实现数据库细粒度访问控制的一种动态查询修改算法

时 杰 朱 虹 冯玉才
(华中科技大学计算机学院 武汉 430074)

摘 要 由于隐私保护和基于Web的安全需求的驱动,数据库细粒度访问控制引起了研究者的广泛关注。细粒度访问控制主要通过查询改写方法实现。然而,在以往的查询改写算法中,没有考虑用户提交的SQL语句的特性和细粒度访问控制策略的特性,从而导致最终执行的SQL中存在大量的冗余,影响了SQL语句执行的性能。在考虑SQL语句和细粒度访问控制策略的特性的前提下,分析了两类冗余,并给出了相应的移除方法。最终提出了一种用于细粒度访问控制实现的查询改写算法。实验证实该算法具有更好的性能。

关键词 关系数据库,细粒度访问控制,查询改写,冗余
中图法分类号 TP309 **文献标识码** A

Dynamical Query Modification Algorithm for Fine-grained Access Control in Databases

SHI Jie ZHU Hong FENG Yu-cai
(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract Fine-grained access control has received much attention from research community due to the requirements of privacy preserving and Web-based applications. It is a promising approach to implement fine-grained access control by query modification. However, in the areing query modification algorithm, the features of the queries issued by users and the features of FGAC policy are not considered. Thus, there are redundancies in the final executed queries which make unnecessary overhead. We first analyzed two different redundancies bases on the feature of queries issued by users and the FGAC policies. Then, we proposed a technique to detect these redundancies and provided a new algorithm to implement FGAC. A comprehensive set of experiments show that the performance is improved by the proposed algorithm.

Keywords Relational database, Fine-grained access control, Query modification, Redundancy

1 引言

数据库细粒度访问控制对用户关于表和视图的访问进行控制,其粒度可以细化到基表和视图中的行、列以及某个元素。近年来,由于隐私保护及基于Web应用程序的安全需求,数据库细粒度访问控制受到了工业界和学术界的广泛关注[1-13]。数据库中传统的访问控制仅支持表级的访问控制,因此在传统应用中,细粒度访问控制主要在应用层实现。然而,该实现方法具有很多缺点[6-8]:应用层实现的细粒度访问控制容易被绕过;增加了对安全策略的管理难度等。因此,迫切需要数据库级的细粒度访问控制机制。

数据库级的细粒度访问控制在近几年内得到了一定的研究,例如,INGRES中的查询改写机制[9]、Oracle中的虚拟隐私数据库机制[10]和新近兴起的Hippocratic数据库[1,2,11]。在这些研究中,数据库级细粒度访问控制机制主要通过查询改写方法实现。查询改写的基本思想是指对用户提交的SQL语句进行动态的、透明的修改,并最终执行修改后的SQL语句,同时要确保修改后的SQL语句无法访问不允许用户访问的信息。

Le Fevre等在文献[11]中提出了一种动态查询改写算法,用于实现元素级的细粒度访问控制。在该算法中,用户提交的SQL语句中涉及到的每个表都被一个新构造的视图所替代。在替代的视图中,通过使用CASE语句,所有不允许用户访问的数据都被"NULL"所替代。然而,该查询改写算法对不同SQL语句中涉及到的相同的基表都采用同样的视图进行替代,没有考虑不同SQL语句的特性和细粒度访问控制策略的特性,影响了SQL语句的执行性能。本文针对上述问题,结合不同SQL语句的特性及细粒度访问控制策略的特性,对上述查询改写算法进行改进,从而提高了SQL语句的执行性能,并通过实验验证了修改后的算法具有更好的性能。

2 基本概念

为了便于下面的讨论,本节首先给出一种元素级细粒度访问控制策略的描述方式。然后,简要介绍文献[11]中的元素级访问控制的实现算法。

定义1(元素级细粒度访问控制策略) 元素级细粒度访问控制策略表示为 $p=(s,(o,f_o),a)$,其中

· s 表示主体,为数据库中的用户或者角色;

