数据库管理系统事务检测综述

崔恒[1)]

[1)](M202376855，软件学院，华中科技大学，武汉，430000)

摘 要 在现代信息系统中, 数据库管理系统（DBMS）扮演着至关重要的角色，事务测试对DBMS而言具有重要意义, 它确保并发操作的隔离性和数据的一致性, 从而防止数据损坏和安全风险的发生, 并为用户提供可靠的数据访问保障。事务测试是指通过搜索测试空间设计测试用例，探索系统的边界条件、异常情况和潜在问题，以发现可能的漏洞。DBMS的事务测试主要面临以下几个挑战：1）商用数据库内核规模庞大，源代码大多未开源，难以从底层直接检测异常；2）传统检测方法时间复杂度高，检测花费时间长。本文介绍了数据库事务测试和隔离级别的概念，按时间顺序归纳总结了现有的典型事务测试方法，包括AGENDA、ELLE、COBRA、TxCheck，并指出了这些方法解决的问题和面临的挑战，为后续研究提供了参考。

关键词 数据库管理系统；事务；隔离级别；并发异常检测

中图法分类号 TP DOI号: *投稿时不提供DOI号

Testing Transactions in Database Management System: A survey

Wei Ru-Kai[1)]

[1)](School of Software Engineering, Huazhong university of Science and Technology, Wuhan, China)

Abstract In modern information systems, database management systems (DBMS) play a vital role, and transaction testing is of great significance to DBMS. It ensures the isolation of concurrent operations and the consistency of data, thereby preventing data corruption and security risks. occurrence, and provide users with reliable data access guarantees. Transaction testing refers to designing test cases by searching the test space and exploring the boundary conditions, anomalies and potential problems of the system to discover possible vulnerabilities. DBMS transaction testing mainly faces the following challenges: 1) Commercial database kernels are large in size and most source codes are not open source, making it difficult to directly detect anomalies from the bottom layer; 2) Traditional detection methods have high time complexity and take a long time to detect. This article introduces the concepts of database transaction testing and isolation levels, summarizes the existing typical transaction testing methods in chronological order, including AGENDA, ELLE, COBRA, and TxCheck, and points out the problems solved and challenges faced by these methods, laying the foundation for subsequent Research provides a reference.

Keywords Database management system; Transaction; Isolation level; Concurrency anomaly detection;

1 引言

随着信息化时代的发展，作为操作管理大规模数据容器，数据库的工具，数据库管理系统(DBMS) 在广泛的数据密集型应用程序中发挥着关键作用。对事务的支持是DBMS的重要特性之一，通过事务用户在使用数据库时无需担心由于并发访问、系统宕机导致的数据错误或数据丢失等问题，简化了上层应用的业务逻辑，因而数据库被广泛应用于银行、企业、金融交易、电子商务等领域。

事务作为数据库最基本的执行单元，在执行过程中，DBMS需保证事务的一些特性（即原子性、一致性、隔离性和持久性，ACID）以确保其执行的正确性。例如，若一些事务是在串行隔离级别（Serializability isolation level，SER）并发执行，那么某一事务执行过程中，其他未提交的事务的操作是不可兼得，且这些事务的执行结果需要与一一执行的结果相同。然而，事务的实现通常涉及复杂的逻辑（如两阶段锁和多版本并发控制），因此很容易引发异常。多数情况下，这些异常不会引发DBMS运行错误，因而难以发现。这些异常会导致客户端应用程序崩溃，给DBMS的正常使用带来极大的挑战。

DBMS定义了隔离级别，即事务与其他事务进行资源或者数据更新的隔离程度, 避免由数据访问冲突造成的数据库状态的不一致和不正确。SER为数据库隔离级别的金标准，但由于数据规模的指数级增加，SER无法满足数据库吞吐量的需求，数据库开发人员不得不放宽事务的隔离需求以满足数据库性能需求。数据库系统的隔离级别实现涉及复杂的数据结构、算法和并发控制机制。不同的隔离级别(如读未提交、读已提交、可重复读和串行化) 需要在数据库内部实现锁定、版本控制和事务管理策略等功能，因此容易出现错误。

为提高DBMS中事务支持的可靠性和正确性，保障用户的使用体验，近年来研究者提出了一些方法测试DBMS的事务支持，以期及时发现并修复错误。然而，由于数据库内核代码规模巨大，且大多数数据库产品不可开源，这些测试方法大多采用黑盒测试的方法，构建测试用例对DBMS进行测试。

