目录

[一、 概述 2](#_Toc536866296)

[二、RQG源码下载及使用 3](#_Toc536866297)

[1.1、下载源码 3](#_Toc536866298)

[1.2、安装DBI模块 3](#_Toc536866299)

[1.3、RQG支持ODBC 3](#_Toc536866300)

[1.2.1、安装DBD-ODBC 3](#_Toc536866301)

[1.2.3、RQG源码修改 4](#_Toc536866302)

[1.2.4、测试 10](#_Toc536866303)

[1.4、RQG支持JDBC 10](#_Toc536866304)

[1.4.1、安装DBD-JDBC 10](#_Toc536866305)

[1.4.2、RQG支持JDBC 11](#_Toc536866306)

[1.4.3、测试 14](#_Toc536866307)

[1.5、代码优化和完善 15](#_Toc536866308)

[1.6、RQG使用说明 15](#_Toc536866309)

[三、语法规则 18](#_Toc536866310)

# 概述

RQG(RandomQueryGenerator)是 MySQL 使用的 QA 测试工具，它是一个开源的 PERL 程序。它可以按照一定的规则自动生成表以及数据，根据指定的语法文件随机生成 SQL 并以一定的并发压力发给指定的一个或几个 MySQL 执行，最后比较各个 MySQL 返回的结果集，如果不一致的话会报错。

官方使用文档：

<https://github.com/RQG/RQG-Documentation/wiki/Category%3ARandomQueryGenerator>

# 二、RQG源码下载及使用

## 1.1、下载源码

下载地址：

<https://www.cpan.org/modules/by-module/DBD/>

RQG主要是perl语言开发，perl支持访问数据库是通过DBI模块以及支持数据库厂家的驱动（DBD），因此在新的环境部署RQG时，需要下载和安装DBI以及测试数据库的驱动。因为没有支持perl语言的trafodion驱动，访问trafodion时可通过DBD-JDBC或者DBC-ODBC，因此移植RQG时只需安装DBC-JDBC或者DBD-ODBC模块即可。RQG支持的工具包下载地址在源码页面。

## 1.2、安装DBI模块

下载地址：<http://www.cpan.org/modules/by-module/DBD/>

如DBI-1.642.tar.gz版本安装：

#tar -zxvf DBI-1.642.tar.gz  
#cd DBI-1.642  
#perl Makefile.PL  
#make  
#make test  
#make install

可能遇到的**问题及解决方法：**

**错误一：**

# perl Makefile.PL

Can't locate ExtUtils/MakeMaker.pm in @INC (@INC contains: /usr/local/lib64/perl5 /usr/local/share/perl5 /usr/lib64/perl5/vendor\_perl /usr/share/perl5/vendor\_perl /usr/lib64/perl5 /usr/share/perl5 .) at Makefile.PL line 11.

BEGIN failed--compilation aborted at Makefile.PL line 11.

**解决方法：**

# yum install perl-ExtUtils-CBuilder perl-ExtUtils-MakeMaker

## 1.3、RQG支持ODBC

### 1.2.1、安装DBD-ODBC

如安装使用的包是DBD-ODBC-1.60.tar.gz，以下是安装步骤：

#tar –xvf DBD-ODBC-1.60.tar.gz

#perl Makefile.PL -x unixODBC

#make

#make install

### 1.2.3、RQG源码修改

Perl连接数据库API是调用DBI的方法DBI->connect，调用格式如：DBI->connect(“DBI:ODBC:traf:xxx.xxx.xxx.xxx”, “user”, “pwd”)；因此RQG访问不同的数据只需修改数据库驱动参数、用户名和密码等。

|  |  |
| --- | --- |
| DBI | 接口类型 |
| ODBC | 数据库驱动 |
| traf | 数据源名称 |
| xxx.xxx.xxx.xxx | 数据库server ip |
| user | 登陆数据库用户名 |
| pwd | 登陆数据库密码 |

为了与MySQL区别开，可拷贝lib/GenTest/Executor/MySQL.pm文件到相同的路径下，然后重命名为TrafODBC.pm

红色部分为主要修改部分：

**注意：以下修改代码只是部分修改，已经支持trafodion的测试，但修改的地方比较多，了解更多修改内容，请使用比较工具对修改版本和原始版本进行比较，然后找到差异的地方。**

#vim lib/GenTest/Executor/TrafODBC.pm

package GenTest::Executor::TrafODBC;

sub init {

my $executor = shift;

my @tmp = split('dsn=', $executor->dsn());

my @tmp1 = split(':', $tmp[1]);

my $dsn\_odbc = $tmp1[0];

