SQLancer 使用手册

0. 引言

SQLancer 是一个针对数据库管理系统（DBMS）的高级模糊测试（fuzzing）工具，旨在发现逻辑错误（返回不正确结果集）与性能问题。它使用基于语法的生成器产生有效的 SQL 查询，并通过创新的“预言机”（oracles）来判断查询结果是否正确。本文档整合了两份原始手册的内容，提供从理论背景、安装部署、运行示例、预言机原理到高级集成与扩展的完整参考。

** 本手册适合对数据库原理和软件测试有基本了解的开发者与测试工程师。 **

1. 快速上手

本节提供最核心的安装和运行步骤，让你可以在几分钟内启动一次测试。

1. 克隆 SQLancer 仓库

```
git clone https://github.com/sqlancer/sqlancer
cd sqlancer
```

2. 使用 Maven 打包项目（跳过测试以加快速度）

```
mvn package -DskipTests
```

3. 运行一个针对 SQLite 的测试示例

```
# --num-threads 4: 使用 4 个并发线程
# sqlite3:       目标数据库
# --oracle NoREC:  使用 NoREC 预言机
java -jar sqlancer-*.jar --num-threads 4 sqlite3 --oracle NoREC
```

当看到 SQLancer 开始输出测试日志时，说明它已成功运行。

2. 环境配置

在深入使用 SQLancer 之前，请确保你的系统满足以下环境要求。

2.1. 必需环境

- Java Development Kit (JDK): SQLancer 是一个 Java 项目，需要 JDK 8 或更高版本来编译和运行。
- Maven: 用于项目构建和依赖管理。大多数包管理器都可以直接安装（例如 `sudo apt install maven`）。
- Git: 用于从 GitHub 克隆项目仓库。
- 目标数据库: 你需要预先安装并运行你希望测试的数据库系统，例如 PostgreSQL, MySQL, SQLite 等。SQLancer 不会为你安装数据库。

2.2. 验证环境

你可以通过以下命令检查环境是否配置妥当：

```
java -version
mvn -version
git --version
```

如果这些命令都能成功返回版本信息，说明你的基础环境已准备就绪。

3. 目录

4. 理论基础与模糊测试方法

现代 DBMS 是复杂软件系统，逻辑错误（Logic Bugs）可能在不崩溃的情况下返回错误结果，难以通过传统测试发现。模糊测试（fuzzing）通过自动化生成大量输入，触发边界条件与未考虑路径，从而找到错误。

模糊测试的三大组件：生成器（Generator/Poet）、传递器（Courier）、预言机（Oracle）。

两类模糊测试方法：

- 基于变异（Mutation-based）：对已有种子输入做随机变更，优点是实现简单、速度快，但多数变异会被解析器早期拒绝，难以测试优化器或执行引擎。
- 基于生成（Generation-based / Grammar-based）：根据 SQL 语法从头生成有效查询，能够深入测试执行引擎与优化器，SQLancer 采用此法以确保生成的查询语法有效。

测试预言机问题（Test Oracle Problem）是模糊测试的核心挑战之一：当目标程序没有崩溃时，如何判定其行为是否正确？SQLancer 的创新在于设计多种预言机（如 TLP、NoREC、PQS 等）来检测逻辑错误而无需预先知道“正确答案”。

5. SQLancer 框架与架构原则

SQLancer 通常采用两阶段流程：

1. 数据库状态生成（Database Generation）：生成表、插入数据、建立索引与视图，创建复杂数据库状态以提高触发错误的概率。
2. 测试（Testing）：在固定的数据库状态上生成并执行查询，使用预言机判断结果是否正确。

将状态生成与查询测试分离的好处是：当第二阶段出现错误时，更容易将其归因于查询处理（规划器、优化器、执行器）而非数据修改交互。

SQLancer 是用 Java 实现并使用 Maven 构建（原始项目）。不过，本手册也包含以 Python/轻量工具实现的变体或衍生项目（例如 `SQLlancer` 文档中所描述的 Python 风格用法），便于用户在不同环境中采用相同方法论。

6. 安装与初始部署

SQLancer 是用 Java 实现并使用 Maven 构建（原始项目）。以下步骤为典型的 SQLancer（Java 版本）安装与运行步骤。

Java / 官方 SQLancer 安装（推荐用于完整功能）

1. 克隆仓库：

```
git clone https://github.com/sqlancer/sqlancer
cd sqlancer
```

2. 打包项目（跳过测试以加快构建）：

```
mvn package -DskipTests
```

3. 运行示例命令：

```
java -jar sqlancer-*.jar --num-threads 4 sqlite3 --oracle NoREC
```

