The Random Query Generator

a cost-effective, emotionally
gratifying database testing

Current state of DB Testing

- QA is not considered fun
- QA is very resource-intensive
  - Some commercial companies claim close to 1:1 Tester-to-Developer ratio

so, sometimes,

  - Insufficient QA is done
  - Products are released with serious issues

Lessons Learned

- Some database products can not run even simple DML
  - “runs DBT2” does not mean bug-free
- ACID and database recovery are difficult
  - repeated deliberate crashing is required
- Easy sequences of SQL can cause crashes, assertions, table corruption

RQG Mode of Operation

1. Generate SQL based on a grammar file;

2. Execute SQL concurrently against one or more databases via Perl DBI/JDBC/ODBC;

3. Validate error codes and result sets with Perl or by comparing between databases

Functional Testing

- Two- and three-way comparisons with other database products
  - including MySQL, JavaDB and PostgreSQL
- API for custom Validators that can check the result set, error code, affected_rows from each query

Stress Testing

- A comprehensive SQL grammar can generate a wide variety of DDL/DML
- Concurrent execution fishes out any indisirable interactions between any number of commands;
- Detects crashes, assertions, deadlocks, valgrind and data corruption;

Transactional Consistency

- Test Repeatable Read via automatic re-execution of all SELECT queries;
- Test recovery via deliberate crashing
  - including crashing the recovery itself
- Test transactional consistency via invariants

Replication Testing

- Execute any workload on master
- Introduce adverse replication situations
  - network outages, server restarts, etc.
- Monitor slave for errors
- Dump and compare master and slave

Performance Testing

- Performance comparisons between databases can be made on a per-query level
- RQG reports queries that have suffered a performance regression above a threshold
- Useful mostly for Optimizer testing, but random data and queries may be unrealistic

Goodies

- Deadlock detection
- Backtraces are dumped on a crash
- Automatic simplification for:
  - grammars
  - individual queries
  - sequences of queries (logs, mysqltest files)
- Automatic generation of test cases
  - currently, in mysqltest format

Automation

- If you have a test result database
  - can report to an XML file
- If you do not have it,
  - log to stdout is verbose enough
  - backtraces are dumped in the log
- Core file, binary, bzr version-info are saved
- Can run the same test repeatedly with various settings

SQL Not Required

Grammar-based approach can be used to generate queries in proprietary languages and APIs

- XML queries
- generate Perl code directly
- generate intermediate representation that a custom Executor can covert to API calls

```
HANDLER t1 OPEN
HANDLER t1 READ idx1 = (’a’,’b’,’c’)
HANDLER t1 CLOSE
```

Open Questions

- Difficult to have concurrent functional testing
- Grammar must be constructed with care
  - using a protocol for grammar creation
- Purely random data and queries may be wildly unrealisic
  - unless data and query patterns come from a customer or a benchmark

Cost-effective

especially if:

- used in comparison testing between two configurations or codebases or
- when comparing against other database products, and
- when using one of the existing grammars