本篇文章综述了自2000年以来的代表性事务异常测试方法，包括AGENDA[2]、ELLE[3]、COBRA[4]、和TxCheck[5]。本文剩余部分组织如下：第二章扼要介绍了数据库管理系统事务异常检测的相关概念，包括DBMS中的事务和隔离级别；第三章总结并对比了各方法的基本思想；第四章是结论与展望部分，总结全文内容并介绍了可能的改进方向。

2 背景

2.1 DBMS事务测试

事务 数据库事务是一个对数据库操作的序列，DBMS 必须保证这些操作的原子性、一致性、隔离性和持久性（即ACID）。本文重点关注关系数据库管理系统(RDBMS)，其中事务通常由一组SQL（即结构化查询语言）语句组成。每个语句执行读操作（例如，SELECT语句）或写操作（例如，INSERT、DELETE和UPDATE语句）。SQL 语句通常涉及谓词（例如，WHERE子句）来选择满足要求的数据库存储对象。

事务的**ACID**特性：

(1) 原子性（Atomicity）：事物是一个不可分割的逻辑工作单元。

(2) 一致性（Consistency）：事务必须使数据库从一个一致性状态变换到另外一个一致性状态

(3) 隔离性（Isolation）：事务未提交前其操作的数据对其他事务不可见

(4) 持久性（Durability）：事务成功提交对数据库的所有更新是永久的

事务测试 通过搜索测试空间设计测试用例，探索系统的边界条件、异常情况和潜在问题，以发现可能的漏洞。

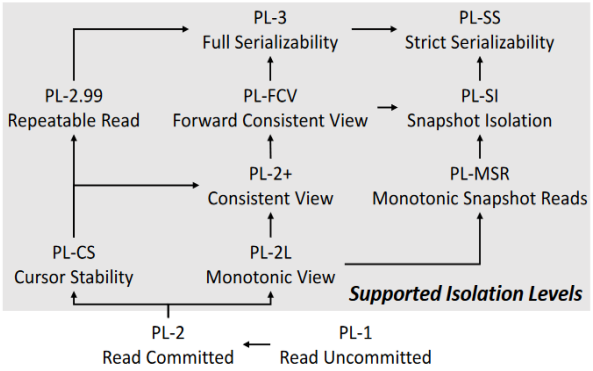

图1　Adya形式化的隔离级别，依照箭头方向隔离级别上升，事务并行程度下降

2.2　ADYA隔离级别

在2000 年 的 论 文“广 义 隔 离 级 别 定 义[1]”中，Adya、Liskov 和O’Neil 根 据 数 据 库 历 史H的禁止行为形式化了各种事务隔离级别，图1展示了Adya等人定义的各层隔离级别。数据库历史由一组事务、编码这些事务中的操作顺序以及版本顺序（每个对象的已修改版本顺序集合）组成。不考虑谓词（predicate）的情况下，Adya 等人定义的依赖关系如下：

1）直接写依赖（Directly write-depends）T_i写入x_i，T_j修改x_i至下一个版本

2）直接读依赖（Directly read-depends）T_i写入x_i，T_j读取x_i

3）直接反依赖（Directly anti-depends）T_i读取x_i，T_j修改x_i至下一个版本

通过这些依赖关系，Adya等人将数据库历史抽象为直接序列化图（Direct Serialization Graph，DSG）。DSG是某个历史H中事务为顶点组成的图，图的边是这些事务间的依赖关系。Adya等人依据DSG定义了如下异常：异常$G1a$（中止读取）和$G1b$（中间读取）分别被定义为读取由中止事务写入的版本或从某个其他事务中间读取版本的事务。剩余异常是根据直接序列化图（Direct Serialization Graph，DSG）上的环来定义的。$G0$异常是DSG中由写入依赖性组成的环。$G1c$异常包括读取依赖性。$G2$异常至少涉及一种反依赖性。

DSG被广泛应用于后续的事务异常检测工具中，这是因为：1）DSG定义的异常相对简单，仅涉及一组特定的事务，因此易于调试并发现为题。2）构建DSG后，检测异常的时间复杂度为O(n)。通过DSG，研究者们可以形式化不同隔离级别下DBMS的预期行为，例如在SER（PL-3）级别下DSG是一个无环图，在可重复读（PL-2.99）级别下DSG不包含任何消除某些谓词依赖性的有向环。Bailis等人进一步扩展了DSG使其能够定义其他隔离级别下DBMS的行为。

