华中科技大学

硕士学位论文

入侵容忍的数据库中数据隔离技术研究

姓名：华学勤

申请学位级别：硕士

专业：计算机软件与理论

指导教师：王元珍

20090529

摘要

数据库是信息系统的核心，集中存放着大量重要而又敏感的数据，是最吸引攻击者的目标，一旦数据被黑客窃取或者是破坏，其损失难以估量。传统的数据隔离技术不能有效地防护黑客的攻击，需要研究合适的数据隔离技术，使得数据库在遭受攻击时既能正常提供服务，又能有效地抵御攻击，保证数据库中数据的正确性。

通过对当前多种隔离技术的分析和比较，在通用数据库平台的隔离方案的基础上，提出了数据库中数据隔离系统的体系结构。该系统由四部分组成：隔离控制模块、代理模块、事务日志模块、版本融合模块。隔离控制模块可为可疑用户建立隔离版本的数据库，包括隔离表构造的时机以及流程、隔离版本的维护以及安全性、隔离版本的控制策略的设计与实现。代理模块是将可疑用户的访问语句定位到不同的数据版本中操作，给出了四种基本语句的代理实现形式。事务日志模块构造了存储事务日志表并给出了安全措施，以便获取数据库中所有事务的操作日志，对版本融合提供数据支持，采用触发器来获得事务写操作的日志，采用写临时表的方式来获取所有事务读操作的日志。版本融合模块是当可疑用户最终被确认是正常用户后，将其维护的隔离版本融合到真实版本的数据库中，在分析了数据不一致的判断方法和解决办法的基础上，设计并实现了单用户隔离版本的融合和多用户隔离版本的融合方案。

测试结果表明，该隔离系统能够正常隔离用户并能保证版本融合后数据的正确性，且对整体系统性能损伤较小，达到了系统设计目标。

关键词： 数据隔离，控制策略，操作日志，数据一致性

Abstract

The database is a core of information system, which stores a large number of important and sensitive data. It has been suffered more and more attacks by hackers, its lost is hard to determine. Traditional data isolation mechanism can not resist the hacker's attack effectively, it is necessary to do research on data isolation technology. The new isolation technology not only enable the database to provide service normally when attacked, but also can resist attacks effectively, guarantee the exactness of the data in the database.

After analyzing and comparing existing isolation technology, based on isolation scheme for the common database platform, proposing system Architecture of data isolation forwards the database. The system is composed by four parts: isolation and control module, agent module, transaction log module, merge module. Isolation and control module is to set up isolated version of database when suspicious users visiting the database. Which including construct time and process of the isolated table, maintenance and security of the isolated version, design and implement control strategy of the isolated version. Agent module is locating SQL statement which suspicious users visit to different data versions to operate, providing implement of four basic type of SQL statement. Transaction log module creates transaction log table and presents safety measures, which to capture all transactions operation history in the database, in order to support data to the merge module. Which including construct system table for storing transaction log, analysis of the security and provide safety measures, use triggers to capture transaction write operations and use writing temporary table to capture transaction read operations. Merge module is to merging isolated version which maintains by suspicious user into normal version when finally confirm the user is normal. First describe data inconsistent judgment method and provide solutions, then provide merging isolated version which maintains by single user into normal version and provide merging isolated versions which maintain by multiple users into normal version.

The test results indicate that isolation system can correctly isolate user and can guarantee the exactness of the data after the suspicious versions merged. It's also almost no decreases the efficiency of the whole system, reach the systematic design object.

Key words: data isolation, control strategy, operation log, data consistent

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□ ，在_____年解密后适用本授权书。

本论文属于

不保密☑。

（请在以上方框内打“ √ ”）

学位论文作者签名：　　指导教师签名：

日期：　年　月　日　　日期：　年　月

1 绪论

1.1 课题背景

随着信息技术的发展以及网络的快速普及，信息系统在社会生活中起着越来越重要的作用。网络系统、操作系统和数据库管理系统(DBMS)是信息系统的主要支撑平台，这三者的安全性直接影响到整个信息系统的安全。在这三者之间，数据库往往是最吸引攻击者的目标[1,2]。这是因为数据库作为信息系统的核心，集中存放了大量的数据。如电子商务系统中保存了大量的商业伙伴和客户的机密信息，由于这些数据的重要性和敏感性，一旦数据被黑客窃取或者是破坏，其损失难以估量。另一方面，网络数据库和分布式数据库的使用也使得数据库的安全显得更加脆弱，数据库安全问题日益突出[2]。

当数据库受到攻击时，传统的数据库安全机制重点在于预防，着眼于对外部用户的身份和权限约束的检查，来保证用户操作的合法性。如包括用户身份验证和识别的用户认证技术、存取控制技术和数据加密技术等。然而，以身份认证和存取控制为主的数据库安全机制对数据的保护能力非常有限，攻击者往往能窃取到合法的身份或权限来执行恶意事务。

基于入侵容忍技术的数据库系统[3,4]是目前的研究热点，该技术能够容忍数据库的损伤并能将数据库恢复到一个正确的状态。该技术主要是通过对用户操纵数据库的行为进行监测和分析，最终确认某一用户是正常用户还是恶意用户。若是正常用户，则不需要做出反应；若是恶意用户，则需要将其对数据库进行的操作进行回滚（undo）操作[5,6]，使数据库恢复到一个正确的状态。但在这个恢复过程中，受损数据会扩散。加上入侵检测的检测期需要一个比较长的时延，恶意事务会在这期间影响到正常事务[7,8]。在恢复期间必须将这些受影响的事务全部做重做（redo）操作，使得数据库性能消耗很大。

依据前面的分析，在此提出数据隔离的思想：在入侵检测的检测期，当发现用户的操作可疑的时候，将该用户的操作放在一个独立的数据库中进行，当最终入侵检测确认该用户是正常的时候，再将该隔离数据库中的数据“融合”到真实数据库

中；如果最终入侵检测确认该用户是恶意用户，只需要将该隔离数据库中的数据丢弃，并且将该用户被判定为可疑用户之前对数据库进行的操作进行回滚操作，使之不会影响到真实数据库中数据的正确性，保证了数据库安全。

本课题来源于华中科技大学数据库与多媒体技术研究所承担的部委预研项目“高安全等级数据库管理系统及其测评关键技术研究”课题。本课题的目的是要实现一个通用数据库的隔离系统，在数据库受到攻击的情况下如何最大限度的控制受损数据的扩散，以及最终如何将数据库恢复到一个正确的状态，从而保证数据库的安全。

1.2 国内外概况

传统的数据库隔离级别[9]只是为了保证事务执行时候数据的一致性，属于一种预防性的举措。当数据库受到攻击时，恶意用户只要能够保证事务中 SQL 语句满足语法要求，还是能对数据库中的数据造成破坏。因此，这种隔离对于数据库遭受攻击的时候毫无用处。

1.2.1 数据库入侵限制中的隔离技术

由于现有的数据库入侵限制技术采用的都是恢复的技术来使数据库回到一个正确的状态。即在用户被确定为恶意用户之前，对该用户的所有操作不做任何的限制。当该用户最终被确认为是恶意用户时，再去找出所有的受感染事务进行回滚操作[10,11]。此时会发现需要回滚大量的事务，对于数据库的性能有很大的影响。

数据库入侵限制的目的是在数据库受到入侵的情况下阻止损坏传播以及入侵者进一步造成破坏。防止损坏传播主要有两种途径[12]，一种是对损坏数据的限制，在损坏数据得到清除或恢复前阻止其它事务存取；一种是对怀疑用户的限制，怀疑用户是一些偏离正常轨道，但在不进一步观察的情况下无法确认是否为正常或恶意用户，限制技术既可以防止过早地停止这些用户对数据库的操纵又可以防止这些用户对数据库的进一步破坏[13-15]。

数据隔离的思想即来源于第二种途径。在隔离的同时，数据库系统本身还要向其他正常用户继续提供服务。也就是说，隔离对于用户而言是透明的。当前对于数

据隔离的研究大多都停留在理论阶段，下面按照时间的顺序，依次介绍其算法思想。

1、静态分区法

静态分区是最早提出来的，其思想就是把数据库中的数据划分为不同大小的区间，当事务操纵的数据在某一个区间的候，其事务的全部操作都限制在该区间内，不允许出现跨界操作。若该事务是恶意事务，则该事务造成的损伤就限定在该区间之内，不会对其他区间的数据造成影响。只有当前区间受到损伤的时候才被允许使用其他的区间。但是在某些数据库当中并不能精确地划分区间，此时可以采用定义区间边界的方法，记录通过边界的传递更新，限制数据通过边界的条件[12]。

2、数据标记法

数据标记是将数据库中的数据打上不同的标签，来区别是正常数据还是损坏数据，从而限制不同的事务对不同标记的数据进行读写处理来达到隔离的目的。Ammann P 等人在文献[16]中从信息系统的隔离策略中得出在数据库中的隔离方法，其思想是使用颜色标签来标记数据受损害的程度，他将数据用四种颜色[16]来标记：红色（Red）、浅红（Off-Red）、浅绿（Off-Green）、绿色（Green）。四种颜色依次向后表示数据的可信赖度越来越高，Red 表示最不可信赖数据，即受损不可使用数据；Green 表示最可信数据，即正常数据。Ammann P 还给出了数据库一致性和事务一致性的定义。在该定义下，给出了三种事务[16]（攻击事务（Attack Transactions）、正常事务（Normal Transactions）、干预事务（Countermeasure Transactions））对上面四种颜色的数据进行处理的详细算法，给出了查找受损数据的范围和改写后的事务处理协议[17,18]以及快照恢复算法使数据库恢复到正确状态。

3、多阶段隔离法

在数据库入侵检测的过程中，确定一个用户的行为是否是异常的需要一个相当长的检测期，同时采用如数据挖掘、统计分析等方法来对事务行为的判断也需要一个比较长的时间。通常情况下，当最终确认该用户是恶意的时候，其最初对数据库操作所产生的受损数据已经将损坏传播开来而导致原来隔离的范围无效。比如在 t_1 时刻用户开始对数据库进行操作，t_2 时刻确定该用户是恶意用户。若在 t_1 到 t_2 时刻之间该用户执行一个恶意事务，对数据 x 进行写操作，同时又有另一正常事务对 x 进行了读操作，又对 y 进行了写操作 $y=x+1$。则当确定隔离 x 时，损坏已经传播到 y 使

得隔离无效。

多阶段隔离的策略的思想是当确定一个用户是恶意的时候，先去查找操作的日志记录，找出所有被事务写操作过的数据，即受损数据。再采用事务间读写依赖的关系来得到所有读取过这些受损数据同时又对其他数据进行写操作的所有受感染数据。将这些数据全部都隔离，确保受损数据不传播。最后在后面的恢复阶段将错误隔离的数据释放。

Liu Peng 等人在文献[19,20]中基于该隔离策略在传统的 COTS[19]（Commercial-Of-The-Shelf）DBMS 框架上面提出了一套隔离的体系结构。当恶意事务被识别时，先将该事务所操作的写数据集和时间戳晚于该事务提交时间的所有数据对象隔离，在后面的受损限制[19]（Damage Confinement）阶段逐步将错误隔离的正常数据释放。在文献[19]中，Liu Peng 给出了 5 种不同的受损限制的方法，并对每种方法的优缺点进行了详尽的介绍。最后基于该受损限制方法，采用依据时间戳的恢复算法使得最终数据库回到了一个正确的状态[21-23]。

4、多版本法

多版本法是基于对可疑用户的隔离。当入侵检测系统发现可疑用户时，通常会采用两种处理方式：（1） 立即停止该用户对数据库的访问，如果在后来的进一步的挖掘该用户行为的规则度[3]时发现该用户是正常的，则停止该用户对数据库的访问就给用户带来不必要的损失，同时也削弱了系统的可用性和效率；（2） 对用户的操作不做任何处理，如果在后来的进一步的挖掘该用户行为的规则度[3]时发现该用户是恶意的，则此时该用户的操作就对数据库中的数据造成了极大的破坏，最坏的情况可能造成整个系统不可用。多版本[24,25]采用的思想是当系统发现某个用户是可疑用户时，为每个用户构造一个由该用户维护的隔离版本数据的方法来将正常用户版本和可疑用户版本分开。当最终可疑用户被证明是正常用户时，该用户的所有操作都合并到正常版本中，否则抛弃该用户的操作。多版本法可以使可疑用户在一定的限制条件下继续执行，同时又不会对数据库造成进一步的损伤。

Liu Peng 等人在文献[14]中采用版本矢量的方法提出了一个在文件系统中使用的具体隔离算法，算法思想主要是当文件 F 被可疑用户修改时，该用户对文件 F 正在进行及其后来的修改操作将被隔离，为了区分可疑用户对 F 的修改和正常用户对

文件 F 的修改，文件 F 的正常版本和每个隔离版本都标记有相应的版本矢量号[14]，可疑用户和正常用户通过对不同版本的文件进行访问来达到隔离的目的。Jajodia 等人在文献[19]中将文件系统中隔离的思想引入到数据库中，给数据库中的记录采用打时间戳标签的方法来标识不同的版本，并且给出了基于版本隔离在数据库中实现的具体协议和算法。Liu Peng 等人在文献[26]中采用 Jajodia 提出的隔离算法实现了一个基于 Oracle 的原型系统 DAIS（Data Attack Isolation System），DAIS 能够对基于 Oracle 的银行应用系统[26]有较好的隔离效果。

Fayad 等人在文献[27]中提出了隔离部分重要数据的设想，并将数据按照重要性和完整性分成三类：无限制和非关键数据项、限制和非关键数据项、关键数据项，并定义了在不同数据项之间所允许的信息流。Fayad 提出了通过检测数据不一致性实现的应用级孤立算法。