@tmp = split('user=', $executor->dsn());

@tmp1 = split(':', $tmp[1]);

my $user = $tmp1[0];

@tmp = split('pwd=', $executor->dsn());

@tmp1 = split(':', $tmp[1]);

my $pwd = $tmp1[0];

@tmp = split('schema=', $executor->dsn());

@tmp1 = split(':', $tmp[1]);

my $schema = $tmp1[0];

if(not defined $dsn\_odbc){

$dsn\_odbc = "traf";

}

if(not defined $user){

$user = "trafodion";

}

if(not defined $pwd){

$pwd = "traf123";

}

if(not defined $schema){

$schema = "SEABASE";

}

say("odbc source info:");

say("dsn:$dsn\_odbc");

say("user=$user");

say("pwd=$pwd");

say("schema=$schema");

my $dbh = DBI->connect("DBI:ODBC:".$dsn\_odbc, $user, $pwd, {

PrintError => 1,

RaiseError => 0,

AutoCommit => 1,

mysql\_multi\_statements => 1

} );

if (not defined $dbh) {

say("connect() to dsn ".$executor->dsn()." failed: ".$DBI::errstr);

return STATUS\_ENVIRONMENT\_FAILURE;

}

$executor->setDbh($dbh);

#

# Hack around bug 35676, optiimzer\_switch must be set sesson-wide in order to have effect

# So we read it from the GLOBAL\_VARIABLE table and set it locally to the session

#

$executor->defaultSchema($executor->currentSchema());

say "Default schema: ".$executor->defaultSchema();

return STATUS\_OK;

$dbh->do("

SET optimizer\_switch = (

SELECT variable\_value

FROM INFORMATION\_SCHEMA.GLOBAL\_VARIABLES

WHERE VARIABLE\_NAME = 'optimizer\_switch'

)

");

# say("Executor initialized, id ".$executor->id());

$executor->defaultSchema($executor->currentSchema());

say "Default schema: ".$executor->defaultSchema();

return STATUS\_OK;

}

sub execute {

my ($executor, $query, $silent) = @\_;

my $dbh = $executor->dbh();

return GenTest::Result->new( query => $query, status => STATUS\_UNKNOWN\_ERROR ) if not defined $dbh;

$query = $executor->preprocess($query);

if (

(not defined $executor->[EXECUTOR\_MYSQL\_AUTOCOMMIT]) &&

(

($query =~ m{^\s\*start\s+transaction}io) ||

($query =~ m{^\s\*begin}io)

)

) {

$dbh->do("SET AUTOCOMMIT=OFF");

$executor->[EXECUTOR\_MYSQL\_AUTOCOMMIT] = 0;

}

my $start\_time = Time::HiRes::time();

my $sth = $dbh->prepare($query);

if (not defined $sth) { # Error on PREPARE

return GenTest::Result->new(

query => $query,

status => STATUS\_UNKNOWN\_ERROR,

err => $dbh->err(),

errstr => $dbh->errstr(),

sqlstate => $dbh->state(),

start\_time => $start\_time,

end\_time => Time::HiRes::time()

);

my $errstr = $executor->normalizeError($sth->errstr());

sub currentSchema {

my ($executor,$schema) = @\_;

return undef if not defined $executor->dbh();

if (defined $schema) {

$executor->execute("SET SCHEMA $schema");

}

return $executor->dbh()->selectrow\_array("VALUES(CURRENT\_SCHEMA)");

}

sub getSchemaMetaData {

## Return the result from a query with the following columns:

## 1. Schema (aka database) name

## 2. Table name

## 3. TABLE for tables VIEW for views and MISC for other stuff

## 4. Column name

## 5. PRIMARY for primary key, INDEXED for indexed column and "ORDINARY" for all other columns

my ($self) = @\_;

my $query =

"select ".

"OBJ.schema\_name as schemaname, ".

"OBJ.object\_name as tablename, ".

"case when OBJ.object\_type='VI' then 'view' ".

"when OBJ.object\_type='BT' then 'table' ".

"else 'misc' end, ".

"COL.column\_name as columnname, ".

"case when cons.CONSTRAINT\_TYPE = 'P' then 'primary' else 'ordinary' end ". ## Need to figure out how to find indexes

"from ".

"\"\_MD\_\".OBJECTS as OBJ ".

"join \"\_MD\_\".COLUMNS as col on OBJ.OBJECT\_UID = col.OBJECT\_UID ".