3　研究进展

事务检测包括验证数据库应用程序中事务的正确使用[1]和DBMS事务支持的正确实现[2,3,4]。数据库应用程序从数据库中选择并处理一些数据，然后更新数据库。为了实现一定的并发性，应用程序通常由一组事务组成，这些事务同时执行可能相互干扰，造成并发问题。

3.1　AGENDA

Deng等人提出了一种基于检测预测器算法的白盒检测方法AGENDA[2]以检测数据库应用程序中事务设计和执行是否正确。AGENDA 将运行应用程序的数据库的数据库架构、应用程序源代码和“示例值文件”作为输入，其中包含一些数据库对象值。测试人员以交互方式选择测试启发式方法，并提供有关测试用例预期行为的信息。使用此信息，AGENDA 填充数据库，生成数据库应用程序的输入，并在这些输入上执行应用程序，最后检查生成的数据库状态和应用程序输出的某些方面的正确性。

AGENDA包含五个模块，其架构如图2所示。第一个模块Agenda Paser用于提取数据库架构、应用程序源代码和测试者提供的示例值文件中的信息，并将信息传递至其他四个模块；第二个模块State Generator使用数据库架构、示例值文件中满

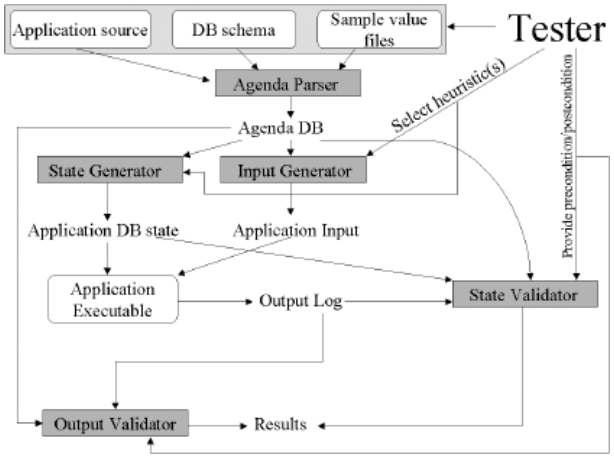

图2 AGENDA方法整体框架

足非空的、唯一的、满足测试人员预定义的数据库表对象间的约束填充数据库；第三个模块Input Generator生成输入应用程序的数据；第四个模块State Validator、第五个模块Output Valiator则分别使用单一的UPDATE和SELECT查询语句修改数据库数据或读取应用程序结果，根据测试人员提供的条件判断是否存在异常。

AGENDA相比于DBUnit等白盒测试工具能自动生成大量测试用例，自动化生成不同的数据库状态等优势，但仍需要测试人员提供应用程序涉及数据库表间的关系等信息对测试用例进行约束，且仅能在SER隔离层级下进行检测。

3.2 ELLE

为克服传统方法需要人工约束测试用例、无法检测弱等级隔离级别（如读未提交）异常的问题，Kyle等人提出了一种基于Adya事务依赖图的黑盒检测方法ELLE[3]以检测DBMS中事务的隔离级别异常。ELLE无需求解事务顺序，而是利用客户端发放的事务、写入的数据库对象以及读取返回的值，以Adya 形式来推理未知数据库的可能事务依赖关系图。

此前的研究发现，Adya定义的依赖关系（见章节2.2）表明在任何历史H中T_j必须在T_i之后，由此可见任何包含环的DSG都不可能是有效的串行历史记录。此外，数据库本身不包含对象版本顺序的定义，或者商业数据库未向客户端开放获取该排序的信息，因此此前并未有方法使用DSG进行事务检测。

Kyle等人证明了通过利用特定的数据类型，我们可以推断出与给定数据库外部观察相兼容的每一个内部历史的属性，这些数据类型允许我们追踪产生的特定版本的版本序列，并能够恢复特定写操作所产生的数据库对象的版本。具体实现细节如下：

Kyle首先建立了可追溯性（traceability）和可恢复性（recoverability）的概念，证明了若满足这些概念，则可以直接从数据库外部观察推理得到内部Adya历史。同时证明了观察中发现的任何异常都必然存在于与该观察一致的Adya历史中。

然后，Kyle定义了对象（Object）为包括一组版本、一个初始版本和一组对象操作的可变数据类型。以对象x的各个版本为顶点、针对x的写操作为边构建有向图v_x，显然，版本顺序与v_x相同，且每个数据库对象观察的版本顺序与Adya历史H的顺序一致。同时重新定义了DSG中的依赖关系，包括直接写-写、读-写和写-读依赖。