到稿日期:2010-01-20 返修日期:2010-03-31 本文受863国家高技术研究发展计划(2006AA01Z430)资助。
时 杰(1983—),男,博士生,主要研究方向为数据库安全等,E-mail:shijie1123@gmail.com;朱 虹(1965—),女,教授,主要研究方向为数据库安全等;冯玉才(1946—),男,教授,主要研究方向为现代数据库等。

• o 表示客体，为数据库中的基表或者视图；

• f_o 为策略函数，以 o 的某个属性为输入参数，并返回某个过滤器(谓词)；

• a 表示访问操作，可以为 *select*，*update*，*insert* 或者 *delete*。

过滤器为数据库中的谓词，用于判定某个元素是否允许被访问。

为了讨论的方便性，下面对细粒度访问控制策略 $p=(s,(o,f_o),a)$，使用 $user(p)$，$object(p)$，$F(p)$，$action(p)$ 分别表示 p 中的主体、客体、策略函数和操作。

与以往的研究保持一致，本文主要讨论 *select* 操作。

例 1　假设存在基表 *employee*(*emp_id*，*emp_name*，*sex*，*dept_id*，*addr*，*phone*)。各属性分别表示员工 *ID*、姓名、性别、部门 *ID*、地址和电话。安全需求要求每个员工都能够查看所有员工的 *emp_id* 信息，但只能查看其所在部门员工的 *emp_name*，*sex* 和 *dep_id* 信息，且只能查看自己的 *addr* 和 *phone* 信息。根据该安全需求，得到细粒度访问控制策略为 $P_1=(Emp,(employee,f_{employee}),select)$，其中 $f_{employee}(emp\_id)=$ TRUE，$f_{employee}(emp\_name)=f_{employee}(dept\_id)=f_{employee}(sex)=r_1$，$f_{employee}(addr)=f_{employee}(phone)=r_2$。$r_1$ 和 r_2 分别为

• $r_1=$"*emp_id* in(select *emp_id* from *employee* where *dept_id*=(select *dept_id* from *employee* where *emp_name*=*USER*())"；

• $r_2=$"*emp_name*=*USER*()"。

文献[11]中提出的元素级细粒度访问控制实现算法称为动态视图替换算法(DVR)。其基本思想是对用户提交的查询语句中的所有基表或者视图 $R(A_1,A_2,\cdots,A_n)$，构建对应的视图 V，并用 V 替换 R。假设细粒度访问控制策略为 $P=(s,(R,f_R),select)$，则视图 V 构建如下：

(SELECT

CASE WHEN $f_R(A_1)$ THEN A_1 ELSE NULL END AS A_1，

CASE WHEN $f_R(A_2)$ THEN A_2 ELSE NULL END AS A_2，

…，

CASE WHEN $f_R(A_n)$ THEN A_n ELSE NULL END AS A_n，

FROM R)

当 $f_R(A_i)$为 TRUE 时，对应的 CASE 语句直接退化为 A_i。

例 2　若在例 1 中的细粒度访问控制策略控制下，用户提交查询 Q_1，根据 DVR 算法，Q_1 将被改写为 Q_1'执行：

• $Q_1=$"SELECT *emp_name* FROM employee WHERE *dept_id*=1002"；

• $Q_1'=$"SELECT *emp*_name FROM(SELECT *emp_id*，CASE WHEN(r_1) THEN *emp_name* ELSE NULL AS *emp_name*，CASE WHEN(r_1) THEN *sex* ELSE NULL END AS *sex*，CASE WHEN(r_1) THEN *dept_id* ELSE NULL END AS *dept_id*，CASE WHEN(r_2) THEN *addr* ELSE NULL END AS *addr*，CASE WHEN(r_2) THEN *phone* ELSE NULL END AS *phone* FROM *employee*) WHERE *dept_id*=1002"

在 DVR 算法中，不允许用户访问的数据都被替换为“NULL”，因此 DVR 算法是安全的。

3　DVR 算法分析

DVR 算法的关键是创建替换视图。然而，DVR 算法在构建替换视图时，没有考虑用户提交的 SQL 语句的特性及细粒度访问控制策略的特性，因此使用 DVR 算法得到的最终执行的查询语句中存在冗余，从而影响了查询语句的执行性能。DVR 算法构造的查询语句中主要存在两种冗余。

