

# **Random Query Generator**

<http://forge.mysql.com/wiki/RQG>

<http://launchpad.net/randgen>

[philip.stoev@sun.com](mailto:philip.stoev@sun.com)

# Random Query Generator

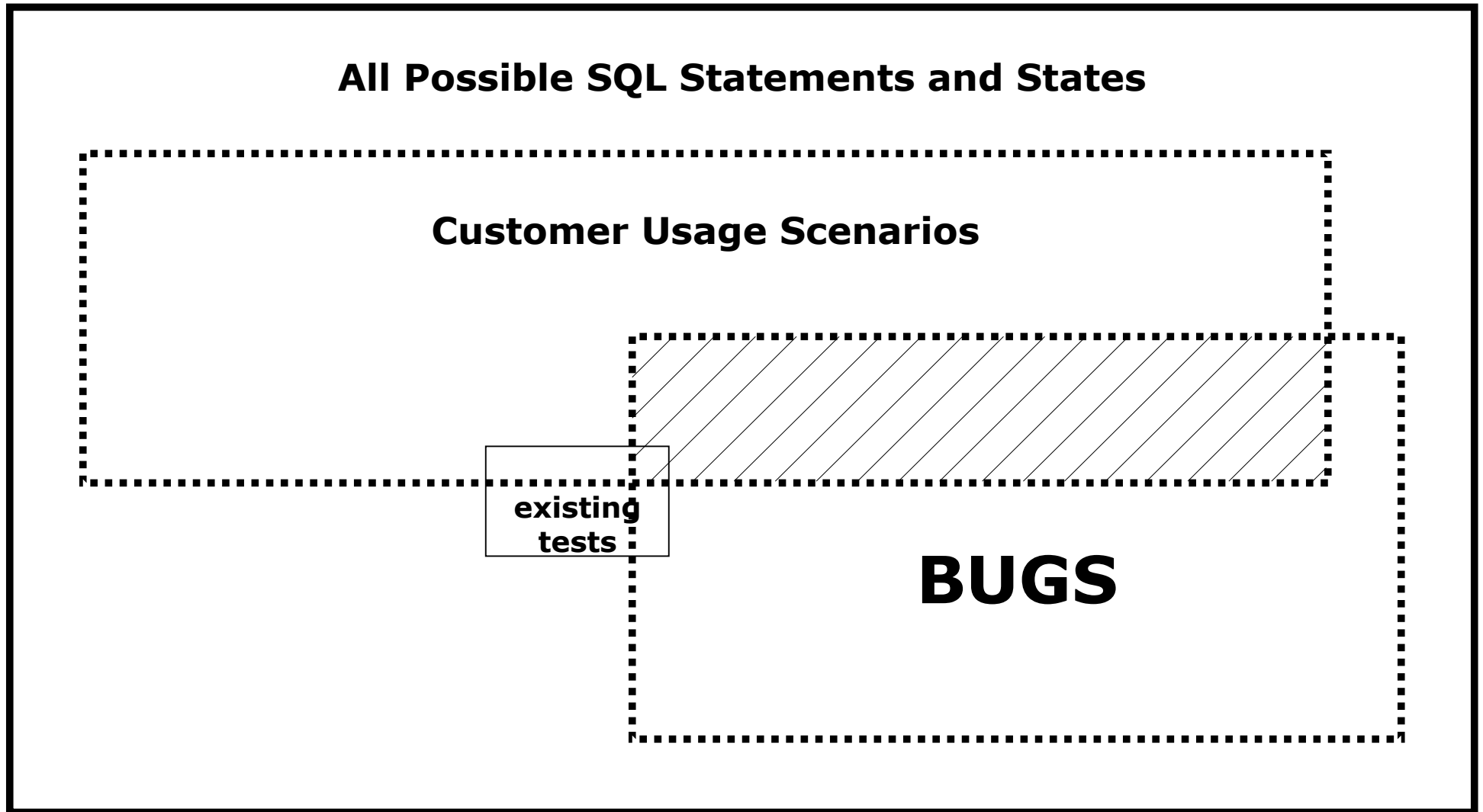
- Executes targeted pseudo-random queries against pseudo-random data
- Monitors query result and server status
- Reports crashes and other deviations

pseudo-random = repeatable tests

- every (single-threaded) run produces the same results.

