南開大學
Nankai University

DynSQL：用于数据库管理系统的支持复杂且有效的 SQL 查询生成的有状态模糊测试

DynSQL: Stateful Fuzzing for Database Management Systems with Complex and Valid SQL Query Generation

答辩人：许洋　　专业：计算机科学与技术　　网络安全

南開大學
Nankai University

目录
CONTENTS


南開大學
Nankai University
第一章

主要内容
Introduction

DynSQL 是一种用于数据库管理系统（DBMS）模糊测试的状态感知自动化测试框架，旨在生成**复杂且有效的** SQL 查询以发现深层次漏洞。它通过一种名为“**动态查询交互**”的机制，在每条 SQL 语句执行后实时捕获数据库状态，用于指导后续语句生成，从而确保语句间的依赖关系正确。同时，DynSQL 引入“**错误反馈机制**”过滤无效查询，进一步提升测试效率和有效性。实验证明，DynSQL 在多个主流 DBMS 上均能**显著提升代码覆盖率**，并成功发现大量之前未被检测到的漏洞。

- **研究背景**：数据库管理系统广泛应用于现代软件，安全性至关重要
- **问题挑战**：现有DBMS fuzzers难以生成复杂且有效的SQL查询，遗漏深层漏洞
- **核心方法**：提出DynSQL，首次引入**状态感知模糊测试**，结合**动态查询交互**与**错误反馈机制**
- **主要成果**：
  - 在6个主流DBMS上发现40个独立漏洞，其中**19个已获CVE编号**
  - **比现有方法提高41%代码覆盖率**
  - 所有生成SQL查询均具备高度语法与语义有效性

