".desc, "uid".asc).show()

// select by expression
people.selectExpr("score", "uid" ).show()
people.selectExpr("score", "uid as keyone" ).show()
people.selectExpr("score", "uid as keyone", "abs(score)" ).show()

// select a few columns
val uid2 = people.select("uid", "score")
uid2.show();

// filter rows
val below60 = people.filter(people("uid") > 20990397 ).show()

// group by
people.groupBy("city").count().show()

// groupby and average
people.groupBy("city").avg().show()

people.groupBy(people("city"))
  .agg(
      Map(
        "score" -> "avg",
        "uid" -> "max"
      )
  )
  .show();
// rollup
people.rollup('city').avg().show()
people.rollup('$city')
  .agg(
    Map(
      "uid" -> "avg",
      "score" -> "max"
    )
  )
  .show();

// cube
people.cube('$city').avg().show()
people.cube('$city')
  .agg(
    Map(
      "uid" -> "avg",
      "score" -> "max"
    )
  )
  .show();

// describe statistics
people.describe("uid", "score").show()

// find frequent items
people.stat.freqItems(Seq("uid")).show()
// join two tables
people.join( people2, "uid" ).show()
people.join( people2, "score" ).show()
people.join( people2 ).where( people("uid") === people2("uid") ).show()
people.join( people2 ).where( people("city") === people2("city") ).show()
people.join( people2 ).where( people("uid") === people2("uid") and people("city") === people2("city") ).show()
people.join( people2 ).where( people("uid") === people2("uid") && people("city") === people2("city") ).show()
people.join( people2 ).where( people("uid") === people2("uid") && people("city") === people2("city") ).limit(3).show()

// union
people.unionAll( people2 ).show()

// intersection
people.intersect( people2 ).show()

// exception
people.except( people2 ).show()

// Take samples
people.sample( true, 0.1, 100 ).show()

// distinct
people.distinct.show()

// same as distinct
people.dropDuplicates().show()

// cache and persist
people.dropDuplicates.cache.show()
people.dropDuplicates.persist.show()

// SQL dataframe
val df = sqlContext.sql("SELECT * FROM int10k where uid < 200000000 and city between 'Alameda' and 'Berkeley'")
df.distinct.show()

The class generated from the above Scala program can be submitted to Spark as follows:

```
/bin/spark-submit --class TestScalaJDBC  
   --master spark://masterhost:7077  
   --driver-class-path /path/to/your/jaguar-jdbc-2.0.jar  
   --driver-library-path $JAGUAR_HOME/lib  
   --conf spark.executor.extraClassPath=/path/to/your/jaguar-jdbc-2.0.jar  
   --conf spark.executor.extraLibraryPath=$JAGUAR_HOME/lib  
   /path/to/your_project/target/scala-2.10/testjdbc_2.10-1.0.jar
```

SparkR with Jaguar

Once you have R and SparkR packages installed, you can start the SparkR program by executing the following command:

```
#!/bin/bash

export JAVA_HOME=/home/jvm/jdk1.8.0_60
LIBPATH=/usr/lib/R/site-library/rJava/libs:$JAGUAR_HOME/lib
```
LDLIBPATH=$LIBPATH:$JAVA_HOME/jre/lib/amd64:$JAVA_HOME/jre/lib/amd64/server
JDBCJAR=$JAGUAR_HOME/lib/jaguar-jdbc-2.0.jar

sparkR \
–driver-class-path $JDBCJAR \
–driver-library-path $LDLIBPATH \
–conf spark.executor.extraClassPath=$JDBCJAR \
–conf spark.executor.extraLibraryPath=$LDLIBPATH

Then in the SparkR command line prompt, you can execute the following R commands:

library(RJDBC)
library(SparkR)

sc <- sparkR.init(master="spark://mymaster:7077", appName="MyTest")

sqlContext <- sparkRSQL.init(sc )


conn <- dbConnect(drv, "jdbc:jaguar://localhost:8888/test", "test", "test")

dbListTables(conn)

df <- dbGetQuery(conn, "select * from int10k where uid > 'anxnfkjj2329' limit 5000;")

head( df )

#correlation
> cor(df$uid,df$score)
[1] 0.05107418

#build the simple linear regression
> model<-lm(uid~score,data=df)
> model

Call:
  lm(formula = uid ~ score, data = df)
Coefficients:
(Intercept) score
2.115e+07 1.025e-03

# get the names of all of the attributes
> attributes(model)

$names
[1] "coefficients" "residuals" "effects" "rank"
[5] "fitted.values" "assign" "qr" "df.residual"
[9] "xlevels" "call" "terms" "model"

$class
[1] "lm"

Spatial Data Management

Jaguar supports spatial data in various forms, including vector geometry objects and raster coordinates. Vector geometry objects include square, rectangle, circle, ellipse, cube, box, sphere, etc. Raster objects include multipoint, linestring, multilinestring, polygon, and multipolygon.

Spatial Data Types

In addition to existing data types in Jaguar, new spatial data types are also supported. In spatial data management, the reference system and direction of an object or surface are important.
factors.

\begin{align*}
0 \leq nx &\leq 1 \\
0 \leq ny &\leq 1 \\
\text{in 2D: } \sqrt{nx^2 + ny^2} &= 1
\end{align*}

\begin{align*}
0 \leq nx &\leq 1 \\
0 \leq ny &\leq 1 \\
\text{in 3D: } \sqrt{nx^2 + ny^2 + nz^2} &= 1
\end{align*}

\begin{align*}
\text{point}(x, y) & \quad \text{point3d}(x, y, z) \\
\text{二维点} & \quad \text{三维点}
\end{align*}

\begin{align*}
\text{Line}(x_1, y_1, x_2, y_2) & \quad \text{Line3D}(x_1, y_1, z_1, x_2, y_2, z_2) \\
\text{二维线段} & \quad \text{三维线段}
\end{align*}

\begin{align*}
\text{Circle}(x, y, a) & \quad \text{Circle3d}(x, y, z, a) \\
\text{二维圆} & \quad \text{三维圆}
\end{align*}

\begin{align*}
\text{Square}(x, y, a, nx) & \quad \text{Square3d}(x, y, z, a, nx) \\
\text{二维正方形} & \quad \text{三维正方形}
\end{align*}
sphere(x y z a)  球体
rectangle(x y a b nx)  矩形
rectangle3d(x y z a b nx ny)  三维矩形面

cube( x y z a nx ny)  a = half edge
Nx ny: direction
立方

box(x y z a b c nx ny)  a = half width
b = half depth
c = half height
Nx ny: direction
长方体
ellipse(x y a b nx)
  a = half width
  b = half height
  Nx: direction

ellipse3d(x y z a b nx ny)
  a = half width
  b = half height
  Nx ny: direction

ellipsoid(x y z a b c nx ny)
  a = half width
  b = half depth
  c = half height
  Nx ny: direction of own Z-axis

 三维椭圆面

 三棱柱

 三棱柱

 三维椭圆体

 三棱柱

 三棱柱
triangle( x1 y1 x2 y2 x3 y3)  
平面三角形

triangle3D( x1 y1 z1 x2 y2 z2 x3 y3 z3)  
三维空间三角面

linestring( x1 y1, x2 y2, x3 y3, x4 y4)  
线串

linestring3D( x1 y1 z1, x2 y2 z2, x3 y3 z3, x4 y4 z4)  
三维空间线串

polygon( (x1 y1, x2 y2, x3 y3,x4 y4,x5 y5,x6 y6, x1 y1), (x6 y6,x7 y7,x8 y8,x6 y6 ) )  
平面多边形 – 可含有一个或多个洞
polygon3D( [x1 y1 z1, x2 y2 z2, x3 y3 z3, x4 y4 z4, x5 y5 z5, x6 y6 z6, x1 y1 z1], [x6 y6 z6, x7 y7 z7, x8 y8 z8, x6 y6 z6] )

三维多边形 - 可含有一个或多个洞
multipoint(x y)
二维多点

multipoint3d(x y z)
三维多点

multilinestring((x1 y1, x2 y2, x3 y3), (x4 y4, x5 y5))
多线串

multilinestring3d((x1 y1 z1, x2 y2 z2, x3 y3 z3), (x4 y4 z4, x5 y5 z5))
三维多线串

multipolygon((polygon1), (polygon2), ...)
多-多边形 – 可含有多个多边形

multipolygon3d((polygon1), (polygon2), ...)
三维多-多边形 – 可含有多个三维多边形
Spatial Data Storage

Creating Table Containing Spatial Data

In creating table containing spatial data types, the type of a column can have a spatial reference identifier (SRID). If no SRID is provided, the default value is zero, meaning it is a simple geometric coordinate system. In addition to the SRID of the column, the number of metrics associated with location point or a shape can be specified with the “metrics:” keyword.

The following examples show how to create tables with spatial columns.

```
cREATE TABLE IF NOT EXISTS geom ( key: a int, value: pt point(srid:4326), b int );
cREATE TABLE IF NOT EXISTS geom2 ( key: a int, value: pt point(srid:wgs84), b int );
cREATE TABLE IF NOT EXISTS geom3 ( key: a int, value: pt point, b int );
cREATE TABLE dot ( key: a int, pt1 point, b int, pt2 point, value: c int, d int, pt3 point3d );
cREATE TABLE cb ( key: a int, q1 cube, b int, q2 cube, value: c int, q3 cube );
cREATE TABLE es ( key: a int, c ellipsoid, value: d int, e ellipse );
cREATE TABLE linestr ( key: lsw linestring(srid:wgs84), a int, value: lss linestring );
cREATE TABLE pol ( key: a int, value: po2 polygon, po3 polygon3d, tm datetime, ls linestring );
cREATE TABLE mline ( key: a int, value: m multilinestring, m3 multilinestring3d );
cREATE TABLE mpg ( key: a int, value: p multipolygon, p3 multipolygon3d );
cREATE TABLE street ( key: a int, value: pt linestring(srid:wgs84,metrics:10), b int );
cREATE TABLE base ( key: a int, value: pt point(srid:wgs84,metrics:20), char(32) );
```

The number of metrics is unlimited, as long as the storage space allows. Each metric has a length of 8 bytes, with default value of zero. The metrics are identified by mN, such as:

```
SELECT col:m1, col:m3 FROM mytab WHERE a=100 AND col:x=200 AND col:y=300;
```
Inserting Spatial Data

Spatial data can be inserted into a table as any other data type. GeoJson formatted data is also accepted by Jaguar. GeoJson format begins with "json" type identifier. The following examples show how to insert spatial data into tables.

```
insert into geom values ( 1, point(23.2222  52.39393), 123 );
insert into geom2 values ( 1, point(22 33), 123, point(99 221) );
insert into geom2 values ( 10, json({"type":"Point", "coordinates": [2,3]}), 123 );
insert into geom3 (b, pt2, pt1, a ) values ( 2, point(25 33), point(23 451), 153 );
insert into d6 ( pt2, a, b, pt1 ) values ( json({"type":"Point", "coordinates": [124,351]}), 209, 13, point(92 19) );
insert into sph1 (s1, b, a, s2 ) values ( sphere( 2 3 4 123), 921, 234, sphere(99 22 33 20000) );
insert into rect1 ( c, a, r1 ) values ( 22, 31, rectangle(29 13 48 19) );
insert into cyn values ( 1, cylinder(1 2 3 4 5 8 8 0.3), 1239 );
insert into cn values ( 1, cone(1 2 3 4 5 8 8 88), 1239, cone(33 22 44 44 99 0.4 0.3) );
insert into eps values ( 1, ellipse(1 2 45 88), 1239, ellipse(22 44 44 99) );
insert into tri values ( triangle3d( 11 33 88 99 23 43 9 8 2), 123 );
insert into linestr values ( 2, linestring( 11.132,2.9 33, 33 44, 5.5 6.6, 55 66, 77 88 ), 210, linestring( 3.3 4.4, 5.5 6.6, 8.9 9 ) );
insert into pol values ( 1, polygon( (0 0, 20 0, 88 99, 0 0) ) );
insert into mp values ( 125, multipoint( 1 2, 3 4, 2 1 ), json({"type":"MultiPoint", "coordinates": [ [1,2,3],[3,4,5] ] });
insert into mline values( 1, multilinestring((0 0,2 0,8 9,0 0),(1 2,2 3,1 2)),multilinestring3d((1 1 1,2 2 2,3 3 3),(2 2 2,3 3 1)));
```

It should be noted that a polygon can have an outer “ring” and several internal “rings” or holes inside the outer ring. The start and end points of a ring must be the same to form a closed shape. A multipolygon just contain multiple polygons where each polygon can have one or more rings. An example of polygon is:

```
polygon( (0 0,2 0,8 9, 23 32, 0 0), (1 2,2 3, 4 5, 1 2), (12 32, 33 44, 50 60, 12 32 ))
```

An example of 3D multipolygon is:

```
multipolygon3d( ((1 1 1,2 2 2,3 3 3, 1 1 1),(2 2 2,3 3 1, 3 5 6, 2 2 2)), ( (1 2, 20 30, 30 40, 1 2),(0 0, 1 2, 2 3, 0 0 ) ));
```
Metric data should be placed following the coordinates and geometric data of a shape. For example,

```
Insert into t values ( square(0 0 100 0.2 m1 m2 m3 ...) );
Insert into ms values ( linestring( 0 0 m1 m2 m3, 3 4 m1 m2 m3, 4 6 m1 m2 m3 ) );
Insert into ms values ( triangle( 0 0, 3 4, 4 6, m1 m2 m3 ) );
```

### Loading Spatial Data

You can prepare a csv-like file containing spatial data and load the file into a table. An example of such file is (input.csv):

```
1,"john doe", point(1 2), 421, linestring( 11 22, 33 44, 55 66)
2,"john doe", point(2 2), 321, linestring( 101 202, 303 494, 550 676), 876
3,"sam", point(3 2), 351, linestring( 151 282, 33 454, 505 666)
4,"dave", point(4 2), 39, linestring( 171 252, 33 424, 575 696)
```

Then you can execute the following command to load the data into a table:

```
$JAGUAR_HOME/bin/jagimportcsv -d DB -t TAB -f input.csv
```

### Spatial Data Query

#### Coordinate

The x, y, z coordinates of a spatial column can be used for data query. For example:

```
select * from points where a > 100 and pt:x between 0.3 and 122 and pt:y between 300 and 400;
```

#### Within
The “within” function checks if a shape is strictly within another shape. For example:

```sql
select * from poly where nm like 'east%' and within(po, square(0 0 10000 ) );
select * from poly3d where within(po, cube(0 0 0 10000 ) );
select * from sq where within(s, polygon( (0 0, 1 1, 2 2, 0 0)) );
select * from sq where within(s, rectangle(0 0 20 30 ) );
```

NearBy

Checks if a shape is near to a location by a distance.

```sql
select * from sq1 where nearby(s1, point( 0 0 ), 2000 );
```

The following table lists the shapes that can be calculated for the nearby relation.
### Intersect

<table>
<thead>
<tr>
<th>NearBy()</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>line3d</td>
</tr>
<tr>
<td></td>
<td>triangle</td>
</tr>
<tr>
<td></td>
<td>triangle3d</td>
</tr>
<tr>
<td></td>
<td>cube</td>
</tr>
<tr>
<td></td>
<td>sphere</td>
</tr>
<tr>
<td></td>
<td>circle3d</td>
</tr>
<tr>
<td></td>
<td>box</td>
</tr>
<tr>
<td></td>
<td>ellipsoid</td>
</tr>
<tr>
<td></td>
<td>cone</td>
</tr>
<tr>
<td></td>
<td>rectangle3d</td>
</tr>
<tr>
<td></td>
<td>circle</td>
</tr>
<tr>
<td></td>
<td>square</td>
</tr>
<tr>
<td></td>
<td>rectangle</td>
</tr>
<tr>
<td></td>
<td>ellipse</td>
</tr>
<tr>
<td></td>
<td>point</td>
</tr>
<tr>
<td></td>
<td>point3d</td>
</tr>
</tbody>
</table>

#### Intersect

- **Intersect**: 相交
- **Disjoint**: 没有任何相交

```sql
select * from sq1 where intersect(s1, square(0 0 200));
select * from rect where intersect(r, rectangle(0 0 200 100));
select * from lstr where intersect(s, rectangle(0 0 200 100));
select * from lstr where intersect(s, circle(0 0 200));
select * from lstr where intersect(s, ellipse(0 0 200 100));
```
CoveredBy

CoveredBy function is similar to “Within” except that some boundary can overlap between two shapes.

