捷豹数据库用户手册

(Rev 2.9.2 2018/05/18)

目录

简介

本说明书目的在于帮助用户顺利安装和使用 Jaguar 捷豹新型分布式数据库。Jaguar 采用与传统数据库不同的数据存储技术，综合大数据平台和数据仓库的特点，让用户在统一的数据平台上进行数据查询和计算。捷豹数据库的独特技术在数据查询、集群扩容方面将速度大大提高。

安装环境

系统将由捷豹服务器和客户端部分程序组成，服务器和客户端可以在不同主机下安装，也可以同一个主机内安装。

服务器端系统的运行最低要求

硬件要求： CPU 至少 8 核，2GHz，32GB RAM 内存，1000GB 磁盘或 SSD
系统软件： Linux CentOS 6 或者 7，RedHat 6 或者 7，Ubuntu，
Windows 服务器 2012,64 位
系统文件系统： ext4，或者 XFS， NTFS

客户端系统的运行最低要求

硬件要求：CPU 至少 4 核，2GHz，16GB RAM 内存，512GB 磁盘或 SSD
系统软件： Linux CentOS 6 或者 7，RedHat 6 或者 7，Ubuntu，
Windows 7，64 位
系统文件系统：ext4，或者 XFS， NTFS

系统安装

按照公司提供的软件包，可以实现一键安装。Serve、client 软件包可以分别安装在不同的主机或都安装到同一个主机上。Jaguar 的服务器进程将在 TCP/IP 端口 8888 监听客户端发起的请求。Jaguar 的服务器进程的名称是 jaguar，此程序应该由 jaguar 用户启动。

在单机上服务进程 jaguar 的启动程序脚本是 jaguarstart。停止运行的脚本程序是 jaguarstop。除了在单节点脚本外，在所有服务器上启动和停止 jaguar server 的脚本程序是 jaguarstart_on_all_hosts.sh 和 jaguarstop_on_all_hosts.sh。

操作系统

捷豹数据库支持 64 位的 LINUX 和 WINDOWS 操作系统以及虚拟主机。

LINUX 操作系统

在 64 位的 LINUX 操作系统（CENTOS6/7 或 REDHAT6/7）内，打开任意一个 TERMINAL 即可安装。

WINDOWS 操作系统

在 WINDOWS 64 位服务器操作系统下，请使用 Msys 或者 Cygwin terminal 进行安装操作。如果你没有安装此类软件， 请通过互联网下载安装。或者使用我们附带提供的 Msys 安装软件 MSYS-1.0.11.exe 进行安装。例如你可以将 c:\msys1 作为目标目录，或者有足够容量的 D 盘进行安装。然后将捷豹数据库的 tar.gz 文件拷贝到 home 目录内，进行安装。

捷豹数据库服务器和客户端软件包安装

Linux 系统

如果您使用的系统是 Linux，在任意一个服务器平台上，以任何用户（生产环境下建议用 jaguar 用户账户）执行下列安装脚本程序：

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

有关文件和程序将解压到 jaguar-n.n.n 目录内。

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

如果你安装的是企业版本，进入到server子目录，执行reqlicense程序，并将其打印出的字符串（申请码）发送给产品销售方，随后会收到产品授权码。

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

将收到的授权码存入同目录的license.txt文件内，以便捷豹数据库能正常启动。对free-trial版本，此授权文件则不需要。

如果你的系统是LINUX并且运行sshd和ssh程序，你只需要执行一条命令即可在所有服务器安装好捷豹数据库软件。如果你的系统是WINDOWS或者没有sshd 和ssh，你需要在每个服务器执行 install.sh 程序安装。

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

安装人员应该产生文件HOSTFILE 并完成配置，文件内应该填写数据库集群内每一个服务器的主机名称。HOSTFILE举例如下：

```
node1
node2
node3
node4
node5
```

另外，**当前用户在所有服务器节点上应该具有相同的密码**，这样才能保证上述脚本在每个服务器都能成功运行。请注意只需要在一个服务器下载 jaguar.tar.gz 文件，并且运行上述 install_jaguar_database_on_all_hosts.sh 脚本程序即可，不必在所有服务器重复此过程。

如果您第一次在集群上安装捷豹数据库，“-f HOSTFILE”是必须提供的，在以后的升级安装新版本时，HOSTFILE 不提供。

如果您在按照过程中出现错误或需要修改配置，可以执行下列脚本程序在所有有关的服务器上删除已经安装好的 jaguar 软件包：

```
./uninstall_jaguar_database_on_all_hosts.sh
```

安装程序将配置文件 server.conf 拷贝到 $JAGUAR_HOME/conf/server.conf，将 jaguar 拷贝到 $JAGUAR_HOME/bin 目录，还有两个启动和停止 Jaguar server 服务器的脚本程序 jaguarstart， jaguarstop 也拷贝到 $JAGUAR_HOME/bin。用户为了执行程序方便，应该将自己的 shell 变量 $PATH 设置为包括 $JAGUAR_HOME/bin 路径，例如 export PATH=$PATH:$JAGUAR_HOME/bin。系统安装完毕后，文件$HOME/.jaguarhome 记录了 $JAGUAR_HOME 的路径。

捷豹数据库在默认情况下将主目录存放到 $HOME下面。如果您要存放到不同目录下，可以通过脚本程序install_jaguar_database_on_all_hosts.sh 的 “-d <DIRECTORY>” 选项来实现。 <DIRECTORY> 就决定了有关脚本程序内 JAGUAR_HOME的取值。

用户的公钥在集群的各个节点设置完成后，$HOME/.jagsetupssh 文件会相应产生。在任何时候，可以运行 “setupsshkeys -f 《conf/cluster.conf》”指令重新设置用户在每个节点的公钥。

Windows 系统

如果您在Windows系统下安装捷豹数据库，请在安装包打开后，执行下列程序：

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

如果有多台Windows主机（或虚拟机）， 请在每个主机上安装捷豹数据库并执行上述脚本程序。安装好后，请参考下列内容配置 conf/cluster.conf 和其它有关参数。关于 conf/license.txt的配置请参照Linux系统的配置方法。

系统配置

系统配置文件 $JAGUAR_HOME*/conf/server.conf* 包含下列参数：

- PORT 是 jaguar 服务程序监听的端口
- LISTEN_IP 是在主机有多个网卡和IP地址情况下，jaguar server 具体监听的IP地址。如果主机只有一个网卡和IP地址，此选项应该被注释起来。
- MEMORY_MODE 指定您的服务器使用内存的模式，取值是high或者是low。如果是high，则捷豹数据库会稍微多用内存来管理数据。如果是low，则内存用量会减少。默认设置是high。