3.1　属性冗余

在现实应用中，用户提交的查询语句具有多种形式。对某些查询语句而言，利用仅被访问的基表或视图中的部分属性。例如，在例 2 中，查询语句 Q 仅仅利用了属性 *emp_name* 和 *dept_id*，而其他属性并未被使用。因此，对该语句而言，其他属性是冗余的。冗余属性上的过滤器同样是冗余的。为了提高查询语句的执行性能，这些冗余需要被移除。

下面，用 $R(A_1,A_2,\cdots,A_n)$表示关系 R，$R.*=\{A_1,A_2,\cdots,A_n\}$。

定义 2(关键属性)　对任意的查询语句 Q，R 是 Q 中涉及的关系。对任意属性 $A\in R.*$，若 A 属于 Q 的 SELECT 子句或者 WHERE 子句，称 A 是关于查询 Q 和关系 R 的关键属性。

定义 3(冗余属性)　对任意的查询语句 Q，R 是 Q 中涉及的关系。对任意属性 $B\in R.*$，若 B 不是关于查询 Q 和关系 R 的关键属性，称 B 为关于查询 Q 和关系 R 的冗余属性。

性质 1　关于查询 Q 和关系 R 的关键属性集合和冗余属性集合分别记为 R_k 和 R_r，则 $R_k\cup R_r=R.*$ 且 $R_k\cap R_r=\varnothing$。

定义 4(冗余过滤器)　对任意的查询语句 Q，R 是 Q 中涉及的关系。若细粒度访问控制策略 $P=(s,(R,f_R),select)$。对任意属性 $B\in R.*$，若 B 是关于查询 Q 和关系 B 的冗余属性，则其对应的过滤器 $f_R(B)$为冗余过滤器。

例 2 中，属性 *emp_name* 和 *dept_id* 是关于查询 Q 和关系 *employee* 的关键属性，而 *emp_id*，*sex*，*addr* 和 *phone* 为冗余属性，过滤器 r_1 和 r_2 为冗余过滤器。

为了提高查询语句的执行性能，冗余过滤器需要被移除，即替换视图中不需要包含冗余过滤器。因此，若主体 s 提交查询 Q，Q 包含关系 $R(A_1,A_2,\cdots,A_n)$，且细粒度访问控制策略为 $P=(s,(R,f_R),select)$，关键属性集合为 $R_k=\{A_{k1},A_{k2},\cdots,A_{kn}\}$，则 R 的替换视图 V 构建如下。

(SELECT

CASE WHEN $f_R(A_{k1})$ THEN A_{k1} ELSE NULL END AS A_{k1}，

CASE WHEN $f_R(A_{k2})$ THEN A_{k2} ELSE NULL END AS A_{k2}，

…

CASE WHEN $f_R(A_{kn})$ *THEN* A_{kn} ELSE NULL END AS A_{kn}，

FROM *R*)

当 $f_R(A_{ki})$为 TRUE 时，对应的 CASE 子句将退化为 A_{ki}。

3.2　相同过滤器冗余

细粒度访问控制策略保护数据库中的信息不被非授权用户访问。在很多情况下，数据库中的信息可以根据信息的敏感程度被划分为几类。如例 2 中，*employee* 表的信息被划分为 3 类：*emp_id* 为一类；*emp_name*，*sex*，*dept_id* 为一类；*addr* 和 *phone* 为一类。同类信息中的属性具有相同的过滤器。该特性在 DVR 算法中没有被考虑。

相同的过滤器(谓词)在 WHERE 子句中的执行性能要远远优于其在 CASE 子句中的执行性能。因此，为了提高查

询语句的执行性能，需要尽可能将 CASE 子句转换为 WHERE 子句执行。在很多情况下，不同 CASE 语句中相同的过滤器可以直接移到 WHERE 子句中。此时，CASE 语句中的多个相同的过滤器可以认为是冗余的。例如，细粒度访问控制策略 $P=(s,(R,f_R),select)$，若所有属性上的过滤器相同，即 $f_R(A_1)=\cdots=f_R(A_n)$，则关系 R 的替换视图 V 可以直接构建如下：