背景和动机
Background and Motivation

现有测试缺乏状态感知

模糊器如SQLsmith在生成SQL时未考虑数据库状态变更，导致无法构造多语句查询，难以深入测试DBMS逻辑。

语义正确性难以保证

查询不仅要语法正确，还需在当前schema下合法。缺乏上下文的语句引用往往导致语义错误，查询被DBMS拒绝。

多语句依赖复杂

复杂SQL查询往往包含嵌套子查询、CTE、JOIN等结构，语句之间依赖紧密，构造难度大，静态方式难以胜任。

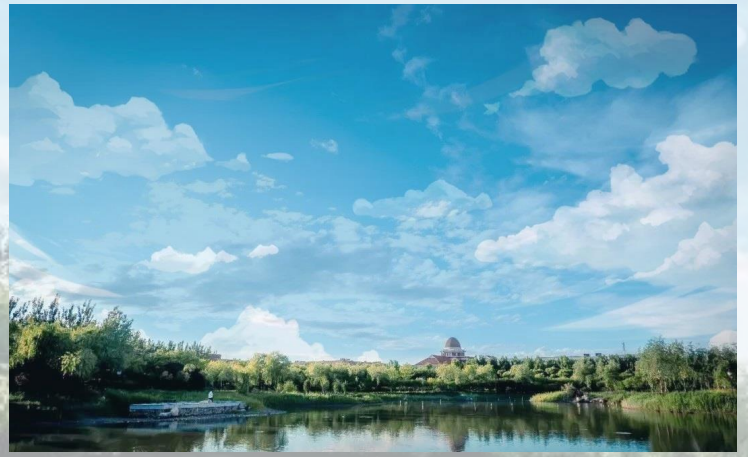
背景和动机
Background and Motivation

SQLsmith

语法丰富，但仅支持单语句，
状态无感

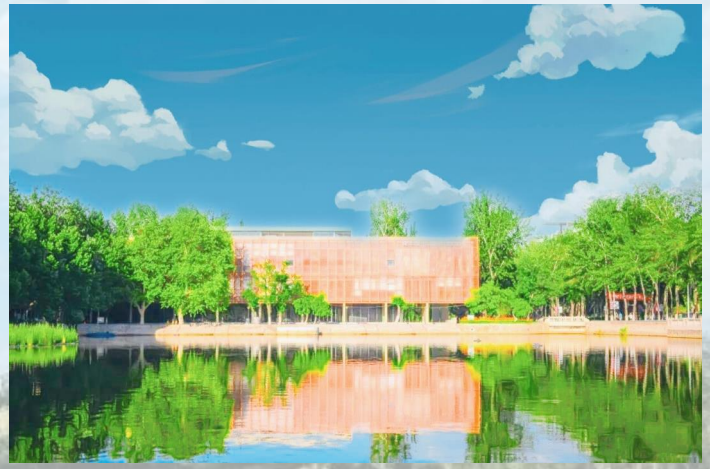
SQLancer

主要关注逻辑/性能bug，无法
生成通用复杂语句



SQUIRREL

支持多语句但推理误差大，错
误率高达50%



南開大學
Nankai University

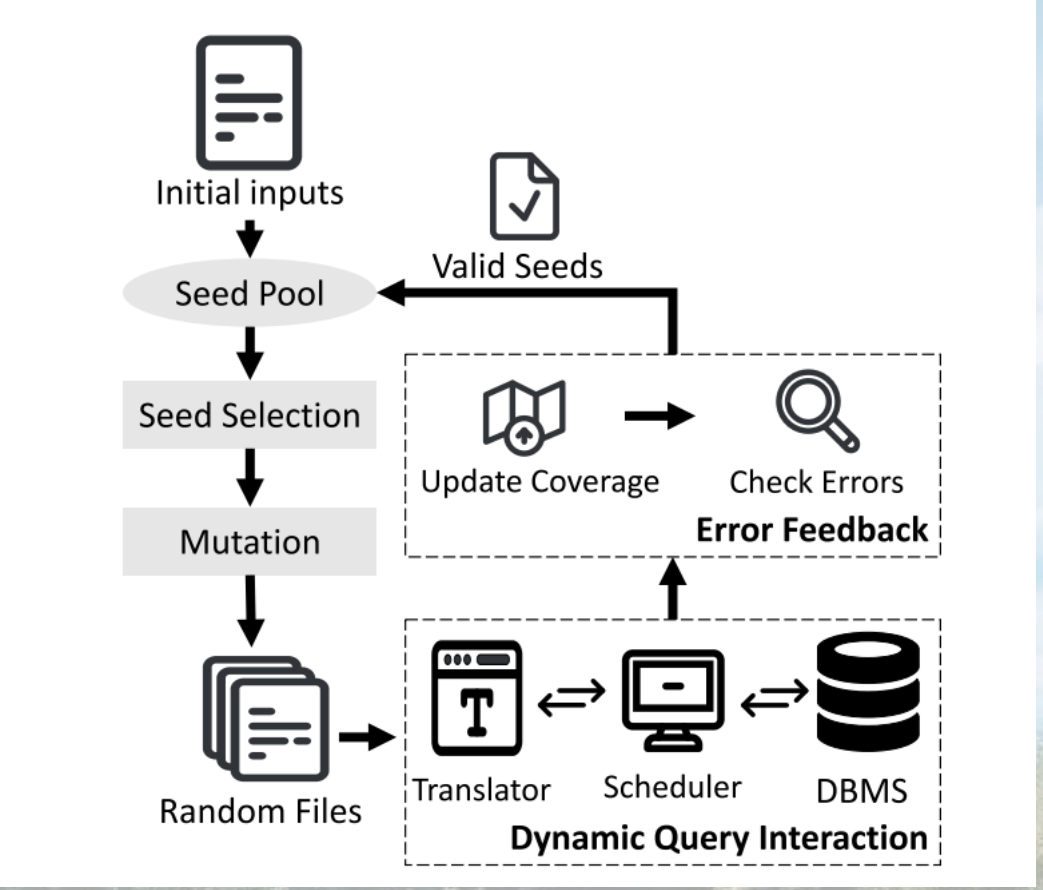
第二章

研究方法及过程:有状态的DBMS模糊测试

Research methods and processes

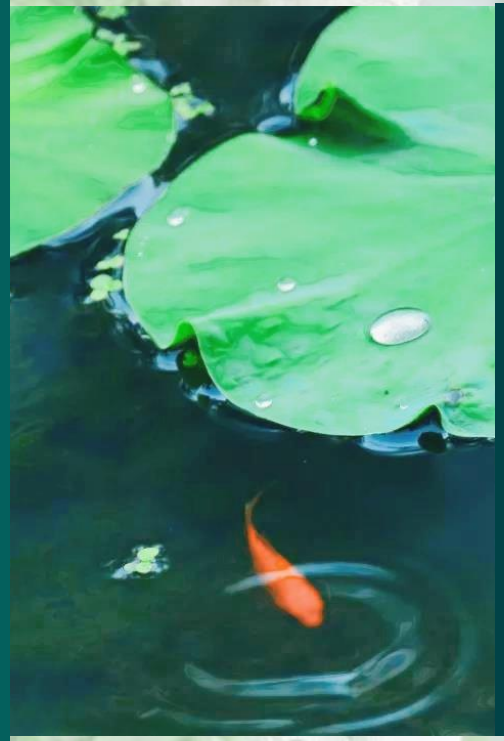
1.初始化输入文件，获取初始随机数种子池
2.从种子池中筛选种子
3.进行种子变异，获取随机文件
4.translator通过随机文件和scheduler传输回来的数据库模式生成sql语句
5.生成sql语句之后进行错误反馈，去除掉触发错误分支的种子
6.将有效种子送回给种子池，并进行种子池中的更新

overview:



研究方法及过程:有状态的DBMS模糊测试

Research methods and processes



- 动态查询交互 (Dynamic Query Interaction)
- 错误反馈 (Error Feedback)

动态查询交互

Research methods and processes

调度器 (Scheduler):