"left join \"\_MD\_\".TABLE\_CONSTRAINTS as cons on OBJ.OBJECT\_UID = cons.TABLE\_UID ".

"where OBJ.object\_name <> 'DUMMY' and OBJ.schema\_name = CURRENT\_SCHEMA";

return $self->dbh()->selectall\_arrayref($query);

}

sub getCollationMetaData {

## Return the result from a query with the following columns:

## 1. Collation name

## 2. Character set

my ($self) = @\_;

my $query ="get tables";

#"SELECT collation\_name,character\_set\_name FROM information\_schema.collations";

return $self->dbh()->selectall\_arrayref($query);

}

修改Constants.pm文件，增加对TrafODBC的支持。红色部分为增加的代码。

#vim lib/GenTest/Executor/Constants.pm

DB\_UNKNOWN

DB\_DUMMY

DB\_MYSQL

DB\_POSTGRES

DB\_JAVADB

DB\_DRIZZLE

DB\_TRAFODBC

DEFAULT\_MTR\_BUILD\_THREAD

);

use constant DB\_UNKNOWN => 0;

use constant DB\_DUMMY => 1;

use constant DB\_MYSQL => 2;

use constant DB\_POSTGRES => 3;

use constant DB\_JAVADB => 4;

use constant DB\_DRIZZLE => 5;

use constant DB\_TRAFODBC => 6;

修改Executor.pm文件，增加对TrafODBC的支持。红色部分为增加的代码。

#vim lib/GenTest/Executor/Executor.pm

sub newFromDSN {

my ($self,$dsn,$channel) = @\_;

if ($dsn =~ m/^dbi:mysql:/i) {

require GenTest::Executor::MySQL;

return GenTest::Executor::MySQL->new(dsn => $dsn, channel => $channel);

} elsif ($dsn =~ m/^dbi:odbc:/i) {

require GenTest::Executor::TrafODBC;

return GenTest::Executor::TrafODBC->new(dsn => $dsn, channel => $channel);

} elsif ($dsn =~ m/^dbi:drizzle:/i) {

require GenTest::Executor::Drizzle;

return GenTest::Executor::Drizzle->new(dsn => $dsn);

} elsif ($dsn =~ m/^dbi:JDBC:.\*url=jdbc:derby:/i) {

require GenTest::Executor::JavaDB;

return GenTest::Executor::JavaDB->new(dsn => $dsn);

} elsif ($dsn =~ m/^dbi:JDBC:.\*url=jdbc:t4jdbc:/i) {

require GenTest::Executor::JavaDB;

return GenTest::Executor::JavaDB->new(dsn => $dsn);

} elsif ($dsn =~ m/^dbi:Pg:/i) {

require GenTest::Executor::Postgres;

return GenTest::Executor::Postgres->new(dsn => $dsn);

} elsif ($dsn =~ m/^dummy/) {

require GenTest::Executor::Dummy;

return GenTest::Executor::Dummy->new(dsn => $dsn);

} else {

say("Unsupported dsn: $dsn");

exit(STATUS\_ENVIRONMENT\_FAILURE);

}

}

sub type {

my ($self) = @\_;

if (ref($self) eq "GenTest::Executor::JavaDB") {

return DB\_JAVADB;

} elsif (ref($self) eq "GenTest::Executor::MySQL") {

return DB\_MYSQL;

} elsif (ref($self) eq "GenTest::Executor::TrafODBC") {

return DB\_TRAFODBC;

} elsif (ref($self) eq "GenTest::Executor::Drizzle") {

return DB\_DRIZZLE;

} elsif (ref($self) eq "GenTest::Executor::Postgres") {

return DB\_POSTGRES;

} elsif (ref($self) eq "GenTest::Executor::Dummy") {

if ($self->dsn =~ m/mysql/) {

return DB\_MYSQL;

} elsif ($self->dsn =~ m/postgres/) {

return DB\_POSTGRES;

} if ($self->dsn =~ m/javadb/) {

return DB\_JAVADB;

} else {

return DB\_DUMMY;

}

} else {

return DB\_UNKNOWN;

}

}

my @dbid = ("Unknown","Dummy", "MySQL", "TrafODBC", "Postgres","JavaDB","Drizzle");

### 1.2.4、测试

使用方法以及语法文件格式请参考conf路径的下的demo以及README文件。

生成query测试：

./gensql.pl --grammar=conf/example.yy --dsn=dbi:odbc:dsn=traf:user=trafodion:pwd=traf123:schema=seabase:sqltype=trafodion