# Testing Coverage

## [Slutz, 1998]



# Design Goals

- Multi-platform
  - supports Linux, Solaris, Windows (native)
- Fully automatic
  - runs unattended
  - provides unequivocal pass/fail indication
- Fully customizable
  - tests all types of queries, tables and fields
  - plug-in components for validating the result
- Repeatable runs

# Installation

```
bzr branch lp:~randgen/randgen/main
```

Required Perl modules:

- DBI
- DBD::mysql
- Math::Random::MT or  
Math::Random::MT::Perl

# The Data Generator

- Creates tables based on configuration
  - various column types, keys, engines, partitions, etc.
- Populates them with random data
  - matching certain desired characteristics
- Default table layout for quick testing
- Future: generate data from schema
  - maintain desired relationship between tables

# Sample Configuration File

```
$tables = {
  rows => [0, 1, 10, 100],
  partitions => [ undef , 'KEY (pk) PARTITIONS 2' ]
};
$fields = {
  types => [ 'int', 'char', 'enum', 'set' ],
  indexes => [undef, 'key' ],
  null => [undef, 'not null'],
  default => [undef, 'default null'],
  sign => [undef, 'unsigned'],
  charsets => ['utf8', 'latin1']
};
$data = {
  numbers => [ 'digit', 'null', undef ],
  strings => [ 'letter', 'english' ],
  blobs => [ 'data' ],
  temporals => ['date', 'year', 'null', undef ]
}
```

# Sample Database Schema

```
CREATE TABLE `table0_int_autoinc` (  
  `int_unsigned` int unsigned,  
  `int_key` int,  
  pk integer auto_increment,  
  `char_utf8` char (1) CHARACTER SET utf8,  
  `int` int,  
  `char_key_utf8` char (1) CHARACTER SET utf8,  
  `char_latin1` char (1) CHARACTER SET latin1,  
  `char_key_latin1` char (1) CHARACTER SET latin1,  
  `int_unsigned_key` int unsigned,  
  key (`int_key` ),  
  primary key (pk),  
  key (`char_key_utf8` ),  
  key (`char_key_latin1` ),  
  key (`int_unsigned_key` )  
);
```

# The Grammar

- Describes the queries to generate, YACC-style:

query:

```
SELECT _field FROM _table where LIMIT _digit |  
UPDATE _table SET _field = _digit where ;
```

where:

```
WHERE _field > _digit |  
WHERE _field IS NULL ;
```

- Provides convenience functions  
\_field , \_table , \_digit, etc.

# Sample Grammar File

query:

```
update | insert | delete ;
```

update:

```
UPDATE _table SET _field = _digit  
WHERE condition LIMIT _digit ;
```

delete:

```
DELETE FROM _table  
WHERE condition LIMIT _digit ;
```

insert:

```
INSERT INTO _table ( _field ) VALUES ( _digit ) ;
```

condition:

```
_field < digit | _field = _digit ;
```

# Sample Generated Query

```
SELECT SUM(DISTINCT OUTR . `varchar_nokey` ) AS X
FROM C AS OUTR
WHERE OUTR . `pk` IN (
    SELECT INNER . `pk` AS Y
    FROM BB AS INNER2
    LEFT JOIN BB AS INNER
    ON ( INNER2 . `datetime_nokey` > INNER . `time_nokey` )
    WHERE INNER . `int_nokey` > INNER . `int_nokey`
    AND OUTR . `time_key` < '2001-04-01`
)
AND OUTR . `int_nokey` > 1
HAVING X <> '18:18:19`
ORDER BY OUTR . `varchar_nokey` , OUTR . `pk`;
```

# Testing Configurations

- Compare two different servers:
  - different versions or binaries:  
5.1 v.s. 6.0
  - different configurations:  
6.0 with and without semijoin
  - different engines:  
Falcon v.s. Innodb
- The future: MySQL v.s. other SQL servers
  - PostgreSQL

# Validators

are plug-in modules which validate the outcome of every query

- check if two servers returned the same result
  - independent of storage engine row ordering
- detect database corruption
  - garbage in result or error message

# Reporters

are plug-in modules that check the overall operation of the server during the test

- run periodically in their own monitoring process
- run at end of test to determine final test outcome
- Detect deadlocks / stalled queries
- Print nice backtraces on multiple platforms
- Force recovery and check for the outcome

# Replication

- monitor slave operation during the test
- compare master and slave data
  - during and/or after the test
- different binaries, engines or configuration on master and slave
- testing modes that detect errors close to the source
- inject random network failures
- use all binary log modes