(SELECT * FROM R WHERE $f_R(A_1)$)

假设由 DVR 算法构建的替换视图为 V'，V 的执行性能要远远优于 V'。

然而，上述情况只是一种特例。下面将结合前述关键属性的概念给出一般性的例子来移除相同过滤器冗余。

考虑属性冗余时，替换视图中仅仅保留关键属性的 CASE 语句。因此，当关键属性所对应的过滤器相同时，相应的 CASE 语句中的过滤器可以直接移到 WHERE 子句中。因此，主体 s 提交的查询 Q 包含关系 $R(A_1,A_2,\cdots,A_n)$，细粒度访问控制策略为 $P=(s,(R,f_R),select)$，且关键属性集合为 $R_k=(A_{k1},A_{k2},\cdots,A_{km})$。若 $f_R(A_{k1})=\cdots=f_R(A_{km})$，则关系 R 的替换视图 V 构建如下：

(SELECT * FROM R WHERE $f_R(A_{k1})$)

另外一个特例为，若查询 Q 的关键属性仅仅包含一个属性 A，A 对应的过滤器可以直接移到 WHERE 子句中。

4 实现

4.1 相同过滤器检测

本文 3.2 节中介绍了相同过滤器冗余及其移除，其基础是相同过滤器检测。本节首先介绍过滤器的描述，进而给出检测相同过滤器的规则。

定义 5(过滤器) 查询 $Q=\langle selList, predicate, relations\rangle$，其中 *selList* 是查询 Q 的 SELECT 子句中的属性集合，*predicate* 表示 Q 的 WHERE 子句，*relations* 表示 Q 中的基表和视图的集合，过滤器即为 *predicate*，其具有以下形式：

- SingToken=⟨attr,Op,const⟩表示属性与常量比较关系；
- DoubleToken=⟨attr,Op,attr⟩表示属性与属性比较关系；
- ExistsToken=⟨exists,Q⟩表示 EXISTS 谓词；
- InToken=⟨InAttr,Q⟩表示 IN 谓词；
- Op∈{≤,<,=,>,≥,LIKE}；
- predicate::= predicate∧predicate | predicate∨predicate |¬predicate | DoubleToken | SingleToken | ExistesToken | InToken。

根据上面谓词定义，下面给出相同过滤器的检测规则。

(1) SingleToken

SingleToken 形式的过滤器，首先将其转换为规范化格式：属性在左边，常量在右边。例如，过滤器 $10<emp\_id$ 被转换为 $emp\_id>10$。

当过滤器 α 为规范化 SingleToken 形式，使用 Attr(α)，Op(α) 和 Const(α)表示 α 中的属性、比较符和常量。例如，Attr($emp\_id>10$)=emp_id，Op($emp\_id>10$)="="和 Const($emp\_id>10$)=10。

规则 1 α 和 β 为规范化 SingleToken 形式过滤器，则

$\alpha=\beta\Leftrightarrow$ Attr(α)=Attr(β)∧Op(α)=Op(β)∧Const(α)=Const(β)

(2)DoubleToken

DoubleToken 的规范化格式为：属性按字母表排序，小属性在左边。例如，过滤器 $y>x$ 被转换为 $x>y$。

当过滤器 α 为规范化 DoubleToken 形式，使用 LAttr(α)，Op(α) 和 RAttr(α)表示 α 中的左边属性、比较符和右边属性。例如，LAttr($x>y$)=x，Op($x>y$)="＞"和 RAttr($x>y$)=y。

规则 2 α 和 β 为规范化 DoubleToken 形式过滤器，则

$\alpha=\beta\Leftrightarrow$ LAttr(α)=LAttr(β)∧Op(α)=Op(β)∧RAttr(α)=RAttr(β)