生成query并执行测试：

./gentest.pl --grammar=conf/example.yy --dsn=dbi:odbc:dsn=traf:user=trafodion:pwd=traf123:schema=seabase:sqltype=trafodion

生成数据，如ddl测试：

./gendata.pl --dsn=dbi:odbc:dsn=traf:user=trafodion:pwd=traf123:schema=seabase:sqltype=trafodion --spec=conf/example.zz

**注：参数sqltype=trafodion用与区分mysql和trafodion在生成query时的语法差异，否则默认使用mysql的语法格式。其他参数值请参考本章节的“1.6 RQG使用说明”。**

## 1.4、RQG支持JDBC

### 1.4.1、安装DBD-JDBC

如：DBD-JDBC-0.71.tar.gz

# tar -xvf DBD-JDBC-0.71.tar.gz

#cd DBD-JDBC-0.71

# perl Makefile.PL

**可能遇到的问题及解决方法：**

**问题一：**

如出现类似的BER.pm模块错误：Can't locate BER.pm in @INC

解决方法：#perl -MCPAN -e "install Convert::BER"

Perl通过JDBC访问数据库是通过DBD\_JDBC server进行连接和操作。因此首先保证DBD JDBC Server正常启动；

下载DBD-JDBC包然后编译成功后，拷贝使用的jdbc jar包到指定的路径（任意路径即可，运行前通过环境变量指定即可），编写启动和停止脚本，脚本如下：

#vim start\_dbd.sh

#!/bin/bash

export CLASSPATH=./dbd\_jdbc.jar:./jdbcT4-2.5.1.jar:log4j-1.2.17.jar

nohup java -Djdbc.drivers=org.trafodion.jdbc.t4.T4Driver -Ddbd.port=9001 -Ddbd.trace=tedious com.vizdom.dbd.jdbc.Server 2>&1 >dbd\_jdbc\_server.log &

停止脚本如下：

#vim stop\_dbd.sh

#!/bin/bash

ps aux | grep com.vizdom.dbd.jdbc.Server | grep -v grep | awk '{system("kill -9 "$2)}'

**完成后执行启动脚本，是否成功查看监听的端口或者查看启动日志。**

### 1.4.2、RQG支持JDBC

修改文件./lib/ GenTest/Executor.pm，在函数newFromDSN增加jdbc:t4jdbc相关的字段，用于trafodion jdbc的调用（主要增加的代码为红色部分）。

**注意：以下修改代码只是部分修改，已经支持trafodion的测试，但修改的地方比较多，了解更多修改内容，请使用比较工具对修改版本和原始版本进行比较，然后找到差异的地方。**

#vim ./lib/ GenTest/Executor.pm

sub newFromDSN {

my ($self,$dsn,$channel) = @\_;

if ($dsn =~ m/^dbi:mysql:/i) {

require GenTest::Executor::MySQL;

return GenTest::Executor::MySQL->new(dsn => $dsn, channel => $channel);

} elsif ($dsn =~ m/^dbi:traf:/i) {

require GenTest::Executor::TrafODBC;

return GenTest::Executor::TrafODBC->new(dsn => $dsn, channel => $channel);

} elsif ($dsn =~ m/^dbi:drizzle:/i) {

require GenTest::Executor::Drizzle;

return GenTest::Executor::Drizzle->new(dsn => $dsn);

} elsif ($dsn =~ m/^dbi:JDBC:.\*url=jdbc:derby:/i) {

require GenTest::Executor::JavaDB;

return GenTest::Executor::JavaDB->new(dsn => $dsn);

} elsif ($dsn =~ m/^dbi:JDBC:.\*url=jdbc:t4jdbc:/i) {

require GenTest::Executor::JavaDB;

return GenTest::Executor::JavaDB->new(dsn => $dsn);

} elsif ($dsn =~ m/^dbi:Pg:/i) {

require GenTest::Executor::Postgres;

return GenTest::Executor::Postgres->new(dsn => $dsn);

} elsif ($dsn =~ m/^dummy/) {

require GenTest::Executor::Dummy;

return GenTest::Executor::Dummy->new(dsn => $dsn);

} else {

say("Unsupported dsn: $dsn");

exit(STATUS\_ENVIRONMENT\_FAILURE);

}

}

修改与数据库交互的pm包，因为MySQL与trafodion存在语法上的差异，因此需修改部分函数的sql语法。以下代码红色部分为主要修改的地方。

#vim lib/GenTest/Executor/JavaDB.pm

sub init {

my $self = shift;