Cover

Cover is the opposite of CoveredBy function.

Contain

The “contain” function checks if a shape strictly contains another shape. It is the opposite of the “Within” function between two shapes.

Disjoint

```
select * from sql where disjoint(s1, rectangle(0 0 200 300));
```

Disjoint is the opposite of “Intersect” function.

Distance

The distance function computes the distance between two shapes.

If the SRID of the shapes are WGS84, then the distance computed is in meters between two shapes described with (longitude, latitude). The X coordinate is the longitude, and Y the latitude. The following table lists the shapes that can be calculated for the distance() function.
Shapes for Location Relation

The following tables list the shapes that can be used to query the location relation among them. Most shapes can be used to query within, contain, cover, coveredby, intersect, disjoint functions.

<table>
<thead>
<tr>
<th>Distance()</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>line3d</td>
</tr>
<tr>
<td></td>
<td>triangle</td>
</tr>
<tr>
<td></td>
<td>triangle3d</td>
</tr>
<tr>
<td></td>
<td>cube</td>
</tr>
<tr>
<td></td>
<td>sphere</td>
</tr>
<tr>
<td></td>
<td>circle3d</td>
</tr>
<tr>
<td></td>
<td>box</td>
</tr>
<tr>
<td></td>
<td>ellipsoid</td>
</tr>
<tr>
<td></td>
<td>cone</td>
</tr>
<tr>
<td></td>
<td>rectangle3d</td>
</tr>
<tr>
<td></td>
<td>circle</td>
</tr>
<tr>
<td></td>
<td>square</td>
</tr>
<tr>
<td></td>
<td>rectangle</td>
</tr>
<tr>
<td></td>
<td>ellipse</td>
</tr>
<tr>
<td></td>
<td>point</td>
</tr>
<tr>
<td></td>
<td>point3d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Between two shapes</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>line3d</td>
</tr>
<tr>
<td></td>
<td>triangle</td>
</tr>
<tr>
<td></td>
<td>triangle3d</td>
</tr>
<tr>
<td></td>
<td>cube</td>
</tr>
<tr>
<td></td>
<td>sphere</td>
</tr>
<tr>
<td></td>
<td>circle3d</td>
</tr>
<tr>
<td></td>
<td>box</td>
</tr>
<tr>
<td></td>
<td>ellipsoid</td>
</tr>
<tr>
<td></td>
<td>cone</td>
</tr>
<tr>
<td></td>
<td>rectangle3d</td>
</tr>
<tr>
<td></td>
<td>circle</td>
</tr>
<tr>
<td></td>
<td>square</td>
</tr>
<tr>
<td></td>
<td>rectangle</td>
</tr>
<tr>
<td></td>
<td>ellipse</td>
</tr>
<tr>
<td></td>
<td>point</td>
</tr>
<tr>
<td></td>
<td>point3d</td>
</tr>
<tr>
<td>Point</td>
<td>point</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>square</td>
</tr>
<tr>
<td></td>
<td>ellipse</td>
</tr>
<tr>
<td>Point3D</td>
<td>point3d</td>
</tr>
<tr>
<td></td>
<td>box</td>
</tr>
<tr>
<td></td>
<td>cone</td>
</tr>
<tr>
<td>MultiPoint</td>
<td>point</td>
</tr>
<tr>
<td></td>
<td>Triangle</td>
</tr>
<tr>
<td></td>
<td>Rectangle</td>
</tr>
<tr>
<td></td>
<td>ellipse</td>
</tr>
<tr>
<td></td>
<td>polygon</td>
</tr>
</tbody>
</table>

| MultiPoint3D | point3d  | line3d   |
|              | cube     | box      |
|              | sphere   | ellipsoid|
|              | cone     |          |

| Line         | triangle | square   |
|              | rectangle| ellipse  |
|              | circle   | linestring|
|              | polygon  |          |
Line3D

- Cube
- Box
- Sphere
- Ellipsoid
- cone
- triangle3D
- square3D
- rectangle3D

Triangle

- triangle
- square
- rectangle
- ellipse
- circle
- linestring
- polygon

Triangle3D

- Cube
- Box
- Sphere
- Ellipsoid
- cone
| Circle | circle  
square  
rectangle  
estepline  
triangle  
linestring  
polygon |
|--------|--------|
| Circle3D | cube  
box  
sphere  
estepline  
ellipsoid  
cone |
| Sphere | cube  
box  
sphere  
estepline  
ellipsoid  
cone  
linestring  
polygon |
<table>
<thead>
<tr>
<th>Square</th>
<th>triangle</th>
<th>square</th>
<th>rectangle</th>
<th>ellipse</th>
<th>circle</th>
<th>linestring</th>
<th>polygon</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Square3D</th>
<th>Cube</th>
<th>Box</th>
<th>Sphere</th>
<th>Ellipsoid</th>
<th>cone</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Cube</th>
<th>Cube</th>
<th>Box</th>
<th>Sphere</th>
<th>Ellipsoid</th>
<th>cone</th>
</tr>
</thead>
</table>
### Rectangle

- triangle
- square
- rectangle
- ellipse
- circle
- linestring
- polygon

![Rectangle Diagram]

### Rectangle3D

- Cube
- Box
- Sphere
- Ellipsoid
- cone

![Rectangle3D Diagram]

### Box

- Cube
- Box
- Sphere
- Ellipsoid
- cone

![Box Diagram]
<table>
<thead>
<tr>
<th>Cylinder</th>
<th>Cone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube</td>
<td>Cube</td>
</tr>
<tr>
<td>Box</td>
<td>Box</td>
</tr>
<tr>
<td>Sphere</td>
<td>Sphere</td>
</tr>
<tr>
<td>Ellipsoid</td>
<td>Ellipsoid</td>
</tr>
<tr>
<td>Cone</td>
<td>cone</td>
</tr>
<tr>
<td>Ellipse</td>
<td>triangle</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ellipsoid</th>
<th>Cube</th>
<th>Box</th>
<th>Sphere</th>
<th>Ellipsoid</th>
<th>cone</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>LineString</th>
<th>triangle</th>
<th>square</th>
<th>rectangle</th>
<th>ellipse</th>
<th>circle</th>
<th>linestring</th>
<th>polygon</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiLineString</td>
<td>triangle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>square</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rectangle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ellipse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>circle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>linestring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>polygon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LineString3D</th>
<th>triangle3D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>square3D</td>
</tr>
<tr>
<td></td>
<td>rectangle3D</td>
</tr>
<tr>
<td></td>
<td>cube</td>
</tr>
<tr>
<td></td>
<td>box</td>
</tr>
<tr>
<td></td>
<td>sphere</td>
</tr>
<tr>
<td></td>
<td>ellipsoid</td>
</tr>
<tr>
<td></td>
<td>cone</td>
</tr>
<tr>
<td></td>
<td>linestring3D</td>
</tr>
</tbody>
</table>
**MultiLineString3D**
- triangle3D
- square3D
- rectangle3D
- cube
- box
- sphere
- ellipsoid
- cone
- linestring3D

**Polygon**
- triangle
- square
- rectangle
- ellipse
- circle
- linestring
- polygon
<table>
<thead>
<tr>
<th>Polygon3D</th>
<th>triangle3D</th>
<th>square3D</th>
<th>rectangle3D</th>
<th>cube</th>
<th>box</th>
<th>sphere</th>
<th>ellipsoid</th>
<th>cone</th>
<th>linestring3D</th>
<th>polygon3d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MultiPolygon</th>
<th>triangle</th>
<th>square</th>
<th>rectangle</th>
<th>ellipse</th>
<th>circle</th>
<th>linestring</th>
<th>Polygon (within)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The area of 2D shapes and surface area of 3D can be computed with the `area()` function, which is valid for the following geometric or geographic shapes:

- Circle
- Circle3D
- Square
- Square3D
- Rectangle
- Rectangle3D
- Triangle
- Triangle3D
- Sphere
- Cube
- Cylinder
- Cone
- Ellipse
- Ellipsoid
- Polygon
- MultiPolygon

For example:

```
jaguar:mydb> select area(sq3) from geotab where key1 < 100;
jaguar:mydb> select col1, area(circle(1 1 10)) as b from geotab;
```
GeoJson

GeoJson is a format to represent shapes using JSON data string. If a GeoJson string is desired from a query result, then the “all” function can be used. The all() function returns all the data of a complex geometric shape in GeoJson format. For example,

```
select all(s) from lstr where intersect(s, circle(0 0 200));
select all(s) from lstr where a=100;
```

The complex shapes include linestring, multilinestring, polygon, multipolygon and their 3D counterparts. In the client programming API, there are methods to get the all data points of a complex shape: getAll(), getAllByName(), getAllByIndex(). Please refer to the file JaguarAPI.h in $JAGUAR_HOME/include/ directory for detailed information.

Dimension

dimension(col) -- get dimension as integer of a shape column. It returns 2 for 2D shapes, and 3 for 3D shapes, and 0 for non-geometric columns. For example:

```
jaguar:test> select dimension(mp) as dim from mpolygon;
jaguar:test> dim:[2]
```

GeoType

geotype(col) -- get type as string of a shape column. For example:

```
jaguar:test> select geotype(mp) as tp from mpolygon;
jaguar:test> tp:[MultiPolygon]
```

PointN

pointn(col,n) -- get n-th point (1-based) of a shape column. It returns in format: [x y] for 2D shapes and [x y z] for 3D shapes. For example:
jaguar:test> select pointn(pt, 3) as p3 from mpoints;
jaguar:test> p3:[10.2 32.7]

Extent

extent(col) -- get bounding box of a shape column.
For 2D shapes, it returns [xmin ymin xmax ymax]
For 3D shapes, it returns [xmin ymin zmin xmax ymax zmax]

jaguar:test> select extent(pd) as bbx from mpoints;
jaguar:test> bbx:[0 0 33.2 49.8]

StartPoint

startpoint(col) -- get the start point of a line string column.
For 2D shapes, it returns [x y], For 3D shapes, it returns [x y z].

jaguar:test> select startpoint(pd) as st from mpoints;
jaguar:test> st:[203.2 178.5]

EndPoint

endpoint(col) -- get the end point of a line string column.
For 2D shapes, it returns [x y], For 3D shapes, it returns [x y z].

jaguar:test> select endpoint(pd) as end from mpoints;
jaguar:test> end:[903.4 778.6]

IsClosed

isclosed(col) -- check if raster points of a line string column is closed. (0 or 1)

jaguar:test> select isclosed(linecol) as ic from mpoints;
jaguar:test> ic:[0]
Number of Points

`numpoints(col)` -- get total number of points of a line string or polygon

```sql
jaguar:test> select * from mpoints where numpoints(pcol) < 100;
```

Number of Rings

`numrings(col)` -- get total number of rings of a polygon or multipolygon. A polygon contains an outer-ring, and also may contain 0 or more inner-rings (holes).

```sql
jaguar:test> select a, numrings(p) r from mpoints;
jaguar:test> a:[123] r:[3]
```

Number of Lines

`numlines(col)` -- get total number of linestrings of a multilinestring, polygon or multipolygon.

```sql
jaguar:test> select a, numlines(p) r from mlstrs;
jaguar:test> a:[123] r:[3]
```

SRID

`srid(col)` -- get SRID of a shape column.

Summary

`summary(col)` -- get a text summary of a shape column. For example:

```sql
jaguar:test> select summary(p) sum from mpoints;
sum=[geotype:Polygon srid:0 dimension:2 numpoints:9 numrings:2 isclosed:1]
```

Minimum and Maximum Points
xmin(col)  -- get the minimum x-coordinate of a shape with raster data
ymin(col)  -- get the minimum y-coordinate of a shape with raster data
zmin(col)  -- get the minimum z-coordinate of a shape with raster data
xmax(col)  -- get the maximum x-coordinate of a shape with raster data
ymax(col)  -- get the maximum y-coordinate of a shape with raster data
zmax(col)  -- get the maximum z-coordinate of a shape with raster data

For example:

```
  jaguar:test> select xmin(p) xm from mpoints;
  jaguar:test> xm=[30.4 332.3 939.9]
```

ConvexHull

`Convexhull(geom)`  -- get the convex hull as polygon of a shape with raster data

For example:

```
  jaguar:test> select convexhull(mline) from multilines;
  jaguar:test> select numpoints(convexhull(lstr)) from linestr;
```

Centroid

`centroid(geom)`  -- get the centroid coordinates of a vector or raster shape

For example:

```
  jaguar:test> select centroid(mline) from multilines;
```

Volume

`volume(geom)`  -- get the volume of a 3D shape

Closestpoint

`closestpoint(point(x y), geom)`  -- get the closest point on geom from point(x y)
Angle

angle(line(x y), geom) – get the angle in degrees between two lines

Buffer

buffer(geom, 'STRATEGY') – get polygon buffer of a shape.

The STRATEGY is:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>distance</td>
<td>Symmetric or asymmetric</td>
<td>RADIUS: length</td>
</tr>
<tr>
<td>join</td>
<td>Round or miter</td>
<td>Number of points</td>
</tr>
<tr>
<td>end</td>
<td>Round or flat</td>
<td></td>
</tr>
<tr>
<td>point</td>
<td>Circle or square</td>
<td>Number of points</td>
</tr>
</tbody>
</table>

‘distance=symmetric/asymmetric:RADIUS,join=round/miter:N,end=round/flat,point=circle/square:N’

Length

length(geom) – get length of line, line3d, linestring, linestring3d, multilinestring, multilinestring3d.

Perimeter

perimeter(geom) – get perimeter length of a closed shape (vector or raster)

Equal

equal(geom1, geom2) – check if shape geom1 is exactly the same as shape geom2

IsSimple
issimple(geom) -- check if shape geom has no self-intersecting or tangent points

isValid

isvalid(geom ) -- check if multipoint, linestring, polygon, multilinestring, multipolygon is valid

IsRing

isring(geom ) -- check if linestring is a ring

IsPolygonCCW

ispolygonccw(geom ) -- check if the outer ring is counter-clock-wise, inner rings clock-wise

IsPolygonCW

ispolygoncw(geom ) -- check if the outer ring is clock-wise, inner rings counter-clock-wise

OuterRing

outerring(polygon) -- the outer ring as linestring of a polygon

OuterRings

outerrings(mpolygon) -- the outer rings as multilinestring of a multipolygon

InnerRings

innerrings(polygon) -- the inner rings as multilinestring of a polygon or multipolygon
RingN

\texttt{ringn}(\texttt{polygon}, \texttt{n}) \ -- \textit{the n-th ring as linestring of a polygon. n is 1-based}

InnerRingN

\texttt{innerringn}(\texttt{polygon}, \texttt{n}) \ -- \textit{the n-th inner ring as linestring of a polygon. n is 1-based}

PolygonN

\texttt{polygonn}(\texttt{multipgon}, \texttt{n}) \ -- \textit{the n-th polygon of a multipolygon. n is 1-based}

Unique

\texttt{unique}(\texttt{geom}) \ -- \textit{geom with consecutive duplicate points removed}

Union

\texttt{union}(\texttt{geom1}, \texttt{geom2}) \ -- \textit{union of two geoms. Polygon outer ring should be counter-clockwise.}

Collect

\texttt{collect}(\texttt{geom1}, \texttt{geom2}) \ -- \textit{collection of two geometric shapes.}

ToPolygon

\texttt{topolygon}(\texttt{geom}) \ -- \textit{converting square, rectangle, circle, ellipse, triangle to polygon}

Text

\texttt{text}(\texttt{geom}) \ -- \textit{text string of a geometry shape}
Difference

difference(geom1, geom2) -- geom1 minus the common part of geom1 and geom2

SymDifference

symdifference(geom1, geom2) -- geom1 - geom2 minus the common part of geom1 and geom2 (symmetric difference)

IsConvex

isconvex(pgon) -- check if the outer ring of a polygon is convex

Interpolate

interpolate(lstr, frac) -- the point on linestring lstr where line length from beginning to the point is equal to frac, which is between 0 and 1.