以上四种方法是当前数据库系统中数据隔离的主要研究方面。静态分区法实现比较简单，但是容易造成过多的正常数据被隔离，且到目前为止没有实现的原型系统。数据标记法不能确切地定义数据被损坏的程度，对于不同标签数据之间的信息流向也没有很好的跟踪解决办法。多阶段隔离法在处理受损限制时并没有给出一个行之有效的方法，只是给出了一些原则，并且处理的精度不高。多版本法由于需要维护多个版本的数据库，因此占用系统的存储空间较大。前面三种方法只是在理论上对其可行性进行了研究，给出了相应的算法，并没有实现基于该算法的原型系统。对其效率和性能怎样，不得而知。Liu Peng 采用多版本法实现了一个基于 Oracle 的银行应用系统 DAIS 也只是在基于特定应用系统中的隔离，无法具有通用性。

1.2.2 数据隔离面临的问题

目前基于数据隔离的研究主要集中在多阶段法和多版本法，到目前为止还没有一套成熟的理论和方法来应用于通用的数据库平台。在国内对于这一领域的研究还比较少，国外研究也仅仅只是在理论上有一些成果，真正转化为实际的应用系统还不是很多。Liu Peng 用基于版本的方法实现了一个基于 Oracle 的原型系统 DAIS[26]，该原型是基于元组级的多版本隔离算法实现的,采用给元组打上时间戳标签的方法来实现版本隔离。DAIS 使用触发器和事务模板来得到事务的读写操作记录，通过

重新改写用户的 SQL 语句使得将事务操纵的数据定位到不同的数据版本。由于其事务模板是基于特定银行应用系统中的事务来提取的，比如说存款操作，取款操作等，这些事务都具有很强的模板性。在改写用户的 SQL 时，DAIS 也采用了大量的 Oracle 专用的语句和技术，这些都是其他的 DBMS 所不具备的，如 DB2 和 SQL SERVER 2000，因此该系统不具有通用性。

基于多版本法的数据隔离技术还有很多有待解决的地方，如入侵检测过程中检测时间和检测精度的平衡问题[28,29]，隔离版本的实现和维护问题[30,31]，隔离版本的数据一致性问题[32,33]，事务操作日志的粒度和完备性问题[34,35]，多用户隔离时的并发控制问题[36]，版本融合时的正确性问题[37,38]，以及系统的效率问题。

1.3 课题主要研究工作

由 1.2.1 节可知，当数据库遭受到攻击的时候，传统的数据库隔离级别无法保护数据库中数据的安全。由 1.2.2 节可知，基于数据库安全通用平台的数据隔离研究还处在起步阶段，算法还不具有通用性，还需要不断地改进和完善。本课题将对多种数据隔离技术进行研究和拓展，使其能够适用于多种数据库平台，建立一个实用性和通用性较强的数据隔离系统，以期在更广泛应用环境中发挥更大的作用。

本课题以 SQL SERVER 2000 为设计平台，研究工作包括以下几个方面。

第一，研究数据隔离的隔离方法和版本融合方法，使之达到以下的目标。

1、能正常的建立相应的隔离版本数据库，并对其进行有效地维护；

2、能有效地阻止受损数据的扩散，使得正常数据能够继续提供给正常用户；

3、能有效且全面地记录数据库中事务的操作日志，尤其是读操作日志的获取；

4、能有效地将隔离版本的数据库融合到正常版本的数据库中，并且保证融合的正确性；

5、能有效地将恶意用户对数据库造成的损伤恢复，使得数据库回到一个正确的状态。

第二，设计了相关的实验，来验证以上提出的目标是否达到，并且对相关的性能进行了分析。

2 入侵容忍的数据库中数据隔离系统的总体设计

本章先从四个方面来对入侵容忍的数据库中数据隔离进行需求分析，然后介绍数据隔离的模型，最后给出数据隔离系统的体系结构和模块划分，分别对每个模块进行功能说明。

2.1 入侵容忍的数据库中数据隔离需求分析

2.1.1 数据隔离中隔离版本的需求

1、隔离版本粒度的选择

当系统确认某个用户是可疑用户时，需要将该用户的操作放入到隔离的数据库中。要求隔离的数据库和真实的数据库具有相同的原始数据。在这里，有几种隔离单位可以供选择：库，表，元组，数据项。

选择库作为隔离单位是明显不合情理的。如果对于每个可疑用户都构造一个与真实数据库一样大小的隔离库，隔离库和真实库中含有相同的数据。那么当有 n 个用户时，数据库中就有 n+1 个库，数据库中的总容量就会成 n 倍增长，尤其是对于现在的应用系统，真实数据库中的数据量本身就很庞大，再要构造多个相同大小的数据库，是系统无法承受的。选择表作为隔离单位也是有相同问题的，当可疑用户只对表中的某几条记录做修改时，需要隔离的数据仅仅只是被可疑用户修改过的数据，选择将整张表的数据复制构造隔离表也是浪费了大量的存储空间的。

选择元组作为隔离单位是可取的，它避免了上面所提到的存储空间浪费的问题。但是和以数据项为隔离单位相比较，数据项的粒度更细，节省的存储容量也更多，且更容易依据数据项之间的读写依赖关系来判断事务之间的依赖关系。因此，应该以数据项为最小隔离单位来构造隔离版本的数据库。

2、隔离版本的生存期

隔离版本构造的时刻应该是可疑用户的事务访问数据项的时刻，这样可以确保不用构造多余的数据项，给出一个例子进行说明。

例 2.1：设库 testDB 中存在 Order_Details 表，该表用于保存订单的详细信息，

共包括 5 个字段(OrderID, ProductID, UnitPrice, Quantity, Discount)，分别表示订单的编号、产品号、单价、数量和折扣，其中 OrderID 是主键。

可疑用户执行事务 T：UPDATE Order_Details SET Quantity = Quantity *1.5 WHERE OrderID=21569;

分析例子可知，在事务访问数据项的时候构造隔离版本，此时只需要为 OrderID 为 21569 这行记录的 Quantity 构造一个隔离版本，而不需要将整张表中的所有记录的 Quantity 构造隔离版本。这样能确保不构造不必要的隔离版本。

隔离版本构造好后，应该由构造它的可疑用户来维护。在该用户没有最终被确认为是正常用户还是恶意用户之前，该隔离版本都不能删除。必须要等到最终确认了用户性质之后才能由构造它的用户来删除。

2.1.2 数据隔离中事务日志的需求

由于数据隔离系统是运行在 DBMS 和应用系统之间，是一个中间件。因此无法使用 DBMS 中的日志记录，为此必须构造一个和 DBMS 具有相同功能的日志文件。事务日志要能够得到事务写操作的所有记录，包括三种操作(INSERT、UPDATE、DELETE)，要获取到事务标识、操作类型、操作前的值、操作后的值、值的类型等等。此外，还必须要得到事务读操作的所有记录，这是因为要依据操作之间的读写依赖来判断数据的受损扩散情况。除了这些和 DBMS 日志系统相同的功能外，数据隔离中的事务日志还必须满足一些其他的条件。比如写到事务日志表中的事务，其执行结果都是成功(commit)的，没有失败的(abort)。这是因为没有成功提交的事务是不会对数据库中的数据进行修改的，也就没有必要对其操作进行隔离。

隔离版本构造的基本单位是数据项，事务日志记录的粒度也必须达到数据项。即每条日志记录都是对某一个数据项进行操作的记录，还必须要记录该数据项被操作时的确切时间值。在事务日志中，数据项 o_i 由一个五元组来唯一标识 $<database\_name, schema\_name, table\_name, column\_name, row\_number>$。其中 $database\_name$ 是数据库名；$schema\_name$ 是模式名；$table\_name$ 是表名，也就是关系；$column\_name$ 是列名，也就是属性列；row_number 是行号，也就是元组的标号。

此外，还要保证日志文件的安全性，日志文件是版本融合中唯一的数据来源，必须确保其安全性，要防止恶意用户随意修改日志文件中的记录。因此，必须采取严格的措施，保证日志文件不被除系统管理员之外的任何用户篡改。

获取日志时还要考虑的一个问题就是性能问题。由于记录日志的单位是数据项，当可疑事务执行全表查询的时候，需要插入到日志表中的记录数目是表的记录行数乘以表的列的数目。当表中的记录数很大的时候，这时候读操作的记录就是一个相当耗时的操作。因此，必须采取一定的处理措施，来保证日志的获取不会对系统的性能造成太大的影响。

2.1.3 数据隔离中版本融合的需求

当可疑用户最终被确认是正常用户时，此时需要将该用户维护的隔离版本的数据库融合到真实版本的数据库中。融合的时候首先要保证的就是融合的正确性，必须依据事务日志所记录的操作日志中的时戳值，得到可疑用户和正常用户对数据项操作的先后次序。在这里核心是要得到事务之间的依赖关系，以便于在融合的时候得到精确反映出事务的执行次序和所对应的操作。

为了确保融合时的正确性，在融合的阶段还必须对真实表中的待融合的记录进行锁定，在融合的时候不允许其他事务对该记录的访问。由于在融合的时候要涉及到查询大量的事务日志操作以及数据一致性的判断问题，因此融合时的性能必须要加以考虑。如果不采取相应的措施，融合的时候对整个隔离系统的影响是比较大的。

当可疑用户最终被确认为是恶意用户时，首先必须将该用户维护的隔离版本抛弃，同时还要将该用户对数据库造成的损伤进行恢复操作，使数据库回到一个正确的状态。在恢复的过程中，如何正确的找出所有受感染的数据是一个难点。依据事务日志找出所有的受损数据和受感染数据，构造回滚事务，使数据项都回到一个正确的状态值。

以上考虑的是单个可疑用户被隔离时的情况，当系统中同时有多个可疑用户被隔离的时候，除了要考虑上面所提到的情况外，还有一些其他的情况要注意。如多个隔离版本维护时，必须考虑到每个隔离版本构造时所依据的真实版本的数据库的状态初值是不一样的。多用户同时操作的并发控制问题以及多用户融合时不同隔离

版本融合的先后次序问题。

2.1.4 数据隔离中性能的需求

数据隔离系统是整个DSRP(Database Security Reinforce Platform)的一个子系统。DSRP 是介于应用系统和 DBMS 之间的中间件，包括监控子系统、审计子系统、恢复子系统等其他的子系统。要使得 DSRP 有一个比较好的性能，就必须要求各个子系统有严格的性能控制。

由于 DSRP 系统主要的性能损失是在审计子系统，为了尽最大限度地提高整个中间件的性能，必然要求其他的子系统性能损失不能太大，这是可行的。因为隔离子系统主要的性能损伤是在隔离版本数据库的构造和版本的维护上，采用的是数据项级别的隔离版本，构造和维护时所占用的资源和消耗的时间都是比较少的。

2.2 入侵容忍的数据库中的数据隔离方案

针对 2.1 节中提出的需求，提出数据隔离方案，首先给出事务模型的描述。

在此将数据库重新定义为数据项 $o_i(i=1,2,...,n)$ 的集合，数据项 o_i 的定义见 2.1.2 节。数据库表示为 $DB=\{o_1,o_2,...,o_n\}$，数据库的状态由数据项的状态来决定。定义事务是执行成功的操作序列 $T_i \subseteq \{(r_i[o_x],w_i[o_x]) \mid o_x \in DB\} \cup \{c_i\}$，其中 $r_i[o_x]$ 表示对数据项 o_x 的读操作，即 SELECT 操作，$w_i[o_x]$ 表示对数据项 o_x 的写操作，包括 INSERT、UPDATE、DELETE 三种操作中的一种；c_i 表示事务的 commit 操作。历史（history）被定义为事务集的执行序列，是一个基于读写操作的偏序集，记为 $H=(\Sigma,\prec_t)$。其中 $\Sigma=\bigcup_{i=1}^{n} T_i$，表示事务集合；$\prec_t$ 表示事务操作之间的时序关系，且有如果 $r_i[o_x] \in \Sigma \wedge w_i[o_x] \in \Sigma$，则有 $r_i[o_x] \prec_t w_i[o_x]$ 或者 $w_i[o_x] \prec_t r_i[o_x]$。两个历史等价是指他们定义在相同的事务集上有相同的操作集并且用相同的方式来定义事务的冲突操作。历史可串行化是指在历史 $H=(\Sigma,\prec_t)$ 中，事务 T_i 和事务 T_j 是历史中任意两个事务，若有 $\bigcup_{i=1}^{n}\{(r_i[o_x],w_i[o_x]) \mid o_x \in DB\} \prec_t \bigcup_{j=1}^{n}\{(r_j[o_x],w_j[o_x]) \mid o_x \in DB\}$，事务 T_i 在事务 T_j 之

前执行；或者$\bigcup_{j=1}^{n}\{(r_j[o_x],w_j[o_x])\mid o_x\in DB\}\prec_t\bigcup_{i=1}^{n}\{(r_i[o_x],w_i[o_x])\mid o_x\in DB\}$，事务$T_i$在事务$T_j$之后执行，则说明历史是可串行化的。正常用户记为$GU_i$，可疑用户记为$SU_i$，恶意用户记为$MU_i$。

在此模型下，给出数据隔离的方案。当用户被确认是可疑用户时，以用户为单位对隔离版本的数据库进行构造和维护。构造隔离版本数据库是以数据项为基本单位进行构造的，并且允许可疑用户在没有自己维护的可疑版本数据项的时候读取正常用户对真实版本数据项的更新值。即一个数据项o_i将含有一个真实版本值和多个隔离版本值。仅当可疑用户对数据项进行写操作时，该数据项才生成一个隔离版本。若该数据项没有被可疑用户进行写操作，该数据项就不会有隔离版本值。所以在一个数据库中，数据项的隔离版本的数量要比真实版本的数量要小得多。由可疑用户SU_i维护的数据项隔离版本，加上其他的真实版本的数据项，就构成了SU_i的虚拟隔离版本的数据库。数据隔离的最终实施是通过控制事务对不同的数据版本进行操作来实现的。