若多个写操作同时操作非“可附加（append）”型的数据类型（如浮点型数据）时，之前的操作的内容会被覆盖，因此Kyle将数据库对象的数据类型限定为列表等可附加类型。通过读操作读取到的版本信息即可得到直接序列图DSG。最后Kyle分析DSG中是否包含环来判断DBMS是否存在事务异常。

ELLE首次使用外部观察恢复内部Adya历史，建立DSG以检测DBMS的事务异常，可将异常检测时间复杂度下降至线性，能够快速分析数十万事务的真实历史。此外，ELLE能够向用户提供简洁的，可解释的异常。实验表明在四个数据库中ELLE发现了多个未被发现的异常，证明了ELLE的通用性.

3.3 COBRA

为检查真实数据库是否满足其生命的SER特性，Tan等人提出COBRA[4]使用合适的SMT求解器检

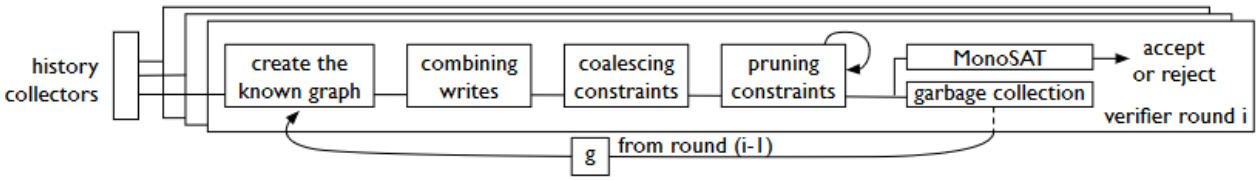

图3 COBRA流程。COBRA迭代的从历史记录中创建依赖图，接着组合写操作并合并约束以对搜索空间进行剪枝，最后使用MonoSMT求解，判断是否存在异常。

测真实DBMS是否支持事务的SER金标准隔离级别。分布式云数据库系统试图为用户提供SER隔离级别同时满足高性能，但由于DBMS需要保证事务原子提交协议交互的共识协议，极容易出现错误。一些研究已经表明现代数据库系统出现违反SER规定的错误。

COBRA包含以下几个组件：一个非假设为”合作“的未经修改的第三方数据库、COBRA修改以连接到数据库的一组客户端、记录对数据库的实际请求和来自数据库的响应的一个或多个历史记录收集器以及一个全面检查可序列化性的验证器。当客户端、收集器和验证器都在同一个信任域中（例如由同一组织部署）时，数据库是不可信的。

COBRA主要解决了两个问题：

（1）有效的异常证据搜索：验证可串行性的一种有效方法是证明事务依赖图G的存在，其节点是历史中的事务，边为事务间的依赖关系，且图需要无环。COBRA利用最先进的SMT求解器MonoSAT，解决了利用SMT编码无环性带来的开销问题。然而单独使用MonoSAT解决搜索问题代价过高，针对该问题，COBRA开发了特定领域剪枝技术，减小了搜索空间的大小，具体包括利用实际工作负载中常见模式（读取-修改-写入事务）有效推断历史记录中事务排列顺序和利用并行GPU测试图上候选边是否会与现有路径生成环。

（2）扩展到连续且不断增长的历史：在线云数据库以连续的方式运行，相应的历史记录不间断且无限增长。为了支持在线数据库，cobra进行轮次验证。验证器逐轮检查历史记录的一部分的可序列化性。然而由于验证器需要验证所有历史记录，但是SER不尊重实时排序，因此未来的事务可以从（在实时视角中）已被覆盖的值中读取。为解决该问题，客户端定期围栏事务（fence transactions），这些epoch添加粗粒度同步，即创建一个允许未来的读取的窗口，允许验证器在窗口前丢弃交易。

Tan等人在具有各种工作负载的现实数据库上进行了事务测试实验。COBRA检测从真实系统的错误报告中收集的所有违反SER要求的违规行为。COBRA在14秒内能检查10k个事务，较SOTA提升了10倍。对于具有连续流量的在线数据库，COBRA在试验的工作负载上实现了2k txn/s的可持续验证吞吐量