For example: select interpolate(lstr, 0.5) from mylines;

LineSubstring

linesubstring(lstr, startfrac, endfrac) -- the substring of linestring lstr where the substring starts at startfrac and ends at endfrac. 0.0 <= startfrac <= endfrac <= 1.0.

LocatePoint

locatepoint(lstr, point) -- fraction of length where a point on linestring is closest to a given point.

For example: select locatepoint(lstr, point(100 200)) from mylines;
AddPoint

addpoint(lstr, point) – add a point at the end of a linestring.

addpoint(lstr, point, position) – add a point at the position of a linestring. Position is 1-based.

For example:

    select addpoint(lstr3, point3d(100 200 300)) from mylines3d;

SetPoint

setpoint(lstr, point, position) – set or replace a point at the position of a linestring. Position is 1-based.

For example:

    select setpoint(lstr3, point3d(100 200 300), 3) from mylines3d;

RemovePoint

removepoint(lstr, position) – remove a point at the position of a linestring. Position is 1-based.

For example:

    select numpoints(removepoint(lstr3, 3)) from mylines3d;

Reverse

reverse(geom) – reverse the order of points in line, linestring, polygon, and multipolygon.

For example:

    select reverse(lstr) from mylines3d;
Scale

scale(geom, facor) – scale the coordinates of geom by a factor.
scale(geom, xfactor, yfactor) – scale the x-y coordinates of 2D geom by two factors.
scale(geom, xfactor, yfactor, zfactor) – scale a 3D geom by three factors.

For example: select scale(ls, 0.5, 0.8) from mylines;

ScaleAt

scaleat(geom, point, fac) – scale the coordinates of geom by a factor relative to a point.
scaleat(geom, point, xfac, yfac) – scale 2D geom by two factors relative to a point.
scaleat(geom, point, xfac, yfac, zfac) – scale a 3D geom by three factors from a point.

For example:

    select scaleat(ls, point(100 100), 0.5, 0.8) from mylines;

ScaleSize

scalesize(geom, fac) – scale the size of vector geom by a factor relative to self-center.
scalesize(geom, xfac, yfac) – scale 2D vector geom by two factors relative to self-center
scalesize(geom, xfac, yfac, zfac) – scale a 3D vector geom by three factors.

For example:

    select scalesize(s2, 0.5, 0.8) from mysquare;

Translate

translate(geom, dx, dy) – translate the position of 2D geom by dx in X and dy in Y axis.
translate(geom, dx, dy, dz) – translate the position of 3D geom by dx, dy, and dz.
For example:

```sql
select translate(s2, 0.5, 0.8) from mysquare;
```

**TransScale**

```sql
transscale(geom, dx, dy, fac) -- translate 2D geom by dx and dy and then scale it by fac.
transscale(geom, dx, dy, xfac, yfac) -- translate geom and then scale it by xfac and yfac.
transscale(geom, dx, dy, dz, xfac, yfac, zfac) -- translate 3D geom and scale.
```

For example:

```sql
select transscale(s2, 100, 200, 2.0, 3.0) from mysquare;
```

**Rotate**

```sql
rotate(geom, N) -- rotate 2D geom by N degrees counter-clock-wise around point(0 0).
rotate(geom, N, 'degree') -- rotate 2D geom by N degrees counter-clock-wise.
rotate(geom, N, 'radian') -- rotate 2D geom by N radians counter-clock-wise.
```

For example:

```sql
select rotate(s, 90, 'degree') from mysquare;
```

**RotateSelf**

```sql
rotateself(geom, N) -- rotate 2D geom by N degrees counter-clock-wise around self-center.
rotateself(geom, N, 'degree') -- rotate 2D geom by N degrees counter-clock-wise.
rotateself(geom, N, 'radian') -- rotate 2D geom by N radians counter-clock-wise.
```

For example:

```sql
select rotateself(s, 90, 'degree') from mysquare;
```
RotateAt

rotateat(geom,N, 'degree’, x, y) – rotate 2D geom by N degrees around point(x y)
rotateat(geom,N, ‘radian’, x, y) – rotate 2D geom by N radians around point(x y)

For example:

    select rotateat(s, 90, ‘degree’, 200, 300) from mysquare;

Affine

affine(geom,a,b,d,e,dx,dy) – apply affine transformation of 2D geom.
affine(geom,a,b,c,d,e,f,g,h,i,dx,dy) – apply affine transformation of 3D geom.

In affine transformation:

2D:

    newx = a*x + b*y + dx
    newy = d*x + e*y + dy

3D:

    newx = a*x + b*y + c*z + dx
    newy = d*x + e*y + f*z + dy
    newz = g*x + h*y + i*z + dz

For example:

    select affine(ls, 10, 20, 30, 40, 100, 200 ) from mylines;

Voronoi Polygons

voronoipolygons(mpoint) – find Voronoi polygons from multipoints
voronoipolygons(mpoint,tolerance) – find Voronoi polygons from multipoints with tolerance
voronoipolygons(mpoint,tolerance,bbox) -- find Voronoi polygons from multipoints with tolerance and bounding box. Default bounding box is 30% larger than the bounding box of all the points.

For example:

```sql
select voronoipolygons(p, 10, bbox(-100, -100, 200, 200) ) as vor from mypoints;
select numpolygons(voronoipolygons(p, 10, bbox(-100, -100, 200, 200) )) as np from mypoints;
```

Voronoi Lines

voronoilines(mpoint) -- find Voronoi linestrings from multipoints
voronoilines(mpoint,tolerance) -- find Voronoi linestrings from multipoints with tolerance
voronoilines(mpoint,tolerance,bbox) -- find Voronoi linestrings from multipoints with tolerance and bounding box. Default bounding box is 30% larger than the bounding box of all the points.

For example:

```sql
select voronoilines(p, 10, bbox(-100, -100, 200, 200) ) as vor from mypoints;
```

Delaunay Triangles

delaunaytriangles(mpoint) -- find Delaunay triangles from multipoints
delaunaytriangles(mpoint,tolerance) -- find Delaunay triangles from multipoints with a tolerance

For example:

```sql
select delaunaytriangles(p) as dt from mypoints;
```

GeoJson

gjson(geom) -- GeoJSON string of geom
gjson(geom,N) -- GeoJSON string of geom, receiving maximum of N points (default 3000).
geojson(geom,N,n) -- GeoJSON string of geom, receiving maximum of N points, n sample points on 2D vector shapes.

For example:

```sql
select geojson(p) as js from mypolygons;
select geojson(p,1000) as js from mypolygons;
```

ToMultipoint

tomultipoint(geom) -- converting geom to multipoint
tomultipoint(geom,N) -- converting geom to multipoint. N is number of points sent to client

For example:

```sql
select tomultipoint(p) as mt from mypolygons;
```

WKT (Well Known Text)

wkt(geom) -- display geom as WKT (Well Known Text) string

For example:

```sql
select wkt(p) from mypolygons;
```

MinimumBoundingCircle

minimumboundingcircle(geom) -- minimum bounding circle of 2D geom

For example:

```sql
select minimumboundingcircle(p) from mypoints;
```
MinimumBoundingSphere

minimumboundingsphere(geom) -- minimum bounding sphere of 3D geom

For example:

    select minimumboundingsphere(p) from mypoints;

IsOnLeft

isonleft(geom1, geom2) -- detects if geom1 is on the left of geom2, for linear shapes only.

For example:

    select isonleft(point, linestr) from mylines;

IsOnRight

isonright(geom1, geom2) -- detects if geom1 is on the right of geom2, for linear shapes

For example:

    select isonright(lstr, linestr) from mylines;

LeftRatio

leftratio(geom1, geom2) -- ratio of geom1 on the left of geom2, for linear shapes

For example:

    select leftratio(point, linestr) from mylines;

RightRatio
rightratio(geom1, geom2) -- ratio of geom1 on the right of geom2, for linear shapes

For example:

```sql
select rightratio(point, linestr) from mylines;
```

KNN (K Nearest Neighbor)

knn(geom, point, K) -- K nearest neighbors on geom to point

knn(geom, point, K, min, max) -- K nearest neighbors on geom to point between distance min and max

For example:

```sql
select knn(linestr, point(30 40)) from mylines;
select knn(linestr3d, point3d(30 40 50), 10, 100) from mylines3d;
```

MetricN

metricn(geom) -- all metrics of vector shape geom (m1#m2#m3#...)

metricn(geom, N) -- metrics of N-th point of raster shapes. For vector shapes, the N-th metric.

metricn(geom, N, m) -- metric of N-th point, m-th metric for raster shapes.

For example:

```sql
select metricn( mysquare ) from squares;
select metricn( mysquare, 3 ) from squares;
select metricn( lstr, 3, 1 ) from linestrings;
```

Spatial Index

Indexes can be built and maintained for spatial data like other types of data. For example,
create index linestr3d1_idx1 on linestr3d1(b, ls2);

Data on the index can be selected using where clauses as regular table.
Time Series Data Management

JaguarDB Time Series

Normally time series is a series of data points indexed in time order. In JaguarDB, the time series has a different meaning: it is both a sequence of data points and a series of tick tables holding aggregated data values at specified time spans. For example, a time series table in JaguarDB can have a base table storing data points in time order, and tick tables such as 5 minute, 15 minute, hourly, daily, weekly, monthly tables to store aggregated data within these time spans.

When a time series table is created, the tick tables are created automatically. The tick tables are used to quickly query aggregated data without doing lengthy computations. As a result, queries for aggregated values in different time periods is extremely fast.

Creating Time Series Tables

The following formats describe commands to create a time series table:

```
create table timeseries(TICK:RETENTION, TICK:RETENTION, ...|BASERENTION)
BASETABLE (key: KEYCOL1, KEYCOL2, ..., value: col rollup VTYPE, ...);
```

Where:

TICK:RETENTION specifies a tick type and retention period of the tick table;
BASERENTION represents the retention period of the base table;

BASETABLE is the name of the base table;

KEYCOL1, KEYCOL2, … are the key columns in the base table;

Rollup specifies the columns whose values will be rolled up to the tick tables;

VTYPE is the type of the column to be rolled up.

The TICK keyword starts with a number and a period type. For example, 15s means a tick table of 15 seconds; 30m means a tick table of 30 minutes.

The letter ‘s’ indicates TICKs in seconds.

The letter ‘m’ indicates TICKs in minutes.

The letter ‘h’ indicates TICKs in hours.

The letter ‘d’ indicates TICKs in days.

The letter ‘w’ indicates TICKs in weeks.

The letter ‘M’ indicates TICKs in months.

The letter ‘q’ indicates TICKs in quarters.

The letter ‘y’ indicates TICKs in years.

The letter ‘D’ indicates TICKs in decades.

Valid TICKs in seconds scale include: 1s, 2s, 3s, 5s, 6s, 10s, 12s, 15s, 20s, 30s.

Valid TICKs in minutes include: 1m, 2m, 3m, 5m, 6m, 10m, 12m, 15m, 20m, 30m.

Valid TICKs in hours include: 1h, 2h, 3h, 4h, 6h, 8h, 12h.

Valid TICKs in days include: 1d, 2d, 3d, 4d, 5d, 6d, 7d, 10d, 15d.

Valid TICKs in weeks include: 1w, 2w, 3w, 4w.

Valid TICKs in months include: 1M, 2M, 3M, 4M, 6M.

Valid TICKs in quarters include: 1q, 2q.

Valid TICKs in years can be any number of years.
Valid TICKs in decades can be any number of decades.

Multiple TICKs are allowed in the same TICK group. For example, you can have 5m and 15m tables, and 1d and 10d tick tables.

The format for the RETENTION is the same as the TICK format, except that it can have any number of retention periods. The RETENTION specifies how long the data points in the base table should be kept. Examples of RETENTION are 15d, 1M, 3M, 1y, etc. If no RETENTION is provided, the data points in the tick table are not deleted. If the retention period is passed, old data will be deleted from the tick tables.

The BASERETENTION specifies how long the data points in the base table should persist. Data points that are older than the retention period are deleted frequently. If no BASERETENTION is provided, the data points in the base table will not be deleted.

A rollup column in a base table indicates that its value will be rolled to the tick tables. In the tick tables the last stored value of the rollup column is saved from the base table. In addition, aggregated values of ‘sum’, ‘min’, ‘max’, ‘avg’, ‘var’ of the column are computed and stored in the tick tables.

The type ‘sum’ indicates that the rollup column in the base table is aggregated into the tick table by taking the cumulative value of the column.

The type ‘min’ indicates that the rollup column in the base table is aggregated to the tick table by taking the minimum value of the column.

The type ‘max’ indicates that the rollup column in the base table is aggregated to the tick table by taking the maximum value of the column.

The type ‘avg’ indicates that the rollup column in the base table is aggregated to the tick table by taking the average value of the column.

The type ‘var’ indicates that the rollup column in the base table is aggregated to the tick table by taking the variance of the column. The variance is the squared value of the standard deviation.

A tick table has the following columns, which are called the heap columns:

    col::sum  -- represents the total sum of the column ‘col’;
col::min -- represents the minimum value of the column ‘col’;
col::max -- represents the maximum value of the column ‘col’;
col::avg -- represents the average of the column ‘col’;
col::var -- represents variance value of the column ‘col’;

The standard deviation of a column can be obtained by taking the square root of the variance. All these statistical values are computed automatically during data ingestion for quick data analysis.

The heap columns hold statistical values of a rollup column. A rollup column is the column on which the aggregating operations are carried out.

VTYPE is the type of the column to be rolled up. The type can only of numerical types: tinyint, smallint, int, bigint, float, and double. In the tick tables, the integer types become bigint type, and the float or double types become double type. String, date, time, location or other columns cannot be rolled up to the tick tables.

The base table can include columns of any type, whose values are not rolled up to the tick tables if they are not marked with the keyword ‘rollup’. Only the columns with the rollup property are rolled up to the tick tables. Without any retention specifications, the create command looks like this:

```
create table timeseries(TICK, TICK, ...)
BASETABLE (key: KEYCOL1, KEYCOL2, ..., value: col rollup VTYPE, ...);
```

It is possible to provide or omit the retention for tick tables and base table independently. That is, one table (base or tick able) may have a retention while others may not have a retention.

BASETABLE is the name for the base table. The tick tables will have a name that include the name of the base table and the TICK type. For example, if the timeseries has 15m and 1d, then the table name for 15m TICK will be BASETABLE@15m; the table name for the 1d TICK will be BASETABLE@1d. The tick tables have the key columns and the rollup columns from the base table. For example, if the base table has a name “traffic”, then the tick tables will have
names “traffic@15m” and “traffic@1d” which all can be queried directly for time series data analysis.