2.3 入侵容忍的数据库中数据隔离体系结构

基于以上的需求和隔离方案，在此给出隔离子系统的体系结构，如图 2.1 所示，主要由四大模块组成，分别是隔离控制模块、代理模块、事务日志模块以及版本融合模块。下面分别对四个模块的功能以及模块之间的信息流向做详细的表述。

当入侵检测子系统发现某用户的行为偏离了正常轨道，但在不进一步观察的情况下无法确认是否为正常或恶意用户，就将该用户设定为可疑用户，并且将其用户名和操纵数据库的语句发送给隔离子系统。

隔离控制模块主要是接收入侵检测子系统的数据，对 SQL 语句进行分析，获取用户想要操纵的表名，依据可疑用户的操作类型来执行不同的操作。若可疑用户的操作类型是 SELECT 操作，则直接读真实表中的数据；若可疑用户执行的是 INSERT、UPDATE、DELETE 中操作的一种，则首先去查系统中是否有该可疑用户维护的隔离表。若有该用户维护的隔离表，则将该语句定位到该隔离表中进行操作；若没有隔离表，则先用存储过程获取真实表的创建语句，建立相同结构的隔离表，再将该

语句定位到该隔离表中进行操作。

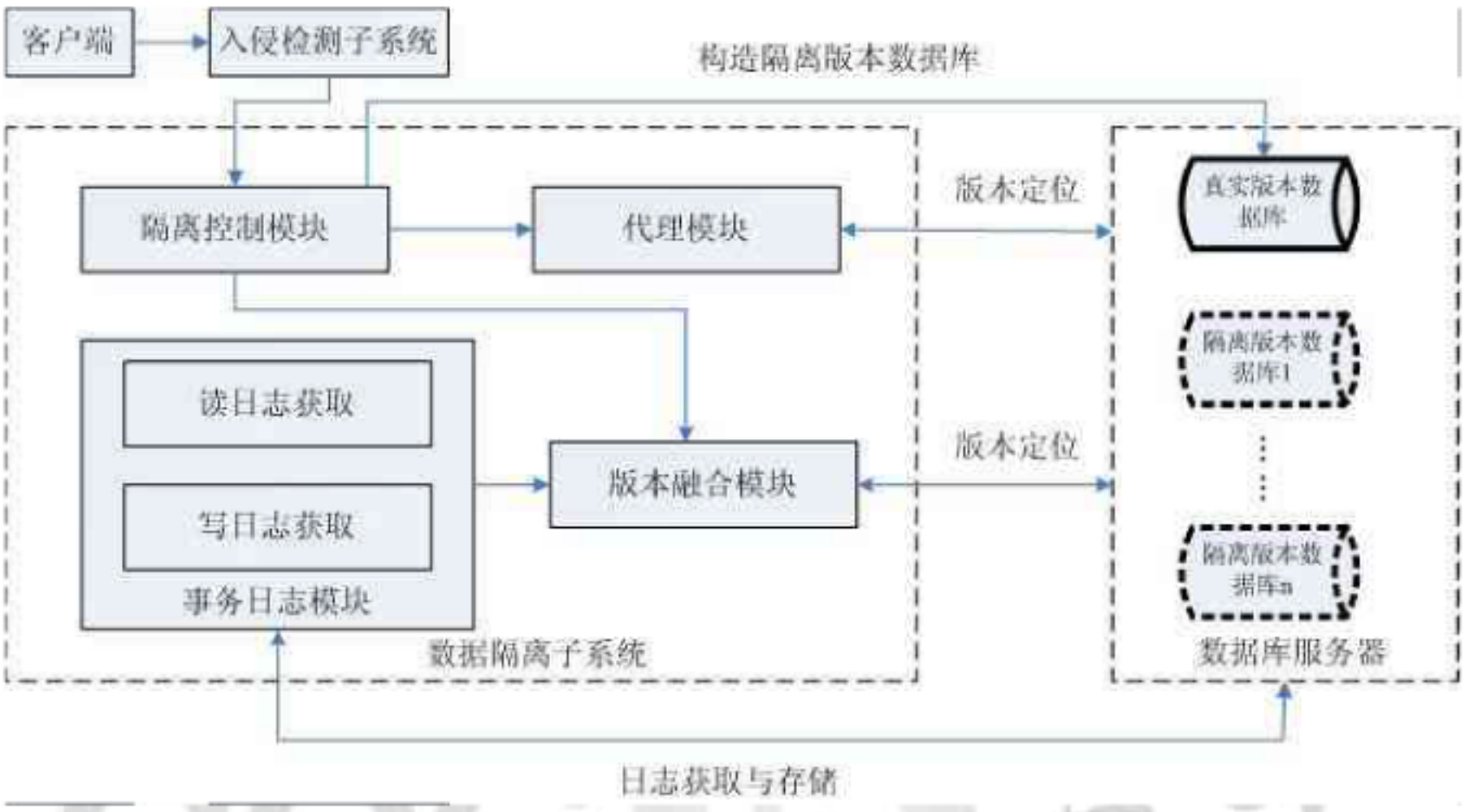

图 2.1 数据隔离子系统体系结构

隔离用户的操作语句经过隔离控制模块的处理后，并不是直接得到返回的结果。而是通过代理模块，代理模块将隔离用户 SQL 语句执行后的结果处理后分别写入到相应版本的隔离表中。

在隔离子系统运行的过程当中，入侵检测子系统仍旧对该用户的行为进行监控和分析，最终会确认该用户的性质。若入侵检测最终发现该用户是恶意用户，则直接将该用户维护的隔离版本的数据丢弃，并且将该用户被判定为可疑用户之前对数据库的操作进行 undo 操作，则该用户不会对数据库中的数据造成任何损伤。若入侵检测最终发现该用户是正常用户时，则要将该用户维护的隔离版本数据库融合到真实版本的数据库当中。融合的前提是要知道该数据库中所有操作的日志记录，通过日志记录去查找所有事务之间的读写依赖关系，为版本融合模块提供数据支持。

事务日志模块主要是记录某一数据项被修改时的值的变化以及修改的精确时间。采用触发器来获取用户的写操作，对表中每个列都设置触发器，当对其有写操作时，利用系统中两张临时表 Inserted 表和 Deleted 表，可以得到写操作的精确信息。对于事务的读操作，采用“读模版”和写临时表的方法来获取。

有了事务日志，就能得到数据库中所有的操作日志，依据此操作日志能够构造

一张“前向图”，用此来识别和解决融合过程当中的数据不一致状态。这样就能够将隔离版本的数据库合并到真实版本的数据库当中，保证了数据的正确性。

与 DSRP 中其他子系统相比，隔离子系统的独立性较高。隔离子系统只需要得到入侵检测子系统发送过来的可疑用户的信息即可，与 Liu Peng 的 DAIS 系统相比较，图 2.1 所描述的隔离子系统有如下特点。

1、DAIS 采取的隔离粒度是元组级别的，本系统中采用的是数据项级别的，隔离级别更细，精度更高。

2、DAIS 体系结构是为专门的银行应用系统来设计的，运用了大量的事务模版表来提取操作事务的特征从而得到相应的读操作；本系统的体系结构是面向任意应用系统的，不局限于某种特定的应用系统，具有较好的通用性。

3、在隔离版本的构造上，本系统不依据特定的数据库服务器。DAIS 是运行在 Oracle 8i 上的，其构造隔离版本数据库时大量运用 Oracle 自身特有的一些语句和技术；本系统则主要运用数据库本身提供的系统表来操作。

2.4 小结

本章首先从四个方面分析了入侵容忍的数据库中数据隔离的需求，首先是隔离版本的需求，其次是事务日志的需求，再次是版本融合的需求，前三个都是功能上的需求，最后是性能上的需求。接着依据这些需求给出了数据隔离的方案。基于给出的隔离方案，最后给出了数据隔离的系统结构图，该结构包括四个模块，分别对每个模块的流程进行了说明，并且将该结构与 Liu Peng 的 DAIS 系统进行了比较，分析了优缺点。

3 入侵容忍的数据库中数据隔离系统的实现

本章将具体介绍入侵容忍的数据库中数据隔离子系统包含的四个模块：隔离控制模块、代理模块、事务日志模块以及版本融合模块。分别对每个模块的所涉及到的算法进行说明、介绍算法流程并给出实例、对算法的正确性进行论述、以及算法的效率等方面来加以介绍。

3.1 隔离控制模块的实现

针对 2.1.1 节中提出的需求，着重阐述隔离控制模块的实现，分为两个小节加以介绍。第一节介绍隔离表的构造方法；第二节主要介绍隔离版本控制的算法实现。

3.1.1 隔离表的构造及维护

1、隔离表构造流程

依据 2.2 节中提出的隔离方案，隔离采用的最小单位是数据项。且只要对数据项进行写操作的时候才会产生隔离版本的数据项，下面给出例子进行说明。

例 3.1：可疑用户执行事务 T，将订单号为 32562 的订单数量设置为和订单号为 32561 的数量一致，事务语句如下：

```
DECLARE @Quantity smallint
BEGIN TRANSACTION
SET @Quantity = (SELECT Quantity FROM Order_Details WHERE OrderID = 32561)
UPDATE Order_Details SET Quantity = @Quantity WHERE OrderID=32562
COMMIT TRANSACTION
```

该事务包含两条操作语句，第一条语句是读取订单号为 32561 中的 Quantity 属性列的值，这是一个读操作，依据隔离方案，读操作是不用生成隔离版本的。因此，订单号为 32561 的这条记录中的 Quantity 属性不用构造相应的隔离版本。第二条语句是写操作，将订单号为 32562 的这条记录中的 Quantity 属性值修改为和订单号为 32561 中的一样。这时候，依据隔离方案，应该要构造一个隔离版本，隔离版本的

值是修改后的值。

相应的隔离版本构造好之后，必须要有一定的存储方式来存储隔离版本。最开始的想法是构造一张系统表，每构造一个隔离版本的数据项，就将该隔离版本插入到系统表中。系统表的字段包括：用户名、库名、模式名、表名、属性名、行号、数据项值、数据项值的类型、版本标识、构造时间。后来在实际的编码当中，发现这种方法是不可行的。首先，版本标识这一个字段很难将数据项隔离版本的信息表达清楚，特别是当系统中有多个可疑用户同时对某一数据项进行写操作的情况，还有单个可疑用户同时对某一数据项进行多次写操作的情况。其次，这种存储方式很难得到数据项之间的读写依赖关系，这对于后面数据一致性的判断是很不利的。最后，多个可疑用户同时在系统中存在的时候，要同时对这张系统表进行大量的插入（隔离版本构造）和读取（版本融合时数据一致性判断）操作，会造成表级锁定问题，对系统的整体性能有很大的损伤。

在这里，采用的是和真实表具有相同结构的隔离表来存储相应的隔离版本。隔离表与真实表有相同表结构、相同的约束条件、相同的索引、触发器和存储过程。所不同的是，隔离表中并不是含有所有真实表中的数据项，只有那些被可疑用户更新过的数据项，才会构造相应的隔离版本存储在隔离表中。因此隔离表和对应的真实表相比较，其容量会小得多。为了区别隔离表和真实表，将真实表的表名和该用户被确定为可疑用户时的时戳值共同作为隔离表的表名。

还是以例 3.1 来说明，当可疑用户执行事务 T 时，通过分析事务 T 中的语句可以得到要访问的表是 Order_Details 表，则构造和 Order_Details 具有相同表结构的隔离表 Order_Details_20090506（假定该用户被发现为可疑时的时间是 2009-05-06 16:24:44.903）。表 Order_Details_20090506 和表 Order_Details 具有相同的约束、索引、触发器和存储过程，构造的订单号为 32562 的记录中的 Quantity 属性的隔离版本的值和这条记录的主键值就存储在表 Order_Details_20090506 中。

采用用户为单位来构造隔离表，下面给出构造隔离表流程的描述：

当可疑用户的 SQL 语句从入侵检测子系统发到隔离控制子系统时，依据该语句的操作类型来确定是否为该用户创建隔离表，假定用户名为 user，用户的 SQL 语句为 sql_str，函数 anay_sql_table(sql_str)用来获取 sql_str 语句所访问的表名集合,用

tableSets 来表示，函数 anay_sql_type(sql_str)用来获取 sql_str 的操作类型，其结果⊂{SELECT,UPDATE,INSERT,DELETE},函数 isola_table_exist(table,user)用来获取在系统中是否存在 user 用户以真实表 table 为基础构造的隔离表。算法 3.1 给出了隔离表建立的伪代码表示。

算法 3.1：隔离表建立算法

输入：sql_str，可疑用户的执行语句

输出：建立相应的隔离表

方法：

```
1   If anay_sql_type(sql_str)＝SELECT
2       DO NOT CREATE ISOLA_TABLE     //不用建立隔离表
3   Endif
4   If anay_sql_type(sql_str)⊂{UPDATE,INSERT,DELETE}
5       tableSets=anay_sql_table(sql_str)      //获取该语句访问的表名集合
6       For i=1 to tableSets.size() do
7           table=tableSets.get(i)                  //取出每个待访问表的表名
8           If isola_table_exist(table,user)==FALSE
                //系统中不存在该用户以表 table 为基础的隔离表
9               get_table_sql(table)            //获取表的创建语句
10              get_trigger_sql(table)          //获取表的触发器语句
11              get_index_sql(table)            //获取表的索引信息
12              get_constraint_sql(table)       //获取表的约束信息
13              CREATE ISOLA_TABLE     //构造相应的隔离表
14          Endif
15      Endfor
16 Endif
```