## The jdbc URL may contain both = and ;. These characters have to

## be encoded in order to pass unharmed through DBI. We assume

## here, that url=.... is the last part of the dsn, and encode

## everything after url=

my ($dsn\_part1,$dsn\_part2) = split(/url=/,$self->dsn());

#$dsn\_part2 =~ s/([=;])/uc sprintf("%%%02x",ord($1))/eg;

$dsn\_part2 =~ s/([=;])/uc sprintf("=",ord($1))/eg;

say("part1:$dsn\_part1");

say("part2:$dsn\_part2");

my @tmp = split('user=', $dsn\_part2);

my @tmp1 = split(':', $tmp[1]);

my $user = $tmp1[0];

@tmp = split('pwd=', $dsn\_part2);

@tmp1 = split(':', $tmp[1]);

my $pwd = $tmp1[0];

@tmp = split('schema=', $dsn\_part2);

@tmp1 = split(':', $tmp[1]);

my $schema = $tmp1[0];

if(not defined $user){

$user = "trafodion";

}

if(not defined $pwd){

$pwd = "traf123";

}

if(not defined $schema){

$schema = "SEABASE";

}

say("jdbc source info:");

say("user=$user");

say("pwd=$pwd");

say("schema=$schema");

my $dbh = DBI->connect($dsn\_part1."url=".$dsn\_part2, $user, $pwd, {

PrintError => 0,

RaiseError => 0,

AutoCommit => 1

});

if (not defined $dbh) {

say("connect() to dsn ".$self->dsn()." failed: ".$DBI::errstr);

return STATUS\_ENVIRONMENT\_FAILURE;

}

$self->setDbh($dbh);

#$self->defaultSchema($self->currentSchema());

#目前参数传递schme时，不能生效，因此根据测试需要修改schema名

$self->defaultSchema($self->currentSchema("ODBCTEST"));

say "Default schema: ".$self->defaultSchema();

return STATUS\_OK;

}

sub currentSchema {

my ($self,$schema) = @\_;

return undef if not defined $self->dbh();

if (defined $schema) {

#set schema语法与mysql有差异

$self->execute("SET SCHEMA $schema");

}

#获取当前schema语法与mysql有差别

return $self->dbh()->selectrow\_array("VALUES(CURRENT\_SCHEMA)");

}

sub getSchemaMetaData {

## Return the result from a query with the following columns:

## 1. Schema (aka database) name

## 2. Table name

## 3. TABLE for tables VIEW for views and MISC for other stuff

## 4. Column name

## 5. PRIMARY for primary key, INDEXED for indexed column and "ORDINARY" for all other columns

my ($self) = @\_;

my $query =

"select ".

"OBJ.schema\_name as schemaname, ".

"OBJ.object\_name as tablename, ".

"case when OBJ.object\_type='VI' then 'view' ".

"when OBJ.object\_type='BT' then 'table' ".

"else 'misc' end, ".

"COL.column\_name as columnname, ".

"case when cons.CONSTRAINT\_TYPE = 'P' then 'primary' else 'ordinary' end ".## Need to figure out how to find indexes

"from ".

"\"\_MD\_\".OBJECTS as OBJ ".

"join \"\_MD\_\".COLUMNS as col on OBJ.OBJECT\_UID = col.OBJECT\_UID ".

"left join \"\_MD\_\".TABLE\_CONSTRAINTS as cons on OBJ.OBJECT\_UID = cons.TABLE\_UID ".

"where OBJ.object\_name <> 'DUMMY' and OBJ.schema\_name = CURRENT\_SCHEMA";

return $self->dbh()->selectall\_arrayref($query);

}

### 1.4.3、测试

运行测试前，保证dbd server正常启动。执行start\_dbd.sh脚本即可。

使用方法以及语法文件格式请参考conf路径的下的demo以及README文件。

生成query测试：

./gensql.pl --grammar=conf/example.yy --dsn 'dbi:JDBC:hostname=localhost:9001;url=jdbc:t4jdbc://10.10.14.65:23400/:user=trafodion:pwd=traf123:schema=seabase:sqltype=trafodion'

生成query并执行测试：

./gentest.pl --grammar=conf/example.yy --dsn 'dbi:JDBC:hostname=localhost:9001;url=jdbc:t4jdbc://10.10.14.65:23400/:user=trafodion:pwd=traf123:schema=seabase:sqltype=trafodion'