管理整个交互过程 。即与目标DBMS交互以捕获最新DBMS状态 ,将数据库 schema 传输给翻译器 。

主要组成器件

翻译器 (Translator):

根据接收到的数据库 schema 将输入文件翻译成SQL语句 。

动态查询交互

Research methods and processes

具体流程:

1.调度器向DBMS查询，获取最新数据库 Schema（如表结构等）

2.调度器将获取到的 Schema 传递给翻译器

3.翻译器根据 Schema 和输入文件（或其未读部分）生成新的SQL语句

4.生成的SQL语句返回给调度器

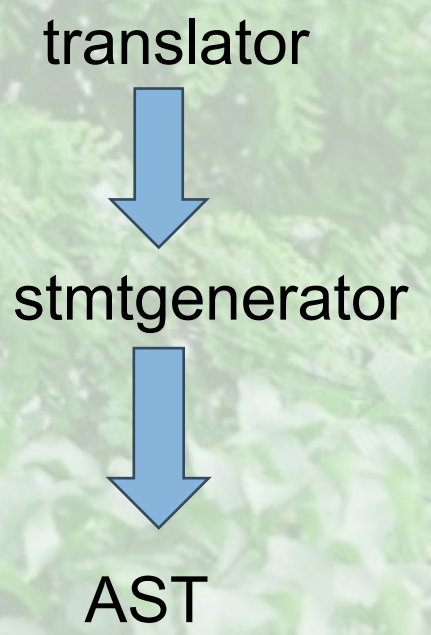
5.调度器将语句提交给DBMS执行

6.调度器收集DBMS的执行结果，包括代码覆盖率和处理状态（成功、错误、崩溃）。

动态查询交互

Research methods and processes

translator生成stmt的策略



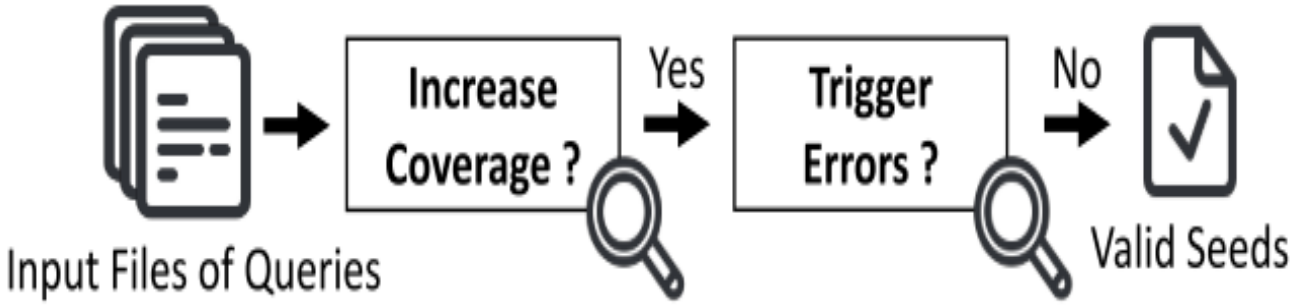
stmtgenerator将输入文件的未读取部分tmp_file直接作为其随机种子 。遍历其AST树以构建SQL语句时，它会根据从tmp_file中读取到的数值来决定选择哪些生成路径 。

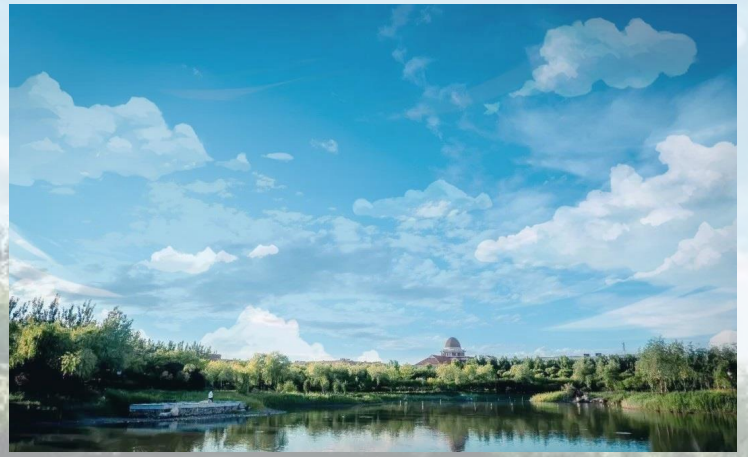
具体来说，它通过计算读取v mod n,其中v是从tmp_file里读取的一个单位,也就是当前,而n是下一个节点可选的数量,这受DBMS实时返回的schema的影响