假定 tableSets 的容量为 n，即 SQL 语句访问的表有 n 个。则算法 3.1 在最坏的情况下比较的次数是 n 次，即算法的时间复杂度为 $O(n)$。

2、隔离表构造实现

在这里，依据用户 SQL 语句所访问的真实表的表名来构造隔离表。由于 DSRP 系统是中间件，并不是在 DBMS 底层上，所以无法得到真实表的创建语句。为此，必须要有能得到表的创建语句的方法。目前流行的商用数据库管理系统中，Oracle 可以利用自带的工具导出某张表的创建语句，在其他的数据库管理系统中，都没有提供系统存储过程或者是工具来得到表的创建语句。但是几乎所有的 DBMS 系统都会提供一系列的系统表。比如 SQL SERVER 2000 中提供以下系统表：

Sysobjects：在数据库内创建的每个对象（约束、默认值、日志、规则、存储过程等）信息；

Syscomments：包含每个视图、规则、默认值、触发器、CHECK 约束、DEFAULT 约束和存储过程的项；

Syscolumns：每个表和视图中的每列在表中占一行，存储过程中的每个参数在表中也占一行；

Systypes：对于每种系统提供数据类型和用户定义数据类型，均包含一行信息。

这些系统表能够反映数据库中的所有信息，对此采用自己创建一个存储过程 SP_GET_TABLE_INFO，通过对这一系列系统表的查询操作，来组合得到本系统所需要表的创建信息。这些语句包括：主键信息、表上面的索引信息、表上面的触发器信息。得到真实表的创建语句后，将真实表中的表名替换成隔离表的表名，再交给服务器执行，至此，隔离表已经创建完毕。

采用存储过程主要是基于性能上的考虑。存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中。用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。由于存储过程会预先编译好放在数据库内，减少编译语句所花的时间。编译好的存储过程会进入缓存，对于经常执行的存储过程，除了第一次执行外，其他执行的速度都会有明显提高。因此存储过程具备一次编译，多次执行的特点。在程序中，将存储过程一次定义后，其他再次需要该功能来获得表的创建语句时，无须重复写代码，只需要调用该存储过程，这样可以提高系统的响应能力和性能。

3、隔离版本的维护

由于以用户为单位来构造隔离版本的数据库，在用户构造单个隔离表之后，必

须以用户为单位来对这些隔离表进行维护，构造了一张表 ISOLA_TABLE_INFO 来存放用户构造的隔离表的信息，表详细描述如表 3.1 所示。

表 3.1 表 ISOLA_TABLE_INFO 结构

列名	类型
DATABASE_NAME	VARCHAR(50)
USER_NAME	VARCHAR(50)
REAL_TABLE	VARCHAR(50)
ISOLA_TABLE	VARCHAR(50)

其中 DATABASE_NAME 为隔离表所在的数据库名，USER_NAME 为用户名，REAL_TABLE 为真实表的表名，ISOLA_TABLE 为该用户依据 REAL_TABLE 构造的隔离表表名。可疑用户在被隔离的过程中每构造一个隔离表，就将该隔离表的信息写入到表 ISOLA_TABLE_INFO 中；在最终用户被确认正常或者是恶意之后，不管是融合操作还是恢复操作，每销毁一张隔离表，就要将该隔离表在该表中维护的记录删除掉。通过查询表 ISOLA_TABLE_INFO，就能很清楚地知道某一用户所维护的隔离数据库中包含哪些隔离表。

以用户为单位来构造和维护隔离表可以很方便地得到数据项的版本号，各个数据项的隔离版本存放在相应不同的隔离表中，这样在融合时候的数据一致性的判定时就不会出现表级锁定的问题。而且通过事务日志可以得到数据项的详细操作的日志，这对于判断数据项之间的读写依赖关系很有帮助。

4、隔离版本的安全性

隔离版本的数据项都是由可疑用户自身生成的，该隔离表也是由该用户来维护的，因此必须确保隔离表中的数据不能被其他的用户操纵。当某个用户想要往隔离表中插入隔离版本的数据时，首先要查询表 ISOLA_TABLE_INFO，看该用户要访问的隔离表记录中的 USER_NAME 是否是这个用户。若查询出的用户名和当前访问的用户名不相同时，系统则拒绝该用户对隔离表的操作。

同样，表 ISOLA_TABLE_INFO 由于是维护隔离表的信息，其安全性也要受到重视，一旦受到恶意用户的篡改，则会造成数据项真实版本和隔离版本之间对应的紊乱，所以必须防止恶意用户篡改隔离表的信息。对该表限制只能允许 SELECT、

INSERT、DELETE 操作，不允许任何用户对表 ISOLA_TABLE_INFO 进行 UPDATE 操作。并且限制只有系统管理员才有权限对其进行 SELECT、INSERT、DELETE 操作，非系统管理员只能对其进行 SELECT 操作。该表是存放在专门的数据库 DSRP 中，对于非系统管理员进入该数据库的用户，可以由监控子系统对用户的行为进行监控，发现可疑行为就立即将该用户判定为恶意用户。

3.1.2 隔离版本控制

在文献[27]中，Jajodia 提出了三种隔离策略，分别为完全隔离（Complete Isolation）、单边隔离（One-way Isolation）、部分隔离（Partial Isolation）。三种隔离的定义如下：

完全隔离策略：其他用户不允许读取可疑用户 SU_1 的更新操作，可疑用户 SU_1 也不允许读取其他正常用户和可疑用户的更新操作；

单边隔离策略：其他用户不允许读取可疑用户 SU_1 的更新操作，但是可疑用户 SU_1 可以读取其他用户的更新操作；

部分隔离策略：其他用户能够部分读取可疑用户 SU_1 的更新操作，可疑用户 SU_1 也能够部分读取其他用户的更新操作。

1、隔离策略的选取

完全隔离策略使得正常用户和隔离用户之间的信息完全不能流通。对于同一数据项 o_i，正常用户和各个可疑用户对其进行的操作都是独立的，这在实现上较为复杂且难度是比较大的[27]。同时在最终版本融合的时候，操纵数据项之间的读写依赖关系和该数据项被更新时的严格的时间顺序是很难得到的，或者说是不确切的。这将导致最终版本融合时数据的正确性是无法保证的。完全隔离由于信息之间的不流通，在最终判断数据一致性的时候会耗费很长的时间，在性能上是比较低的。部分隔离策略中所说的部分读取是一个很模糊的概念，并没有一个衡量准则来说明什么样的数据是可以同时被正常用户和可疑用户访问，什么样的数据只能被正常用户访问不能被可疑用户访问。因此，这种隔离策略在实现上是不太可行的。

本系统选取的是单边隔离策略，这是基于实现的可行性和性能上来考虑的。单边隔离在实现上较完全隔离要简单一些，而且在性能上要好得多。依据单边隔离策

略的思想，提出自己的隔离版本控制的准则。特别要提到的是，可疑用户 SU_1 的事务只能更新由 SU_1 生成的隔离版本。当 SU_1 的事务想读取数据项 o_i 的时候，如果这时候数据项 o_i 没有由 SU_1 生成的隔离版本，那么就去读真实版本，否则就读相应的隔离版本。具体流程如下：

（1） 在数据库系统开始执行用户操作之前，每个数据项 o_i 有一个唯一的版本，也就是本身的真值，用 $o_i(main)$ 表示；

（2） 当正常用户 GU_i 的事务想要读或者更新 o_i 的时候， $o_i(main)$ 就给 GU_i ；

（3） 当可疑用户 SU_i 提交的事务想要更新 o_i 时，若此时该用户没有 o_i 的隔离版本，则由算法 3.1 构造隔离版本，记为 $o_i(SU_{ti})$，并将该版本给事务做更新操作；若此时有该用户维护的可疑版本，则直接将隔离版本给事务做更新操作；

（4） 当可疑用户 SU_i 提交的事务想要读取 o_i 的时候，若此时有相应的隔离版本 $o_i(SU_{ti})$，则事务读取 $o_i(SU_{ti})$ 的值，否则事务读取 $o_i(main)$ 的值；

（5） 若存在可疑用户 SU_i 和 SU_j $(i \neq j)$，对于同一数据项 o_i，分别维护有相应的隔离版本 $o_i(SU_{ti})$ 和 $o_i(SU_{tj})$，则 SU_i 不能操纵 $o_i(SU_{tj})$ 且 SU_j 不能操纵 $o_i(SU_{ti})$。

2、版本控制实现

依据 1 中给出的版本控制流程，给出版本控制的算法。隔离版本控制的算法思想主要是在执行可疑用户的 SQL 语句之前，先将该语句要操纵的真实表 R 和有相同表结构的隔离表 S 进行一个预处理，使得隔离表 S 中的记录“映射”到真实表 R 中，待相关的语句执行完毕后再将真实表 R 中的记录还原。算法 3.2 主要是将隔离表 S 的信息“映射”到真实表 R 中后，再对真实表 R 进行读操作，操作完毕后再将真实表 R 中的记录还原。算法 3.2 给出了版本控制算法的伪代码表示。

算法 3.2：隔离版本控制算法

输入：sql_select，可疑用户的查询语句

输出：table_record_sql_redelete，先将记录集插入到真实表中后需要删除该记录集的语句

table_record_sql_reinsert，先将记录集从真实表中删除后需要插入该记录集的语句

sql_rewrite_list，改写后的 SQL 语句序列

方法：

```
1   tableSets=anay_sql_table(sql_select)          //获取该语句访问的表名集合
```

```
2  For i=1 to tableSets.size() do
3   table=tableSets.get(i)                          //取出每个待访问表的表名
4   If(存在与 table 相同结构的隔离表 isol_table)
5      get_primary_keys(table)                      //获取该表的主键
6      查询日志记录，依据主键值查找 table 中含有 isola_table 中没有的记录集 records
7      从 table 中删除该 records，并将删除的 records 改写成重新插入的 sql 语句 sql_delete
8      table_record_sql_reinsert.add(sql_delete) //将语句加入到链表中，供最后恢复用
9    Endif
10 Endfor
11 For i=1 to tableSets.size() do
12   table=tableSets.get(i)                          //取出每个待访问表的表名
13   If(存在与 table 相同结构的隔离表 isol_table)
14     get_primary_keys(table)                       //获取该表的主键
15     查询日志记录，依据主键值查找 isola_table 中含有 table 中没有的记录集 records
16     从 table 中插入该 records，并将插入的 records 改写成重新删除的 sql 语句 sql_insert
17     table_record_sql_redelete.add(sql_insert)  //将语句加入到链表中，供最后恢复用
18   Endif
19 Endfor
20 查询表 ISOLA_TABLE_INFO，得到 tableSets 所对应的隔离表集 isol_tableSets
21 For i=1 to tableSets.size() do
22   table=tableSets.get(i)                          //取出每个待访问表的真实表名
23   isol_table=isol_tableSets.get(i)                //取出每个待访问表的隔离表名
24   sel=SELECT table.* INTO #tempi FROM table, isol_table WHERE table.primary_key
     IN(SELECT isol_table.primary_key FROM isol_table)
25   upd=UPDATE table ta SET(all fields)=(SELECT tb.* FROM isol_table tb WHERE
     ta.primary_key = tb.primary_key) WHERE table.primary_key IN (SELECT
     isol_table.primary_key FROM isol_table)
26   sql_rewrite_list.add(sel)                       //改写后的语句添加到 sql_rewrite_list
```

```
27    sql_rewrite_list.add(upd)                    //改写后的语句添加到 sql_rewrite_list
28 Endfor
29 sql_rewrite_list.add(sql_select)                //将用户自己的语句添加到 sql_rewrite_list
30 For i=1 to tableSets.size() do
31    table=tableSets.get(i)                       //取出每个待访问表的真实表名
32    isol_table=isol_tableSets.get(i)             //取出每个待访问表的隔离表名
33    res=UPDATE table ta SET(all fields) = (SELECT tb.* FROM #tempi tb WHERE
          ta.primary_key=tb.primary_key);DROP TABLE #tempi
34    sql_rewrite_list.add(res)                    //改写后的语句添加到 sql_rewrite_list
35 Endfor
```

假定 tableSets 的容量为 n，即 SQL 语句访问的真实表有 n 个，假定每个真实表都有一个对应的隔离表，则算法 3.2 在最坏的情况下比较的次数是 $4n$ 次，即算法的时间复杂度为 $O(n)$。

3、版本控制分析

假定数据库中的每个真实表 R 都有一个相同结构与之对应的隔离表 S。算法 3.2 中的第一个 for 循环的作用是将从真实表 R 中删除和隔离表 S 中有相同主键值的记录；第二个 for 循环的作用是将隔离表 S 中已经删除但是和真实表 R 中主键值不相同的记录插入到真实表 R 中；第三个 for 循环中有两条语句，sel 语句和 upd 语句。这两条语句是在前两个 for 循环已经对真实表 R 进行处理的基础上再次对 R 进行操作。sel 语句的作用是将真实表 R 中含有和隔离表 S 具有相同主键值的记录存放在一个临时表中，upd 语句的作用是将真实表 R 中含有和隔离表 S 具有相同主键值的记录所有属性的值更改为隔离表 S 中的值。算法进行到这里已经将隔离表 S 中的记录“映射”到真实表 R 中，此时就可以执行可疑用户本身的 SELECT 语句了，这也满足单边隔离的策略。即将隔离表 S 中的记录“映射”到真实表 R 中的目的是为了能够使隔离用户能够读取到正常版本数据项的值。第四个 for 循环中的 res 语句的作用是将 upd 语句改写过的数据项的值恢复为原来真实表 R 中的真值。