(3)ExistsToken

当 α 为 ExistsToken=⟨exists,Q⟩形式的过滤器时，使用 $EQ(\alpha)$表示 α 中子查询 Q。

规则 3 α 和 β 为 ExistsToken 形式过滤器，则

$\alpha=\beta\Leftrightarrow EQ(\alpha)=EQ(\beta)$

(4) InToken

当 α 为 InToken=⟨InAttr,Q⟩，使用 IAttr(α)和 IQ(α)表示 α 中的属性 InAttr 和子查询 Q。

规则 4 α 和 β 为 InToken 形式过滤器，则

$\alpha=\beta\Leftrightarrow$ IAttr(α)=IAttr(β)∧IQ(α)=IQ(β)

(5) AND 过滤器

当过滤器 α 为 predicate∧predicate 时，称为 AND 过滤器。使用 Prd(α)表示 α 中的子过滤器。

规则 5 α 和 β 为 AND 过滤器，则

$\alpha=\beta\Leftrightarrow(\forall c\in Prd(\alpha),\exists d\in Prd(\beta)\wedge c=d)\wedge(\forall d\in Prd(\beta),\exists c\in Prd(\alpha)\wedge c=d)$

(6) OR 过滤器

当过滤器 α 为 predicate∨predicate 时，称为 OR 过滤器。使用 Prd(α)表示 α 中的子过滤器。

规则 6 α 和 β 为 OR 过滤器，则

$\alpha=\beta\Leftrightarrow(\forall c\in Prd(\alpha),\exists d\in Prd(\beta)\wedge c=d)\wedge(\forall d\in Prd(\beta),\exists c\in Prd(\alpha)\wedge c=d)$

(7) 查询语句

查询语句 Q 形式为⟨*selList*, *predicate*, *relations*⟩，使用 Attr(Q)，Pred(Q)和 Rel(Q) 表示 Q 中的 SELECT 子句中的属性集合、WHERE 子句的谓词和关系集合。

规则 7 Q_1 和 Q_2 为两个查询语句，则

$Q_1=Q_2\Leftrightarrow$Attr(Q_1)=Attr(Q_2)∧Pred(Q_1)=Pred(Q_2)∧Rel(Q_1)=Rel(Q_2)

(8) NOT 过滤器

当过滤器 α 和 β 为¬predicate 形式过滤器时，称为 NOT 过滤器，首先使用 De Morgan 规则移除否定符合，进而使用规则 1－规则 7 进行相同性检测。

此处，给出了过滤器的构造规则和相同性检测规则。由于过滤器本质上是数据库中的谓词，其形式多样。此处仅仅考虑了在细粒度访问控制策略中使用的常见情况。

根据上述规则可以检测出相同的过滤器。然而，检测是一项耗时的工作。为了改善查询语句的执行性能，过滤器相同性检测在策略被定义时进行，并将其存储在数据库中。

定义 6 细粒度访问控制策略 $p=(s,(o,f_o),select)$，相同

过滤器信息采用下面的同一性集合(Identical Set)结构存储：

$IS_p=\{(p,A,filter)\mid A\in 2^{\omega,*},\forall a\in A,f_o(a)=filter\}$

任意的同一性集合 IS，对 $r\in IS$，使用 $r.A$ 和 $r.PR$ 表示 r 的属性集合和过滤器。

因此，对于任意定义成功的细粒度访问控制策略 p，经过相同过滤器检测，都存在一个对应的同一性集合 IS_p 与之对应。

性质 2 对任意的细粒度访问控制策略 $p=(s,(o,f_o),select)$，其相对应的同一性集合为 IS_p：

- $\bigcup_{r\in IS_p} r.A=o.*$
- $\forall r_1,r_2\in IS_p,r_1\neq r_2,r_1.A\cap r_2.A=\emptyset\wedge r_1.PR\neq r_2.PR$。