COBRA仍存在一些问题。首先，无法保证COBRA在合理的时间内终止。其次，COBRA仅支持键值API，因此不处理范围查询操作，例如“JOIN”和“SUM”。第三，COBRA尚不支持客户端中的异步（事件驱动）I/O 模式。第四，COBRA主要强调验证者和收集者的容错性（尽管存在模块化解决方案）。

3.4 TxCheck

为进一步检测DBMS是否支持复杂操作和包含谓词的事务执行，Jiang等人提出基于语句级依赖图的黑盒检测方法TxCheck[5]。此前方法如ELLE需要限定特定的数据类型，通过简单的append操作生成测试用例，同时无法在操作中编码谓词操作，导致错过了许多错误；此外，先前方法聚焦于测试DBMS的隔离级别保证，遗漏了其他的错误。

不同于此前方法基于数据库历史构造DSG，判断其中是否有环来检测是否存在异常，TxCheck基

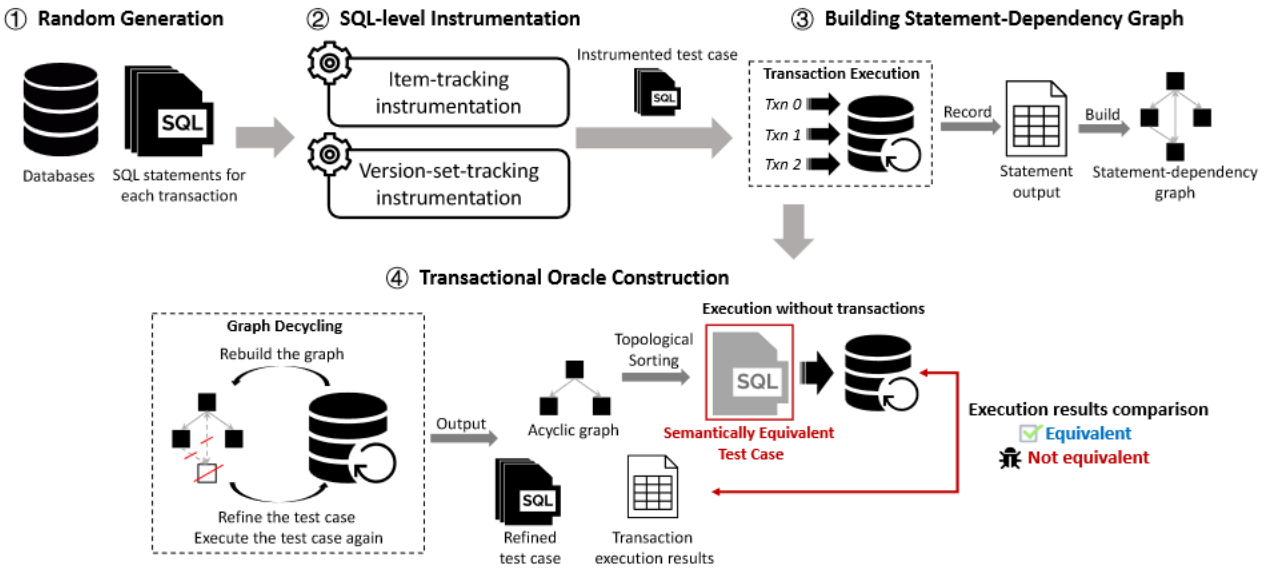

图4 TxCheck方法流程。首先随机生成测试用例；接着使用SQL级别工具提取SQL依赖关系，生成原始语句依赖图；然后迭代地重构SDG和原始测试用例，直至最终SDG中不包含环；而后通过拓扑排序生成语义等价测试用例；最后分别执行最终测试用例和等价测试用例，判断他们地语句输出内容和数据库存放内容是否相同。

于测试-语言机的范式通过提取事务中SQL语句间的依赖关系，生成语义上等价的、未被包装成事务的测试用例，判断与原始事务的执行结果是否有区别。TxCheck的整体流程如图四所示

TxCheck首先定义了七种SQL语句间的依赖关系，描述了事务中不同SQL语句间的联系，包括读取或写入相同对象的依赖、谓词操作的依赖和直接STMT-vlue-write依赖三种。

接着，TxCheck制定了一些约束，即：向数据库的对象添加PrimaryKey和VersionKey两个属性，在执行执行写操作（如UPDATE 和INSERT）的语句应将所处理的项的VersionKey 更改为新值，执行读操作（如SELECT）的语句需要输出项目的PrimaryKey和VersionKey。