版本控制的实质是将隔离表的信息“映射”到真实表中后，再对真实表进行读操作，操作完毕后再将真实表中的记录还原。在这个过程当前后，并没有对真实表

中数据项的值做出任何修改，而且还满足了 1 中的隔离版本流程中的（4）（5）。因此，版本控制算法是正确的。下面给出一个实例来对算法进行说明。

例 3.2 还是依据 2.1 中的表结构，假定此时可疑用户已经依据算法 3.1 构造了和 Order_Details 有相同结构的隔离表，表名为 order_details_20090508，用户执行事务 T：

T：SELECT * FROM Order_Details WHERE OrderID=32562 and ProductID=1002

经过算法 3.2 的处理，如图 3.1 所示得到改写后的 sql_rewrite_list 语句。

改写后的sel:

select order_details.* into #temp0 from order_details,order_details_20090508 where order_details.OrderID = order_details_20090508.OrderID ;

改写后的upd:

```
update order_details ta set
OrderID=(select OrderID from order_details_20090508 tb where ta.OrderID =tb.OrderID ),
ProductID=(select ProductID from order_details_20090508 tb where ta.OrderID=tb.OrderID ),
UnitPrice=(select UnitPrice from order_details_20090508 tb where ta.OrderID =tb.OrderID ),
Quantity=(select Quantity from order_details_20090508 tb where ta.OrderID =tb.OrderID ),
Discount=(select Discount from order_details_20090508 tb where ta.OrderID =tb.OrderID )
from order_details,order_details_20090508
where order_details.OrderID=order_details_20090508.OrderID;
```

改写后的用户语句：

```
select * from order_details where OrderID=32562 and ProductID=1002
```

改写后的res:

```
update order_details set
OrderID=(select OrderID from #temp0 where order_details.OrderID=#temp0.OrderID ),
ProductID=(select ProductID from #temp0 where order_details.OrderID=#temp0.OrderID ),
UnitPrice=(select UnitPrice from #temp0 where order_details.OrderID=#temp0.OrderID ),
Quantity=(select Quantity from #temp0 where order_details.OrderID=#temp0.OrderID ),
Discount=(select Discount from #temp0 where order_details.OrderID=#temp0.OrderID )
from order_details, #temp0
where order_details.OrderID=#temp0.OrderID;
drop table #temp0;
```

图 3.1 例 3.2 中的 sql_rewrite_list 语句

在实现的过程中，采用了临时表。在 SQL SERVER 2000 中，临时表与永久表相似，但临时表存储在 tempdb 库中，当不再使用时会自动删除。使用临时表也是从性能上来考虑的，只需要使用如 SELECT * INTO #TEMP FROM TABLE 这样简单的 SQL 语句便可以将表 TABLE 中的数据导入到临时表#TEMP 中。这在算法 3.2 中的第三个 for 循环时保存真实表和隔离表中具有相同主键值的记录时很方便，而且速度也很快，可以提高整个隔离子系统的效率。

至此，可疑用户的隔离版本数据库已经创建完毕，可疑用户执行的操作语句也已经定位到了相应的隔离版本中，接下来就可以交给代理模块来真正执行用户的操作语句。

3.2 代理模块的实现

SQL 语句包括数据操纵语句（DML），数据定义语句（DDL），事务控制语句以及会话控制语句。在 DSRP 系统中，针对的是数据操纵语句的安全性的分析。虽然数据操纵语句有很多种，但是最基本的操作还是增、删、查、改四种操作，因此，在隔离子系统的实现中，考虑的也是这四种情况的操作语句。

版本控制模块只是对隔离用户的操作语句进行一个前期的预处理过程，此时只是获得了相应的版本。对于 SELECT 操作，可以将算法 3.2 中输出的 sql_rewrite_list 直接发送给数据库服务器执行后返回相应的结果。但是对于写操作集{UPDATE、INSERT、DELETE}中的任何一种写操作，都不能直接将版本控制处理后的 SQL 语句发送给数据库服务器直接执行。如果不用代理，则写操作将会直接对真实表中的数据进行改写，这是违背单边隔离策略的，也是不允许的。

在隔离版本控制的基础上，将可疑用户的 SQL 语句的写操作全部要“改写”成读操作的形式。用读操作获取相应的记录后，再对这些记录进行处理，将更新后的数据项的值写入到相应的隔离表中。

下面分别介绍四种语句（SELECT、INSERT、UPDATE、DELETE）的代理模块的实现。

1、SELECT 语句

这种情况比较简单，实质上算法 3.2 已经差不多完成了读操作时的代理实现情

况。这也是因为要满足单边隔离的策略，算法 3.3 给出了 SELECT 语句代理操作时的伪代码表示。

算法 3.3：SELECT 语句代理实现算法

输入：sql_select，可疑用户的读操作语句

table_record_sql_redelete，算法 3.1 执行 sql_select 的结果

table_record_sql_reinsert，算法 3.1 执行 sql_select 的结果

sql_rewrite_list，算法 3.1 执行 sql_select 的结果

输出：resultSets，返回给用户相应的查询结果

方法：

```
1   send sql_rewrite_list to the DB server and return resultSets
        //将语句序列发送给服务器执行并返回结果
2   send table_record_sql_redelete to the DB server
        //将原先插入到真实表中的记录删除
3   send table_record_sql_reinsert to the DB server
        //将原来从真实表中删除的记录重新插入到真实表中
```

算法 3.3 只执行了有限步，因此算法的时间复杂度为 $O(1)$。

算法 3.3 比较简单，其正确性是显而易见的。算法中的 2、3 两步是为了使在算法 3.2 中预处理中真实表中的修改后的记录恢复到没有修改之前的状态。使得真实表在可疑用户的读操作时既能让可疑用户读取到真实版本的数据值，又不破坏真实表中的数据值。

2、INSERT 语句

当可疑用户执行的是 insert 操作时，因该操作是写操作，必须先判断是否有该用户维护的隔离表。若没有，则由算法 3.1 建立相关的隔离表。接着要判断该插入语句的赋值语句中是否含有子查询的表。若有子查询的表，则对这些表运用算法 3.3 进行处理后再改写用户执行的插入语句；若没有子查询的表，则直接改写用户执行的插入语句。插入操作一般是插入一条记录，即含有多个数据项的值。因此，只需要将待插入的数据项的值直接插入到相应的隔离表中即可，不需要依据真实表中的数据项来构造隔离版本的数据项。其隔离版本的数据项的值直接就是用户插入操作

的值。算法 3.4 给出了 INSERT 语句代理操作时的伪代码表示。

算法 3.4：INSERT 语句代理实现算法

输入：sql_insert，可疑用户插入操作语句

输出：status，操作后的状态，1 表示成功，0 表示失败

方法：

```
1   table=anay_sql_table(sql_insert)     //获取该语句要插入的真实表的表名
2   If(不存在与 table 相同结构的隔离表 isol_table)
3       运用算法 3.1 创建隔离表          //创建该用户维护的隔离表
4   Endif
5   If(sql_insert 中的赋值子句中含有子查询)
6       将 sql_insert 中的"insert into table"去掉，改写后的语句记为 sql_insert_subquery
                                         //获取要插入的数据，用 select 来代替
7       将 sql_insert_subquery 运用算法 3.3 来处理，得到结果 resultSets
                  //将插入语句改写为读操作后，获取要操作的集合，再写入到相应的隔离表中
8       将 resultSets 插入到相应的隔离表中，并将 1 赋值给 status 返回
9   Endif
10 If(sql_insert 中的赋值子句中不含有子查询)
11      将 sql_insert 语句中真实表表名 table 改写为隔离表表名 isol_table
12      将改写后的语句发送给服务器执行，并返回结果给 status
13 Endif
```

算法 3.4 中除了运用算法 3.1 和算法 3.3 外，其余的步骤都是有限步。因此，其算法的时间复杂度与算法 3.1 和算法 3.3 相同，为 $O(n)$。

3、DELETE 语句

当可疑用户执行 delete 语句时，因是写操作，同样必须先判断是否存在由该用户维护的隔离表。由于可疑用户只能对自己维护的可疑版本的数据进行写操作。因此，不允许隔离用户对真实表进行删除操作。必须改写用户的语句，使其只能删除与真实表对应的并且是由自己维护的隔离表中的记录。若要删除的记录存在于隔离表中，则从隔离表中删除该记录；若要删除的记录只存在于真实表而不存在于隔离

表中，则向事务日志记录中插入该操作，但实质上是不做任何删除操作。这样做的目地是为了在版本融合的时候，能够得到事务之间的操作序列，为正确融合提供数据支持。算法 3.5 给出了 DELETE 语句代理操作时的伪代码表示。

算法 3.5：DELETE 语句代理实现算法

输入：sql_delete，可疑用户删除操作语句

输出：status，操作后的状态，1 表示成功，0 表示失败

方法：

```
1   table=anay_sql_table(sql_delete)        //获取该语句要删除的真实表的表名
2   If(不存在与 table 相同结构的隔离表 isol_table)
3      运用算法 3.1 创建隔离表              //创建该用户维护的隔离表
4   Endif
5   将 sql_delete 中的"delete"改为"select * "，改写后的语句记为 sql_delete_rewrite
6   将 sql_delete_rewrite 运用算法 3.3 来处理，得到结果集 resultSets
                                          //将删除语句改写为读操作后，获取要操作的集合，
                                          //再从相应的隔离表中删除有相同主键值的记录
7   For i=1 to resultSets.size() do
8      result= resultSets.get(i)
9      If(result 在相应的隔离表 iso_table 中)
10        delete this record from iso_table //从相应的隔离表中删除该记录
11     Endif
12     Else   insert a Delete operation to the AUDIT_TABLE   //向事务日志中插入操作记录
13  Endfor
```

假定算法 3.5 中运用算法 3.1 时执行了 n 次，运用算法 3.3 时执行了 m 次，resultSets 的容量为 k，则算法 3.5 在最坏的情况下执行了 $m+n+k$ 次。故算法的时间复杂度为 $O(n)$。

4、UPDATE 语句

当可疑用户执行 update 语句时，因是写操作，同样必须先判断是否存在由该用户维护的隔离表。同样，本系统中也不能直接在真实表中进行修改操作，必须在相

应的隔离表中进行。首先必须将用户要修改的数据所在表中的相关记录复制到该用户维护的隔离表中，再改写用户的操作语句，在该隔离表中对某些数据项进行修改操作。算法 3.6 给出了 UPDATE 语句代理操作时的伪代码表示。

算法 3.6：UPDATE 语句代理实现算法

输入：sql_update，可疑用户修改操作语句

输出：status，操作后的状态，1 表示成功，0 表示失败

方法：

```
1  table=anay_sql_table(sql_update)  //获取该语句要修改的真实表的表名
2  If(不存在与 table 相同结构的隔离表 isol_table)
3     运用算法 3.1 创建隔离表        //创建该用户维护的隔离表
4  Endif
5  改写 sql_update 语句，将真实表中要更新属性所在的记录复制到临时表#temp_insert 中
6  将临时表#temp_insert 中的所有记录赋值到隔离表 isol_table 中
7  将 sql_update 中的真实表名替换为隔离表名，改写后的语句记为 sql_update_rewrite
8  将 sql_update_rewrite 发送给服务器执行并返回执行状态给 status
```

假定算法 3.6 中运用算法 3.1 时执行了 n 次，其余操作执行了有限次，则算法 3.6 在最坏的情况下执行了 n 次。故算法的时间复杂度为 $O(n)$。

3.3 事务日志模块的实现

针对 2.1.2 节中提出的需求，着重阐述读写操作日志的记录方式，分为三个小节加以介绍。第一小节介绍事务日志的存储；第二小节介绍写操作日志的获取；第三小节介绍读操作日志的获取。

3.3.1 事务日志的存储

在版本融合时必须精确得到数据库中所有事务执行的详细日志记录，这就要求日志文件必须记录所有事务操作的“痕迹”。对于写操作，要记录其修改的是哪一列，修改前的值和修改后的值，操作的确切时间等。对于读操作，也需要记录读的是哪一列和操作的确切时间。这是因为读操作可以引起受损数据的扩散，因此，必须得

到事务所有操作的详细记录。

由 2.1.2 节中提出的需求可知，本系统无法运用 DBMS 中的日志记录。这就要求必须自己来构造事务日志，首先要考虑的就是日志文件的存储以及结构问题。类比于传统的 DBMS 中的日志信息，设计了一张表 AUDIT_TABLE 来存放日志信息，该表结构如表 3.2 所示。

表 3.2 表 AUDIT_TABLE 结构

列名	类型
USER_NAME	VARCHAR(50)
DATABASE_NAME	VARCHAR(25)
SCHEMA_NAME	VARCHAR(25)
TABLE_NAME	VARCHAR(25)
COLUMN_NAME	VARCHAR(25)
IDENTITY_ID	INT
TRANSACTION	VARCHAR(25)
OLD_VALUE	VARCHAR(25)
NEW_VALUE	VARCHAR(25)
VALUE_TYPE	VARCHAR(25)
OPER_TYPE	VARCHAR(10)
TIMESTAMP	DATETIME

其中 USER_NAME 为用户名；DATABASE_NAME 为用户操纵的数据库名；SCHEMA_NAME 为用户的模式名；TABLE_NAME 为用户操纵的表名；COLUMN_NAME 为用户操纵的表中的列名；IDENTITY_ID 为用户操纵的表中记录的行号，能够唯一标示某一行；以上这 5 项能够唯一标识数据库中的某一个数据项。TRANSACTION 为用户操纵的事务编号；OLD_VALUE 为该数据项被操作前的值；NEW_VALUE 为该数据库被操作后的值；VALUE_TYPE 为该数据项所在列的数据类型，如 int、varchar 等；OPER_TYPE 为用户操纵的类型，为集合{INSERT,DELETE,UPDATE,SELECT}中的任一元素；TIMESTAMP 为用户操纵数据项的确切时间。

