Jaguar User Manual

(Rev 2.9.2 05/18/2018)

Contents

Introduction

The user manual is an introduction to the highly-scalable and fast NoSQL database. Jaguar is not like any other NoSQL database. Jaguar uses a brand-new data storage model which facilitates fast point query as well as range query from big datasets.

Environment

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

Minimum System Requirements for Jaguar Server

Hardware：CPU 8 Core, 2GHz, 32GB RAM, 1000GB HD
Software： Linux, CentOS 6 or 7, RedHat 6 or 7, Ubuntu,
Windows Server 2012, x86_64
File systems： ext4, XFS, or NTFS

Minimum System Requirements for Jaguar Client

Hardware：CPU 4 Core, 2GHz, 16GB RAM, 512GB HD
Software： Linux CentOS 6 or 7, RedHat 6 or 7, Ubuntu,
Windows 7, x86_64
File systems：ext4, XFS, or NTFS

Jaguar 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”. 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 and Windows 64 bit operating systems. There are Linux and Windows jaguar server and client libraries in the download files.

Linux System

In a Linux 64 bit system (such as Centos 6/7 or Redhat 6/7), you can open a terminal (or putty, xterm, etc.) and saved the downloaded file in any directory.

Windows System

In a Windows server system, please use Msys or Cygwin terminals to install and manage the Jaguar server. If you do not have Msys nor Cygwin terminal, please install Msys1 terminal software by downloading MSYS-1.0.11.exe from the Web. Alternatively, you can click on the included binary software MSYS-1.0.11.exe to install Msys terminal. For example, after you click on MSYS-1.0.11.exe, you can enter c:\msys1 as target installation directory (or using any drive that has enough disk storage space). Then you can copy the Jaguar tar.gz files to your home directory in Msys terminal. If you are running Jaguar in a cluster of servers, please make sure such environment is setup on every server in the cluster.

Jaguar Server and Client Setup

Linux System

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

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

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

```
$ cd jaguar-n.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. If you are on a Windows system which does not have sshd and ssh, then you need to execute the install.sh script individually on each server to install Jaguar.

If you are installing the enterprise version, in the server subdirectory, please execute the reqlicense program and send the output string to your vendor who will send you back a license string.

```
$ cd server
$ ./reqlicense
   2a086efe8a5aa1d09c35a79acb082b552
```

You need to copy the license string received from your vendor to the license.txt file in the same directory so that jaguar server can be started successfully. For the free-trial version, the license.txt file is not used.

```
$ ./ 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:
node1
node2
node3
node4
node5
```

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 Jaguar 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.

Windows System

You are recommended to open a Msys terminal, and execute the script in the tar package:

```
$ ./install.sh
```

This will install Jaguar server and client software on your Windows system. Also you can copy the client lib/JaguarClient.dll file to Windows system32 directory for Java client programs to load the Jaguar client library properly. The format of HOSTFILE is same as in Linux systems. The configuration of conf/license.txt should follow the same procedure as in Linux system.

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. So is the server.conf file (except LISTEN_IP is different in case it is used).

Configuration file $JAGUAR_HOME/*conf/datacenter.conf* is config 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
```

Windows System

On a Windows server which does not have sshd and ssh, you must execute the jaguarstart command on each server.

You may also check the status of Jaguar on a host:

```
$ $JAGUAR_HOME/bin/jaguarstatus
```

All jaguar server processes on all the hosts can be stopped with:

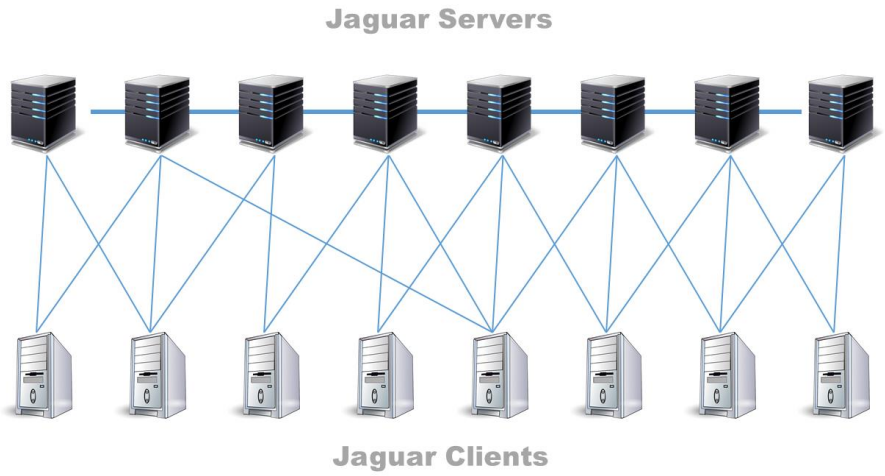
```
$ $JAGUAR_HOME/bin/jaguarstop -all
```