Examples of Time Series Tables

Food Delivery Time Series

The following command will create a base table named “delivery”, and two ticks:

```sql
create table timeseries(1M:1y,1y)
delivery (key: ts timestamp, courier char(32), customer char(32),
    value: meals rollup bigint, addr char(128) );
```

The base table “delivery” has no retention, so its records are persisted forever unless explicitly deleted by the user. One TICK is of monthly (M), having a retention period of one year (y). Another TICK is of yearly (y) with no retention (kept forever). The column ‘meals’ will be rolled up to ticks ‘1M’ and ‘1y’, but the column ‘addr’ is not rolled up to the ticks. Key columns are always rolled up to the tick tables.

Traffic Monitoring Time Series

The following command will create a base table named “traffic”, and five ticks:

```sql
create table timeseries(15m:3h,1h:48h,1d:3M,1q:3y)
traffic (key: ts timestamp, line char(32),
    value: volume rollup bigint,
    driver char(32)
);
```

The base table “traffic” has a retention of 3 years, so its records are persisted for 3 years and any older records will be deleted. The column ‘volume’ is used to collect the traffic volume since the last observation time. The value of this column will be rolled up to all the tick tables. One tick ‘15m’ is to aggregate data for every 15 minutes, persisted for a total of 3 hours. Tick ‘1h’ is to
aggregate passengers by hourly, with retention of 48 hours. Tick ‘1d’ stores aggregated data from the base table in a daily schedule, persisting for 3 months. Tick ‘1q’ stores quarterly (3 months) aggregated data, without any retention (meaning persisting forever).

In the traffic base table, timestamp ‘ts’ is the leading key, line (which can be a bus line, train line, or any transportation vehicle line number) is the second key. The key columns will all be rolled up to the tick tables for fast data searching. The column ‘volume’ is also rolled up to the tick tables as data of interest for time series data analysis. The driver’s name represented by the column ‘driver’ will not be present in any of the ticks. Except the key columns, columns that are not marked ‘rollup’ will not be present in any of the ticks. Importantly, only numeric columns (such as int, bigint, float, double) can be rolled up to the tick tables, because it does not make sense to aggregate non-numeric fields.

**IoT Sensor Time Series**

JaguarDB can manage data generated by IoT (Internet of Things) sensors. To monitor individual sensors and their captured data, the following time series table structure can be used:

```
create table timeseries(5m:1d,1h:48h,1d:3M,1M:20y|5y)
sensorstat (key: sensorID char(16), ts timestamp,
    value: temperature rollup float,
    pressure rollup float,
    windspeed rollup float,
    rpm  rollup float,
    fuel rollup float,
    model char(16),
    type  char(16)
);
```

The base table “sensorstat” has a retention of 5 years, so its records are persisted for 5 years and older records will be deleted. The column ‘temperature’ is used to measure the temperature at the location of the sensor. The value of this column will be rolled up to the tick tables. The column ‘pressure’ measures the pressure at the location of the sensor. The value of this column will also be rolled up to the tick tables. The column ‘windspeed’ measures the wind speed at the location of the sensor. The value of this column will also be rolled up to the tick tables. The
column ‘rpm’ is used to measure the revolutions per minute (RPM) of an engine at the location of the sensor. The value of this column will be rolled up to the tick tables. The column ‘fuel’ measures fuel consumption at the location of the sensor. The value of this column will be rolled up to the tick tables. The columns ‘model’ and ‘type’ records the model and type of the sensor or the device that the sensor is equipped for. The last two columns, however, are not rolled up to the ticks.

The tick tables will have keys and rollup columns in the base table, and the aggregated heap columns. For example, the tick table ‘sensorstat@1d’ has the following structure:

```
table test.sensorstat@1d

  (key:
    sensorid char(16),
    ts datetimesec,
  value:
    temperature float(36.6),
    temperature::sum double(40.10),
    temperature::min double(40.10),
    temperature::max double(40.10),
    temperature::avg double(40.10),
    temperature::var double(40.10),
    pressure float(36.6),
    pressure::sum double(40.10),
    pressure::min double(40.10),
    pressure::max double(40.10),
    pressure::avg double(40.10),
    pressure::var double(40.10),
    windspeed float(36.6),
    windspeed::sum double(40.10),
```
In this time series, since the sensorID is the first key, looking up various data associated with a sensor is very fast. The ticks contain aggregate values in different time windows so that data analysis in various time windows can be completed very quickly without conducting full scan of tables nor doing complex computations.
Base Table and Ticks

In JaguarDB, a base table stores the detailed time series data. A tick (or a tick table) stores aggregated data from the base table according to the predefined tick length. When a time series table is created, the associated tick tables are automatically created. When time series data is written to the base table, the aggregated data is also automatically written to all the tick tables. User can then directly query the tick tables for data in different time windows.

For example, in the above sensorstat time series table, when this table is created, the following tick tables are also created:

sensorstat@5m
sensorstat@1h
sensorstat@1d
sensorstat@1M
They can be show in the jag client program:

```
jaguar:mydb> desc sensorstat;
table timeseries(5m:1d,1h:48h,1d:3M,1M:20y|5y) test.sensorstat
  (  
    key:
      sensorid char(16),
      ts timestamp,
    value:
      temperature rollup float(36.6),
      pressure rollup float(36.6),
      windspeed rollup float(36.6),
      rpm rollup float(36.6),
      fuel rollup float(36.6),
      model char(16),
      type char(16),
      spare_ char(76),
  );

jaguar:mydb> desc sensorstat@5m;
table test.sensorstat@5m|1d
  (  
    key:
      sensorid char(16),
      ts datetimesec,
    value:
      temperature float(36.6),
      temperature::sum double(40.10),
      temperature::min double(40.10),
      temperature::max double(40.10),
      temperature::avg double(40.10),
      temperature::var double(40.10),
      pressure float(36.6),
```
Table test.sensorstat@1h|48h
(
key:
    sensorid char(16),
    ts datetimesec,
);
value:

temperature float(36.6),
temperature::sum double(40.10),
temperature::min double(40.10),
temperature::max double(40.10),
temperature::avg double(40.10),
temperature::var double(40.10),
pressure float(36.6),
pressure::sum double(40.10),
pressure::min double(40.10),
pressure::max double(40.10),
pressure::avg double(40.10),
pressure::var double(40.10),
windspeed float(36.6),
windspeed::sum double(40.10),
windspeed::min double(40.10),
windspeed::max double(40.10),
windspeed::avg double(40.10),
windspeed::var double(40.10),
rpm float(36.6),
rpm::sum double(40.10),
rpm::min double(40.10),
rpm::max double(40.10),
rpm::avg double(40.10),
rpm::var double(40.10),
fuel float(36.6),
fuel::sum double(40.10),
fuel::min double(40.10),
fuel::max double(40.10),
fuel::avg double(40.10),
fuel::var double(40.10),
counter bigint DEFAULT '1',


spare_char(386),
);

Jaguar:mydb> desc sensorstat@1M;
table test.sensorstat@1M|20y ( key: 
    sensorid char(16),
    ts datetimesec,
value: 
    temperature float(36.6),
    temperature::sum double(40.10),
    temperature::min double(40.10),
    temperature::max double(40.10),
    temperature::avg double(40.10),
    temperature::var double(40.10),
    pressure float(36.6),
    pressure::sum double(40.10),
    pressure::min double(40.10),
    pressure::max double(40.10),
    pressure::avg double(40.10),
    pressure::var double(40.10),
    windspeed float(36.6),
    windspeed::sum double(40.10),
    windspeed::min double(40.10),
    windspeed::max double(40.10),
    windspeed::avg double(40.10),
    windspeed::var double(40.10),
    rpm float(36.6),
    rpm::sum double(40.10),
    rpm::min double(40.10),
rpm::max double(40.10),
rpm::avg double(40.10),
rpm::var double(40.10),
fuel float(36.6),
fuel::sum double(40.10),
fuel::min double(40.10),
fuel::max double(40.10),
fuel::avg double(40.10),
fuel::var double(40.10),
counter bigint DEFAULT '1',
spare_ char(386),
);

The keyword “20y” in “table test.sensorstat@1M|20y” means the retention period for tick
sensorstat@1M is 20 years. The string “test” means the table is created in the “test” database. It
should be noticed that the tick tables contain only the key columns and the rollup columns.

In addition, there is an extra column ‘counter’ added automatically in the tick tables. It tracks
the number of records in a period of time. In a tick table, the datetime or timestamp key column
will not have the detailed date time value. Instead. The data time column will have values at the
start of the tick. For example, in an hour tick table, the date time key column will not have
minutes and seconds. It will have time values that are rounded to hours, e.g. “2022-10-12
13:00:00”, “2022-08-15 16:00:00”.

Inserting Data Into Time Series Tables

A user can insert data into the base tables just like any other tables. Data in tick tables will be
automatically prepared and inserted by JaguarDB. The following example shows how to insert
data into the base table:

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'dronel-sid1', '20.0', '35.5', '30.2', '1300', '1.3', 'AA212', 'DH' );
insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model,
type ) values ( 'dronel-sid1', '20.5', '35.8', '30.7', '1320', '1.5', 'AA212', 'DH' );
insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model, type) values ('drone1-sid2', '21.0', '35.7', '30.8', '1304', '1.2', 'AA213', 'DH');
insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model, type) values ('drone2-sid1', '22.0', '36.4', '30.3', '1404', '2.2', 'AB213', 'DF');

Data from different sensors which may be attached to different devices can be stored in the base table 'sensorstat'. Here the key column 'ts' is omitted and a default client's local time will be inserted automatically.

When the base table is populated, the four tick tables are automatically populated:

- sensorstat@5m -- will have data aggregated every 5 minutes
- sensorstat@1h -- will have data aggregated every hour
- sensorstat@1d -- will have data aggregated every day
- sensorstat@1M -- will have data aggregated every month

Reading Data From Time Series Tables

Reading From Base Table And Tick Tables

A user can read data from the base tables as well as the tick tables just like any other tables. The following example shows how to read data from the base table:

```
> select sensorid, ts, temperature, pressure, rpm from sensorstat where sensorid='drone1-sid1';
```

```
> select sensorid, ts, temperature, pressure, rpm from sensorstat where sensorid='drone1-sid2';
```

The following example shows how to read aggregated data from the tick tables:
> select sensorid, ts, temperature::avg, pressure::avg, rpm::max from sensorstat@1d
where sensorid='drone1-sid1';

sensorid=[drone1-sid1] ts=[2021-03-25 01:00:00] temperature::avg=[20.0]
pressure::avg=[35.5] rpm::max=[1300.0]

> select sensorid, ts, temperature::avg, pressure::avg, rpm::max from sensorstat@1d
where sensorid='drone1-sid2';

sensorid=[drone1-sid2] ts=[2021-03-25 01:00:00] temperature::avg=[21.0]
pressure::avg=[35.7] rpm::max=[1304.0]

> select sensorid, ts, temperature::avg, pressure::avg, rpm::min, rpm::max from sensorstat@1d where sensorid='*';

sensorid=['*'] ts=[2021-03-25 01:00:00] temperature::avg=[20.875] pressure::avg=[35.85]
rpm::min=[1300.0] rpm::max=[1404.0]

Note that the condition sensorid='*' can be used to select data for all the possible values of the
sensorid column. Only on key columns can a user apply the '*' condition. This predicate is
described below in detail.

Grouping Data In Windows

The window(length, column) function takes a datetime (including time of different
granularities) column and breaks the column into time windows. The length argument
represents the length of time for the windows, and column is a table column name. The window
column must have a time type. The following examples demonstrate how to make queries
based on the time windows:

```
select pickup_datetime, window(5m, pickup_datetime)
from rides
where date(pickup_datetime)='2021-02-11';
```

The above query breaks the pickup_datetime into a series of 5 minutes intervals and gets the
start time of the intervals in the day of '2021-02-11'. The window() function can appear anywhere
in the query statement but only one is required.

The get aggregated values of numeric columns in a table, the group by clause can be used to
get the aggregated values in each of the time windows:
The window function creates time windows of 5 minutes based on the column ‘pickup_datetime’. The average of value of total_amount is taken in the 5 minute windows by the group by method.

### All Key Values in Tick Table

JaguarDB precomputes values for the ‘*’ condition of every key column in the tick table. Also, all combinations of the ‘*’ value of each key column are calculated. For example: If there are three key columns (named A, B, C for instance), then the following combinations are prepared:

<table>
<thead>
<tr>
<th>Key A</th>
<th>Key B</th>
<th>Key C</th>
<th>Value Rollup Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>B</td>
<td>C</td>
<td>V1, V2, V3, ...</td>
</tr>
<tr>
<td>A</td>
<td>*</td>
<td>C</td>
<td>V1, V2, V3, ...</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>*</td>
<td>V1, V2, V3, ...</td>
</tr>
<tr>
<td>A</td>
<td>*</td>
<td>*</td>
<td>V1, V2, V3, ...</td>
</tr>
<tr>
<td>*</td>
<td>B</td>
<td>*</td>
<td>V1, V2, V3, ...</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>C</td>
<td>V1, V2, V3, ...</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>V1, V2, V3, ...</td>
</tr>
</tbody>
</table>

Selection of data can choose all combinations of the key values. For example, the following select patterns can be applied:

```sql
select * from ticktable where A='*' and B='value-of-B';
select * from ticktable where A='*' and B='value-of-B' and C='value-of-C';
select * from ticktable where A='value-of-A' and B='*' and C='value-of-C';
select * from ticktable where A='*' and B='*' and C='value-of-C';
select * from ticktable where A='*' and B='*';
select * from ticktable where A='*' and B='*' and C='*';
```
A condition with value of "*" outputs only a single record, instead of multiple records of all possible values. The following query:

```sql
select * from ticktable where A='*' and B='*';
```

may outputs records of all possible values of key C. However, the following query:

```sql
select * from ticktable where A='*' and B='*' and C='*';
```

will outputs a single record, if such data exists in the table. The column values are aggregated in runtime under the key entry of "*" for all possible values of the key column. Queries for aggregation values are fast because only a single record is read to retrieve the aggregated data without scanning the tables to get the result.