表 AUDIT_TABLE 存放在 DSRP 数据库中，这也是由安全性方面来考虑。事务日志的准确性涉及到事务操作序列之间的严格时序关系以及操作的值的更新变化，

必须确保其安全性。不能让恶意用户篡改日志表中的日志记录，否则会影响最终版本融合时候的正确性。对表 AUDIT_TABLE 限制只能由隔离系统对其进行 SELECT 和 INSERT 操作，不允许做 UPDATE 和 DELETE 操作。不允许普通用户对其进行写操作，只允许普通用户对其进行 SELECT 操作。一旦发现有普通用户对该表进行写操作，立即停止该用户的操作，并将该用户设置为恶意用户。

3.3.2 写操作日志获取

写操作日志的获取主要依靠触发器来实现。触发器分为三种触发器：INSERT 触发器、UPDATE 触发器、DELETE 触发器。触发器在被激活的时候能够生成两张只读的临时表，Inserted 表和 Deleted 表。这两张表里面能够保存更新操作时候数据项的操作前后的值。这点正好满足日志记录中要求的必须精确记录数据项操作前后的值。

因此，在用户要操纵的数据库中的表中都建立三种类型的触发器，用来获取用户写操作的日志。在真实表上建立触发器的另外一个优点是，当可疑用户建立隔离表的时候，它能够获取到真实表中建立触发器的语句，同时在隔离表中也建立相应的触发器。这样的话，可疑用户在隔离表中所进行的写操作的日志记录也会及时保存下来。利用触发器来获取写操作的日志是正确的，这是由触发器的性质[39]来决定的。如果触发器被用户定义的事务激活，但是该事务回滚，则触发器对数据库产生的影响也被撤销。即若用户定义的写操作的事务由于某些原因执行了回滚操作，则该更新操作也不会形成日志记录写入到表 AUDIT_TABLE 中。

3.3.3 读操作日志获取

Liu Peng 等人在文献[26]中采用的是事务模板机制来获取读操作的日志。DAIS 是实现在银行应用系统中，Liu Peng 依据银行系统中操作事务的特征提取了读集模板表[26]（Read Set Template Table），实质上就是事务操作的模板。当获取到用户的读操作语句时，将用户语句与该模板表进行匹配，就可以获取到用户的读操作记录。

这种方法是不具有通用性的，本系统无法得知所有应用系统的操作事务的特征模板。这里借鉴写操作日志的生成方法，在读操作的时候，也生成一张和触发器一

样类似的临时表。待读操作完成后，再将临时表中的记录写入到表 AUDIT_TABLE 中。

采用重构用户读操作语句来实现，将用户的一条读操作语句改写为两条语句。第一条语句是将读操作的记录写入到一个临时表中，第二条语句是用户查询的语句。待用户的查询语句完成后，再将临时表中的记录写入到表 AUDIT_TABLE 中，这两条语句必须是同时执行。在这里，可以采用一些措施来减少写临时表时带来的系统的性能损耗，具体方法可以参见文献[40]。

3.4 版本融合模块的实现

针对 2.1.3 节中提出的需求，着重阐述版本融合时的数据项合并，分为三个小节加以介绍。第一小节介绍版本冲突的检测与处理；第二小节介绍单隔离用户的版本融合实现；第三小节介绍多隔离用户的版本融合实现。

3.4.1 版本冲突检测与处理

在隔离的过程中，本系统遵循的是单边隔离的策略，即数据项的真实值和可疑值分别在正常事务和可疑事务中独立的执行。这种情况可能导致的后果是真实版本的数据库和隔离版本的数据库的状态不一致。这也是版本融合中的难点所在。

采用构造前向图的方式来判断数据一致性问题。假定可疑用户对数据库操纵的历史为 H_s，系统中除去可疑用户之外的用户对数据库操纵的历史为 H_e，将前向图记为 $G(H_e,H_s)$，则前向图按照以下准则来构造。

（1） T_i 和 T_j 同为可疑事务或者同为正常事务，他们同时对一数据项进行操作产生冲突，即至少有一个事务对该数据项进行了写操作。如果 T_i 在 T_j 之前执行，则构造有向边 $T_i \rightarrow T_j$

（2） 如果可疑事务 T_s 被隔离的时候读取了正常事务 T_g 的更新操作，则构造有向边 $T_g \rightarrow T_s$。这条有向边被称为读边（read edge），加入读边的目的是为了支持单边隔离策略

（3） 如果正常事务 T_g 读取了被可疑事务 T_s 更新过的数据项，而且 T_g 到 T_s 在当前的图中没有路径可达，则构造有向边 $T_g \rightarrow T_s$

（4） 如果可疑事务T_s读取了被正常事务T_g更新过的数据项，而且T_g到T_s没有包括读边在内的路径，则构造有向边$T_s \rightarrow T_g$

依据以上准则构造好前向图之后，可以通过判断前向图中是否存在环来判断数据一致性是否产生冲突。若有向图中存在环，则说明数据一致性存在冲突，此时必须回退一些事务来打断环。要找到环中回退某个事务对整个系统的性能影响最小是很困难的，只能通过随机地选取环中某个事务进行回退，若回退的是正常事务T_g，则除了回退T_g之外，还要回退从前向图中以该事务为起点的读边$T_g \rightarrow T_s$中可疑事务的T_s和受T_s感染的所有事务。

至此，已经消除了单个可疑用户在数据库中的不一致情况，下一步可以对单个可疑用户进行版本融合处理。

3.4.2 单用户隔离时版本融合

首先考虑简单情况，即系统中只有一个可疑用户被隔离的情况。

当最终可疑用户被确认是恶意用户时，首先将该用户维护的可疑版本的数据库删除，再从事务日志中找出可疑用户初次进入数据库的时间t_1和该用户被确认为可疑的时间t_2，采用二阶段来对数据库进行恢复操作。第一阶段正向扫描t_1到t_2时段事务日志文件，采用读写依赖的方式来得到所有被恶意事务更新过的数据项（脏数据）和被恶意用户感染的正常事务更新过的数据项（受感染数据），并且得到事务之间的严格的执行序列；第二阶段反向扫描t_2到t_1时段的事务日志，依据前面的事务操作序列和待恢复的数据项构造 undo 事务提交给后台数据库系统执行。即可使数据库恢复到一个正确的状态。

当最终用户被确认是正常用户时，在数据一致性的前提下，要将该用户维护的可疑版本的数据库融合到真实版本的数据库中，事务日志模块中得到事务各个操作确切的时戳值，依据该时戳值去判定版本值之间的时序关系。将该数据项的真实值替换为最近正常事务修改的值。算法 3.7 给出了单用户隔离时版本融合的实现。

算法 3.7：单用户隔离时版本融合实现算法

输入：isolaTableSets，可疑用户维护的隔离表

AUDIT_TABLE，事务日志

输出：数据库的正确状态

方法：

```
1   For i=1 to isolaTableSets.size() do
2       isolaTable= isolaTableSets.get(i)          //获取该可疑用户维护的每个隔离表
3       get_all_records(isolaTable) to recordSets //获取表 isolaTable 中所有的记录给 recordSets
4       serach_real_table(isolaTable) to table
                  //查询表 ISOLA_TABLE_INFO 获取隔离表 isolaTable 对应的真实表给 table
5       For j=1 to recordSets.size() do
6           record= recordSets.get(j)              //获取记录集中的每条记录
7           If(table 存在与 record 有相同主键的记录)
8             get_real_record(table, record) to real_record
               //从真实表 table 中得到与 record 有相同主键值的记录给 real_record
9             For k=1 to record.size() do
10              item=record.get(k)                 //对于记录 record 中的每个数据项 item
11              If(real_record 中对应的数据项的值和 item 不同)
12                查询事务日志 AUDIT_TABLE 得到 item 最迟被修改的值 item_late
13              If(item_late 为空)                 //没有查询到该数据项的操作日志
14                item= real_record.get(k)         //取 item 为真实表中的值
15              Endif
16                Else item=item_late              //将查询到的值赋值给 item
17              Endif
18              Else item=real_record.get(k)       //取 item 为真实表中的值
19            Endfor
20            将组合好后的记录插入到真实表 real_record 中
21          Endif
22          insert record to table                 //将该记录插入到真实表 table 中
26      Endfor
27      将表 isolaTable 从系统中删除，并删除在 ISOLA_TABLE_INFO 中的记录值
```

28 Endfor

假定该可疑用户维护的隔离表的数量是m，即 isolaTableSets 的容量是m。各个隔离表中记录集数目最多的是n，每条记录中含有的属性列的数目最多的是k，则在最坏的情况下，算法 3.7 要执行的次数是$m\times n\times k$次，故算法的时间复杂度为$O(m\times n\times k)$。但是在实际的情况中，隔离表中存放的记录数目一般是不大的，因为里面存放的都是可疑用户对数据项的更新操作。在这种前提下，算法的时间复杂度为$O(m\times k)$。

版本融合实现中对于隔离表中每条记录，首先查询相应的真实表中是否有主键值相同的记录，若没有则直接将该记录插入到真实表中。若真实表中有相同主键值的记录，将真实表中的记录和隔离表中的记录中的每个数据项的值分别比较，首先判断隔离表中的值是否为空，若为空则直接将真实表的值赋值给数据项，再去比较二者值是否相同，若相同则取真实表中的值，若不相同则去查找事务日志中该数据项最迟被修改的值，将其赋值给数据项，最后将该条记录插入到真实表中。在版本融合正在进行的时候，当将某一个隔离表 isolaTable 中的记录融合到与之对应的真实表 table 中，此时必须将真实表中的正在被融合的记录锁定。采取这样的措施是为了保证融合的正确性。

3.4.3 多用户隔离时版本融合

当系统中同时有出现多个可疑用户时，此时的情况要比单个可疑用户要复杂一些。主要表现在两个方面：（1）、不同的可疑用户是在不同的时刻被发现和被隔离。（2）、在将一个可疑用户的隔离版本融入到真实版本中的时候，会对其它正在被隔离的用户产生影响，所以在版本融合的时候必须与其它正在被隔离的用户保持同步，即并发性问题。

对于第一方面的问题，不同的可疑用户在不同的时刻被发现和隔离，而在这些不同的时刻，真实版本的数据库值可能会随着正常用户的访问而发生改变。采用的方法是将所有可疑用户构造隔离版本数据库时所依据的真实版本的数据库的初始状态设定为一致。即数据库中所有可疑用户构造隔离版本时对应的真实版本的状态都是以系统中第一个可疑用户被发现时构造隔离版本时所依据的真实版本的值。使得数据库中所有隔离版本所对应的真实版本都有一个相同的初始状态值。这样做的目

的是为了减少数据不一致性发生的概率，同时可以使得各个可疑用户在将自己维护的隔离版本融合到真实版本的时候能够确保融合的正确性。

对于第二个方面的问题，当一个可疑用户的隔离版本被融合到真实版本的时候，必须采取措施来保证对其他的可疑用户的操作不产生影响。通常当一个可疑用户的隔离版本（即操作的历史）融合到真实版本中的时候，会在系统维护的前向图中产生环，即会发生数据不一致性的情况，这个时候就要采取 3.4.1 中的措施来打断环来使数据保持一致性。只有在保持系统维护的前向图中不存在环的前提下，多个可疑用户才能正确有序地将自己维护的可疑版本的数据库融合到真实版本的数据库中。

在前面两个问题得到解决后，可疑用户融合时都可以采用 3.4.2 中的方法来使得数据库在融合后保持数据的正确性。

3.5 小结

本章主要介绍了入侵容忍的数据库中数据隔离子系统的实现，主要工作包括。

1、介绍了隔离模块的实现，首先介绍了隔离表的建造及维护，给出了实现的算法、存储及安全性分析；接着介绍了隔离版本的控制，从隔离策略的选取、版本控制的实现及分析来分别加以介绍。

2、介绍了代理模块的实现，给出了 4 种基本语句的代理实现算法及性能分析。

3、介绍了事务日志模块的实现，分别从事务日志的存储方式及安全性、写操作日志的获取、读操作日志的获取三个方面来介绍。

4、介绍了版本融合模块的实现，分别从版本冲突检测、单用户版本融合、多用户版本融合三个方面来加以介绍。

4 入侵容忍的数据库中数据隔离系统测试与分析

为了验证 2.1 节中提出的需求是否被所设计系统满足，本章主要设计两个方面的实验来验证。首先是隔离系统的正确性实验，接着是隔离子系统对整个 DSRP 系统性能的影响实验。

隔离系统是在 windows 平台下面采用 java 语言编写完成的，采用的工具有 Eclipse 3.3 和 JDK 6.0。

4.1 隔离子系统正确性实验

1、实验环境

为了测试隔离系统隔离版本生成的正确性和版本融合后数据的正确性，在此自行设计了数据库 testDB，主要设计了库中的两张表，orders 表和 order_details 表，分别是订单表和订单详细表。由于表中的数据与要测试的隔离的正确性没有太大关系，在此就不介绍表的详细结构。

2、正确性实验步骤及结果分析

（1）首先设定一个用户为可疑用户，假定为 William，构造一个事务 T。

```
INSERT INTO Orders VALUES(20,'g',2,3,4,'g','g','g','g','g','g') ;
UPDATE Orders SET shipcountry='Russian' WHERE OrderID=10538;
INSERT INTO Order_Details VALUES (20000,3698,2.569,300,0.85);
UPDATE Order_Details SET Quantity=999 WHERE OrderID=10252;
```