- REPLICATION 是每份数据的总备份数目。在企业版捷豹数据库中，此参数最大值是3。如果服务器的个数小于3，此值位服务器的个数。系统一旦运行，此参数不能修改。在免费版中，此参数固定为1，即没有数据备份。
- BUFF_READER_BLOCKS 系统从表中读取数据时预读的数据块数，默认值是4096。
- JAG_LOG_LEVEL 日志详细程度。最小值是0，最大值是9。数值越大记录的信息越多。
- LOCAL_BACKUP_PLAN 指定数据如何在本机进行数据快照备份。有5个时间间隔：每15分钟、每小时、每天、每星期、每个月。备份数据保存时，保存方式有：SNAPSHOT （快照，旧的备份数据不删除）或 OVERWRITE （保持最新备份，旧的数据被删除）。LOCAL_BACKUP_PLAN 参数的设置格式是：

  时间间隔1:方式|时间间隔2:方式|时间间隔3:方式|…

  其中时间间隔取值为15MIN, HOURLY, DAILY, WEEKLY, MONTHLY。方式取值为 SNAPSHOT 或 OVERWRITE。如果需要有多个时间段的备份，用竖线 “|”分隔。
- REMOTE_BACKUP_SERVER and REMOTE_BACKUP_INTERVAL: 这两个参数指定将数据远程备份到其它服务器主机的IP地址和时间间隔（秒）。远程备份的主机代表SAN存储设备系统，此系统应该具有足够大的容量。

配置文件 $JAGUAR_HOME/*conf/cluster.conf* 由HOSTFILE 文件自动生成，生产过程通过运行install_jaguar_database_on_all_hosts.sh 脚本程序完成。文件cluster.conf 包括HOSTFILE 里所有主机名称：

```
server1
server2
server3
server4
……
```

请确保cluster.conf 文件在每一个服务器内容都一致。配置文件server.conf 也要在各个服务器上一致（如果启用了LISTEN_IP，LISTEN_IP是唯一的不同）。

配置文件 $JAGUAR_HOME/*conf/datacenter.conf* 是为了支持多个数据中心数据多活同步。其中必需列出每个数据中心中任意一个服务器的IP地址。 格式如下：

```
数据中心1集群内任意一个服务器的IP地址或主机名:<Port>:<Type>
数据中心2集群内任意一个服务器的IP地址或主机名:<Port>:<Type>
数据中心3集群内任意一个服务器的IP地址或主机名:<Port>:<Type>
 ……
```

其中Port时捷豹服务器监听额端口号，（默认值是8888），Type 是服务器的类型，包括：HOST，GATE，或 PGATE。通常数据的路径是： HOST→GATE→GATE→HOST。一个 GATE 服务器保护一个或多个HOST服务器，并且转送给HOST数据。

例如(conf/datacenter.conf)：

192.168.1.100:8888:HOST
212.18.191.130:8888:GATE
192.168.1.130:8888:PGATE

启动和停止

LINUX 系统

下列指令将在所有服务器上启动 jaguar server（在一个服务器运行即可）

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

上述指令完成后，所有服务器的 Jaguar server 即在 port 8888 监听客户端发出的请求，执行数据库操作。在服务器启动后，管理员用户可以 “admin”账号登陆，默认密码为“jaguar”。我们建议您尽快修改此密码。管理员可创建更多的数据库和用户账户。创建的数据库和用户立即生效。服务器的日志文件存入到 $JAGUAR_HOME/log/ 目录内。

您还可以检查各个服务器 jaguar server 的运行状态：

```
$JAGUAR_HOME/bin/jaguarstatus_on_all_hosts.sh
```

下列指令可以停止各个服务器 jaguar server：

```
$JAGUAR_HOME/bin/jaguarstop_on_all_hosts.sh
```

WINDOWS SERVER 系统

如果你的系统是 WINDOWS 或者没有 SSHD SSH，请在每个服务器执行 jaguarstart 命令启动捷豹数据库。

在一个服务器上还可以查看本机捷豹数据库的运行状况:

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

停止所有捷豹数据库服务器执行:

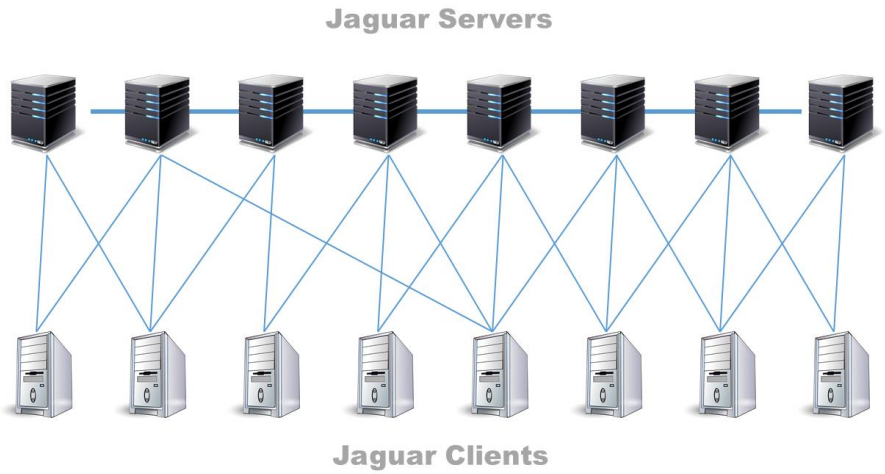
```
$ $JAGUAR_HOME/bin/jaguarstop -all
```

只停止本机捷豹数据库服务器执行:

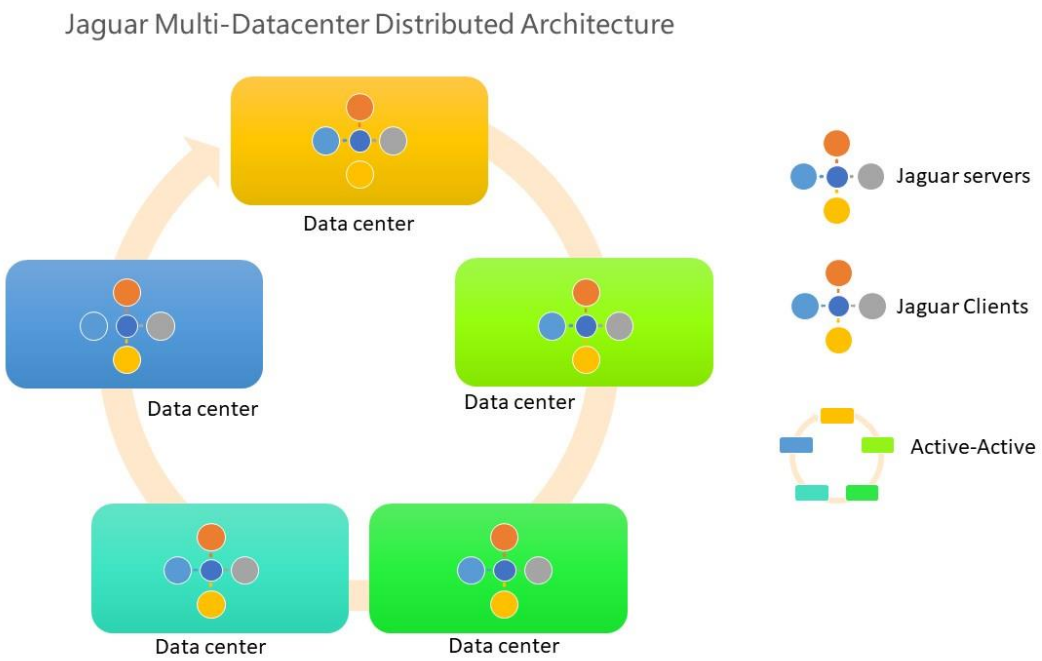
```
$ $JAGUAR_HOME/bin/jaguarstop -s
```

系统架构

J 捷豹分布式数据库为扁平式、多主架构。客户端可以连接任何一个服务器，实现完全线性扩展。



捷豹数据库还支持多个数据中心（或云）。每个数据可一旦单独运行捷豹数据库集群。集群内的数据更新时，其它数据中心的数据也会相应得到更新。



条件和限制

Mount noatime

数据库系统运行时，IO 的性能是非常重要的因素。因此建议关闭文件系统的 access time 功能。具体操作步骤是 root 编辑 /etc/fstab 文件在默认选项值内 加入 noatime:

```
defaults,noatime
```

资源限制

系统打开文件最大个数

我们建议您采用 root 或者sudo 权限将文件 /etc/security/limits.conf 修改如下:

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

个人用户最大进程或线程数

/etc/security/limits.conf:

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

内核最大线程数

```
/etc/sysctl.conf:
    kernel.threads-max = 1000000
```

进程个数最大数目

```
/etc/sysctl.conf:
    kernel.pid_max = 1000000
```

有关文件修改后，重启系统，新产生的进程会自动读取这些系统参数生效。如果你不想重新启动系统，可以运行 sysctl -p 使 /etc/sysctl.conf 生效，打开一个新的terminal让/etc/security/limits.conf生效。请注意，Linux系统内部 /etc/security/limits.d/目录下某些文件内的设置会刷新(override) /etc/security/limits.conf 文件的设置，请注意修改/etc/security/limits.d/ 内的文件配置。对nproc 参数不要设置太高（或设为unlimited）。 nproc 设为unlimited 时会导致用户不能登录。

安装确认

Jaguar Server 安装完毕后，确认下列文件设置正常：

1) Port 8888 没有与其它服务或应用程序冲突
2) 用户主目录$JAGUAR_HOME 成功建立
3) 下列文件存在并且文件模式正确（可执行）：
   $JAGUAR_HOME/bin/jaguarstart
   $JAGUAR_HOME/bin/jaguarstop

测试运行

测试途径

针对 Jaguar 的测试可以通过两种途径：1）交互式的 Client-Server 会话　2）通过利用编程 API 方式。第一重方式启用 jag 客户端程序进行，测试用户输入 JQL（类似 SQL 的语言）语句，server 接收到指令后，执行相关操作。第二种方式， 由用户或开发人员利用我们提供的 C++/JAVA API，编写 C++/JAVA 语言程序，请求数据库服务器进行数据写入、查询操作。

基本性能

测试首先可针对 Jaguar 的优势部分进行测试，不需要测试产品的完备性和稳定性。测试方面包括一些几个基本性能：

- 数据插入速度
- 数据查询速度
- 索引性能
- 内存用量

数据插入速度

针对数据插入速度的测试有二种方式：server 进行 batch load；client 连接 server 进行单个的数据插入。在测试之前应该保证 server，client 软件包都已经正确安装，并且正常运行。

预备工作：

1. 建立用户 admin
   $ jaguar> createuser test:test;

2. 建立表格
   $ jag -u test -p test -d test -h 127.0.0.1:8888

   jaguar> create table test (key: uid char(16), value: v1 char(16), v2 char(16), v3 char(16));

(A) 文件导入

在 Jaguar server 主机上可以测试 load 3 百万条数据。样品数据可以由 client 软件包附带的 genrand 程序产生:

```
$ genrand 3000000 71
Please enter a line of header which contains name of keys and values.
Example: uid,v1,v2,v3
Your input: uid,v1,v2,v3   (输入回车键)

$ mv genrand.out /tmp/3M.txt
```

然后在 jag 客户端（任何用户可以运行 jag 客户端）输入下列指令将 3 百万条数据从 /tmp/3M.txt 插入到 test 表格中:

```
jaguar> load /tmp/3M.txt into test;
```

预期值: 等待一段时间后（大约 2~3 分钟），jag 将会报告 load 数据占用了多少 milliseconds（毫秒）。

你可以将 SQL 指令写入到一个文件内，然后运行 jag 程序自动执行这些指令:

```
$ vi mycommands.rql

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

然后调用这些 SQL 命令:

```
$ jag < mycommands.rql
```

(B) 客户端单个数据插入

客户端的软件包附带的 jbench 程序用来实现从客户端向 server 插入、修改、查询数据。下列指令将产生 10000 个随机数，连接 server，数据传输给 server，server 将数据插入到数据库的 abenct 表内:

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

其中双引号内以冒号分隔的数字第一个是插入次数，第二个是修改次数，第三个是查询次数，第四个是删除次数。 选项“-k 0”代表 test 数据表清空且重新产生。

预期值：利用 jbench 插入数据，速度大约为 8000-15000 个/秒。jbench 包括的网络数据传输的 overhead。

注意：jbench 测试程序利用 test 数据库，以及自带的 jbench 表。 表 jbench 包含 uid key (16 字节)，city value (32 字节)。

数据查询速度

jbench 客户端测试查询速度：

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

上述指令将对数据库 server 查询 10000 次。

基本功能

下列基本功能可以通过上述两种测试途径完成：

1) create database 创建新的数据库
2) create user 创建新用户，改变权限，身份
3) create table 创建表格
4) insert data into table 写入数据
5) select data from table 查询数据
6) update data in table 更新数据
7) delete data from table 删除数据
8) join tables and select data 交互查询多个表格
9) load file into table 批量 load 数据
10) drop a table 删除表格

11) drop database 删除数据库

利用会话式客户端 jag 上述测试的方法通过 help 指令可以得到帮助:

```
jaguar> help;
You can enter the following commands:
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 batch load data)
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 func         (how to use Jaguar built-in functions)
```

请注意在执行一条指令时，指令语句里面的关键词（例如 select, where, table, from）之间只能用空格、‘\t’、‘\r’、‘\n’字符来分隔。不允许其它不可打印的字符，这些会造成语句执行错误。

编程指导

Shell 指令

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

举例: `$ $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 语言支持

```
#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 语言支持

```
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 支持

```
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 语言支持

```
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 语言支持

确保环境变量 PYTHONPATH 指到 jaguarpy.so 所在的目录:

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

然后在你的 python 程序中:

```
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 语言支持

为了支持 PHP 编程，请用使用 root 或者 sudo 将捷豹数据库 conf/jaguar.ini 文件拷贝到系统的/etc/php.d 目录(Centos)，或者/etc/php5/mods-available (Ubuntu)，或 PHP 设置要求的不同目录。同时将 lib/jaguarphp.so 和 lib/libJaguarClient.so 拷贝到合适的目录。

$ php -i | grep additional　　提供 jaguar.ini 应该拷贝到的目录。

$ php -i |grep extension_dir　提供 jaguarphp.so 应该拷贝到目录。

例如：

Centos 系统　# cp -f　conf/jaguar.ini　/etc/php.d/

Centos 系统　# cp -f　lib/jaguarphp.so　/usr/lib64/php/modules

Ubuntu 系统　# cp -f conf/jaguar.ini /etc/php5/mods-available/

Ubuntu 系统　# 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 语言支持

请将 lib/jaguarrb.so 拷贝到 $JAGUAR_HOME/lib directory 然后 export RUBYLIB 环境变量:

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

Ruby 程序:

```
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 语言支持

请将 lib/jaguarnode.node 拷贝到$JAGUAR_HOME/lib 目录。

```
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 语言支持

使用 GO 语言， 你需要建立你自己的 project 目录， 例如 myproject. 下面举例说明如何设置开发环境:

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

   确认下面设置正确:

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

6. 执行 $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()
}
```

索引查询

举例说明如果表 mytable 含有主键 uid 以及非键 v1, v2, v3. 如果想按照非主键查询，可以建立以一个或多个非主键列的索引。例如：

```
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();
```

客户端 API

下列的函数在 C++, Java, Scala, Python, PHP, Ruby, Go, NodeJS 均有支持:

1. bool connect(String host, int port, String uid, String pass, String db) 连接服务器. Returns True for success, False for failure.

2. bool execute(String command)
   执行 DML 语句，例如 create table, drop table, update, delete, insert 语句. 此函数不能执行 select 和其他查询语句。

3. bool query(String query)
   查询语句，select 应该使用此函数。

4. Bool reply()
   配合 query() 函数，将每一条记录返回客户端查询。此函数应当置于 while loop 之内，将每一条记录获取，直至结束。结束时此函数返回值是 false。

5. void printRow()
   打印一条记录数据。

6. void close()
   关闭连接并释放资源。

7. String getDatabase()
   获取当前会话的数据库名称。

8. bool hasError()
   检查前面查询有无错误。

9. String error()
   如果有错误，返回错误信息。

10. String getNthValue(int N)
    取得当前行第 N 个列的字符串值，从 1 开始。

11. String getValue(String columnName)
    获取当前行所给列名称的列字符串值。例如如果 uid 是某个列的名称，getValue("uid") 返回 uid 列的字符串值。

12. String getMessage( )
    当前行的全部数据。有些查询不涉及到列的结构信息，例如 desc table, 应当使用 getMessage() 得到全部信息。

13. long getLong(String columnName)
    .得到某个列的 long 数值。

14. double getFloat(String columnName)
    得到某个列的 float 数值。

15. int getColumnCount()
    当前行的列个数。

16. String getColumnName(int col)
    得到第几个列的名称。

17. int getColumnType(int col)
    得到第几个列的类型数值(以 JDBC 为标准)

18. String getColumnTypeName(int col)
    得到第几个列的字符串名称(以 JDBC 为标准)

19. String getTableName(int col)
    得到某个列的表名称。

系统运营

网页管理集群

捷豹数据库集群的管理软件包通过 httpd 服务给管理员提供 WEB 界面，用来管理数据库服务器集群。此软件包的名称为 jaguar-admin-nnn.tar.gz 格式。从 github 网站下载此软件包后，

1) 在当前服务器运行 install_jaguar_admin_on_all_hosts.sh 然后 jagadmin 程序将会安装在所有数据库服务器上
2) 启动 jagadmin server $JAGUAR_HOME/jagadmin/bin/jaguarallstart
3) 将 index.cgi 拷贝到/var/www/cgi-bin/ 并且将 html/* 拷贝到/var/www/html/
4) 在当前服务器启动 Apache server, # systemctl start httpd
5) 在浏览器打开网页 http://<IP>，其中 IP 是当前服务器的 IP 地址

以 admin 用户和密码登录后，管理员能检查各个服务器的运行状态、数据传输的统计数字、资源的利用率等信息。管理员还能创建数据库、新用户。

配置远程备份

捷豹数据库在各个服务器上的数据可以定时备份到一个中心的 REMOTE_BACKUP_SERVER 主机山。此主机应该是具有足够容量的 SAN 存储或云存储设备。

第一个 Jaguar 数据库服务器上进行的配置

打开 conf/server.conf 文件设置好远程主机的 IP 地址 REMOTE_BACKUP_SERVER 和时间间隔（秒） REMOTE_BACKUP_INTERVAL 参数。这项配置只需要在集群的第一个服务器完成即可，不需要在其它服务器进行。配置文件 conf/syncpass.txt 含有 jaguar 用户登录到其它服务器的密码。为了安全起见此文件的模式应该是 600（chmod 600 conf/syncpass.txt）。此密码可以与操作系统的用户密码不同。

举例 conf/syncpass.txt:

mypassword888

其中 mypassword888 是 jaguar 用户能连接到其它捷豹数据库服务器专门进行数据备份的密码。

远程备份存储服务器配置

在远程备份服务器上，rsync 服务应该进行配置。其配置文件/etc/rsyncd.conf 应该具有下列格式:

```
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
```

其中 “[jaguardata]” 不能修改， 但是“path = /home/jaguar/jaguarbackup” 里面的备份目录 jaguarbackup 可以修改成不同名称。此目录的所属用户必须是 jaguar。

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

配置文件/etc/rsyncd.secrets （ root 用户 chmod 600 ） 应该包含 jaguar 用户的数据备份密码:

```
jaguar:mypassword888
```

文件/etc/rsyncd.secrets 中的密码应该与第一个 jaguar 服务器上 conf/syncpass.txt 内容一致。完成上述配置后，在 CentOS 和 Redhat 系统上，可以在远程备份服务器上重新启动 rsynd 服务:

```
# systemctl restart rsyncd
```

重启故障节点

如果发现某个节点出现故障， 根据不同情况请按照如下方法处理：

（1） 如果是硬件故障（例如磁盘损坏、零件损坏等），请修复硬件或更换节点设备，确保其 IP 地址和有关配置正确，然后运行这个脚本程序：
$JAGUAR_HOME/bin/jaguarstart_dorecover
此程序将指示服务器进行数据修复，然后一直运行进行正常数据管理工作。
（2） 如果 jaguar 进程不小心被停止，运行 jaguarstart_dorecover 程序即可。
（3） 如果只是网络出现故障，服务器本身还正常运行，那么无需进行任何操作。网络正常后，服务器集群会自行进行数据恢复。

数据类型

Jaguar 支持如下数据类型：

1. Character 字符串
   char(长度) -- 在 KEY 字段是固定长度的字符串， 在 VALUE 部分是任意长度的字符串。varchar(length)与 char(length) 等同。

2. Boolean
   boolean -- 单个数字

3. Integer
   int or integer - 范围在 -9999999999 和 +9999999999 之间的数字

4. Big integer
   bigint -- 范围在 -9999999999999999999 和 +9999999999999999999 之间的数字

5. Small integer
   smallint -- 范围在 -99999 和 +99999 之间的数字

6. Tiny integer
   tinyint – 范围在 -999 和 +999 之间的数字

7. Medium integer
   mediumint – 范围在 -9999999 和 +9999999 之间的数字

8. Float

float(L,d) -- L 为总长度，d 为小数点后面的位数。

9. Double

double(L,d) -- 与 float 类似， L 为总长度，d 为小数点后面的位数。

10. DateTime

datetime - 16 位的精确到微秒的时间数值。写入到数据库时，必须采用下面的格式:

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

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

其中 YYYY 代表年，例如 2005
MM 是月，例如 10
DD 是日期，例如 04
hh:mm:ss 是 小时：分钟：秒 ，例如 02:23:21
uuuuuu 是选项，代表分数秒（最多六位数，精确到微秒）

+HH:MM and -HH:MM 是时区标志的选项， 以 GMT 为标准。如果时区标志没有提供，客户端输入的数据认为是客户端本地的时间，服务器会自动辨认调整。输入时区是为了能让客户端输入其它时区的时间。

举例:

客户端位于美国加州:

```
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 - 与 datetime 相似但精确度是 nano seconds (纳秒，千分之一微秒)。其格式为:

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

12. 日期 date

date 类型的输入输出格式为 YYYY-MM-DD
例如:
insert into sales (uid, datecol) values (1234, '2015-03-12');
select * from sales where datecol='2015-12-23';

13. Time 时间
time(*) - 时间的表述，从年月日到微秒。在写入时间和查询时间时，必须采用如下格式:

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

其中 YYYY 是四个数字的年数，例如 2005
MM 是两位是的月份，例如 10
DD 是日期， 例如 04
hh:mm:ss 是 小时:分钟: 秒， 例如 02:23:21
uuuuuu 是可选项， 代表秒的分数（最多 6 位，即微秒级别）

+HH:MM and -HH:MM 是与格林威治标准时间的时差（小时: 分钟）。

举例:

在美国加州:
insert into sales (uid, sttime) values (1232, '2014-11-23 16:32:21 -08:00');
insert into sdata (deviceid, logtime) values (1232, '2015-10-23 13:32:21.234019 -08:00');
select * from sales where sdate > '2014-12-10 03:12:23';

14. TimeNano

timenano --- 精确到纳秒的时间数值。输入格式为:

HH:MM:SS.[nnnnnnnnn] - 其中 nnnnnnnnn 代表 nanoseconds （纳秒）

15. Timestamp

timestamp -- 与 datetimestamp 相同。 两者都接受 ‘yyyy-mm-dd HH:MM:SS.123456 HH:MM’ 格式或者自 1970 年 1 月一日 00:00:00 开始计时的微秒数，例如 1482000884000000。

16. Real
    Real – 此类型是 double(38,8)，共有 38 位，小数点后面有 8 为数字。

17. Text
    Text -- 等同于 char(1024)

18. TinyText
    TinyText -- 等同于 char(256)

19. MediumText
    MediumText -- 等同于 char(2048)

20. LongText
    LongText -- 等同于 char(10240)

21. Blob
    Blob -- 等同于 char(1024)

22. TinyBlob
    TinyBlob -- 等同于 char(256)

23. MediumBlob
    MediumBlob -- 等同于 char(2048)

24. LongBlob
    LongBlob -- 等同于 char(10240)

25. String
    String – 等同于 char(64)

26. Varchar
    Varchar(N) 或者 varchar2(N) – 等同于 char(N)

27. Bit

Bit – 取值 1 或者 0

28. Enum
    列的值只能取指定的数值。COLUMN Enum ('v1', 'v2', 'v3', …)

29. File
    此列可以存贮文件数据。数据的大小没有限制，可以是任何形式的文件（音频、视频、图像、照片、DOC、PDF、WORD 等）

创建表默认值

在创建表时，每个列可以制定默认值。如果写入数据（insert）时，列的名称和赋值没有提供，默认值会被写入系统。一般列的默认值时一个字符，时间列的默认值可以为当前时间（CURRENT_TIMESTAMP）。当一行的数据被更新时，时间列可以相应改变。例如：

```
Create table tab123 (
    Key:  id uuid,
    Value:
        a int default  '0',
        b char(16) default 'anonymous',
        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 ,
        stype enum ( 'high', 'med', 'low' ) default 'low'
    );
```

Jaguar 与 Java 的数据类型映射

下列表格列出了捷豹数据库数据类型与 Java 的数据类型等同效果：

Jaguar 类型	格式	Java 类型
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 函数调用

Jaguar 支持多个内部函数，可对一个或多个列进行数学运算或其它操作。下面内容是对这些函数的详细描述。

语法：

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

EXPR(COL): 列操作的运算

Numeric columns: columns with arithmetic operation

\+ addition

- subtraction)

* multiplication)