错误反馈

Research methods and processes

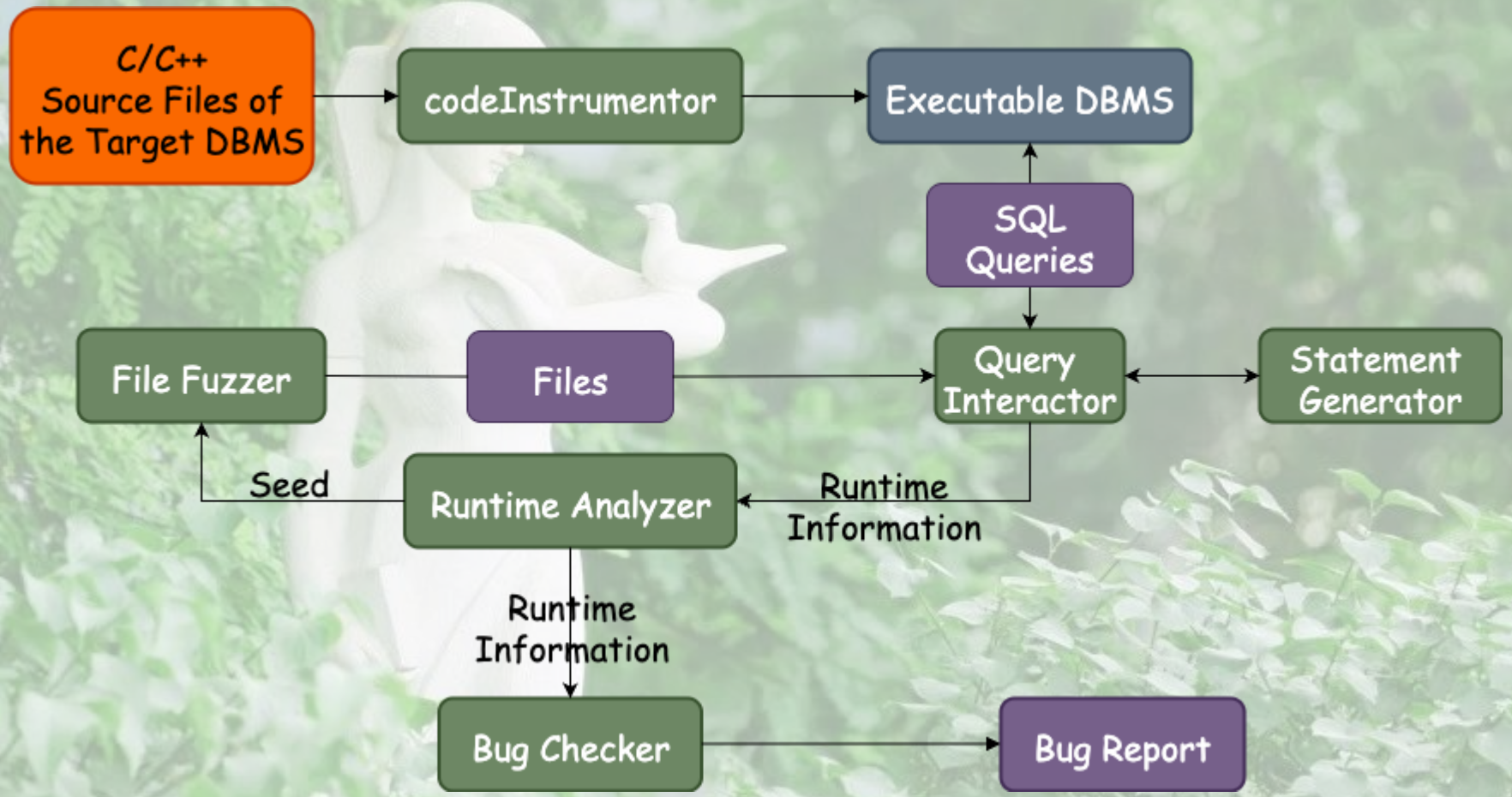
现有模糊测试器在发现导致新的语法或语义错误的无效查询时，会将其保存为种子。然而，这些种子很可能生成大量重复触发相同无效错误的查询，无法有效提升代码覆盖率。





双重筛选: 只有当查询增加了代码覆盖率且不导致任何语法或语义错误时，其对应的输入文件才会被识别并保存为有效的种子。

DynSQL整体流程框图



DynSQL框图模块介绍

框架与实施
Framework and Implementation

1.代码检测器

编译并检测目标 DBMS 的代码，
并生成接收和处理 SQL 查询的可执行程序

2.查询交互器

从文件模糊测试器接收输入文件，并执行动态查询
交互以生成复杂有效的查询。
收集目标 DBMS 所需的运行时信息，进行动态分析

3.语句生成器

使用内部 AST 模型生成语法正确的 SQL 语句，
这些语句仅引用给定数据库模式中声明的数据

4.运行时分析器

分析收集到的运行时信息，根据错误反馈识别种子，并为
下一轮模糊测试选择种子

5.文件模糊测试器

执行常规文件模糊测试，根据给定的种子生成文件。
作者主要参考 AFL 实现了该模块。

6.错误检查器

根据收集到的运行时信息检测错误并生成相应的错误报告

DynSQL实施细节

框架与实施
Framework and Implementation

DBMS 支持
01

DynSQL 需要DBMS提供的接口来启动 DBMS、连接 DBMS、停止 DBMS、发送 SQL 语句、获取语句处理结果以及查询数据库模式。DBMS 通常为这些操作提供了定义明确的接口，因此用户可以方便地在不同的 DBMS 中采用 DynSQL。