说明：

- `--num-threads 4`：使用 4 个并发线程。
- `sqlite3`：目标 DBMS，可替换为 `postgres`，`mysql` 等（需对应 provider 支持）。
- `-- 注：Java 版本与轻量 Python 版本在功能与可扩展性上不同。Java 版本（官方 SQLancer）提供成熟的预言机与 provider 生态，适合深入研究 DBMS；Python 版本适用于快速原型或特定语法驱动的测试。

7. 操作参数与配置

SQLancer 的命令行参数遵循一个重要规则：通用选项（例如 `--num-threads`）必须位于 DBMS 名称之前，而 DBMS 特定选项（例如 `--oracle`）必须位于 DBMS 名称之后。

常用参数示例：

```
 --num-threads <n>           # 通用：在 DBMS 之前
--timeout-seconds <secs>    # 通用：设置整个运行或单条查询超时时间
--num-tries <n>             # 通用：在发现 n 个错误后退出
--log-each-select           # 通用：记录每条 SELECT（默认启用）
[dbms-name]                 # 例如 sqlite3, postgres
--oracle <OracleName>       # 在 DBMS 名称之后指定使用的预言机，例如 NoREC、TLP、PQS
```

一些用于自动化归约与调试的实验性选项（部分仅在官方 Java 版本可用）：

```
--use-reducer
--ast-reducer-max-steps <n>
--ast-reducer-max-time <secs>
```

此外，日志目录下会保存触发错误的 `.log` 文件，包含用于重现错误的最小 SQL 语句集。

8. 核心预言机详解

SQLancer 最有价值的贡献之一是其多样的预言机（Oracles），它们将查询转换为可验证的断言，从而在不需要“正确答案”的情况下发现错误。

1) 三值逻辑划分（Ternary Logic Partitioning - TLP）

原理：对布尔谓词 p 的求值存在三个可能：TRUE、FALSE、NULL。TLP 将原始查询 Q 拆分为三个子查询：WHERE p、WHERE NOT p、WHERE p IS NULL。三个结果的并集应与没有谓词的查询结果等价。若不等价则发现错误。

以下是 TLP 的工作原理示意图：

```
原始查询 Q: SELECT * FROM t WHERE p

拆分为：
- Q1: SELECT * FROM t WHERE p          (TRUE 行)
- Q2: SELECT * FROM t WHERE NOT p      (FALSE 行)
- Q3: SELECT * FROM t WHERE p IS NULL  (NULL 行)

验证: Q1 ∪ Q2 ∪ Q3 == SELECT * FROM t
若不等，则发现逻辑错误
```

适用：检测 WHERE、GROUP BY、HAVING、聚合函数、DISTINCT 等对 NULL 与布尔逻辑敏感的场景。

2) 非优化引用引擎构建（NoREC）

原理：将查询改写为一个强制全表扫描、难以被优化的等价查询。比如把 `SELECT * FROM t WHERE p` 转换为 `SELECT (p IS TRUE) FROM t`，并比较结果中 TRUE 的计数与原查询的返回行数是否匹配。若不匹配则判定为优化器或执行错误。

NoREC 原理示意图：

```
优化查询: SELECT * FROM t WHERE p  (可能使用索引)

转换为非优化查询: SELECT (p IS TRUE) FROM t  (强制全表扫描)

比较:
- 优化查询结果行数 == 非优化查询 TRUE 计数
- 若不匹配，则优化器错误
```

适用：主要用于发现优化器相关错误，尤其是索引使用与过滤条件相关的错误。

3) 枢轴查询合成（Pivoted Query Synthesis - PQS）

原理：随机选取一行（pivot row），生成一个谓词并保证对该行为 TRUE。执行查询后，枢轴行必须出现在结果集，否则存在执行或语义错误。

PQS 工作流程：

```
1. 从表中随机选择一行 (Pivot Row)

2. 生成谓词 p，使 p 对 Pivot Row 求值为 TRUE

3. 执行查询: SELECT * FROM t WHERE p

4. 检查: Pivot Row 是否在结果集中？
   - 是: 正常
   - 否: 发现逻辑错误
```

适用：对复杂 WHERE 子句、索引和执行引擎错误检测非常有效。

4) 基数估计约束测试（CERT）

原理：校验查询规划器估计的基数与实际行数之间的显著偏差，从而发现性能相关错误（选择了次优计划）。

适用：用于性能回归、查询计划选择错误的检测（非逻辑错误）。

5) 其他预言机

- 差分查询计划（DQP）：对连接与子查询等场景的逻辑错误检测。
- 崩溃/隐式崩溃：当 DBMS 进程终止时记录为崩溃错误。
- 非预期内部错误：将返回但不在预期错误列表中的内部错误视为潜在问题。