4.2 ADVR 算法

为了移除 DVR 算法所生成的查询语句中的冗余，本节提出 ADVR 算法(Adaptive DVR)，如算法 1 所示。

算法 1 Adaptive DVR Algorithm

```
Input: Q: the query issued by user U
Output: Q_modified: the modified query
1: for each R_i referred in Q do
2:   AS_Ri = GetKeyAttributes(R_i, Q);
/* Search all key attributes of R_i for Q and add them into set AS_R */
3:   P = GetFGACPolicy(R_i, U);
/* Search the FGAC policy for relation R_i and user U */
4:   if ∃r∈IS_p, AS_Ri = r.A then
5:     V_Ri = "SELECT AS_Ri[1], ..., AS_Ri[AS_Ri.length] FROM R_i
           WHERE r.PR"
6:   else
7:     V_Ri = "SELECT CASE WHEN F(P)(AS_Ri[1]) THEN AS_Ri
           [1] ELSE NULL END AS AS_Ri[1],
8:           ...,
9:           CASE WHEN F(P)(AS_Ri[AS_Ri.length] THEN AS_Ri
               [AS_Ri.length]) ELSE NULL END AS AS_Ri
               [AS_Ri.length]
10:        FROM R_i"
11:  END if
12: END for
13: Replacing R_i with V_Ri in Q to form Q_modified
```

ADVR 算法为用户提交的查询语句中的每个关系 R_i 建立替换视图 V_{Ri}(line 1)，以获取关系 R_i 的关键属性和细粒度访问控制策略(line 2,3)；然后判断关键属性上的过滤器是否相同(line 4)。若相同，则按本文 3.2 节介绍的方法建立替换视图 V_{Ri}(line 5)，否则按 3.1 节介绍的方法建立替换视图 V_{Ri}(line 7—10)，最后用替换视图 V_{Ri} 替换关系 R_i。

5 实验与分析

本节将通过实验说明 ADVR 相对于 DVR 算法而言具有更好的性能。实验中主要考虑两个参数：

- Table Size：表中的元组数目；
- Key Attribute Size：关键属性的个数。

与文献[8,11-13]保持一致，实验数据集(见表 2)产生于 Wisconsin Benchmark[14]。实验环境为：2.8GHz Intel(R) CPU，2GB 内存，320GB 硬盘，Windows XP 操作系统和 Oracle 10g 数据库管理系统。

表 1 实验数据集

Attribute	Description
ID1(number)	Primary key, sequential order
ID2(number)	Candidate key, random order
VA1(number)	Values 0-k, k=TableSize/100, Filter PA
VA2(number)	Values 0-k, Filter PA
VA3(number)	Values 0-k, Filter PA
VA4(number)	Values 0-k, Filter PA
VA5(number)	Values 0-k, Filter PA
VB1(number)	Values 0-k, Filter PB
VB2(number)	Values 0-k, Filter PB
VB3(number)	Values 0-k, Filter PB
VC1(number)	Values 0-k, Filter PC
select_25(number)	Values 0-1(25%), indexed
select_50(number)	Values 0-1(50%), indexed
select_75(number)	Values 0-1(75%), indexed
stringu1(32-byte str)	Unique character string
Stringu2(32-byte str)	Unique character string

ADVR 算法根据关键属性和细粒度访问控制策略的特性移除 DVR 算法产生的冗余。由于并非所有由 DVR 算法生成的查询语句中都存在冗余，是否存在冗余直接与查询语句的关键属性集和细粒度访问控制策略相关。因此，对待测试的查询语句，随机生成其关键属性集，而细粒度访问控制策略是固定的。在实验中，对每个待测试的查询语句，关键属性集从{VA1,VA2,VA3,VA4,VA5,VB1,VB2,VB3,VC1}中随机生成。待测试的查询语句结构如下：

SELECT list FROM Datasets WHERE clause

Datasets 是表名，list 为随机选择的属性，clause 为查询选择条件。当指定关键属性集大小为 n 时，随机在集合{VA1,VA2,VA3,VA4,VA5,VB1,VB2,VB3,VC1}中选择 n 个属性构成关键属性。进行 6 次选择，即构造了 6 个待测语句。每个待测语句执行 3 次，最终取其平均执行时间。每次执行时，清空缓存。

细粒度访问控制策略主要存在 3 种不同的过滤器。VA1,VA2,VA3,VA4 和 VA5 具有相同的过滤器 PA，VB1,VB2 和 VB3 具有相同的过滤器 PB，VC1 上的过滤器为 PC。