SQL 语句生成
02

作者参考 SQLsmith实现了基于 AST 的语句生成器，并额外支持一些 SQL 特性，例如 GROUP BY、UNION 等。为了解决DBMS方言的问题，作者根据 SQL 标准修复了支持的 SQL 特性中通用的部分，并将其他部分设为可选。

错误检测
04

DynSQL 使用 ASan作为其默认检查器来检测关键内存错误。DynSQL 还会分析动态查询交互中收集的异常错误，以检测导致 DBMS 报告奇怪错误消息的错误。

查询最小化
03

为了更方便地重现和定位 DBMS 错误，作者对每个触发新错误的查询执行最小化操作。最小化过程主要参考了 APOLLO和 C-Reduce。开发人员也可以利用他们的专业知识进一步最小化触发错误的查询。

南開大學
Nankai University

第三章

测试与评估
Test and Evaluation

Q1

DynSQL可以通过生成复杂而有效的查询来发现现实世界dbms中的错误吗?
Can DynSQL find bugs in real-world DBMSs by gener-ating complex and valid queries?

Q2

DynSQL发现的漏洞对安全的影响如何?
How about the security impact of the bugs found by DynSQL?

Q4

DynSQL能胜过其他最先进的DBMS模糊器吗?
How do dynamic query interaction and error feedback contribute to DynSQL in DBMS fuzzing?

Q3

动态查询交互和错误反馈如何有助于DBMS模糊化中的DynSQL?
Can DynSQL outperform other state-of-the-art DBMS fuzzers?

测试与评估
Test and Evaluation

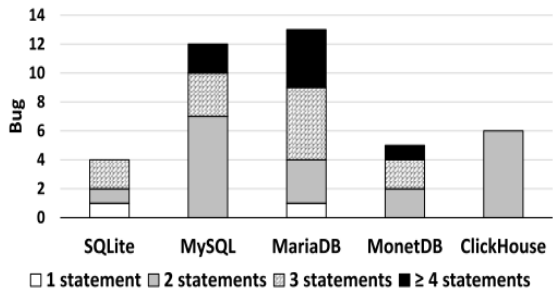
使用DynSQL对每个目标DBMS进行五次模糊处理，并计算平均值以获得可靠的结果，运行24小时。

DBMS	Bug			Validity	
	Found	Confirmed	Fixed	Statement	Query
SQLite	4	3	3	279K/286K	24K/30K
MySQL	12	12	6	91K/96K	9.4K/13K
MariaDB	13	13	6	170K/175K	17K/21K
PostgreSQL	0	0	0	154K/160K	14K/18K
MonetDB	5	5	5	70K/72K	7.0K/8.6K
ClickHouse	6	5	1	74K/77K	7.5K/10K
Total	40	38	21	838K/866K	79K/101K

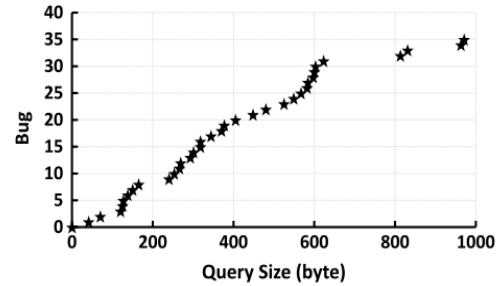
生成查询和语句

101K条SQL查询，有效率78%
866K条SQL语句，有效率97%
发现40个独特漏洞
2个漏洞可通过单条语句触发
19个漏洞需要2条语句触发
其余漏洞需要至少3条语句触发。

发现bug



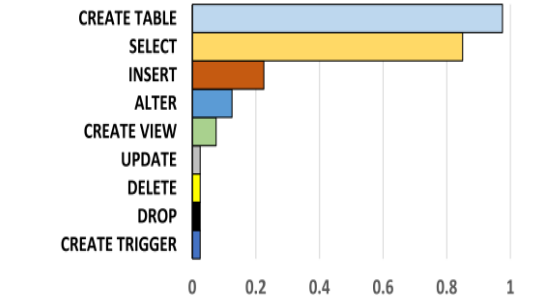
触发bug查询的语句



触发bug的查询的大小

35个漏洞的触发查询大小小于1000字节
0到600字节，触发的漏洞数量几乎呈线性增长
重点关注触发21个已修复错误的查询，应用开发者的补丁来修复相应的错误，发现所有这些查询都是有效的。
CREATE TABLE主导
SELECT关键作用
INSERT、ALTER、CREATE VIEW改变DB状态，为深层漏洞创造条件。