### Indexes of Time Series Tables

Automatically managing the aggregation tick tables in JaguarDB is not the end of the story. JaguarDB also enables a user to create indexes on the timeseries tables for flexible queries, then JaguarDB will automatically create index records and apply them on the tick tables. The following paragraphs demonstrate how indexes are created and used. For example, suppose we have inserted data records into the base table 'delivery':

```sql
insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JohnDoe', '3', '110 A Street, CA 90222' );
insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JaneDoe', '5', '110 B Street, CA 90001' );
insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'MaryAnn', '3', '100 C Street, CA 92220' );
insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'PaulD', '12', '550 Ivy Road, CA 90221' );
```

```sql
> select * from delivery;

<table>
<thead>
<tr>
<th>ts</th>
<th>courier</th>
<th>customer</th>
<th>meals</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-03-14 20:51:06.457043</td>
<td>QDEX</td>
<td>JohnDoe</td>
<td>3</td>
<td>110 A Street, CA 90222</td>
</tr>
<tr>
<td>2021-03-14 20:51:41.282601</td>
<td>QDEX</td>
<td>JaneDoe</td>
<td>5</td>
<td>110 B Street, CA 90001</td>
</tr>
</tbody>
</table>
```

Indexes of Time Series Tables
Then if we read the data from the tick tables, we see column statistics are shown:

```sql
> select * from delivery@1M;
```

```plaintext
<table>
<thead>
<tr>
<th>ts</th>
<th>courier</th>
<th>customer</th>
<th>meals</th>
<th>meals::sum</th>
<th>meals::min</th>
<th>meals::max</th>
<th>meals::avg</th>
<th>meals::var</th>
<th>counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-03-14</td>
<td>QSEND</td>
<td>MaryAnn</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2021-03-14</td>
<td>QSEND</td>
<td>PaulD</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

```sql
```plaintext
<table>
<thead>
<tr>
<th>ts</th>
<th>courier</th>
<th>customer</th>
<th>meals</th>
<th>meals::sum</th>
<th>meals::min</th>
<th>meals::max</th>
<th>meals::avg</th>
<th>meals::var</th>
<th>counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-03-01</td>
<td>QSEND</td>
<td>MaryAnn</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2021-03-01</td>
<td>QSEND</td>
<td>PaulD</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

```sql
> select * from delivery@ly;
```

```plaintext
<table>
<thead>
<tr>
<th>ts</th>
<th>courier</th>
<th>customer</th>
<th>meals</th>
<th>meals::sum</th>
<th>meals::min</th>
<th>meals::max</th>
<th>meals::avg</th>
<th>meals::var</th>
<th>counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-03-01</td>
<td>QSEND</td>
<td>MaryAnn</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2021-03-01</td>
<td>QSEND</td>
<td>PaulD</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```
The leading key column is the timestamp in the base and tick tables. If we want to look up records by courier names, we can create an index using the courier column as the leading column in the index:

create index delivery_index_courier on delivery(courier, customer, meals);

After the create command, the index for the base table can be directly queried:

> select * from delivery_index_courier;


The above “create index” command creates only an index on the base table, not indexes on the tick tables. Recall that a timeseries table is a cluster of tables that include the base table and a series of tick tables. In this case, only one index is created on the basetable. However, if a user wishes to create a cluster of indexes on the timeseries tables, then the following command can used to create a cluster of indexes:

create index delivery_index_courier ticks on delivery(courier, customer, meals);

An index named “delivery_index_courier@1M” is created based on the tick table “delivery@1M”. Another index named “delivery_index_courier@1y” is created based on the tick table
“delivery@1y”. The indexes for the tick tables, however, do not contain the heap columns such as meals::min, meals::max, and meals::avg columns, because the index column “meals” is a key in the index. If the meals column is to be treated as a heap column, then the index can be created with the following command:

create index deliv_index_cour ticks on delivery(key: courier, customer, value: meals);

The above command creates a cluster of indexes that include all the statistical values of the field “meals”.

If only certain heap columns were to be tracked by indexes, indexes on tick tables can be created with selected heap columns:

create index delivery_index2_courier on delivery@1M(courier, customer, meals::min, meals::max, meals::sum);

> select * from delivery_index2_courier;

<table>
<thead>
<tr>
<th>courier</th>
<th>customer</th>
<th>meals::min</th>
<th>meals::max</th>
<th>meals::sum</th>
<th>ts</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>3</td>
<td>12</td>
<td>23</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>*</td>
<td>JaneDoe</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>*</td>
<td>JohnDoe</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>*</td>
<td>MaryAnn</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>*</td>
<td>PaulD</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>QDEX</td>
<td>*</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>QDEX</td>
<td>JaneDoe</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>QDEX</td>
<td>JohnDoe</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>QSEND</td>
<td>*</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>QSEND</td>
<td>MaryAnn</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2021-03-01 00:00:00</td>
</tr>
<tr>
<td>QSEND</td>
<td>PaulD</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>2021-03-01 00:00:00</td>
</tr>
</tbody>
</table>

The query below selects all deliveries from all couriers to the customer ‘PaulD’:

> select * from delivery_index2_courier where courier='*' and customer='PaulD';
Like any other indexes in JaguarDB, query by the leading column in the index is very fast because it is ordered first in the storage structure of the index.

Delete Data From Time Series

Normally users store time series data and do not expect to delete the data. However, in case a user wishes to delete records in time series tables, the user can execute the delete command. Records in the base tables are removed but records in the tick tables are not deleted. The user however can execute the delete command on a tick table specifically.

For example:

delete from delivery where courier='QSEND';
select * from delivery;

We can see that records of courier 'QSEND' are deleted in the base table 'delivery', but the tick tables still have them:

> select * from delivery@1M;
ts=[2021-03-01 00:00:00] courier=[*] customer=[*] meals=[12] meals::sum=[23]
ts=[2021-03-01 00:00:00] courier=[*] customer=[JohnDoe] meals=[3] meals::sum=[5]
ts=[2021-03-01 00:00:00] courier=[*] customer=[MaryAnn] meals=[3] meals::sum=[3]
ts=[2021-03-01 00:00:00] courier=[*] customer=[PaulD] meals=[12] meals::sum=[12]


> select * from delivery_index;


How about the records in the index? Are they deleted from the indexes?

> select * from delivery_index_courier;

We can see the records in the index for the base table are deleted.

If we delete the records in the tick tables:

delete from delivery@1M where courier='QSEND';
delete from delivery@1y where courier='QSEND';

then the records in the tick tables and the records in the indexes for the tick tables are deleted:

> select * from delivery@1M;
ts=[2021-03-01 00:00:00]  courier=[*]  customer=[*]  meals=[23]  counter=[4]
ts=[2021-03-01 00:00:00]  courier=[*]  customer=[JaneDoe]  meals=[5]  counter=[1]
ts=[2021-03-01 00:00:00]  courier=[*]  customer=[JohnDoe]  meals=[3]  counter=[1]
ts=[2021-03-01 00:00:00]  courier=[*]  customer=[MaryAnn]  meals=[3]  counter=[1]
ts=[2021-03-01 00:00:00]  courier=[*]  customer=[PaulD]  meals=[12]  counter=[1]
ts=[2021-03-01 00:00:00]  courier=[QDEX]  customer=[*]  meals=[8]  counter=[2]

> select * from delivery@1y;
ts=[2021-01-01 00:00:00]  courier=[*]  customer=[*]  meals=[23]  counter=[4]
ts=[2021-01-01 00:00:00]  courier=[*]  customer=[JaneDoe]  meals=[5]  counter=[1]
ts=[2021-01-01 00:00:00]  courier=[*]  customer=[JohnDoe]  meals=[3]  counter=[1]
ts=[2021-01-01 00:00:00]  courier=[*]  customer=[MaryAnn]  meals=[3]  counter=[1]
ts=[2021-01-01 00:00:00]  courier=[*]  customer=[PaulD]  meals=[12]  counter=[1]
ts=[2021-01-01 00:00:00]  courier=[QDEX]  customer=[*]  meals=[8]  counter=[2]

In general, if the data in a table is deleted, then data in the associated indexes will be deleted.
Truncate Time Series

A base table can be truncated (deleting all records but keeping the table structure) and the indexes associated with the base table will be automatically truncated.

```sql
truncate table delivery;
select count(*) from delivery;
test.delivery has 0 rows
select count(*) from delivery@1M;
test.delivery@1M has 11 rows
select count(*) from delivery@1y;
test.delivery@1y has 11 rows

select count(*) from delivery_index_courier;
test.delivery_index_courier has 0 rows

select count(*) from delivery_index_courier@1M;
test.delivery_index_courier@1M has 11 rows
select count(*) from delivery_index_courier@1y;
test.delivery_index_courier@1y has 11 rows

If a base table is truncated, its tick tables are left untouched. A tick table can be manually truncated and its associated index will be truncated accordingly:

```sql
truncate table delivery@1M;
select count(*) from test.delivery@1M;
test.delivery@1M has 0 rows
select count(*) from delivery_index_courier@1M;
test.delivery_index_courier@1M has 0 rows

truncate table delivery@1y;
select count(*) from test.delivery@1y;
test.delivery@1y has 0 rows
Drop Time Series

If a base table is dropped, all its data are permanently deleted. Also, all its tick tables are dropped, and all associated indexes including the indexes for the base table, and the indexes for the tick tables are all dropped. Be cautious when you want to drop tables.

Space and Time Data Management

The following example illustrates how a user can manage time series data and location-based data in one JaguarDB 'rides' table. The rides table is created by the following command:

```
CREATE TABLE timeseries(5m,30m,1d,1M) rides {
    key:
    pickup_datetime datetimesec,
    dropoff_datetime datetimesec,
    driver_name char(16),
    rate_type char(8),
    payment_type char(1),

    value:
    passenger_count rollup int,
    trip_distance rollup float(8.2),
    pickup_location point(srid:wgs84),
    dropoff_location point(srid:wgs84),
    fare_amount rollup float(8.2),
    tip_amount rollup float(6.2),
    tolls_amount float(6.2),
    total_amount rollup float(8.2),
```
Here the ‘rides’ is the base table, and there are four tick tables created for ticks of five minutes, thirty minutes, one day, and one month. Each rollup column will generate five heap columns in the tick tables. Passenger pickup location and drop off location are represented by points having longitude and latitude coordinates in degrees. Data can be inserted by the following example:

```sql
insert into rides values ( '2021-02-11 09:22:12', '2021-02-11 09:50:42', 'DriverAHM', 'REG', '1', '2', '48.6', point(122.036 37.7), point(122.385 37.622), '56.5', '10.5', '5.0', '72.0' );

insert into rides values ( '2021-02-11 09:32:12', '2021-02-11 09:58:42', 'DriverJHS', 'HYP', '1', '3', '49.2', point(122.035 37.369), point(122.381 37.621), '73.5', '12.5', '5.8', '91.8' );

insert into rides values ( '2021-02-12 09:32:12', '2021-02-12 13:50:42', 'DriverAHM', 'REG', '1', '2', '66.8', point(121.8864 37.3382 ), point(122.382 37.622), '96.1', '20.5', '8.0', '124.6' );
```

With the data we have, we can answer the following questions:

1. How many rides took place on each day?

   ```sql
   select pickup_datetime as day, counter as rides from rides@1d where driver_name='*' and rate_type='*' and payment_type='*';
   ```

   Answer:
   ```
   day=[2021-02-11 00:00:00] rides=[2]
   day=[2021-02-12 00:00:00] rides=[1]
   ```

2. How many rides took place on the day of ‘2021-02-12’?

   ```sql
   select pickup_datetime as day, counter as rides from rides@1d where driver_name='*' and rate_type='*' and payment_type='*' and pickup_datetime='2021-02-11 00:00:00';
   ```

   Answer:
   ```
   day=[2021-02-11 00:00:00] rides=[2]
   ```

3. What is the average fare amount?

   ```sql
   select avg( fare_amount::avg) avg_fare_mount from rides@1M where driver_name='*' and rate_type='*' and payment_type='*';
   ```

   Answer:
   ```
   avg_fare_mount=[75.366667]
   ```

4. What is the average fare amount in February of year 2021?
select pickup_datetime as month, fare_amount::avg avg_fare_mount from rides@1M where driver_name='*' and rate_type='*' and payment_type='*' and pickup_datetime='2021-02-01 00:00:00';

Answer:
month=[2021-02-01 00:00:00] avg_fare_mount=[75.3666666667]

(5) What is the average fare amount for each driver?

select driver_name, avg( fare_amount::avg) avg_fare_mount from rides@1M where driver_name != '*' and rate_type='*' and payment_type='*' group by driver_name;

Answer:
driver_name=[DriverAHM] avg_fare_mount=[76.3]
driver_name=[DriverJHS] avg_fare_mount=[73.5]

(6) How many rides took place for each rate type?

select rate_type, sum(counter) rides from rides@1M where rate_type != '*' and driver_name='*' and payment_type='*' group by rate_type;

Answer:
rate_type=[HYP] rides=[1.0]
rate_type=[REG] rides=[2.0]

(7) What are the monthly average trip distance for all drivers?

select pickup_datetime as month, trip_distance::avg from rides@1M where rate_type='*' and payment_type='*' and driver_name='*';

Answer:
month=[2021-02-01 00:00:00] trip_distance::avg=[54.8666666667]

(8) What are the monthly average trip distance and maximum average distance for each driver?

select driver_name driver, pickup_datetime as month, avg(trip_distance::avg) avg_distance, max(trip_distance::avg ) max_avg_distance from rides@1M where rate_type='*' and payment_type='*' and driver_name != '*' group by driver_name;

Answer:
driver=[DriverAHM] month=[2021-02-01 00:00:00] avg_distance=[57.7] max_avg_distance=[57.7]
driver=[DriverJHS] month=[2021-02-01 00:00:00] avg_distance=[49.2] max_avg_distance=[49.2]

(9) How many rides took place every 5 minutes for the day of '2021-02-11'?

select pickup_datetime time, counter rides from rides@5m where driver_name='*' and rate_type='*' and payment_type='*' and pickup_datetime >= '2021-02-11 00:00::00' and pickup_datetime < '2021-02-12 00:00:00';

Answer:
(10) How many rides on the day of '2021-02-11' originated from within 10 kilometers of Sunnyvale, California in 30 minute buckets?

    select pickup_datetime as day from rides where distance(pickup_location, point(122.035 37.369 ), 'center' ) < 18000;

(11) What is the average total amount by 5 minutes for the day of 2021-02-11?

    select pickup_datetime start5min, window(5m, pickup_datetime), avg(total_amount) avg_total_amount
    from rides
    where date(pickup_datetime)='2021-02-11'
    group by pickup_datetime;

The window function creates time windows of 5 minutes based on the column 'pickup_datetime'. The average is taken in the 5 minute windows by grouping the windows.

Summary

Jaguar is a massive linearly scalable NoSQL database that can be used as a high-performant operational data store (ODS), search engine, indexing engine, or any big data search database. Jaguar has strong support for timeseries data and location-based data. It allows fast data ingestion, indexing, and easy analysis of timeseries data and location data. Jaguar integrates timeseries data, location data, documents into one for the full control of IoT systems.