生成数据，如ddl测试：

./gendata.pl --dsn 'dbi:JDBC:hostname=localhost:9001;url=jdbc:t4jdbc://10.10.14.65:23400/:user=trafodion:pwd=traf123:schema=seabase:sqltype=trafodion' --spec=conf/example.zz

**注：参数sqltype=trafodion用与区分mysql和trafodion在生成query时的语法差异，否则默认使用mysql的语法格式。其他参数值请参考本章节的“1.6 RQG使用说明”。**

## 1.5、代码优化和完善

因MySQL与trafodion存在语法上的差异，因此语法生成文件lib/GenTest/Generator/FromGrammar.pm修改根据trafodion的语法特征进行修改，因为修改地方比较多，可以通过比较工具对比下lib\GenTest\Generator\FromGrammar.pm文件与源码文件的差异。

其他完善和优化的代码块涉及的代码比较多，请使用比较工具对比和查阅。

## 1.6、RQG使用说明

RQG工具使用样例可参考README文件。

**生成query：**

gensql.pl主要是用于生成query测试，该程序会根据指定的语法文件，生成query并终端输出； 语法格式请参考conf路径下的配置文件。

**部分参数使用说明：**

**--grammar**

指定的语法文件。

**--queries**

总的生成query数，默认是1000.

**--dsn**

连接数据库的dsn。

**--mask-level**

语法生成时，关键字段嵌套的深度，默认最大值是500，当参数值小于默认值时，使用默认值；即该参数值大于500时才有效，否则使用默认最大值500。

**--help**

打印使用帮助。

**生成query并执行：**

gentest.pl主要用于生成query并执行测试，该程序会根据指定的语法文件，生成query并执行。语法格式请参考conf路径下的配置文件。

**部分参数使用说明：**

**--grammar**

指定的语法文件。

**--queries**

总的生成query数，默认是1000.

**--dsn**

连接数据库的dsn。

**--mask-level**

语法生成时，关键字段嵌套的深度，默认最大值是500，当参数值小于默认值时，使用默认值；即该参数值大于500时才有效，否则使用默认最大值500。

**--help**

打印使用帮助。

**--threads**

并发执行的线程数。

**生成数据测试：**

gendata.pl主要用于生成ddl并执行的测试等；语法格式请参考conf路径下的配置文件。测试ddl时并非只有gendata.pl支持，gensql.pl或者gentest.pl同样可以作为ddl的测试，详细使用参数请参考官方文档。

**部分参数使用说明：**

**--spec**

指定的语法文件。

**--dsn**

连接数据库的dsn。

**--help**

打印使用帮助。

**其他主要参数值说明：**

|  |  |
| --- | --- |
| dbi:JDBC | 使用JDBC驱动连接数据库，注意”JDBC”必须是大写字母。 |
| hostname:9001 | DBD-JDBC服务IP及端口 |
| url=jdbc:t4jdbc://10.10.14.65:23400/: | Jdbc连接数据库使用的url，主要修改数据库服务IP。**注：”url=xxx”前以分号”;”隔开，其他字段值以冒号”:”间隔。** |
| dbi:odbc:traf | 使用odbc方式连接数据库，其中”traf”为ODBC数据源名称，可自定义，使用和配置请参考ODBC的使用。 |
| user | 登陆数据库的用户名 |
| pwd | 登陆数据库的密码 |
| schema | 测试使用的schema。**注：使用ODBC时，此参数无效，由ODBC配置的数据源决定。** |
| sqltype | 等于”trafodion”时，表示使用trafodion的语法，否则默认使用mysql语法，使用mysql语法时，生成的query，表名和列名等会带有”’”，trafodion不支持这种写法，因此需注意。 |

# 三、语法规则

语法规则主要参考官方指导文档。

<https://github.com/RQG/RQG-Documentation/wiki/Category%3ARandomQueryGenerator>

RQG约定了很多的字段，不同的字段代表着不同的意义，如”\_table”、”\_field”和”\_digit”等字段是RQG约定的关键字，当语法文件有这些字段时，会根据数据库已存在的表进行替换，然后生成query。字段的设计需根据不同数据库特性及语法规则设计。

设计语法文件时请参考conf路径下的模板或者官网的使用文档。

部分字段说明：

\_table

替换当前schema下的表名。

\_field

替换当前表的列名。

\_digit

替换为数字。

\_schema、\_database

替换为当前的schema名。

**注：目前，语法了解甚少，后续再逐步完善文档，敬请谅解！**