在执行事务 T 之前，可以查询表 ISOLA_TABLE_INFO，看此时是否存在相应的隔离表，结果如图 4.1 所示。

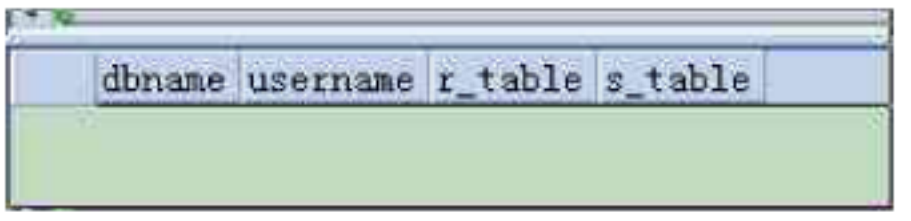

图 4.1 表 ISOLA_TABLE_INFO 内容示意图

可以看到，此时库中是不存在任何的隔离表的，但是在执行 William 的事物 T

之后，再次来查询表 ISOLA_TABLE_INFO，可以看到已经构造了相应的隔离表，结果如图 4.2 所示。

	dbname	username	r_table	s_table
1	testDB	William	order_details	order_details_20090512
2	testDB	William	orders	orders_20090512

图 4.2 表 ISOLA_TABLE_INFO 内容示意图

（2） 此时已经构造了隔离表，则用户 William 事务 T 中的更新操作都已经写入到了相应的隔离表中，在最终确认该用户是正常用户做版本融合之前，先执行下面的查询操作，看相应的真实表中是否有相同的记录存在。执行的 SQL 语句如下：

```
SELECT * FROM [testDB].[dbo].[Orders] where OrderID=20
SELECT * FROM [testDB].[dbo].[Orders] where OrderID=10538
SELECT * FROM [testDB].[dbo].[Order_Details] where OrderID=20000
SELECT * FROM [testDB].[dbo].[Order_Details] where OrderID=10252
```

执行后得到的结果如图 4.3 所示。

	OrderID	CustomerID	EmployeeID	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCode	ShipCountry	Orders_sys_id
1	10538	BSBEV	9	3	4.8700	Bs Beverages	Fauntleroy Circus	London	NULL	EC2 5NT	China	291

	OrderID	ProductID	UnitPrice	Quantity	Discount	Order_Details_sys_id
1	10252	20	64.8000	40	5.000000000000003E-2	17
2	10252	33	2.0000	25	5.000000000000003E-2	18
3	10252	60	27.2000	40	0.0	19

图 4.3 查询 SQL 语句内容示意图

从上面的结果可以看到，隔离用户对数据库的操作没有写入到真实表中，数据都存放在相应的隔离表中，下面是查询 orders 对应的隔离表和 order_details 对应的隔离表所得到的结果，得到的结果如图 4.4 所示。

	OrderID	CustomerID	EmployeeID	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCode	ShipCountry	Orders_sys_id
1	20	g	2	3	4.0000	g	g	g	g	g	g	1
2	10538	BSBEV	9	3	4.8700	Bs Beverages	Fauntleroy Circus	London	NULL	EC2 5NT	russian	291

	OrderID	ProductID	UnitPrice	Quantity	Discount	Order_Details_sys_id
1	10252	20	64.8000	688	5.000000000000003E-2	17
2	10252	33	2.0000	688	5.000000000000003E-2	18
3	10252	60	27.2000	688	0.0	19
4	20000	3698	2.5690	300	0.84999999999999998	1

图 4.4 查询隔离表内容示意图

（3） 假定最终确认用户 William 是正常用户，则要将事务 T 的操作融合到真实版本中，融合后，再次查询（2）中的语句，得到的结果如图 4.5 所示。

	OrderID	CustomerID	EmployeeID	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCode	ShipCountry	Orders_sys_id
1	20	g	2	3	4.0000	g	g	g	g	g	g	1

	OrderID	CustomerID	EmployeeID	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCode	ShipCountry	Orders_sys_id
1	10538	BSBEV	9	3	4.8700	Bs Beverages	Fauntleroy Circus	London	NULL	EC2 5NT	russian	291

	OrderID	ProductID	UnitPrice	Quantity	Discount	Order_Details_sys_id
1	20000	3698	2.5690	300	0.850000023841857 91	1

	OrderID	ProductID	UnitPrice	Quantity	Discount	Order_Details_sys_id
1	10252	20	64.8000	688	0.050000000745058 06	17
2	10252	33	2.0000	688	0.050000000745058 06	18
3	10252	60	27.2000	688	0.0	19

图 4.5 查询 SQL 语句内容示意图

由表中的数据可以看到，事务 T 对数据库的操作已经融合到了真实表中，可以得出结论：该隔离系统能够正常隔离用户并能保证版本融合后的数据的正确性。

4.2 隔离子系统对DSRP系统性能实验

本实验主要测试隔离子系统对整个 DSRP 系统性能的影响，测试工具采用 TPC-W 和 DSRP 系统以及 SQL SERVER 2000。

1、测试环境

该实验在TPC-W环境下进行，TPC-W是TPC委员会2000年发布的一个电子商务应用基准[41,42]。TPC-W测试平台由中间系统、后台数据库服务器、web服务器和客户端四部分构成。主要是通过测量单位时间内所处理的WEB交互数量（Web Interaction Per Second,WIPS），来衡量数据库网络事务处理能力，从而衡量数据库系统的性能。TPC-W平台制定3种不同的交易模式。（1） WIPSo模式，读事务和写事务各占50%；（2） WIPS模式，读事务和写事务各占的比例为80%和20%；（3）、WIPSb模式，读事务和写事务各占的比例为95%和5%。

在本实验中，选择 SQL SERVER 2000 作为后台数据库服务器，web 服务器为 weblogic8.1，rbe 客户端远程模拟浏览器，用户通过该浏览器进行网上交易。

2、实验步骤及结果分析

本实验主要测试TPC-W在三种不同的模式下测试隔离子系统对DSRP系统的性能的影响。

图 4.6 到图 4.8 给出了实验的数据。

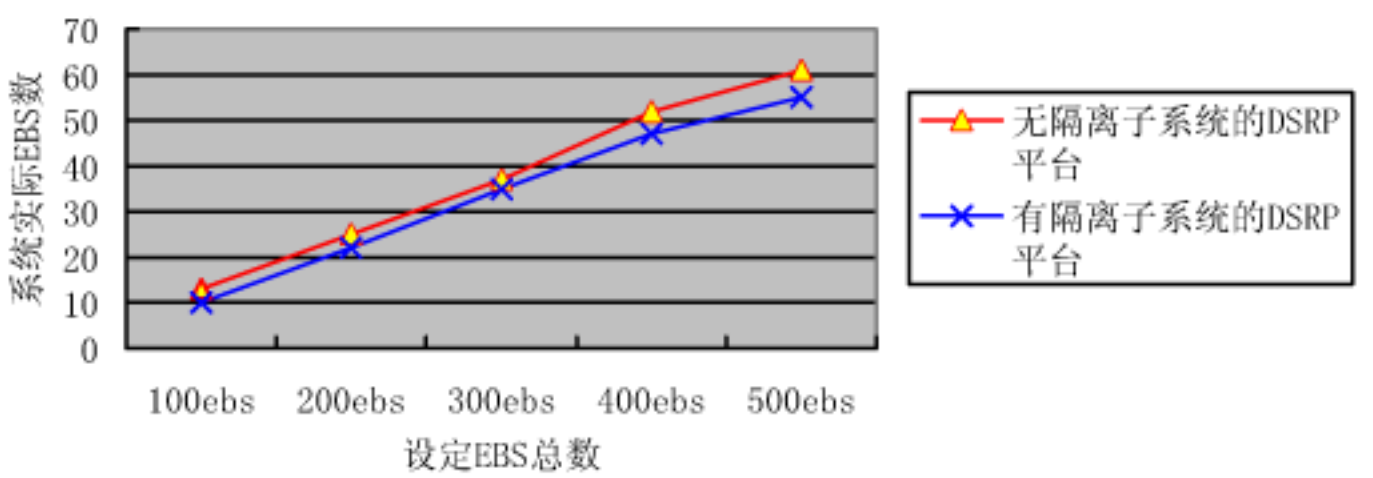

图 4.6 wips 模式下系统性能对比图

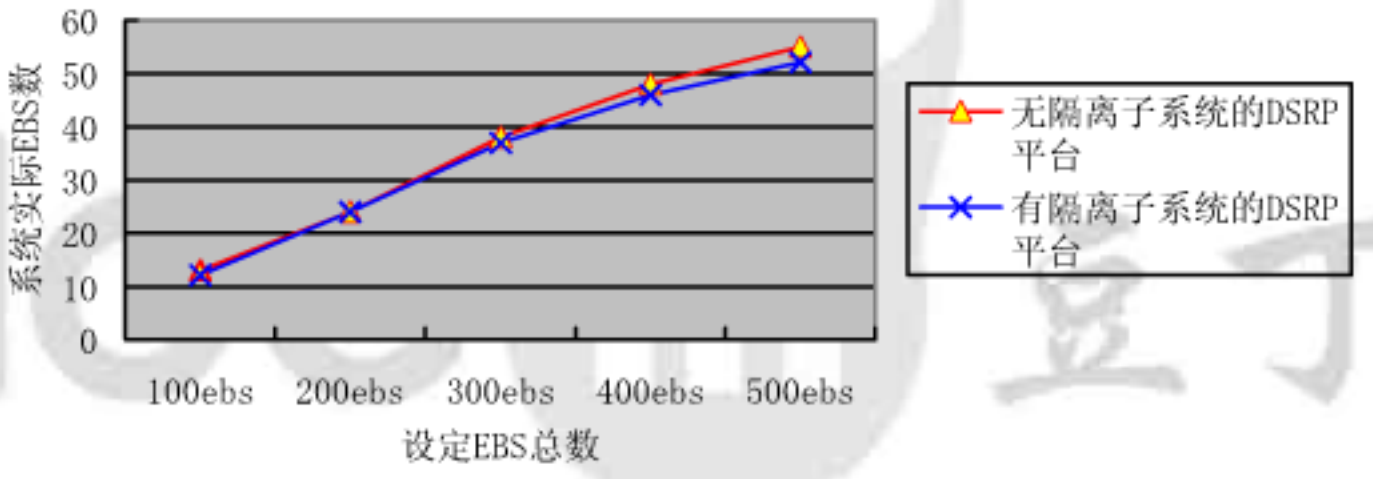

图 4.7 wipsb 模式下系统性能对比图

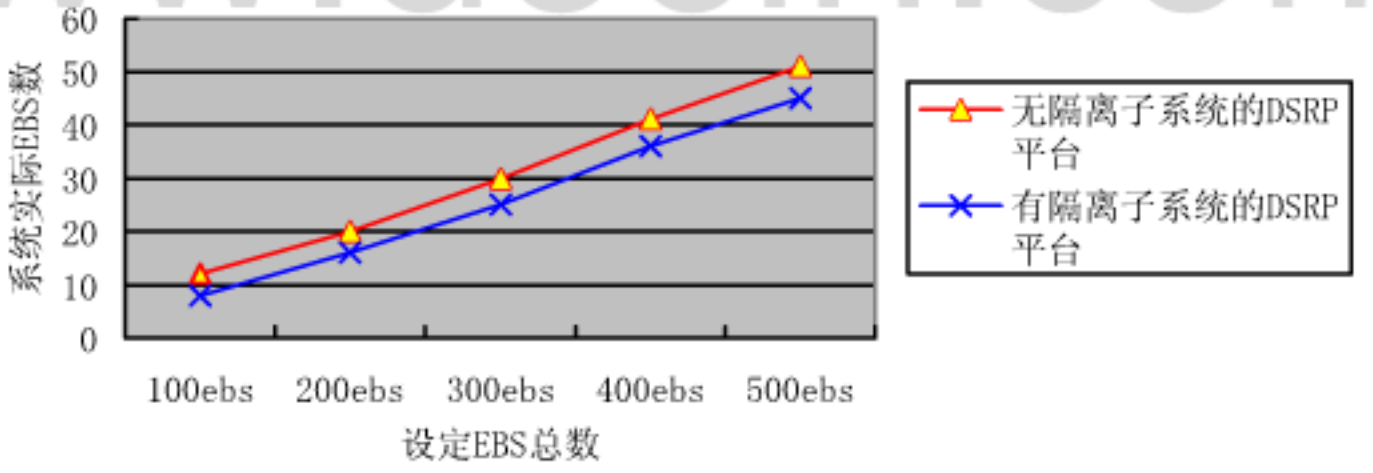

图 4.8 wipso 模式下系统性能对比图

从图 4.6，图 4.7，图 4.8 三张图中的数据可以看出。

（1） 整体上看，三种模式下面，隔离子系统对整个 DSRP 系统性能的影响是不大的。

（2） 从图中对比来看，三种模式下面，wipso 模式下面对系统的影响是最大

的。这是因为三种模式中，wipso 模式中写事务的比例是最大的。写事务过多就必然会导致创建更多隔离版本的数据库，以及要创建多个用于维护数据一致性的前向图。这些都会导致更多的资源被消耗，同时也会影响系统的性能。

4.3 小结

本章主要介绍了入侵容忍的数据库中隔离系统的正确性及性能实验。实验分为三个部分，首先是测试了可疑用户隔离版本构造和版本融合的正确性，接着是测试了隔离系统对整个 DSRP 系统性能的影响。