/ division

% modulo

^ power

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)): 函数

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

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

substring(EXPR(COL), start, length)

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 time column

minute(TIMECOL) -- value of minute in a time column

hour(TIMECOL) -- value of hour in a time column

date(TIMECOL) -- value of date in a time column

month(TIMECOL) -- value of month in a time column

year(TIMECOL) -- value of year in a time column

datediff(type, TIMECOL, TIMECOL) -- difference of two time 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)

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

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

dayofyear(TIMECOL) -- day of the year in a time column (1-366)

curdate() -- current date (yyyy-mm-dd)

curtime() -- current time (hh:mm:ss)

now() -- current date and time (yyyy-dd-dd hh:mm:ss)

Example: 举例

select sum(amt) as amt_sum from sales limit 3;

select cos(lat), sin(lon) from map limit 3;

select tan(lat+sin(lon)), cot(lat^2+lon^2) from map limit 3;

select uid, uid+addr, length(uid+addr) from user limit 3;

select price/2.0 + 1.25 as newprice, lead*1.25 - 0.3 as newlead from plan limit 3;

Jaguar SQL 语句

捷豹数据库支持的 SQL 指令可以通过 jag 客户端程序的 help 命令的得到：

```
jaguar:test> help;
```

```
You can enter the following commands (以分号结束):

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)
```

Admin 管理员指令

下列指令应该由 admin 账户登录执行:

```
createdb DBNAME;
dropdb [force] DBNAME;
createuser UID;    -- 创建新用户
createuser UID:PASSWORD; -- 创建新用户（附带密码）

dropuser UID;
showusers;
```

举例:

```
createdb mydb;
dropdb mydb;
createuser test;
dropuser test;
```

Grant 指令

管理员 admin 创建一般用户账户后，应该赋予此用户的权限。Grant 指令用来赋予用户的权限。

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];

PERM 可以是: all/create/insert/select/update/delete/alter/truncate

All 代表所有的权限。在赋予 select 权限时，可以附加 where 条件，对表的行进行过滤。

举例:

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 指令

用户的权限可以由 admin 通过 revoke 指令取消。

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;

PERM 可以是: all/create/insert/select/update/delete/alter/truncate

举例:

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 指令

客户端要求更换数据库:

```
use DATABASE;
```

举例:

```
use myuserdb;
```

Describe 指令

描述表或索引的具体结构:

```
desc TABLE;
desc INDEX;
```

举例:

```
desc usertab;
desc addr_index;
```

Show 指令

显示数据库特性:

```
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)
```

举例:

```
show databases;
show tables;
show indexes from mytable;
show indexes;
show task;
```

Create 指令

创建表或索引:

```
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]);
```

举例:

```
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 bigint, value: audio file, video file );
```

创建表时，如果 key 没有提供，默认的 UUID 类型的列将作为主键，其名称为 _id。

创建索引时，用户可以添加几个 value 列，这样只通过此索引就可以读取到有关数据。缺点是占用了存储资源，优点是读取有关数据速度快，不必再到原始表去查询数据。

Insert SQL 指令

```
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’);

insert into TAB values ( k1, k2, ‘/file/path1’, ‘/file/path2’ );
```

举例:

```
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/myvideo.mov' );
```

如果某个列的类型是 UUID， insert 语句里不能包含此列，因为数据库服务器会自动产生独特的值，写入数据库内。UUID 列占 40 个字节。

对于 datetime，datetimenano，timestamp 列，如果时区信息没有提供，数据库将采用客户端本地的时区。

如果在写入数据的列当中包含 load_file(FPATH) 指令，文件 FPATH 的内容会先经过 base64 编码，然后写入到相应的列内。文件 FPATH 字符串之内可以包含用户的环境变量，例如 $HOME/img/a.jpg， $HOME 会被解析成用户的主目录。

Load 指令

将一个数据文件加载到数据库的某个表：

```
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.)
```

默认值：

```
columns terminated by: ','
```

```
lines terminated by: '\n'
```

```
column values can be quoted by singe quote (') character.
```

举例：

```
load /tmp/input.txt into user columns terminated by ',';
```

此指令可以将 CSV 文件加载到数据库。

Select SQL 指令

从数据库表中读取数据有各种方式:

```
(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;
```

如果没有提供 limit，系统默认在屏幕显示 1000 条数据。在 API 编程程序内则没有 1000 条的限制。 Timeout 参数是一个选项，它是数据库服务器超时限定(秒)。如果没有提供此参数，其默认值为 60 秒。如果其数值为零，那么数据库服务器不会中断，一个 SELECT 查询可能会花费很长世间。我们建议对复杂查询先实验用一定的 TIMEOUT，估计花费时间长短后， 再决定提供超时限制的数值。

举例:

```
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 指令

表内有属于 file 的列，其文件数据存储在服务器端。如果想要将文件数据读取出来，可以使用 getfile 指令：

```
Getfile COL into localfilapath from table where key=…;
```

其中 localfilepath 是客户端本地的文件路径和名称。请注意 where 条件必须指定唯一的一条记录。

getfile 指令还可以一次读取多个文件：

```
Getfile COL1 into fpath2, COL2 into fpath2 from table where key=…;
```

getfile 指令还可以读取文件的属性：

```
Getfile COL1 size, COL2 time, COL1 md5 from table where key=…;
```

结果： COL1.size:[322] COL2.time:[…] COL1.md5:[JJURJFFNFNFJ]

SQL Join 表联结

捷豹数据库支持两个表的 INNER JOIN 操作。任意两个表可以按照任意列联结，外加 WHERE 条件语句。

```
(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];
```

举例:

```
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;
```

请注意 JOIN 与前面 SELECT 语句同样有超时（TIMEOUT）设置，如果语句结尾没有提供 TIMEOUT 参数，则默认超时时间是 60 秒钟。

Update SQL 指令

```
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='...', ...;
```

举例:

```
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 指令

```
delete from TABLE;
```

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

举例：

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

Drop 指令

永久删除表和索引：

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

举例：

```
drop table force usertab;
drop index user_idx1 on user;
```

Truncate 指令

清空表内的数据，但保留表的结构（表的索引数据也会被清空）：

```
truncate table TABLE;
```

举例：

```
truncate table user;
```

Alter 指令

修改列的名称：

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

举例：

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

Spool 指令

指定查询数据后数据输出的文件：

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

举例：

```
   spool /tmp/myout.txt;
   select * from mytab limit 10000;
   spool off;
   (The above command will stop writing output data to any file)
```