TxCheck通过在写入操作语句前后插入Read语句（BWR、AWR）和在每条语句后插入读取语句（VSR），判断他们的上输入有没有重叠部分，再根据依赖关系一一比较来得到语句间的依赖关系，并以依赖关系为边、语句为节点形成语句依赖图SDG。由于VSR可能会输出目标语句未引用的项目，若这些项目与其他语句的输出重叠，则会捕捉虚假的依赖项，作者随后证明了得到的为实际SDG的超图，同时，虚假依赖关系并不影响故障判断的正确性。

得到SDG之后，TxCheck首先迭代地通过广度优先搜索判断图中是否有环，若有则移除节点对应的SQL语句及其附带的的BWR、AWR、VSR，而后重新在数据库执行；若没有环则输出测试用例和对应的SDG。随后TxCheck在最终的SDG上执行拓扑排序，生成等价的测试用例。最后在数据库上分别执行原始测试用例和等价测试用例，查看执行后的语句输出和数据库内容是否分别相同，若不相同则证明有异常出现。

Jiang等人在SQLsmith的架构上编写了TxCheck，依照SQLsmith的方案生成随机测试用例，测试用例中包含多个事务；针对每个事务，TxCheck都会设置一个客户端会话进行执行以避免并发执行引入的不确定性。此外，为解决在真实DBMS可能会在不同的隔离层级上阻止语句的执行，导致SQL-level instrumentation无法正常执行的问题，TxCheck采用阻塞调度机制，保证插入语句和目标语句始终按照原始语句执行顺序执行，否则便删除原始SQL语句。

TxCheck在MySQL、MariaDB和TiDB三个广泛使

用的真实数据库上进行测试，发现了56个新的异常。与之对比的是，在19个异常测试用例中，ELLE仅能发现一个简单的、不包含谓词操作的异常。

4 总结与展望

4.1 总结

数据库管理系统的事务支持已成为用户选择数据库产品的重要考虑因素，因此事务检测的研究越发注重于真实数据库管理系统的检测而不是人工设计异常事务数据集的检测。同时，由于数据库需要执行的事务越发复杂，规模日益庞大，能自动生成复杂、规模庞大的测试用例并能快速、准确测试数据库管理系统事务支持的方法已成为研究者关注的热点问题。由于数据库内核规模庞大且代码通常未开源，黑盒测试已成为研究者的首选测试方法。图方法在检测异常时表现出线性时间复杂度，因此基于DSG的方法被广泛应用在DBMS隔离级别测试中，但这些方法仍然需要对数据类型做出一些限制，且无法检测谓词异常。最近，研究者提出以测试-语言机架构为基础，检测语义等价的非事务的测试用例是否与原始执行结果相同来检验数据库管理系统的事务支持，在真实数据库中取得了较好的效果。本文按时间顺序归纳并总结了经典数据库管理系统事务检测方法，指出了各种方法的主要优点和可能的改进方向，为后续的研究提供了参考。

4.2 展望

（1）现有的测试用例是随机生成的，可能会错过一些错误。模糊测试是一种广泛应用于软件和系统测试的方法, 通过搜索测试空间并生成多样化的测试案例, 探索系统的边界条件、异常情况和潜在问题, 以发现可能的漏洞。但传统的模糊测试技术无法直接应用于DBMS事务测试。

（2）恢复涉及谓词的历史时间复杂度较高带来的整体性能的下降。TxCheck[5]方法检查所有引用的数据库表中的条目，但对于大规模数据库检测效率极低。可针对事务中使用的具体语句，参考相应的SQL语法对涉及谓词语句的引用做出限制。

参 考 文 献

[1] ADYA A, LISKOV B, O'NEIL P. Generalized isolation level definitions[C]//Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073). IEEE, 2000: 67-78.

[2] DENG Y, FRANKL P, CHAYS D. Testing database transactions with agenda[C]//Proceedings of the 27th international conference on Software engineering. 2005: 78-87.

[3] KINGSBURY K, ALVARO P. Elle: Inferring isolation anomalies from experimental observations[A]. 2020.

[4] TAN C, ZHAO C, MU S, et al. Cobra: Making transactional Key-Value stores verifiably serializable[C/OL]//14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, 2020: 63-80. https://www.usenix.org/conference/osdi20/presentation/tan.

[5] JIANG Z M, LIU S, RIGGER M, et al. Detecting transactional bugs in database engines via graph-based oracle construction[J]. Transactions, 2023, 1: 20.