从实现的结果可以看出，隔离版本的构造和融合是正确的。同时隔离系统对整个 DSRP 系统性能的影响是不大的，基本上达到了 2.1.4 中的性能需求。

5 总结和展望

5.1 全文总结

随着信息技术的发展，信息的安全性越来越受到人们的关注。数据库作为信息存储的载体，其安全性更应当受到关注。在面对黑客攻击和恶意用户对数据库造成的损伤面前，传统的数据隔离技术显得无能为力。为此，在入侵检测技术的基础上，提出了数据隔离的思想。在充分分析了隔离技术发展的状况以及存在的问题后，提出了运行在通用数据库平台上面的数据隔离系统体系结构，进而实现了该系统。本文的主要工作可以归纳为以下三个方面。

1、设计了入侵容忍的数据库中数据隔离系统

从入侵容忍的数据库中数据隔离的需求出发，主要是隔离版本需求、事务日志需求、版本融合需求以及系统性能需求四个方面来阐述数据隔离系统应该要达到的需求。接着依据需求，提出了数据隔离的思路。该思路给出了数据隔离的基本单位、隔离版本的建立、隔离版本的维护以及隔离版本与真实版本之间的信息流动。最后基于该思路给出了数据隔离的体系结构。体系结构包括隔离控制模块、代理模块、事务日志模块以及版本融合模块四个模块。

2、实现了入侵容忍的数据库中数据隔离系统

依据体系结构中的四个模块，对于每个模块给出了详细的算法流程描述以及实现和性能的分析。

（1） 隔离控制模块。阐述了隔离表构造的时机、流程、实现细节，阐述了以用户为单位来对隔离表进行维护的原因，分析了隔离版本的安全性以及采取的措施，提出了隔离版本控制时隔离策略准则，给出了版本控制实现的算法以及实例分析，并对性能做了优化。

（2） 代理模块。即将可疑用户的 SQL 语句的写操作全部要“改写”成读操作的形式。用读操作获取相应的记录后，再对这些记录进行处理，将更新后的数据项的值写入到相应的隔离表中。介绍了四种典型语句（SELECT、UPDATE、INSERT、DELETE）的详细代理实现过程，给出了详细算法并加以分析。

（3） 事务日志模块。要获取数据库中所有的事务操作日志对版本融合提供数据支持。先介绍了事务日志的存储，构造了事务日志存储的日志表并对其安全性加以分析，给出了安全措施。接着介绍了写日志操作的获取，主要是采用触发器。最后介绍了读操作日志的获取。

（4） 版本融合模块。版本融合主要是将最终被确定为正常用户的可疑用户所维护的隔离版本的数据库融合到真实版本的数据库中。先介绍了数据不一致性的判断以及解决办法。在数据一致性的前提下，先介绍了单隔离用户版本融合的实现，给出了算法描述和分析，最后介绍了多隔离用户时的版本融合的实现。

3、测试分析了系统性能

设计了两组实验。第一组实验是验证隔离版本构造的正确性以及版本融合的正确性，第二组实验是验证隔离系统对整个 DSRP 系统性能的影响。

5.2 展望

为了继续完善数据隔离系统，本课题还需要在多个方面进行进一步研究。

首先，隔离版本数据库以及事务日志的安全性还有待进一步加强。本文中只是给出了一些基本的措施，需要研究更多的安全措施来保证其安全性。

其次，隔离版本控制只是给出了实现的算法，其正确性证明并没有给出。这在以后的研究中有待进一步给出证明过程。

再次，事务日志模块中读操作的获取是借鉴了写操作的思想，这种方法在性能上有很大的损失。因此，如何找出一个高效且能正确获取事务读操作的算法是今后工作的重点。

最后，版本融合中多用户隔离版本融合时如何确保融合的正确性是今后研究的难点，本文中只给出了简单的情况。

由于作者水平有限，本文中存在错误在所难免。敬请各位老师和同学批评指正。

致谢

回顾两年的硕士学习生活，我深深地体会到自己取得的每一份成绩都离不开许多关心、帮助我的人。正是得到他们无私的帮助，我的各项工作才得以顺利完成。在此，我谨向他们表示最真诚、最衷心的感谢！

首先，我要感谢我的导师王元珍教授。感谢王老师在这两年的时间里，在科研工作上对我的启发和引导，在学习上对我的悉心指导，在生活上对我的亲切关怀。她渊博的知识、严谨的治学态度让我受益无穷。从论文的选题到修改、定稿都给予了我悉心的教导和耐心的帮助，并提出了许多宝贵的意见，值此论文完成之际，谨向恩师表示深深的谢意和良好的祝愿！

特别要感谢我的课题指导老师朱虹教授。感谢朱老师给了我一个参与课题项目的机会，让我从项目中学到了很多东西。朱老师对我课题研究给予了很好的引导和帮助，使我能顺利完成课题研究工作。在本文的撰写过程中，朱老师也花费了大量的心血来为我修改论文，并给出了很多中肯的意见。在此，向朱老师表达深深的谢意！

感谢冯玉才教授、曹忠升副教授、吴永英副教授、许向阳副教授、吴恒山副教授在这两年对我的帮助。从他们身上，我学会了很多知识和经验。

衷心感谢我的家人给我照顾和关爱，感谢我的女朋友潘福霞，在我最困难的时候给我的支持和在生活上对我的照顾。

感谢课题组其他同学对我的帮助和照顾。

感谢各位专家和评委的耐心审阅，他们提出了很多宝贵的意见和建议。

参考文献

[1] 罗华钧, 孙长军. 现代数据库安全概论. 微机发展, 2002, 12 (4): 633~637

[2] 郭渊博, 马建峰. 容忍入侵的国内外研究现状及所存在的问题分析. 信息安全与通信保密, 2005, 32 (7): 337~341

[3] 刘大勇, 张玉清. 事务级数据库入侵检测系统的设计. 中国农业大学学报, 2006, 11 (4): 109~113

[4] 蔡亮, 杨小虎, 董金祥. 信息战下的数据库安全——我国的特殊需求分析和对策. 计算机研究与发展, 2002, 39(5): 568~573

[5] P.Ammann, S.Jajodia, P.Liu. Recovery from Malicious Transactions. IEEE Transactions on Knowledge and Data Engineering, 2002, 14 (5): 1167~1185

[6] P.Liu, H.Wang, L.Liu. Real-time data attack isolation for commercial database applications. Journal of Network and Computer Applications, 2006, 20(4): 164~189

[7] R. Graubart, L. Schlipper, C. McCollum. Defending Database Management Systems Against Information Warfare Attacks. Technical report, The MITRE Corporation, 1996. 69~84

[8] Y. Zhong, X. L. Qin. Database Intrusion Containment and Recovery Techniques under Information Warfare. Chinese Jouinal of Computers, 2005, 32(7): 68~73

[9] 钱雪忠, 黄向前. 基于SQL Server2000 的实用并发控制技术. 江南大学学报, 2003, 2 (3): 23~28

[10]K. Bai, P. Liu. A Fine-grained Damage Management Scheme in a Self-Healing PostgreSQL System. in: 2008 11th IEEE High Assurance Systems Engineering Symposium. Nanjing, China. 2008. Beijing: ACM Press, 2008. 373~382

[11]P. A. Bemstein, V. Hadzilacos, N. Goodman. Concurrency Control and Recovery in Database Systems. First Edition. Menlo Park :Addison-Wesley, 1987. 288~301

[12]钟勇，秦小麟. 信息战条件下的数据库入侵限制和恢复技术. 计算机科学，2005, 32 (7): 78~82

[13]Fraga J, Powell D. A Fault- and Intrusion-Tolerant File System. in:Proceedings of the 3rd International Conference on Computer Security, IFIP/SEC85. Dublin, Ireland, 1985. Ireland: ACM Press, 1985. 203~218

[14]P. Liu, S. Jajodia, and C. D. McCollum. Intrusion Confinement by Isolation in Information Systems. Journal of Computer Security, 2000, 90(15): 67~85

[15]K. Bai, M. Yu, P. Liu. TRACE:Zero-Down-Time Database Damage Tracking, Quarantine,and Cleansing with Negligible Run-Time Overhead.Computer Science, 2008, 25 (8): 161~176

[16]P. Ammann, S. Jajodia, C. D. McCollumet. Surviving information warfare attacks on databases. in: Proceedings of the IEEE Symposium on Security and Privacy. New York, USA. 2000. New York:ACM Press, 2000. 1~12

[17]J. McDermott, D. Goldschlag. Towards a model of storage jamming. in: Proceedings of the IEEE Computer Security Foundations Workshop. Kenmare, Ireland, 1996. Kenmare: ACM Press, 1996. 176~185

[18]C. Pu. On-the-fly,incremental,consistent reading of entire databases. Algorithmica, 1986, l0(3): 271~287

[19]P. Liu and S. Jajodia. Multi-Phase Damage Confinement in Database Systems for Intrusion Tolerance. in: Proceedings of the 14th IEEE Computer Security Foundations Workshop. Washington DC, USA: IEEE Computer Society Press, 2001. 98~120

[20]J. Lin, J. Jing, P. Liu. Framework for Intrusion Tolerant Certification Authority System Evaluation. In: International Symposium on Reliable Distributed Systems. Brijing, China, 2007. Beijing: IEEE Computer Society Press, 2007. 231~241

[21]Jajodia S, Ammann P, McCollum C D. Surviving information warfare attacks. IEEE Computer Society, 1999,32(4): 57~63

[22]Tripathy S, Panda B. Post-Intrusion Recovery Using Data Dependency Approach. in: Proc. of the 2001 IEEE Workshop on Information Assurance and Security. United States Military Academy, West Point, New York ,USA,2001.USA: IEEE Computer Society Press, 2001. 1131~1138

[23]K. G. Popstojanova, F. Wang, R. Wang et al, Characterizing intrusion tolerant systems using a state transition model, in: DARPA Information Survivability Conference and Exposition, Washington DC, USA,2001. Washington DC: IEEE Computer Society Press, 2001. 211~221

[24]P. Liu, J. Jing. Architectures for Self-healing Databases under Cyber Attacks, International Journal of Computer Science and Network Security, 2006, 6(1): 204~216

[25]Liu Peng. Architectures for Intrusion Tolerant Database Systems. In: Proc. of 18th Annual Computer Security Applications Conf. Las Vegas, Nevada, 2002. Las Vegas: ACM Press, 2002. 301~310

[26]Liu Peng. DAIS: A Real-Time Data Attack Isolation System for Commercial Database Applications. in: 17th Annual Computer Security Applications Conference. New Orleans, 2001. USA: ACM Press, 2001. 1~11

[27]Fayad A, Jajodia S, McCollum C D. Application-level isolation using data inconsistency detection. In: Proc. 15th Annual Computer Security Applications Conf. Phoenix, AZ, 1999. USA: ACM Press, 1999. 119~126

[28]Alexey Smirnov, Tzi cker Chiueh. A Portable Implementation Framework for Intrusion Resilient Database Management Systems. in:2004 International Conference on Dependable Systems and Networks. Florence,Italy. 2004. Italy: ACM Press, 2004. 543~558

[29]H. Wang, P. Liu, and L. Li, Evaluating the Survivability of Intrusion Tolerant Database Systems and the Impact of Intrusion Detection Deficiencies, International Journal of Information and Computer Security, 2007, 1 (3): 315~340

[30]Meng Yu, Wanyu Zang, Peng Liu: Database Isolation and Filtering against Data Corruption Attacks. in: Proc. 23rd Annual Computer Security Application Conference. New York, USA, 2007. USA: ACM Press, 2007. 97~106

[31]Panda B, Giordano J. Reconstructing the Database after Electronic Attacks. in: S.Jajodia. Database Security XII: Status and Prospect, USA: Kluwer Academic Publishers, 1999. 143~156

[32]Ammann P, Jajodia S, Mavuluri P. On-the-fly reading of entire databases. IEEE Trans. on Knowledge and Data Engineering,1995,7 (5): 834~838

[33]Pramote Luenam, Peng Liu. ODAR: An On-the-fly Damage Assessment and Repair System for Commercial Database Applications. in: Proc. 15th IFIP WG 11.3 Working Conference on Data and Application Security. New York, USA, 2001. USA: ACM Press, 2001. 239~252

[34]S. Patnaik and B. Panda. Dependency Based Logging for Database Survivability from Hostile Transactions. in: Proceedings of the 12th International Conference on Computer Applications in Industry and Engineering. New York, USA, 1999. New York: ACM Press, 1999. 169~202

[35]P. Ragothaman, B. Panda. Hybrid Log Segmentation for Assured Damage Assessment. ACM Transactions on Database System(TODS), 2004, 29(2): 363~402

[36]P. Liu. Trusted Recovery from Malicious Attacks: [Ph.D.Dissertation]. Virginia : George Masson University, 1999

[37]Sani T, Brajendra P. Post-Intrusion Recovery Using Data Dependency Approach. in: Proceedings of the 2001 IEEE Workshop on Information Assurance and Security. Piscataway, NJ, 2001. USA: IEEE Computer Society Press, 2001.18~28

[38]B. Panda, S. Tripathy. Data Dependency Based Logging for Defensive Information Warfare. in: Carrol, J. Symposium on Applied Computing Proceedings of the 2000 ACM Symposium on Applied Computing. New York: ACM Press, 2000. 361~365

[39]刘敏贤，汤娟. SQL Server 数据库应用系统中数据完整性的设计与实现. 微机发展, 2002, 12(4): 50~53

[40]陈敏. 抗数据库恶意事务的日志系统研究：[硕士学位论文]。武汉：华中科技大学图书馆，2006

[41]D. F. Garcia, J. Garcia. TPC-W e-commerce benchmark evaluation. Computer, 2003, 35(2): 42~48

[42]D. A. Menasce. TPC-W: a benchmark for e-commerce. Internet Computing, IEEE, 2002, 6(3): 83~87

入侵容忍的数据库中数据隔离技术研究

作者： 华学勤
学位授予单位： 华中科技大学