The Jaguar server process on the local host can be stopped with:

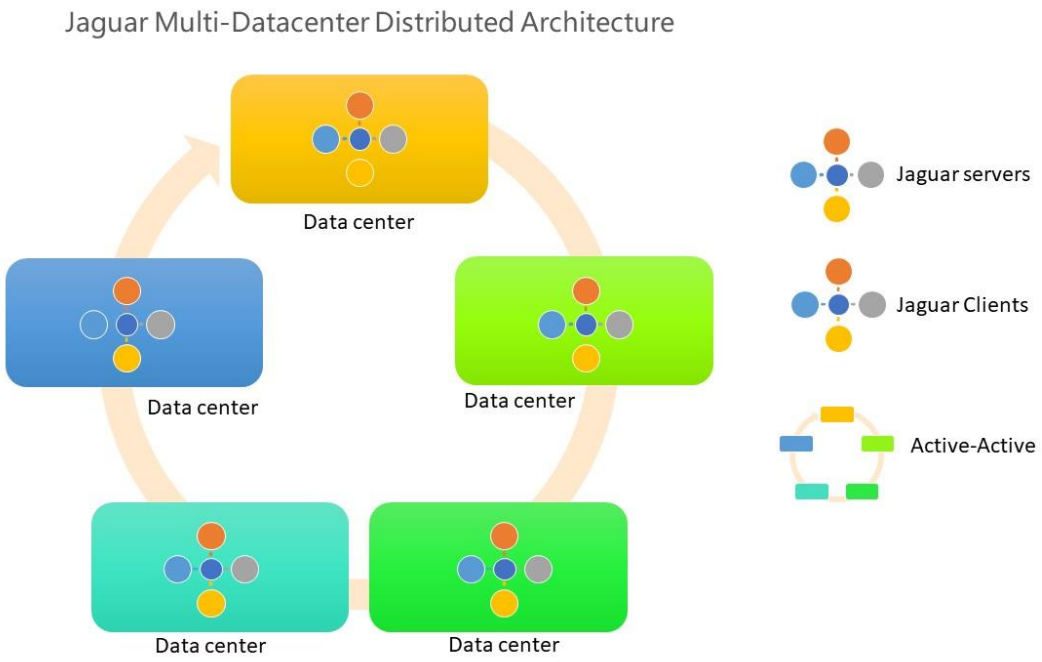
```
$ $JAGUAR_HOME/bin/jaguarstop -s
```

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 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 window 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
$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':
$ 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

$ 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:
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 < 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  –r "10000:0:0:0"  | 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

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

```
#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

```
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: " + val + " 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 );
```

PHP

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
$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:

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

```
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:

```
var homedir=process.env.JAGUAR_HOME;

var libname = homedir + "/jaguar/lib/jaguarnode";

const Jag = require( libname );

var jdb = Jag.JaguarAPI();

jdb.connect("127.0.0.1", 8888, "admin", "jaguar", "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");
end
```

Go

To use Go language client API, you need your project directory, for example myproject. The following steps illustrated how to setup the development environment:

1. `mkdir -p $HOME/myproject/src/jaguargo`
2. `cp  $HOME/jaguar/doc/example.go    $HOME/myproject`
3. `cp  $HOME/jaguar/doc/goexample.sh  $HOME/myproject`
4. `cp $HOME/jaguar/lib/ jaguargo.go   $HOME/myproject/src/jaguargo/`
5. `vi $HOME/myproject/src/jaguargo/jaguargo.go`

   Make sure the location of Jaguar include header files and shared library files are correct:

   ```
   //#cgo CFLAGS: -I/home/jaguar/jaguar/include
   //#cgo LDFLAGS: -L/home/jaguar/jaguar/lib -lJaguarClient -ljaguargo -ldl
   ```

6. Execute `$HOME/myproject/goexample.sh`

```
package main
import(
  "jaguargo"
  "fmt"
  "os"
)

func main() {
   var rc int = 0
   jdb := jaguargo.New()
   rc = jdb.Connect("127.0.0.1", 8888, "admin", "jaguar", "test", "NULL", 0)
```

```
    if rc < 0 {
        fmt.Println("Error connect")
        os.Exit(1)
    }

    jdb.Execute("create table gotab123 ( key: uid char(32), value: addr char(128) )" )
    jdb.Query("show databases" )
    for {
            rc := jdb.Reply()
            if rc > 0 {
                jdb.PrintRow()
            } else {
                break
            }
    }
    jdb.Close()
}
```

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 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
dont compress   = *.gz *.tgz *.zip *.z *.Z *.rpm *.deb *.bz2
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. DateTime
    datetime – 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 2005
    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. So 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:

```
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';
```

11. 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:

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

12. Date

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

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

13. 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
```

14. TimeNano

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

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

15. 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.

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

17. Text
    Text -- is the same as char(1024)

18. TinyText
    TinyText -- is the same as char(256)

19. MediumText
    MediumText -- is the same as char(2048)

20. LongText
    LongText -- is the same as char(10240)

21. Blob
    Blob -- is the same as char(1024)

22. TinyBlob
    TinyBlob -- is the same as char(256)

23. MediumBlob
    MediumBlob -- is the same as char(2048)

24. LongBlob
    LongBlob -- is the same as char(10240)

25. String
    String – is the same as char(64)

26. Varchar
    Varchar(N) – is same as char(N)

27. Bit
    Bit – is one byte column, with value of 1 or 0

28. Eum

    COLUMN enum ('val1', 'val2', 'val3', …) -- A column can take certain values only

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

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:

Jaguar Type	Format	Java Type
bool	bool	boolean
char	char(length)	java.lang.String
char	char(length)	byte[]
int	int	int
smallint	smallint	int
tinyint	tinyint	int
mediumint	mediumint	int
bigint	bigint	long
double	double(m,n)	double
float	float(m,n)	float
timestamp	timestamp (in milliseconds)	java.util.Date
datetime	datetime (in milleseconds)	java.util.Date
datetimenano	datatimenano (in microseconds)	Java.sql.Timestamp
time	time	SimpleDateFormat
timenano	timenano	SimpleDateFormat

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

^ 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'

FUNC(EXPR(COL)):

min(EXPR(COL)) -- minimum value of column expression

max(EXPR(COL)) -- maximum value of column expression

avg(EXPR(COL)) -- average value of column expression

sum(EXPR(COL)) -- sum of column expression

count(1) -- count number of rows

stddev(EXPR(COL)) -- standard deviation of column expression

first(EXPR(COL)) -- first value of column expression

last(EXPR(COL)) -- last value of column expression

abs(EXPR(COL)) -- absolute value of column expression

acos(EXPR(COL)) -- arc cosine function of column expression

asin(EXPR(COL)) -- arc sine function of column expression

ceil(EXPR(COL)) -- smallest integral value not less than column expression

cos(EXPR(COL)) -- cosine value of column expression

cot(EXPR(COL)) -- inverse of tangent value of column expression

floor(EXPR(COL)) -- largest integral value not greater than column expression

log2(EXPR(COL)) -- base-2 logarithmic function of column expression

log10(EXPR(COL)) -- base-10 logarithmic function of column expression

log(EXPR(COL)) -- natural logarithmic function of column expression

ln(EXPR(COL)) -- natural logarithmic function of column expression

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 radian

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

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 END_TIMECOL – BEGIN_TIMECOL.

dayofmonth(TIMECOL) -- the day of the month in a datetime column (1-31)

dayofweek(TIMECOL) -- the day of the week in a datetime column (0-6)

dayofyear(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:

```
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;
```

Jaguar SQL Statements

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

```
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)
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;
      dropdb mydb;
      createuser test;
      dropuser test;
```

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;

   show task;
```

Create command

Commands for creating table and index:

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


create table TABLE ( COL1 TYPE(size), COL2 TYPE(size), ...  );


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 );
```

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.

Insert SQL Commands

```
insert into TABLE (col1, col2, col3, ...) values ( 'va1', 'val2',
intval, ... );
insert into TABLE values ( k1, k2, 'va1', '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');
```

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%';
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.