触发bug的查询的有效性



触发bug的查询的语句分布

测试与评估
Test and Evaluation

MariaDB的整数溢出

DynSQL使用大于300K字节的SELECT语句触发n_elems的整数溢出。 为了修复这个错误，开发人员使用size_t来定义n_elems，以增加其最大值（Linux 64从2^32-1到2^64−1）。

MariaDB的use-after-free

由于误用回滚机制导致的，在处理由DynSQL生成的特定查询时，服务器释放更改的内容，但忘记删除列表中的这些更改，这会触发回滚机制，导致列表中释放的更改被取消引用。

MonetDB的子查询结果缺失

由于捕获动态查询交互中收集的异常错误而出现。 触发错误的查询包含一个CREATE TABLE语句和一个带有CTE的SELECT语句。当该错误被触发时，MonetDB服务器报告一个错误消息，指出“子查询结果丢失”。

测试与评估
Test and Evaluation

敏感性分析：验证动态查询交互（DQI）和错误反馈（EF）对DynSQL的贡献

Table 5: Results of sensitivity analysis

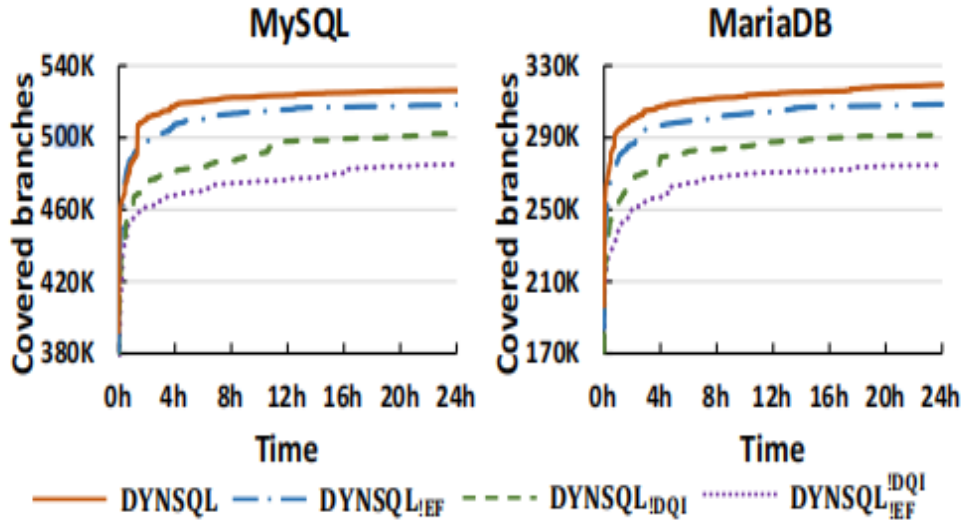
DBMS	$DynSQL_{!DQI}^{!EF}$				$DynSQL_{!DQI}$				$DynSQL_{!EF}$				DynSQL			
	Statement	Query	Coverage	Bug	Statement	Query	Coverage	Bug	Statement	Query	Coverage	Bug	Statement	Query	Coverage	Bug
SQLite	175K/307K	11K /30K	49K	1	208K/305K	16K/35K	51K	1	285K/293K	21K/29K	53K	3	279K/286K	24K/30K	54K	4
MySQL	65K/100K	2.8K/12K	485K	4	65K/96K	7.5K/13K	502K	6	89K/94K	9.2K/13K	518K	10	91K/96K	9.4K/13K	526K	12
MariaDB	123K/177K	7.9K/18K	275K	3	136K/176K	12K/22K	291K	6	165K/174K	13K/21K	309K	9	170K/175K	17K/21K	319K	13
PostgreSQL	103K/160K	6.4K/17K	125K	0	123K/162K	11K/18K	132K	0	144K/157K	13K/18K	141K	0	154K/160K	14K/18K	147K	0
MonetDB	41K/80K	3.2K/6.9K	126K	2	53K/78K	7.1K/11K	137K	2	66K/70K	4.9K/6.6K	145K	5	70K/72K	7.0K/8.6K	149K	5
ClickHouse	53K/83K	1.9K/7.8K	435K	3	55K/82K	6.3K/11K	458K	4	72K/76K	8.3K/12K	466K	6	74K/77K	7.5K/10K	476K	6
Total	560K/907K	33K/92K	1495K	13	641K/899K	61K/110K	1571K	17	821K/864K	69K/101K	1632K	33	838K/866K	79K/101K	1671K	40

DQI贡献更大：显著提升语句有效性和漏洞发现。

EF作用：过滤无效种子，提高查询有效性和漏洞发现。

测试与评估
Test and Evaluation

敏感性分析：覆盖率增长曲线与时间开销分配



DBMS	Schema Querying	Query Generation	Query Execution
SQLite	1.17%	3.16%	95.67%
MySQL	0.15%	0.44%	99.41%
MariaDB	1.29%	4.67%	94.04%
PostgreSQL	1.20%	10.04%	88.76%
MonetDB	0.13%	2.20%	97.67%
ClickHouse	0.19%	1.42%	98.39%
Average	0.69%	3.66%	95.65%