Reference

Location Data

The following statements are examples of JaguarDB geospatial data management.

drop table if exists geom1;
create table if not exists geom1 ( key: a int, value: pt point(srid:4326), b int );
desc geom1 detail;
insert into geom1 values ( 1, point(22 33), 123 );
insert into geom1 (a, pt, b ) values ( 2, point(22 33), 123 );
insert into geom1 (b, pt, a ) values ( 222, point(22 33), 12 );
select * from geom1;
drop index if exists geom1_idx1 on geom1;
create index geom1_idx1 on geom1(b,pt);
select * from geom1_idx1;

drop table if exists geom2;
create table if not exists geom2 ( key: a int, value: pt1 point, b int, uid uuid, pt2 point(srid:wgs84) );
desc geom2;
insert into geom2 values ( 1, point(22 33), 123, point(99 221) );
insert into geom2 values ( 10, json({"type":"Point", "coordinates": [2,3]}), 123,
json({"type":"Point", "coordinates":[5,9]}) );
insert into geom2 (a, pt1, pt2, b ) values ( 2, point(22 33), point(23 421), 123 );
insert into geom2 (b, pt2, pt1, a ) values ( 222, point(22 33), point(90 21), 17 );
select * from geom2;
drop index if exists geom2_idx1 on geom2;
create index geom2_idx1 on geom2(b,pt2);
select * from geom2_idx1;

drop table if exists geom3;
create table if not exists geom3 ( key: pt1 point, value: b int, uid uuid, a int, pt2 point );
desc geom3;
insert into geom3 values ( point(22 33), 123, 2, point(99 221) );
insert into geom3 (b, pt2, pt1, a ) values ( 2, point(25 33), point(23 451), 153 );
select * from geom3;
drop index if exists geom3_idx1 on geom3;
create index geom3_idx1 on geom3(b,pt2,uid);
select * from geom3_idx1;

drop table if exists d5;
create table if not exists d5 ( key: a int, pt1 point3d, b int, pt2 point3d, value: c int, pt3 point3d, d int, pt4 point3d(srid:wgs84) );
desc d5;
insert into d5 values( 1, point3d(22 33 4), 23, point3d(99 22 1), 244, point3d(8 2 3),
23, point3d(8 2 3) );
insert into d5 values( 2, point3d(32 83 0), 23, point3d(94 82 1), 214, point3d(9 7 2),
23, point3d(1 2 3) );
select * from d5;
drop index if exists d5_idx on d5;
create index d5_idx on d5(pt3, pt4, d, c );
select * from d5_idx;

drop table if exists d6;
create table if not exists d6 ( key: a int, pt1 point, b int, pt2 point, value: c int,
pt3 point, d int, pt4 point3d );
desc d6;
insert into d6 values( 1, point(22 33 ), 23, point(99 1), 244, point(8 3), 234,
point3d(8 2 3) );
insert into d6 values( 2, point(32 83 ), 23, point(94 82 ), 214, point(9 2), 234,
point3d(1 2 3) );
insert into d6 ( pt2, a, b, pt1 ) values ( json({"type":"Point", "coordinates":
[123,321]}), 208, 12, point(91 17) );
insert into d6 ( pt2, a, b, pt1 ) values ( json({"type":"Point", "coordinates":
[124,351]}), 209, 13, point(92 19) );
select * from d6;
drop index if exists d6_idx on d6;
create index d6_idx on d6(pt4, pt3, d, c );
select * from d6_idx;

drop table if exists cir1;
create table if not exists cir1 ( key: a int, c1 circle(4326), b int, c2 circle,
value: c int, c3 circle, d int, c4 circle );
desc cir1 detail;
insert into cir1 values ( 100, circle( 22 33 100), 123, circle(99 22 191), 2133,
circle(99 22 12), 123, circle(88 33 2211 ) );
insert into cir1 values ( 101, circle( 92 33 140), 523, circle(99 42 191), 2133,
circle(99 42 12), 823, circle(38 43 811 ) );
drop index if exists cir1_idx1 on cir1;
create index cir1_idx1 on cir1(c3, d, c2);
select * from cir1_idx1;

drop table if exists sph1;
create table if not exists sph1 ( key: a int, s1 sphere, b int, s2 sphere, value: c int, s3 sphere );
desc sph1 detail;
insert into sph1 values ( 100, sphere( 2 3 4 123), 321, sphere(99 22 33 20000), 321, sphere(99 223 12020 29292) );
insert into sph1 values ( 102, sphere( 2 3 4 123), 321, sphere(99 22 33 20000), 321, sphere(99 223 12020 29292) );
insert into sph1 (s1, b, a, s2 ) values ( sphere( 2 3 4 123), 921, 234, sphere(99 22 33 20000) );
insert into sph1 (s1, b, a, s2 ) values ( sphere( 2 32 5 123), 951, 534, sphere(99 22 33 20000) );
drop index if exists sph1_idx1 on sph1;
create index sph1_idx1 on sph1(c, s2, a, s1);
select * from sph1_idx1;

drop table if exists sq1;
create table if not exists sq1 ( key: a int, s1 square, b int, s2 square, value: c int, s3 square );
desc sq1 detail;
insert into sq1 values ( 100, square( 22 453 22222), 100, square(9 3 123), 299, square(82 332 1212) );
drop index if exists sq1_idx1 on sq1;
create index sq1_idx1 on sq1(s3);

drop table if exists cb1;
create table if not exists cb1 ( key: a int, q1 cube, b int, q2 cube, value: c int, q3 cube );
desc cb1 detail;
insert into cb1 values ( 111, cube( 2 3 4 1233), 1234, cube(233 22 55 9393), 3212, cube(92 92 82 2345) );
select * from cir1;
select * from sph1;
select * from sq1;
select * from cb1;
drop index if exists cb1_idx1 on cb1;
create index cb1_idx1 on cb1(q3,q2, b);
select * from cb1_idx1;

drop table if exists rect1;
create table if not exists rect1 ( key: a int, r1 rectangle, value: c int );
desc rect1 detail;
insert into rect1 values ( 1, rectangle(22 33 88 99), 233 );
insert into rect1 ( c, a, r1 ) values ( 22, 31, rectangle(29 13 48 19) );
drop index if exists rect1_idx1 on rect1;
create index rect1_idx1 on rect1(c, r1);

drop table if exists bx1;
create table if not exists bx1 ( key: a int, b1 box, value: c int, b2 box );
desc bx1 detail;
insert into bx1 values ( 1, box(22 33 44 88 99 123), 233, box(9 9 9 22 22 33) );
insert into bx1 ( c, a, b1 ) values ( 22, 31, box(29 13 48 19 21 12) );
select * from bx1;
select distance( point3d(0 0 0), b2, 'max') as maxdist from bx1;
select distance( point3d(0 0 0), b2, 'min') as mindist from bx1;
drop index if exists bx1_idx1 on bx1;
create index bx1_idx1 on bx1(b2, c);

drop table if exists cyn1;
create table if not exists cyn1  ( key: a int, c1 cylinder, value: c int );
desc cyn1 detail;
insert into cyn1 values ( 1, cylinder(1 2 3 45 88 0.3), 1239 );
insert into cyn1 ( c, c1, a ) values ( 13, cylinder(1 2 3 45 88), 139 );
select * from cyn1;
drop index if exists cyn1_idx1 on cyn1;
create index cyn1_idx1 on cyn1(c);
select * from cyn1_idx1;
drop table if exists cn1;
create table if not exists cn1  ( key: a int, c1 cone, value: c int, c2 cone );
desc cn1 detail;
insert into cn1 values ( 1, cone(1 2 3 45 88), 1239, cone(33 22 44 44 99 0.4 0.3) );
insert into cn1 ( c, c1, a ) values ( 13, cone(1 2 3 45 88), 139 );
select * from cn1;
drop index if exists cn1_idx1 on cn1;
drop index if exists cn1_idx2 on cn1;
create index cn1_idx1 on cn1(c2);
create index cn1_idx2 on cn1(c,c2);
select * from cn1_idx1;
select * from cn1_idx2;

drop table if exists el1;
create table if not exists el1  ( key: a int, c1 ellipse, value: c int, c2 ellipse );
desc el1 detail;
insert into el1 values ( 1, ellipse(1 2 45 88), 1239, ellipse(22 44 44 99) );
insert into el1 ( c, c1, a ) values ( 13, ellipse(2 3 45 88), 139 );
select * from el1;
drop index if exists el1_idx1 on el1;
drop index if exists el1_idx2 on el1;
create index el1_idx1 on el1(c2,c);
create index el1_idx2 on el1(c,c2);
select * from el1_idx1;
select * from el1_idx2;

drop table if exists es1;
create table if not exists es1  ( key: a int, c1 ellipsoid, value: c int, c2 ellipse );
desc es1 detail;
insert into es1 values ( 1, ellipsoid(1 2 3 45 88 99), 1239, ellipse(22 44 44 99) );
insert into es1 ( c, c1, a ) values ( 13, ellipsoid(2 3 4 45 88 99), 139 );
select * from rect1;
select * from bx1;
select * from cyn1;
select * from cn1;
select * from el1;
select * from es1;

drop index if exists es1_idx1 on es1;
drop index if exists es1_idx2 on es1;
drop index if exists es1_idx3 on es1;
create index es1_idx1 on es1(c2);
create index es1_idx2 on es1(c,c2);
create index es1_idx3 on es1(c1,a,c2);
select * from es1_idx1;
select * from es1_idx2;
select * from es1_idx3;

drop table if exists line1;
create table if not exists line1 ( key: a int, c1 line, value: c int, c2 line );
desc line1 detail;
insert into line1 values ( 1, line(1 2, 45 8.3), 1239, line(44 99, 291 9.1 ) );
insert into line1 values ( 3, line(1 20, 4 3), 139, line(4 9, 91 9 ) );
select * from line1;
drop index if exists line1_idx1 on line1;
create index line1_idx1 on line1(c2, c, c1 );
select * from line1_idx1;

drop table if exists line3d2;
create table if not exists line3d2 ( key: a int, c1 line3d, value: c int, c2 line3d );
desc line3d2 detail;
insert into line3d2 values ( 1, line3d(1 2 45 8.3 22 3.3), 1239, line3d(44 99 291 9.1 33 44 ) );
select * from line3d2;
drop index if exists line3d2_idx1 on line3d2;
create index line3d2_idx1 on line3d2(c2, c, c1);
select * from line3d2_idx1;

drop table if exists tri1;
create table if not exists tri1 ( key: t1 triangle, value: a int );
insert into tri1 values ( triangle( 11 33 88 99 21 32), 123 );
insert into tri1 values ( triangle( 31 33 18 99 33 44), 223 );
drop index if exists tri1_idx1 on tri1;
create index tri1_idx1 on tri1( a );
select * from tri1_idx1;

drop table if exists tri31;
create table if not exists tri31 ( key: t1 triangle3d, value: a int );
insert into tri31 values ( triangle3d( 11 33 88 99 23 43 9 8 2), 123 );
insert into tri31 values ( triangle3d( 31 33 18, 99 12 34, 9 9 1), 223 );
drop index if exists tri31_idx1 on tri1;
create index tri31_idx1 on tri1( a );
select * from tri31_idx1;

### queries
select * from cir1 where within(point(10 22), c1);
select * from cir1 where coveredby(point(10 22), c1);
select * from cir1 where contain(c1, point(10 22));
select * from cir1 where cover(c1, point(10 22));

select * from cir1 where within( c1, rectangle(1 2 23 34 0.1) );

select * from cir1 where disjoint( c1, rectangle(1 2 23 34 0.1) );
select * from cir1 where nearby( c1, rectangle(1 2 23 34 0.1), 200 );
select distance( c1, point(22 33), 'center' ) as dist from cir1;
select distance( c1, point(22 33), 'max' ) as dist from cir1;
select distance( c1, point(22 33), 'min' ) as dist from cir1;
select distance( point(22 33), c1, 'center' ) as dist from cir1;
select distance( point(22 33), c1, 'max' ) as dist from cir1;

select distance( point(22 33), c1, 'min' ) as dist from cir1;

select * from cb1 where within( point3d(100 200 300), q1 );

###                                      x  y    z
select * from cb1 where cover( q1, sphere(11 234 234 100) );

###                                        x   y   z    r
select * from cb1 where nearby( q2, sphere(31 434 235 100), 3000 );

select * from cb1 where nearby( q2, ellipsoid(31 434 235 100 200 200), 3000 );

select * from cb1 where nearby( q3, ellipsoid(31 434 235 100 200 300 0.1 0.2), 3000 );

###                                           x   y   z    a   b   c   nx ny

### linestring

drop table if exists linestr1;

create table linestr1 ( key: a int, value: ls1 linestring(wgs84), b int, ls2 linestring );

desc linestr1 detail;

insert into linestr1 values ( 1, linestring( 11 2,2 33 , 33 44, 55 66, 55 66, 77 88 ), 200, linestring( 33 44, 55 66, 8 9 ) );

insert into linestr1 values ( 2, linestring( 11.13 2,2.9 33 , 33 44, 5.5 6.6, 55 66, 77 88 ), 210, linestring( 3.3 4.4, 5.5 6.6, 8.9 9 ) );

insert into linestr1 values ( 2, json({'"type":"LineString","coordinates": [[2,3],[3,4]]} ), 121, json({'"type":"LineString","coordinates": [[2,3],[3,4]]} ) );

select * from linestr1;

select * from linestr1 where within( ls1, square( 10 10 78.1 ) );

create index linestr1_idx1 on linestr1( b, ls2 );

desc linestr1_idx1 detail;

select * from linestr1_idx1;

drop table if exists linestr21;
create table linestr21 ( key: ls1 linestring(wgs84), value: a int );
desc linestr21 detail;
insert into linestr21 values ( linestring( 11 2,2 33 , 33 44, 55 66, 55 66, 77 88 ),
200 );
select * from linestr21;
select * from linestr21 where within( ls1, square( 10 10 78.1 ) );
create index linestr21_idx1 on linestr21( a );
desc linestr21_idx1 detail;
select * from linestr21_idx1;
drop table if exists linestr2;
create table linestr2 ( key: ls1 linestring(wgs84), a int, value: ls2 linestring );
desc linestr2 detail;
insert into linestr2 values ( linestring( 1 2,2 33 , 33 44, 55 66, 55 66, 77 88 ),
200, linestring( 33 44, 55 66, 8 9 ) );
insert into linestr2 values ( linestring( 1.13 2,2.9 33 , 33 44, 5.5 6.6, 55 66, 77
88 ), 210, linestring( 3.3 4.4, 5.5 6.6, 8.9 9 ) );
select * from linestr2;
select * from linestr2 where within( ls1, square( 10 10 78.1 ) );
create index linestr2_idx1 on linestr2( ls2 );
desc linestr2_idx1 detail;
select * from linestr2_idx1;
drop table if exists linestr3;
create table linestr3 ( key: ls1 linestring(wgs84), a int, value: ls2 linestring, b
int );
desc linestr3 detail;
insert into linestr3 values ( linestring( 211 2,2 33 , 33 44, 55 66, 55 66, 77 88 ),
200, linestring( 33 44, 55 66, 8 9 ), 804 );
insert into linestr3 values ( linestring( 211.13 2,2.9 33 , 33 44, 5.5 6.6, 55 66, 77
88 ), 210, linestring( 3.3 4.4, 5.5 6.6, 8.9 9 ), 805 );
select * from linestr3;
select * from linestr3 where within( ls2, square( 10 10 78.1 ) );
select geojson(ls1) from linestr3 where intersect( ls1, square( 10 10 78.1 ) );
select geojson(ls1), geojson(ls2) from linestr3 where intersect( ls1, square( 10 10 78.1 ) ) and intersect( ls2, square( 10 10 1000 ) );
create index linestr3_idx1 on linestr3( b, ls2 );
desc linestr3_idx1 detail;
select * from linestr3_idx1;
drop table if exists linestr3d1;
create table linestr3d1 ( key: ls1 linestring3d(wgs84), a int, value: ls2 linestring, b int );
desc linestr3d1 detail;
insert into linestr3d1 values ( linestring3d( 1 2 2,1 2 33 , 8 33 44, 8 55 66 ),
200, linestring( 33 44, 55 66 ), 804 );
insert into linestr3d1 values ( linestring3d( 1.1 2 2, 2 2.9 3 , 3 3 4, 2 5 6 ),
210, linestring( 3.3 4, 5 6 ), 805 );
insert into linestr3d1 values ( linestring3d( 0 -10 0, 0 10 0, 2 2.9 3 , 3 3 4, 2 5 6 ),
310, linestring( 3.3 4, 5 6 ), 805 );
insert into linestr3d1 values ( linestring3d( 0 -20 0, 0 20 0, 2 2.7 3.8 ), 315,
linestring( 3.3 4.2, 5.1 6.7 ), 808 );
select * from linestr3d1;
select ls2:x, ls2:y from linestr3d1;
select geo:id, geo:col, geo:i, ls2:x, ls2:y from linestr3d1 where ls2:x > 0;
select * from linestr3d1 where within( ls1, cube( 10 10 78.1 ) );
select * from linestr3d1 where intersect( ls1, cube( 10 10 78.1 ) );
select * from linestr3d1 where intersect( ls1, linestring3d( 0 0 -10, 0 0 10, 10 10 78.1 ) );
select geojson(ls1) from linestr3d1 where intersect( ls1, linestring3d( 0 0 -10, 0 0 10, 10 10 78.1 ) );
select geojson(ls2) from linestr3d1 where within( ls2, square( 0 0 1000000 ) );
create index linestr3d1_idx1 on linestr3d1( b, ls2 );
desc linestr3d1_idx1 detail;
select * from linestr3d1_idx1;
drop table if exists lstr;
create table lstr ( key: a int, value: ls linestring );
insert into lstr values ( 1, linestring(0 0, 20 0) );
insert into lstr (ls, a) values ( linestring(0 0, 20 0), 121 );
insert into lstr (a, ls) values (124, linestring(1 1, 20 0));
select * from lstr;
select geojson(ls) from lstr where intersect(ls, linestring(10 -10, 10 10));
select geojson(ls) from lstr where within(ls, square(0 0 10000));