```
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 fname < 'Zack' and ( zipcode =
94506 or zipcode = 94507);

select * from user where fname >= 'Sam' and zipcode in ( 94506, 94582 );

select * from t1_index where uid='frank380' or uid='davidz';

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;
```

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.

```
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:

```
Getfile col1 size, col2 time, col2 md5 from t123 where …;
Output is: col1.size:[38393] col2.time:[…] col2.md5:[IEdjJDDKKDnxnE]
```

SQL Join Support

Jaguar provides inner join or simply 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];
```

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:

```
alter table TABLE rename OLDKEY to NEWKEY;
Rename a key name in table TABLE.

Example:
   alter table mytable rename mykey1 to userid;
```

Spool command

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

```
spool LOCALFILE;
spool off;

Example:
   spool /tmp/myout.txt;
   (The above command will make the output data to be written to file
    /tmp/myout.txt)

   spool off;
   (The above command will stop writing output data to any file)
```

Change password

A user can change the login password:

```
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 SQ Lfile 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.ql 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:

```
  $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

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

For example:

```
$JAGUAR_HOME/bin/jagimport -d mydb -t salestab
```

Import table data from a SQL file on client side

$ $JAGUAR_HOME/bin/jagimportsql DATABASE.TABLE.sql

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

For example:

```
$JAGUAR_HOME/bin/jagimportsql   mydb.salestab.sql
```

Import table data from a CSV file on client side

$ $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:

```
$JAGUAR_HOME/bin/jagimportcsv -d mydb -t t1 -f mydb.t1.csv
```

Schema Change

Create Extra Columns

When creating a new table, we suggest you create a few spare columns for future use. For example:

```
Create table mytab ( key: k1 char(16), k2 char(16),
                     value:  col1 char(10), spare1 char(8), spare2 char (16), spare3 char(32) );
```

Columns spare1, spare2, spare3 are not used for now but can be used in the future. They could be simply renamed to a more meaningful name.

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 -d DB -h :8888
   jaguar> import into DB.TABLE complete;

Repair Table

Normally the table records in Jaguar database are orderly organized and you do not need to repair a table. However, in unexpected situations, a table might be corrupted (misplaced records but no loss of data). If a table is corrupted, you may see strange results from select or group by operations. If you suspect a table structure is corrupted, you can first run the check command and, if necessary, repair the table.

Check Table

```
$ bin/jag -u admin -p -h localhost:8888
  jaguar> check mydb.mytable123;
  jaguar> quit;
```

The syntax for checking a table is "check DATABASE.TABLE;". You have to provide the database name and table name, connected with a period. If your table is not corrupted, it will tell you so and there is no need to repair it.

Repair Table

If a table is corrupted, you can connect to a server with admin account in exclusive mode (-x yes). Because the repair operation modifies the table records, you need to start with the maintenance mode so there will no conflicts with other clients:

```
$ bin/jag -u admin -p -h localhost:8888 -x yes
   jaguar> repair mydb.mytable123;
   jaguar> quit;
```

The repair command will report progress of the repairing process.

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 increase 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:
   $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. In the future, each new cluster of servers can be added with the same method.

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.**

If you are on Windows system, please install Msys1 terminal which has bash support.

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

## print more debug info
# debug=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:

```
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:

```
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" -> "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 )

drv <- JDBC("com.jaguar.jdbc.JaguarDriver", "/home/jaguar/jaguar/lib/jaguar-jdbc-2.0.jar", "`")

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”

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. Its core data structure allows fast data ingestion, indexing, and query. Jaguar is better than conventional NoSQL database because the keys in Jaguar are globally sorted by order. This unique feature enables Jaguar to execute fast range query by index keys. Jaguar is also well integrated with Spark, SparkR for data analysis and modeling.