组合使用多个预言机能覆盖 DBMS 的不同子系统（解析器、规划器、优化器、执行引擎、存储引擎）。

9. 扩展与开发指南

对新 DBMS 或新 SQL 方言的支持通常需要在 SQLancer 中实现 provider：包含连接管理、Schema 生成、表达式生成器（ExpressionGenerator）以及预言机的适配实现。

概览步骤：

1. 确定目标 DBMS 及其 SQL 方言。
2. 在项目中新增 provider 包（Java 项目中为一个新的 package）。
3. 实现 Provider 类（管理连接与选项）。
4. 实现 Schema 生成器以创建表、索引、视图等。
5. 实现 ExpressionGenerator，为目标方言生成合法的表达式（这是最复杂的部分）。
6. 适配并实现需要的预言机（如 NoREC、PQS 等）。

示例参考：DataFusion 的 SQLancer 支持分支（`datafusion-sqlancer`）展示了如何为 DataFusion 实现 `DataFusionExpressionGenerator.java` 与 `DataFusionNoRECOracle.java`。

设计要点与权衡：

- SQLancer 的生成器采用递归与类型化的策略（生成布尔表达式可能需要生成数字表达式等）。
- 该设计虽不如声明式语法（如 ANTLR）易于扩展，但能提供更细粒度和性能优化的生成逻辑，这对预言机（尤其是 PQS 等需要与生成器紧密协作的预言机）非常重要。

10. 在 CI/CD 中部署

模糊测试通常是长期运行的任务，不适合与短时间的同步 CI 步骤直接绑定。将 SQLancer 集成到异步 CI 流程的建议：

1. 触发方式：按计划（夜间）、或在合并到主分支后触发。
2. 环境准备：在隔离容器或 VM 中构建并启动目标 DBMS 的最新版本。
3. 启动 SQLancer：以后台进程或异步作业运行，并设置超时（例如 `--timeout-seconds 7200` 表示运行 2 小时）。
4. 日志聚合：将 `logs/` 或结果目录写到持久存储（如 S3）。
5. 异步错误分类：独立进程扫描日志目录，发现 `.log` 后自动在 Issue Tracker 中创建问题并附带 `.log` 文件。
6. 自动化报告：将触发的 `.log` 与构建元信息一起提交到错误跟踪系统。

最佳实践：

- 将模糊测试作为异步任务运行，避免阻塞普通流水线。
- 使用容器化（Docker）保证可复现的环境。
- 维护语法/种子库以指导数据库状态生成与语句组合。
- 不同运行使用不同预言机以最大化覆盖（例如夜间运行 NoREC，周末运行 TLP）。

11. 结果分析与日志

SQLancer 在日志目录下生成多类文件：

- `*-cur.log`：当 `--log-each-select` 启用时，包含当前会话执行的所有 SELECT 语句日志。
- `*.log`（发生错误时）：包含用于重现错误的最小 SQL 语句集合（DDL/DML + 触发的查询）。这类文件设计为可执行脚本以便开发者复现并调试问题。

最小化 `.log` 文件通常由工具自动归约生成（实验性 AST 归约器可进一步缩小测试用例）。

复现错误的步骤示例：

```
 # 在干净环境启动数据库实例
# 执行日志中提供的 .sql 或 .log 文件
# 观察是否复现崩溃或逻辑差异
psql -h localhost -U testuser -d testdb -f reproduced_error.sql
```

对性能问题（hangs）建议使用 `EXPLAIN` 或 `EXPLAIN ANALYZE` 来查看查询计划并找出规划器/优化器的瓶颈。

12. 最佳实践与策略

- 隔离环境：在独立测试环境运行，避免破坏生产数据。
- 优先处理崩溃类（Crashes），其次是死锁/挂起（Hangs），最后是逻辑错误（Logic Bugs）。
- 记录随机种子（seed）以便复现。
- 语法优先：花时间设计高质量的语法规则比无限制运行更有效率。
- 结合多种预言机以提高覆盖：NoREC 针对优化器，TLP 针对 NULL/布尔逻辑，PQS 对枢轴行不变性非常敏感。

13. 附录：常见命令与代码块

构建与运行（Java 版本）示例：

```
 git clone https://github.com/sqlancer/sqlancer
cd sqlancer
mvn package -DskipTests
java -jar sqlancer-*.jar --num-threads 4 sqlite3 --oracle NoREC
```