覆盖率增长曲线：

DynSQL在MySQL和MariaDB上覆盖分支更快、更多。

开销可忽略：

模式查询和生成耗时仅占3.66%，执行占95.65%。

与现有的工具进行对比

Table 7: Results of comparison

DBMS	SQLsmith			SQUIRREL			DynSQL		
	Statement	Query	Bug	Statement	Query	Bug	Statement	Query	Bug
SQLite	265K/267K	265K/267K	1	31M/45M	2.4M/9.8M	1	279K/286K	24K/30K	4
MySQL	100K/102K	100K/102K	3	506K/854K	17K/171K	3	91K/96K	9.4K/13K	12
MariaDB	148K/152K	148K/152K	3	245K/392K	425/78K	2	170K/175K	17K/21K	13
PostgreSQL	192K/197K	192K/197K	0	8M/10M	35K/560K	0	154K/160K	14K/18K	0
Total	705K/718K	705K/718K	7	40M/56M	2.5M/11M	6	695K/716K	64K/82K	29

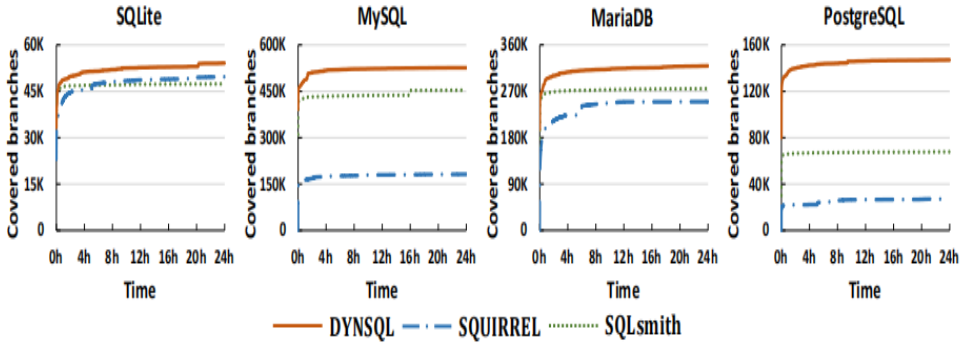
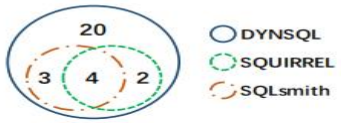
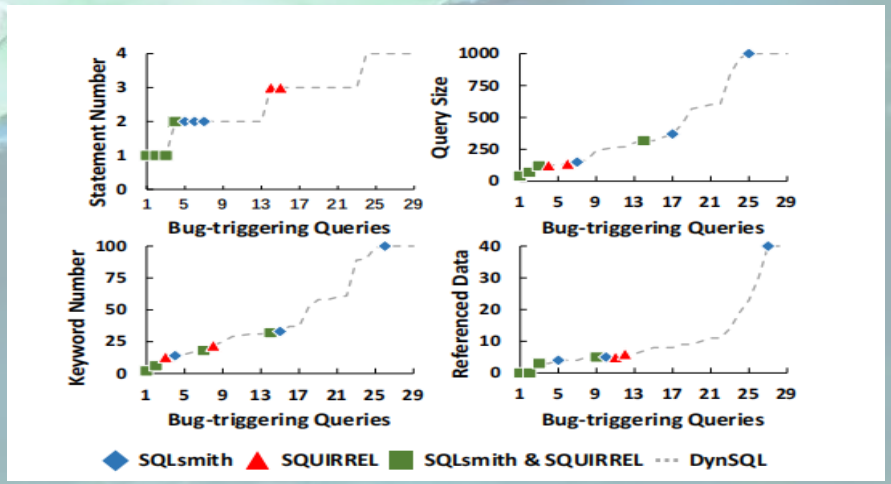


Figure 13: Code coverage of DynSQL and the other DBMS fuzzers.