用户修改密码

登录的用户可以修改自己的密码：

```
changepass;
```

举例：

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

用户修改密码时还可以直接提供密码（不太安全）：

```
jaguar > changepass mypassword888;
```

Group By 语句

对表中数值型的列能进行综合操作。

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

Group By LastValue 语句

某些组的最后一个记录可以通过 “group by lastvalue” 语句实现。

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

上述语句目的是以键 k1, k2, k3 对记录分组，并且只读取每个组的最后一个记录。

Order By 语句

查询的数据可以做排序:

```
order by COL1, COL2, COL3 [ASC/DESC]…
```

默认的排序次序是 ASC（依次增加）. 依次递减是 DESC。

Aggregation 语句

针对表或索引的一个或多个列可以进行综合计算。

举例:

```
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;
```

数据库系统设置

表内列个数限制

表或索引最多可有 4096 个列。

数据库名称字符限制

数据库名称最多包含 64 个英文字母。

列名称限制

列名称最多包含 32 个英文字母。

记录字符限制

表中每条记录最多 20 亿字节。 文件 file 类型的列数据大小没有限制。

数据导出和导入

捷豹数据库表中的数据可以导出到插入格式的 SQL 文件，此文件之后可以导入到捷豹数据库或其它数据库中。

Export 导出

数据导出到各个服务器中

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

举例:

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

在各个服务器上，jaguar/export 目录下会有导出的数据。

数据导出到客户端的一个文件中

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

数据将会存入到 DATABASE.TABLE.sql 文件。

举例:

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

导出的数据保存在客户端的 mydb.salestab.sql 文件。请注意如果表非常大，要确保客户端具有足够的存储空间。

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

数据将会存入到 DATABASE.TABLE.csv 文件，为 CSV 格式的文件。

举例:

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

导入数据

从所有服务器导入事先导出的数据

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

举例:

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

请注意 jagexport 必须事先完成，各个服务器具有准备好的数据。

从客户端文件导入数据

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

文件 DATABASE.TABLE.sql 是实现通过 jagexportsql 指令导出的文件。

举例:

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

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

文件 DB.TAB.sql 是 CSV 格式的文件。

举例:

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

修改 Schema

第一种方法是：为了方便修改 schema，我们建议用户在创建表时，多创建几个备用的列。例如:

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

列 spare1, spare2, spare3 暂时可能不用，但将来可能会用到，只需修改列的名称即可。

第二种方法是：利用 spare_ 列的额外空间增加列。指令为

```
Alter table TABLE add COLUMN TYPE;
```

举例： alter table tab123 add spacex int;

第三种种方法是：在项目开发早期，表的数据量不大时，可以采取下面的步骤修改 Schema：

1) 运行 jagexport 指令把表的数据暂存在各个服务器
2) Drop 有关表
3) 重新创建表并且遵循下面规则：
   a) 可以增加新的一个或多个列
   b) 一些旧的列可以去掉
   c) 存储空间小的列可以扩展成存储空间大的列，例如 int 改为 bigint
   d) 其余的列名称保持不变
4) 执行 jagimport 指令
5) Jagimport 成功完成后，执行下列指令清空临时数据：

```
$ jag -u admin -p -d DB -h :8888
     jaguar> import into DB.TABLE complete;
```

修复表

通常情况下数据库的表内数据是按照规则排列，不需要数据修复。但是如果有意外情况发生，表内的数据排列顺序可能会被破坏。如果表的结构发生故障，可以先检查再修复表。

检查表

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

检查表的语法是 “check DATABASE.TABLE;”。用句号分隔数据库名称和表名称。如果表结构正常，检查结果会显示表不需要修复。

修复表

如果表需要修复，admin 管理员可以连接数据库服务器（但必须以 exclusive 模式进入，--x yes ）。Exclusive 模式保证只有一个 admin 同时以 exclusive 模式进入。

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

Repair 指令将会报告修复进展程度，

系统容错性

捷豹数据库集群运行时，如果一个或多个服务器宕机、或者网络中断，捷豹数据库集群将会继续运行，数据会备份到合适的服务器。故障修复后，备份的数据会拷贝到有关的服务器内，整个系统恢复到正常运行状态。我们建议系统管理员平时应该配备好一个或多个服务器，将相关软件包括捷豹数据库软件（相同版本）安装就绪，完成 conf/cluster.conf 文件配置。一旦某个服务器发生永久性损坏，则把备用的 IP 地址更换为损坏的主机 IP 地址，将备用的服务器替换旧的服务器即可。如果只是网络暂时中断，则不需要任何工作。网络通畅后，系统会自动恢复。

系统扩容

随着数据库的数据量增长，现有的集群可能需要增加更多的服务器来达到扩容的目的。捷豹数据库允许一次增加多个服务器（并且是推荐的做法），并且扩容过程非常快。遵循下面的几个步骤可以实现快速扩容：

1. 将所有新的服务器配置到一个新的集群中。配置新的集群跟配置任何一个捷豹数据库集群采取同样的步骤，无任何区别，只是要保证这个新的集群能与现有的捷豹数据库集群能互通网络。启用这个新的集群（启动集群中所有 jaguar server）。
2. 将新集群的 conf/cluster.conf 文件拷贝到现有集群的任意一个主机内并且重新命名为 conf/newcluster.conf。
3. 在 conf/newcluster.conf 文件所在的主机上，连接捷豹数据库并且执行下列指令：

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

执行完成“addcluster”命令后，新的集群就会与现有的集群联通，开始接收新数据和数据查询任务。

捷豹数据库安全保护

如何保障用户的数据安全也是捷豹数据库的重要技术指标。捷豹数据库在多个方面保护用户的数据不会被泄露、更改或破坏。

局域网保护

在数据库服务器运行的网络环境下，配置硬件防火墙，或设置云环境的安全策略，只有符合安全策略的数据流才能通过防火墙，防止黑客 DOS 攻击，保证数据库局域网的安全运行。企业还可配备数据库专用防火墙，禁止向数据库发送的非法 SQL 语句，提供针对数据库恶意攻击的阻断能力。

服务器保护

捷豹数据库所运行的操作系统可以采用安全 LINUX 系统（SELINUX）。SElinux 在 linux 内核级别上提供了一个灵活的强制访问控制子系统，为每一个用户、程序、进程、还有文件定义了访问还有传输的权限。

用户权限保护

捷豹数据库的数据全部为一个合法系统用户（jaguar）所拥有，其它任何用户均不具备查看、修改、删改捷豹数据库数据的权限。此合法系统用户具有系统登录密码，我们建议用户使用高复杂度的密码，并且妥善保管，经常更换，以最大程度保护数据的安全性。