drop table if exists poll1;
create table poll1 (key: a int, value: poll polygon);
insert into poll1 values (1, polygon((0 0, 20 0, 88 99, 0 0)));
insert into poll1 values (21, polygon((0 0, 80 0, 80 80, 0 80, 0 0)));
insert into poll1 values (2, json({"type":"Polygon", "coordinates": [[[0,0], [2,0], [8,9], [0,0]], [[1,2], [2,3], [1,2]]]}));
insert into poll1 values (3, json({"type":"Polygon", "coordinates": [[[0,0], [2,0], [8,9], [0,0]], [[1,2], [2,3], [1,2]]]}));
select * from poll1;
select * from poll1 where intersect(pol, line(0 10 80 10));
select geojson(pol) from poll1 where intersect(pol, linestring(10 -10, 10 10));
select geojson(pol) from poll1 where within(pol, square(0 0 10000));
select geojson(pol) from poll1 where intersect(pol, square(0 0 10000));

drop table if exists pol2;
create table pol2 (key: a int, value: pol2 polygon, pol3 polygon3d, tm timestamp default current_timestamp, ls linestring);
insert into pol2 values(1, polygon((0 0, 2 0, 8 9, 0 0), (1 2, 2 3, 1 2)), polygon3d((1 1 2, 2 2 3, 3 3, 1 1 1), (2 2 2, 3 3 1, 2 2 2)), linestring(30 40, 40 50, 5 6));
insert into pol2 values(2, json({"type":"Polygon", "coordinates": [[[0,0], [2,0], [8,9], [0,0]], [[1,2], [2,3], [1,2]]]}), polygon3d((4 1 2, 2 2 2, 3 3 3, 1 9 1, 4 1 2), (2 2 2, 3 3 1, 8 2 9, 2 2 2)), linestring(30 40, 40 50, 5 6));
select * from pol2;
select geojson(pol3) from pol2 where within(pol3, cube(0 0 0 10000));
select geojson(pol2) from pol2 where within(pol2, square(0 0 10000));
select geojson(pol2) from pol2 where intersect(pol2, square(0 0 10000));

drop table if exists mp;
create table mp (key: a int, value: m1 multipoint, m2 multipoint3d);
desc mp detail;
insert into mp ( m1, a, m2 ) values ( multipoint(1 2, 3 4, 2 1), 100, multipoint3d(1 2 3, 3 4 5, 2 2 1));

insert into mp values (123, multipoint(1 2, 3 4, 2 1), multipoint3d(1 2 3, 3 4 5, 2 2 1));

insert into mp values (125, multipoint(1 2, 3 4, 2 1), json({"type":"MultiPoint", "coordinates": [[1,2,3],[3,4,5]]}));

select * from mp;

drop table if exists mline;
create table mline ( key: a int, value: l1 multilinestring, l2 multilinestring3d );
desc mline detail;
insert into mline values(1, multilinestring((0 0, 2 0, 8 9, 0 0),(1 2, 2 3, 1 2)), multilinestring3d((1 1 1, 2 2 2, 3 3 3),(2 2 2, 3 3 1)));
insert into mline values(1024, multilinestring((1 1, 2 3, 4 5, 4 9)), multilinestring3d((0 0 0, 1 9 9, 11 12 13, 33 32 34)));

insert into mline values(3, json({"type":"MultiLineString","coordinates": [[0,0],[2,0],[8,9],[0,0]],[1,2],[2,3],[1,2]}), multilinestring3d((1 1 1, 2 2 2, 3 3 3),(2 2 2, 3 3 1)));
select * from mline;
select geojson(l1) from mline where intersect(l1, square(0 0 100000));
select geojson(l2) from mline where intersect(l2, cube(0 0 0 100000));

drop table if exists mpg;
create table mpg ( key: a int, value: p1 multipolygon, p2 multipolygon3d );
desc mpg detail;
insert into mpg values(1, multipolygon(((0 0, 2 0, 8 9, 0 0),(1 2, 2 3, 7 8, 1 2))),(0 0, 2 2, 3 3, 0 0)), multipolygon3d(((1 1 1, 2 2 2, 3 3 3, 1 1 1),(2 2 2, 3 3 1, 3 5 6, 2 2 2)));

insert into mpg values(2, multipolygon(((0 0, 2 0, 8 9, 0 0),(1 2, 2 3, 7 8, 1 2))),(0 0, 2 2, 3 3, 0 0)), multipolygon3d(((1 1 1, 2 2 2, 3 3 3, 1 1 1),(2 2 2, 3 3 1, 3 5 6, 2 2 2)));

insert into mpg values(3, multipolygon(((0 0, 2 0, 8 9, 0 0),(1 2, 2 3, 7 8, 1 2))),(0.1 0.2, 2.2 2.2, 5 5, 0.1 0.2));
multipolygon3d(((1 1 1, 2 2 2, 3 3 3, 1 1 1), (2 2 2, 3 3 1, 3 5 6, 2 2 2)));

insert into mpg values(30, json( "type":"MultiPolygon","coordinates": [ [[[4,0], [2,0], [8,9], [4, 0]], [[1, 5], [2, 3],[1, 5]]], [[[4,4], [2,0], [8,9], [4, 4]], [[1, 2], [2, 3],[1, 2]]] )
);

multipolygon3d(((1 1 1, 2 2 2, 3 3 1, 3 5 6, 2 2 2)));

insert into mpg values(32, json( { "type":"MultiPolygon","coordinates": [ [[[4,0], [2,0], [8,9], [4, 0]], [[1, 5], [2, 3],[1, 5]]], [[[4,4], [2,0], [8,9], [4, 4]], [[1, 2], [2, 3],[1, 2]]] } )
);

select * from mpg;
select geojson(p1) from mpg where intersect(p1, square(0 0 100000));
select geojson(p2) from mpg where intersect(p2, cube(0 0 0 100000));

drop table if exists rg2;
create table rg2 ( key: a int, value: dt datetime, d date, t time, r range(datetime) );

insert into rg2 values (1, '2018-10-10 01:01:01', '2018-12-12', '12:11:11', range( '2015-10-10 01:01:01', '2028-10-10 01:01:01' ));
insert into rg2 values (2, '2014-10-10 01:01:01', '2015-12-12', '14:11:11', range( '2010-10-10 01:01:01', '2028-12-31 01:01:01' ));
select * from rg2;
select * from rg2 where within(d, range('2000-10-10', '2030-01-01'));
select * from rg2 where within(t, range('01:01:01', '13:13:11'));
select * from rg2 where within(dt, range('1980-01-1 01:01:01', '2019-08-09 13:13:11'));
select * from rg2 where intersect(r, range('1980-01-1 01:01:01', '2019-08-09 13:13:11'));
select * from rg2 where intersect(r, range('1980-01-1 01:01:01', '1999-08-09 13:13:11'));

drop table if exists pold;
create table pold ( key: a int, value:name char(64), pol polygon(wgs84));
insert into pold values(1, "California", json("type":"Polygon","coordinates":[[[-123.23225,42.006187],[-122.37885,42.01166],[-121.037,41.99523],[-120.00186,41.999348],[-118.71478,38.10128],[-117.4989,37.21934],[-115.85034,35.970596],[-114.63446,35.00118],[-114.63446,34.87521],[-114.47015,34.44801],[-114.136055,34.305607],[-114.25655,34.174164],[-114.41538,34.108437],[-114.53587,33.933174],[-114.497536,33.697666],[-114.52492,33.549799],[-114.72757,33.40739],[-114.66184,33.034958],[-114.52492,33.02948],[-114.72209,32.717297],[-114.04751,32.624187],[-117.126465,32.536556],[-117.24696,32.668003],[-117.25243,32.876125],[-117.32912,33.12259],[-117.47151,33.29785],[-117.7837,33.538837],[-118.18352,33.76339],[-118.26019,33.703144],[-118.41355,33.74148],[-118.39164,33.84007],[-118.5669,34.02713],[-118.802414,33.998898],[-119.21866,34.14678],[-119.27891,34.26727],[-119.55823,34.415146],[[-119.87589,34.40967],[-120.13879,34.47539],[-120.47288,34.44801],[-120.64814,34.579456],[-120.6098,34.85878],[-120.67005,34.902596],[-120.6371,35.099766],[-120.8946,35.247643],[-120.905556,35.45029],[-120.00144,35.461243],[-121.16845,35.63505],[-121.28346,35.674843],[-121.332756,35.78438],[-121.71614,36.195152],[-121.89688,36.315643],[-121.93522,36.638786],[-121.85854,36.6114],[-121.787346,36.803093],[-121.92974,36.978355],[-122.105,36.956448],[-122.33504,37.11528],[-122.41719,37.24125],[-122.400764,37.36174],[-122.51578,37.520573],[-122.51578,37.783466],[-122.32956,37.783466],[-122.406235,38.15042],[[-122.488396,38.112083],[-122.50482,37.931343],[-122.701996,37.893005],[-122.9375,38.029926],[-122.97584,38.265434],[-123.129196,38.451653],[-123.33184,38.565667],[-123.44138,38.698112],[-123.73714,38.95553],[-123.68784,39.032207],[-123.82477,39.366302],[-123.76452,39.552517],[-123.85215,39.83184],[-124.109566,40.105686],[-124.3615,40.25904],[-124.4108,40.43978],[-124.15886,40.877937],[-124.109566,41.025014],[-124.15886,41.14083],[-124.06575,41.442062],[-124.1479,41.715908],[-124.25745,41.78163],[-124.21363,42.000071],[-123.23325,42.006187]]))));

drop table if exists lstrm;
create table lstrm ( key: a int, b int, value: ls linestring(srid:4326,metrics:10) );
insert into lstrm values ( 100, 200, linestring(0 80 100 200 300, 0.1 80.2 300 400 550 600 700, 0.2 80.5 1000 2000 23456, 0.8 80.9 10000 30000) );

select area(pl), area(p2) from mpg;
select dimension(pl), dimension(p2) from mpg;
select geotype(pl) from mpg;
select pointn(ls,1) from lstr;
select pointn(ls,2) from lstr;
select extent(pl) from mpg;
select extent(p2) from mpg;
select startpoint(ls) from lstr;


select endpoint(ls) from lstr;
select isclosed(ls) from lstr;
select isclosed(p1) from mpg;
select numpoints(ls) from lstr;
select numpoints(l2) from mline;
select numsegments(l2) from mline;
select numrings(l2) from mline;
select numrings(p1) from mpg;
select numrings(p2) from mpg;
select numpolygons(p1) from mpg;
select numpolygons(p2) from mpg;
select srid(p1) from mpg;
select srid(p2) from mpg;
select summary(p1) from mpg;
select xmin(ls) from lstr;
select ymin(ls) from lstr;
select xmax(ls) from lstr;
select ymax(ls) from lstr;
select convexhull(p1) from mpg;
select convexhull(pol) from pol1;
select convexhull(ls) from lstr;
select centroid(p1) from mpg;
select volume(po3) from pol2;
select volume(q1), volume(q2) from cb;
select closestpoint( point(1 1 ), ls) from lstr;
select closestpoint( point(1 1 ), pol) from pol1;
select angle(c1, c2) from line1;
select angle(line(0 0, 2 5), c2) from line1;
select angle(line3d(0 0 0, 3 4 5), l) from line3d;
select buffer(c2, 'distance=symmetric:2,join=round:10,end=round,point=circle:20') from line1;
select buffer(c2, 'distance=asymmetric:2,join=miter:10,end=flat,point=square:20') from line1;
select length(c2) from line1;
select perimeter(pol) from pol1;
select perimeter(s1) from sq1;
select equal(s1,s2) from sq1;
select issimple(c2) from line1;
select issimple(pol) from pol1;
select isvalid(pol) from pol1;
select isring(pol) from pol1;
select isring(c2) from line1;
select ispolygonccw(pol) from pol1;
select ispolygoncw(pol) from pol1;
select outerring(pol) from pol1;
select outerrings(p1) from mpg;
select innerrings(p1) from mpg;
select ringn(p1,1) from mpg;
select ringn(p2,2) from mpg;
select ringn(pol,1) from pol1;
select ringn(pol,2) from pol1;
select innerringn(pol,1) from pol1;
select polygonn(p1,1) from mpg;
select unique(c2) from line1;
select union(c1,c2) from line1;
select union(p1,p2) from mpg;
select union(pol,'polygon((0 0, 2 3, 2 4, 8 2, 3 9, 0 0))') from pol1;
select union(pol,polygon((0 0, 2 3, 2 4, 8 2, 3 9, 0 0))) from pol1;
select union(pol,polygon((0 0, 2 3, 2 4, 8 2, 3 9, 0 0))) from pol1;
select topolygon(cl,30) from cir1;
select topolygon(s1,30) from sql;
select topolygon(s1) from sql;
select topolygon(s1) s1pgon, topolygon(s2) s2pgon from sql;
select text(s1) from sq1;
select difference(line(0 0, 2 2), point(2 2) ) df;
select difference(linestring(0 0, 2 2, 3 4), point(2 2) ) df;
select difference(linestring(0 0, 2 2, 3 4, 4 6), line(2 2, 3 4) ) df;
select difference(pol, polygon((0 0, 8 0, 800 800, 80 80, 0 0), (3 4, 4 6, 4 2, 3 4))) df from pol1;
select difference('polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))', pol) df from pol1;
select symdifference(line(0 0, 2 2), point(2 2)) df;
select symdifference(linestring(0 0, 2 2, 3 4), point(2 2)) df;
select symdifference(linestring(0 0, 2 2, 3 4, 4 6), line(2 2, 3 4)) df;
select symdifference(pol, polygon((0 0, 8 0, 800 800, 80 80, 0 0), (3 4, 4 6, 4 2, 3 4))) df from pol1;
select symdifference(pol, polygon((0 0, 8 0, 800 800, 80 80, 0 0), (3 4, 4 6, 4 2, 3 4))) df from pol1;
select symdifference('polygon((0 0, 8 0, 800 800, 80 80, 0 0), (3 4, 4 6, 4 2, 3 4))', pol) df from pol1;
select intersection('polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))', 'polygon((1 1, 9 1, 9 9, 1 9, 1 1))') dd;
select intersection('polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))', p1) dd from mpg;
select intersection(p1, 'polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))') dd from mpg;
select intersection('polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))', p2) dd from mpg;
select union('polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))', 'polygon((1 1, 9 1, 9 9, 1 9, 1 1))') dd;
select union('polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))', p1) dd from mpg;
select union(p1, 'polygon((0 0, 8 0, 8 8, 0 8, 0 0), (3 4, 4 6, 4 2, 3 4))') dd from mpg;
select isconvex(pol) from pol1;
select interpolate(ls, 0.5) from lstr;
select linesubstring(ls, 0.2, 0.8) from lstr;
select locatepoint(ls, point(3 9)) from lstr;
select addpoint(ls, point(234 219), 2) from lstr;
select setpoint(ls, point(234 219), 1) from lstr;
select removepoint( ls, 2 ) from lstr;
select reverse( ls ) from lstr;
select scale( ls, 3 ) from lstr;
select scale( ls, 10, 20 ) from lstr;
select scaleat( ls, point(10 20), 10 ) from lstr;
select scaleat( ls, point(10 20), 10, 20 ) from lstr;
select scalesize( ls, 10 ) from lstr;
select scalesize( ls, 10, 20 ) from lstr;
select translate( ls, 10, 20 ) from lstr;
select transscale( ls, 200, 300, 10 ) from lstr;
select transscale( ls, 200, 300, 10, 20 ) from lstr;
select rotate( ls, 180 ) from lstr;
select rotate( ls, 1.0, 'radian' ) from lstr;
select rotateself( ls, 180 ) from lstr;
select rotateself( s1, 180 ) from sq1;
select rotateself( s1, 1.80, 'radian' ) from sq1;
select rotateat( s1, 1.80, 'radian', 100, 300 ) from sq1;
select rotateat( ls, 1.80, 'radian', 100, 300 ) from lstr;
select affine( ls, 1, 2,3, 4, 500, 600 ) from lstr;