DynSQL覆盖分支数比SQLsmith高41%，比SQUIRREL高166%



DynSQL发现20个独有漏洞，但是SQLsmith和SQUIRREL因无法生成复杂多语句查询而遗漏



DynSQL的漏洞触发查询：字节数更大关键词和数据依赖更多

南開大學
Nankai University

第四章

论文归纳

The induction and summary of the thesis

未来方向

引入约束求解器 & 机器学习

自动学习AST规则

AST规则依赖手工构造

需根据不同DBMS特性手动适配

性能Bug难以发现

如执行缓慢、资源消耗异常

仍有部分无效SQL语句生成

约3% statements，22% queries

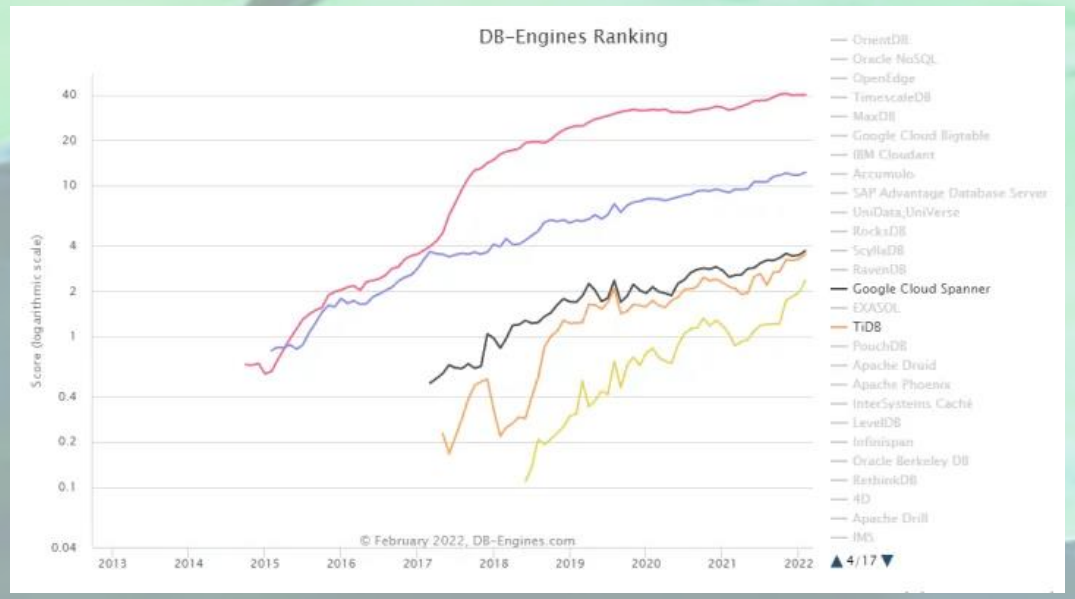
约束条件推理能力有限

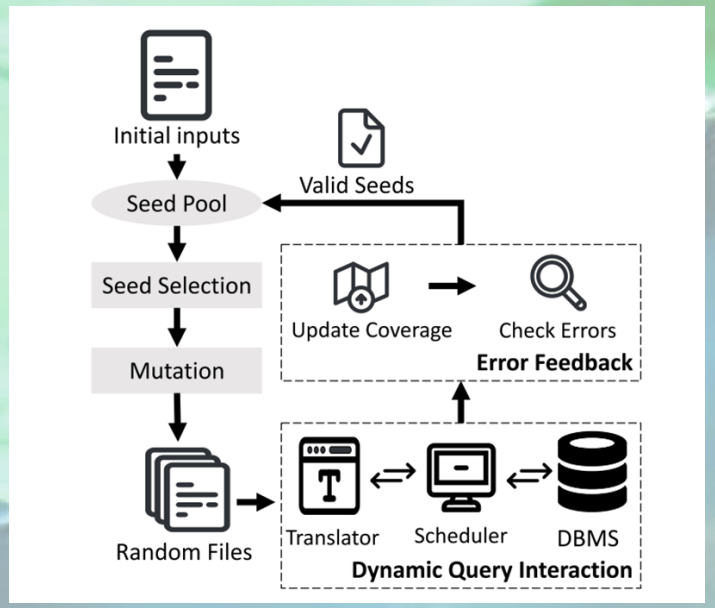
复杂表达式 & 隐式CHECK约束难以处理

语义Bug检测能力不足

无法发现不报错但逻辑错误的查询结果

创新思考





文章只进行了传统的关系型数据库的测试
而当前有很多的采用的是分布式存储的数据库
可以将这种动态交互的思想应用到分布式数据库的测试中

在种子的选择上，考虑结合强化学习的奖励策略
论文丢弃了所有引发错误和重复查询的种子
这可能导致种子中一些变种导致的查询没有呈现
采用强化学习策略，给不同种子进行权重计算，
实现逐步减少错误方向，而不是直接丢弃

总结

The induction and summary of the thesis

本文工作：

提出**DynSQL**：首个**动态交互式、状态感知**的数据库模糊测试框架

创新设计：

- **动态查询交互**（Dynamic Query Interaction）
- **错误反馈机制**（Error Feedback）

实验成效：

发现**40个漏洞**（38确认，21修复，19分配CVE）

代码覆盖率提升41%，超越SQLsmith和SQUIRREL

平均每个有效查询包含**8.6条SQL语句**

未来工作方向：

解决**约束表达式引起的语义错误**（如CHECK、UNIQUE等）

拓展支持更多类型的**语义错误和性能bug检测**

利用**机器学习自动构造AST规则**，降低人工负担

南開大學
Nankai University

谢谢观看 欢迎批评指正

答辩人：许洋 杨博涵 付家权 张耕嘉 周重天