用户登录保护

通过数据库客户端连接到数据库服务器的用户必须通过身份验证才能成功登录数据库。身份验证包括用户名和密码。我们建议用户妥善保管用户名和密码，并且拒绝复杂度不符合安全标准的密码。

用户级别保护

登录访问捷豹数据库的用户分为两种类型：1）管理员账户；2）一般用户。只有管理员账户才能创建数据库、删除数据库、创建一般用户、删除一般用户。一般用户只能创建表、查询数据、修改、删除表数据。

数据库服务器关联保护

由于捷豹数据是分布式数据库集群，各个数据库服务器之间需要经常交换数据库信息。在交换信息的过程中，服务器之间需要令牌（SERVER_TOKEN）交换验证，才能有效交流数据。在安装捷豹数据库系统时，此令牌可以随机产生并保存使用。因此每一个企业客户的 Token 都是独特的，企业客户的捷豹数据库系统得到独立安全运行。

数据库访问保护

捷豹数据库具有黑名单（blacklist.conf）、白名单（whitelist.conf）安全设置。只有 IP 地址或网段在百名单内的服务器和客户端才能访问数据库服务器，在白名单范围内，还可以设置 IP 地址或网段黑名单，防止某些客户端登录访问。如果系统没有设置白名单和黑名单，则允许所有 IP 地址登录。在生产环境下，我们建议使用白名单和黑名单，加强系统安全性。

数据库日志监控

捷豹数据库保存数据每个用户的登录信息、表格创建和删除信息，数据库管理员可以经常和定时监控这些数据库日志，并且检查每个捷豹数据库服务器的日志，判别是否存在异常登录和数据修改情况。

数据导入和实时同步

捷豹数据库 github 代码库里面存有最新的从其它数据库导入数据到捷豹数据库的代码，并且还有两个数据库并行运行保持数据同步的代码，例如从甲骨文数据库和 MySQL 数据库导入数据和同步数据。数据同步的工作原理是： 1） 捷豹数据库必须产生与其它数据库类似的表格（结构不完全相同），（2）其它数据库产生 changelog 表反映某个表的变化和 3 个 trigger，trigger 将变化的数据写入 changelog 表，（3）其它数据库表的数据导入到捷豹数据库相应的表， （4） 捷豹数据库服务器端开启监听 java 服务进程将其它数据库表的 changelog 的数据写入捷豹数据库的相应表。此 java 进程一直循环不停，持续数据同步工作,并且可以监听多个表的变化，更新捷豹数据库里面的相应的表。

建议采用下面的步骤设置建立数据同步功能：

第一步 捷豹数据库创建表

捷豹数据库必须产生与其它数据库类似的表格，**此步骤在捷豹服务器上进行。**

举例：使用 github.com/datajaguar/jaguardb: importsync/databaseimport/from_oracle/ create_jaguar_table.sh 程序在捷豹数据库集群上任意一个服务器上创建与甲骨文数据库相同的表。在 example1 目录里面 create_jaguar_table_example1.sh 程序可以作为参考，在捷豹数据库产生相应的表 table234。需要预先编译 JDBC 库，cd importsync/jdbc; ./compile.sh

第二步 其它数据库创建 Changelog Trigger

在其它数据库创建 changelog 表反映原始表的变化和 3 个 trigger，trigger 将变化的数据写入 changelog 表。**此步骤在其它数据库服务器上进行。**

如果在 Windows 系统，请先安装 Msys1 窗口系统。

举例参照 github 内如下程序在甲骨文书库创建 changelog 和三个 trigger。
importsync/databasesync/oracle/OracleToJaguar/oracle_create_changelog_trigger.sh

结果：对应原始表 table234 的 changelog 产生，table234 如果有新数据写入、数据修改、或删除，changelog 会增加一条或两条数据。

第三步 导入数据

其它数据库表的数据导入到捷豹数据库相应的表，**此步骤在捷豹服务器上进行。**

举例参照 github 内此程序：

importsync/databaseimport/from_oracle/example1/ import_from_oracle.sh

请注意修改 appconf.oracle 配置文件 source_jdbcurl ， dest_jdbcurl，以及其它参数。

第四步 更新捷豹数据库

捷豹数据库服务器端开启监听 java 服务进程将其它数据库表的 changelog 数据写入捷豹数据库的相应表。 **此步骤在捷豹服务器上进行。**

举例参照 github 内程序启动对一个或多个表的监控和更新程序。
importsync/databasesync/oracle/OracleToJaguar/example1/start_sync_oracle_to_jaguar.sh

请修改 appconf.oracle 里面设置，符合自己系统的配置。

appconf.oracle:

source_jdbcurl=jdbc:oracle:thin:@//192.168.7.120:1522/test

（192.18.7.120 为甲骨文服务器 IP 地址）

```
source_table=table234|table345   (多个表用竖线分隔)
source_user=test
source_password=test
sleep_in_millis=3000
(每隔三秒扫描一次 changelog 的数据)
keep_rows=10000
(同步后 changelog 暂时保留一定量的数据)

dest_jdbcurl=jdbc:jaguar://localhost:8888/test
(8888 为捷豹数据库的端口号)
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
```

如果只是从其它数据库将数据导入到捷豹数据库，则值需要完成上述第一步（创建表）和第三步（导入数据），其它步骤不要进行。第四步的更新捷豹数据库可以任意时刻进行，如果 java sync 服务器进程中断，重新启动后数据会正常进行更新。

Spark 数据分析

Jaguar 提供了 JDBC 接口，Apache Spark 或任何其它数据平台可以通过 JDBC 接口调用 Jaguar 里面的数据。下面的例子说明了如和结合 Spark 与 Jaguar 对数据进行各种分析。在下面的例子中， 表格的结构如下：

```
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) );

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()
```

如上所述源代码产生的类（class），可利用如下命令提交给 Spark:

```
/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
```

Jaguar 平台进行 SparkR 分析

如果你已经安装了 R 和 SparkR 软件包，即可按照如下方式启动 SparkR 程序:

```
#!/bin/bash

export JAVA_HOME=/usr/lib/java/jdk1.7.0_75
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
```

启动后，在运行环境里输入下列命令：

```
library(RJDBC)
library(SparkR)

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

sqlContext <- sparkRSQL.init(sc )

drv <- JDBC(“com.jaguar.jdbc.JaguarDriver”, “/home/exeray/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”

结束语

捷豹数据库是一款新型的 NoSQL 大型分布式数据库系统，采用了创新技术提高数据的写入和查询，并且大幅提高数据库扩容的速度，使数据库系统无限扩展而不遭遇瓶颈。捷豹数据库还提供多种数据查询方法，不仅数据录入速度高，还具备强大的数据查询分析功能。