select ls:x, ls:y, ls:m1, ls:m2, ls:m3, ls:m4 from lstrm where a < 10000;
select voronoipolygons(tomultipoint(ls) ) vp from lstrm;
select voronoipolygons(tomultipoint(ls,100) ) vp from lstrm;
select voronoipolygons(tomultipoint(ls),100,bbox(0 80 0.2 80.2) ) vp from lstrm;
select voronoilines(tomultipoint(ls) ) VL from lstrm;
select voronoilines(tomultipoint(ls),100) ) VL from lstrm;
select voronoilines(tomultipoint(ls),100,bbox(0 80 0.2 80.2) ) VL from lstrm;
select delaunaytriangles(tomultipoint(ls) ) dt from lstrm;
select delaunaytriangles(tomultipoint(ls,100) ) dt from lstrm;

select geojson(ls) from lstrm;
select geojson(ls, 10000) from lstrm;
select geojson(c1, 10000,300) from cir1;
select tomultipoint(ls) from lstrm;
select tomultipoint(c1, 300) from cir1;
select wkt(ls) from lstrm;
select minimumboundingcircle(ls) from lstrm;
select minimumboundingsphere(pt3) from d5 where a < 1000;
select isonleft(point(30 40), ls) from lstrm;
select leftratio(point(30 40), ls) from lstrm;
select isonright(point(30 40), ls) from lstrm;
select rightratio(point(30 40), ls) from lstrm;
select knn(ls, point(30 40), 10) from lstrm;
select knn(ls, point(30 40), 10, 10, 100) from lstrm;
select metricn( ls, 2 ) from lstrm;
select metricn( ls, 2, 3 ) from lstrm;

Timeseries Data

The following statements demonstrate timeseries data management.

create table timeseries(5m) tc1 (  key: k1 int, c1 char(2), ts timestamp, value: v1 rollup int, v2 int );
create table timeseries(5m) ts1 (  key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts1 ( k1, v1, v2 ) values ('5', '103', '247' );
insert into ts1 ( k1, v1, v2 ) values ('5', '303', '253' );
insert into ts1 ( k1, v1, v2 ) values ('5', '503', '553' );
insert into ts1 ( k1, v1, v2 ) values ('5', '903', '153' );
insert into ts1 ( k1, v1, v2 ) values ('6', '10', '29' );
insert into ts1 ( k1, v1, v2 ) values ('6', '100', '29' );
select * from ts1;
select * from ts1@5m;

create table timeseries(10s) ts1002 ( key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts1002 ( k1, v1, v2 ) values ('5', '100', '200' );
select * from ts1002;
select * from ts1002@10s;

drop table ts2;
create table timeseries(5m) ts2 ( key: k1 int, ts timestamp, value: v1 rollup int, v2 int, v3 rollup int );
insert into ts2 values ('5', '2021-02-12 13:35:12', '100', '200', '111' );
select * from ts2;
insert into ts2 values ('5', '2021-02-12 13:35:13', '100', '200', '1123' );
insert into ts2 values ('6', '2021-02-12 13:36:14', '100', '200', '213' );
insert into ts2 values ('7', '2021-02-12 13:36:17', '100', '200', '233' );
insert into ts2 values ('6', '2021-02-12 13:37:17', '100', '200', '322' );
select * from ts2;
select * from ts2@5m;

create index ts2idx1 on ts2(v1, k1);
create index ts2idx2 ticks on ts2(v3, v1, k1);
create index ts2idx3 ticks on ts2(v3, k1, v1);
insert into ts2 (k1, v2 ) values ('10', '243' );
select * from ts2idx1;
select * from ts2idx2;
select * from ts2idx3;

create index ts2idx4 ticks on ts2(key: v3, k1, value: v1);
desc ts2idx4;
select * from ts2idx4;
select * from ts2idx4@5m;
drop table ts2002;
create table timeseries(5m|10m) ts2002 ( key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts2002 values ('5', '2021-02-12 13:35:12', '100', '200' );
insert into ts2002 values ('5', '2021-02-12 13:35:13', '100', '200' );
insert into ts2002 values ('6', '2021-02-12 13:35:14', '100', '200' );
insert into ts2002 values ('6', '2021-02-12 13:36:14', '100', '200' );
insert into ts2002 values ('7', '2021-02-12 13:36:17', '100', '200' );
insert into ts2002 values ('6', '2021-02-12 13:36:17', '100', '200' );
insert into ts2002 values ('6', '2021-02-12 13:37:17', '100', '200' );
insert into ts2002 values ('6', '2021-02-12 13:37:17', '100', '200' );

select * from ts2002;
select * from ts2002@5m;
create table timeseries(1h:0h) ts3 ( key: k1 int, ts timestamp, k2 int, k3 char(10), k4 char(12), k5 char(23), value: b rollup int, c int, c2 int, c3 rollup int, d rollup int, e int, f rollup int );
insert into ts3 values ('1', '2020-12-14 12:12:12', '100', 'k3k', 'k4k', 'k5k', '200', '300', '400', '456', '222', '333', '321' );
insert into ts3 values ('2', '2020-12-14 12:12:12', '100', 'k3k', 'k4k', 'k5k', '200', '300', '400', '456', '222', '333', '321' );
insert into ts3 values ('2', '2020-12-14 12:52:12', '100', 'k3k', 'k4k', 'k5k', '200', '300', '400', '456', '222', '333', '321' );
insert into ts3 values ('3', '2020-12-14 12:52:12', '100', 'k3k', 'k4k', 'k5k', '200', '300', '400', '456', '222', '333', '321' );
insert into ts3 values ('2', '2020-12-14 12:13:12', '101', 'k3k', 'k4k', 'k5k', '202', '304', '400', '457', '223', '353', '421' );
insert into ts3 values ('2', '2020-12-14 12:14:12', '101', 'k3k', 'k4k', 'k5k', '202', '304', '400', '457', '223', '353', '421' );
select * from ts3;
select * from ts3@1h;

drop table ts4;
create table timeseries(1h:0h, 3M:2y ) ts4 ( key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts4 ( k1, v1, v2 ) values ('1', '123', '321' );
insert into ts4 ( k1, v1, v2 ) values ('2', '123', '321' );
insert into ts4 ( k1, v1, v2 ) values ('3', '123', '321' );
insert into ts4 (k1, v1, v2) values ('3', '123', '321');
select * from ts4;
select * from ts4@1h;
alter table ts4 add tick(1d);
alter table ts4 add tick(1D:10D);
alter table ts4 drop tick(1D);
create index ts4_idx1 on ts4(v1, ts);
select * from ts4_idx1;
create index ts4_idx2 on ts4(ts, k1);
select * from ts4_idx2;
alter table ts4 add tick(1q);
select * from ts4@1q;
alter table ts4 retention 0;
alter table ts4 retention 12M;
alter table ts4@3M retention 3y;
desc ts4;
desc ts4@3M;
create table timeseries(1d) ts5 (key: k1 int, ts timestamp, value: v1 rollup int, v2 int);
insert into ts5 values ('5', '2020-12-14 12:13:12', '100', '200');
insert into ts5 values ('5', '2020-12-14 12:14:12', '100', '200');
insert into ts5 values ('5', '2020-12-14 12:15:12', '100', '200');
insert into ts5 values ('5', '2020-12-14 12:16:12', '100', '200');
insert into ts5 values ('6', '2020-12-14 12:17:12', '100', '200');
insert into ts5 values ('6', '2020-12-14 12:18:12', '100', '200');
select * from ts5;
select * from ts5@1d;
```sql
create table timeseries(3d) ts5002 (  key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts5002 values ('5', '2020-12-14 12:13:12', '100', '200' );
insert into ts5002 values ('5', '2020-12-14 12:14:12', '100', '200' );
insert into ts5002 values ('6', '2020-12-14 12:17:12', '100', '200' );
insert into ts5002 values ('6', '2020-12-14 12:18:12', '100', '200' );
insert into ts5002 values ('7', '2020-12-15 12:18:12', '100', '200' );
select * from ts5002;
select * from ts5002@3d;

create table timeseries(1w) ts6 (  key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts6 values ('5', '2020-12-14 12:13:12', '100', '200' );
insert into ts6 values ('5', '2020-12-14 12:14:12', '100', '200' );
insert into ts6 values ('5', '2020-12-14 12:15:12', '100', '200' );
insert into ts6 values ('5', '2020-12-14 12:16:12', '100', '200' );
insert into ts6 values ('5', '2021-02-04 12:16:12', '100', '200' );
select * from ts6;
select * from ts6@1w;

create table timeseries(1month) ts7 (  key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts7 values ('5', '2020-12-14 12:13:12', '100', '200' );
insert into ts7 values ('5', '2021-02-04 12:16:12', '100', '200' );
select * from ts7;
select * from ts7@1M;

create table timeseries(1year) ts8 (  key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts8 values ('5', '2020-12-14 12:13:12', '100', '200' );
insert into ts8 values ('5', '2021-02-04 12:16:12', '100', '200' );
select * from ts8;
select * from ts8@1y;
```
create table timeseries(1decade) ts9 ( key: k1 int, ts timestamp, value: v1 rollup int, v2 int );
insert into ts9 values ('5', '2020-12-14 12:13:12', '100', '200' );
insert into ts9 values ('5', '2021-02-04 12:16:12', '100', '200' );
select * from ts9;
select * from ts9@1D;
create table timeseries(15s:60s|1h) ts10 ( key: ts timestamp, a int, value: b int default '1000', c rollup int default '234' );
insert into ts10 ( a )values ( 100 );
insert into ts10 ( a )values ( 200 );
insert into ts10 ( a )values ( 300 );
insert into ts10 ( a )values ( 400 );
insert into ts10 ( a )values ( 600 );
insert into ts10 ( a )values ( 700 );
insert into ts10 values ( 600 );
select * from ts10;
select * from ts10@15s;
drop table tss1001;
create table tss1001 ( key: ts timestamp, a int, value: b int default '1000' );
insert into tss1001 ( a )values ( 100 );
select * from tss1001;
drop table tss1;
create table tss1 ( key: a int, value: b timestamp );
insert into tss1 values ( 100 );
select * from tss1;
drop table tss2;
create table tss2 ( key: a int, value: b timestampsec );
insert into tss2 values ( 100 );
insert into tss2 values ( 200 );
select * from tss2;
drop table tss3;
create table tss3 ( key: a int, value: b timestampnano );
insert into tss3 values ( 100 );
insert into tss3 values ( 200 );
select * from tss3;
drop table tss4;
create table tss4 ( key: a int, value: b timestampmill );
insert into tss4 values ( 100 );
insert into tss4 values ( 200 );
insert into tss4 values ( 300 );
select * from tss4;
drop table tspace1;
create table timeseries(5m) tspace1 (  key: k1 int, ts timestamp, loc point, k2 int default '23', value: v1 rollup int, v2 int, v3 rollup int );
insert into tspace1 (k1, loc, v2 ) values ('10', point(2 3), '243' );
insert into tspace1 (k1, loc, v2, v1, v3 ) values ('10', point(2 3), '243', '1222', '3456' );
insert into tspace1 (k1, loc, v2, v1, v3 ) values ('11', point(4 5), '643', '2222', '4456' );
insert into tspace1 (k1, loc, v2, v1, v3 ) values ('12', point(4 5), '643', '2222', '4456' );
select * from tspace1;
select * from tspace1@5m;
select * from tspace1@5m where nearby(loc, point(34 12), 100 ) and k1=12;
create index tspace1idx1 on tspace1(v1, k1);
create index tspace1idx2 on tspace1(v3, v1, k1);
create index tspace1idx3 on tspace1(v3, k1, v1);
select * from tspace1idx1;
select * from tspace1idx1@5m;
select * from tspace1idx2;
select * from tspace1idx3;
drop table tspace2;
create table timeseries(5m) tspace2 (  key: k1 int, ts timestamp, loc circle, k2 int default '23', value: v1 rollup int, v2 int, v3 rollup int );
insert into tspace2 (k1, loc, v2, v1, v3) values ('10', circle(2 3 30), '243', '1292', '3456');

insert into tspace2 (k1, loc, v2, v1, v3) values ('11', circle(4 5 50), '643', '2262', '4456');

insert into tspace2 (k1, loc, v2, v1, v3) values ('12', circle(4 5 45), '645', '2422', '4056');

select * from tspace2;
select * from tspace2@5m;
select * from tspace2@5m where nearby(loc, point(34 12), 100) and k1=12;
select * from tspace2@5m where nearby(loc, point(34 30), 40) and k1=12;

create index tspace2idx1 on tspace2(v1, k1);
create index tspace2idx2 on tspace2(v3, v1, k1);
create index tspace2idx3 on tspace2(v3, k1, v1);
select * from tspace2idx1;
select * from tspace2idx1@5m;
select * from tspace2idx2;
select * from tspace2idx3;
select * from tspace2idx3 where v3=4456;

drop table sensorstat;
create table timeseries(5m:1d,1h:48h,1d:3M,1M:20y|5y)
sensorstat (key: sensorID char(16), ts timestamp,
   value: temperature rollup float,
       pressure rollup float,
       windspeed rollup float,
       rpm rollup float,
       fuel rollup float,
       model char(16),
       type char(16)
);

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model, type) values ( 'drone1-sid1', '20.0', '35.5', '30.2', '1300', '1.3', 'AA212', 'DH' );

insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model, type) values ( 'drone1-sid1', '20.5', '35.8', '30.7', '1320', '1.5', 'AA212', 'DH' );
insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model, type ) values ( 'drone1-sid2', '21.0', '35.7', '30.8', '1304', '1.2', 'AA213', 'DH' );
insert into sensorstat (sensorid, temperature, pressure, windspeed, rpm, fuel, model, type ) values ( 'drone2-sid1', '22.0', '36.4', '30.3', '1404', '2.2', 'AB213', 'DF' );

drop table delivery;
create table timeseries(1M:1y,1y)
    delivery (key: ts timestamp, courier char(32), customer char(32),
    value: meals rollup bigint, addr char(128) );

insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JohnDoe', '3', '110 A Street, CA 90222' );
insert into delivery ( courier, customer, meals, addr ) values ( 'QDEX', 'JaneDoe', '5', '110 B Street, CA 90001' );
insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'MaryAnn', '3', '100 C Street, CA 92220' );
insert into delivery ( courier, customer, meals, addr ) values ( 'QSEND', 'PaulD', '12', '550 Ivy Road, CA 90221' );

select * from delivery;
select * from delivery@1M;
select * from delivery@1y;
create index delivery_index_courier on delivery(courier, customer, meals );
select * from delivery_index_courier;
select * from delivery_index_courier@1M;
select * from delivery_index_courier@1M where courier='*' and customer='JohnDoe';
select * from delivery_index_courier@1y;
create index delivery_index2_courier on delivery@1M(courier, customer, meals::min, meals::max, meals::sum );
select * from delivery_index2_courier;
select * from delivery_index2_courier where courier='*' and customer='PaulD';