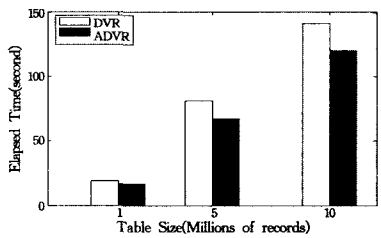

图 1 Table Size 对性能的影响，其中关键属性个数为 4

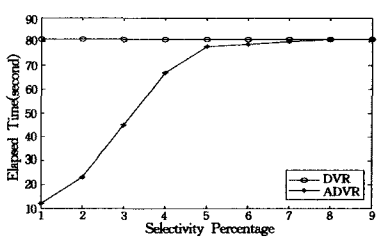

图 2 关键属性个数对性能的影响，其中 Table Size=5000000

实验结果如图 1 和图 2 所示。结果表明，ADVR 算法在性能上优于 DVR 算法。其主要原因是 ADVR 算法移除由 DVR 算法构造的查询中包含的冗余。同时，由图 2 可见，关

(下转第 142 页)

ACM Press,2000:71-80

[4] Oza N C,Russell S. Online bagging and boosting[C]// Proc of Artificial Intelligence and Statistics. Morgan Kaufmann. San Francisco,2001:105-112

[5] Oza N C, Russell S. Experimental comparisons of online and batch versions of bagging and boosting[C]// Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, California: ACM Press, 2001: 359-364

[6] Gama J, Rocha R, Medas P. Accurate decision trees for mining high-speed data streams[C]// Proc. of the 9th ACM SIGKDD International Conference in Knowledge Discovery and Data Mining. New York: ACM Press, 2003: 523-528

[7] Hulten G, Spencer L, Domingos P. Mining Time-changing Data Streams[C]// Proc. of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco: ACM Press, 2001: 97-106

[8] Street W, Kim Y. A *streaming ensemble algorithm* (SEA) for large-scale classification[C]// Proc. of the 7th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM Press, 2001: 377-382

[9] Wang Haixun, Fan Wei, Yu P S, e al. Mining concept-drifting data streams using ensemble classifiers[C]// Proc. of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington: ACM Press, 2003: 226-235

[10] Kolter J Z, Maloof M A. Dynamic weighted majority: an ensemble method for drifting concepts[J]. Machine Learning Research, 2007, 8: 2755-2790

[11] Zhang P, Zhu X Q, Shi Y. Categorizing and mining concept drifting data streams[C]// Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Las Vegas, Nevada, USA: ACM Press, 2008: 812-820

[12] Bifet A, Holmes G, Pfahringer B, et al. New ensemble methods for evolving data streams[C]// Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2009: 139-148

[13] Chu Fang, Wang Yizhou, Zaniolo C. An adaptive learning approach for noisy data streams[C]// Proc. of 4th IEEE International Conference on Data Mining. Brighton: IEEE Computer Science, 2004: 351-354

[14] Wang Yong, Li Zhanhuai, Zhang Yang. Classifying noisy data streams[C]// Proc. of 3rd International Conference of Fuzzy Systems and Knowledge Discovery (FSKD). Xi'an, China: Springer, 2006: 549-558

[15] Gama J, Fernandes R, Rocha R. Decision trees for mining data streams[J]. *Intelligent Data Analysis*, 2006, 10: 23-45

[16] Li Peipei, Hu Xuegang, Wu Xindong. Mining concept-drifting data streams with multiple semi-random decision trees[C]// Proc. of 4th International Conference on Advanced Data Mining and Applications. Chengdu, China: Springer, 2008: 733-740

[17] Tan Pang-ning, Steinbach M. 数据挖掘导论[M]. 范明, 等译. 北京: 人民邮电出版社, 2006

(上接第 133 页)

键属性个数越少,ADVR 算法的优越性就越大。其主要原因是关键属性个数越少,DVR 算法构造的查询语句中存在冗余的可能性就越大,即 ADVR 算法移除冗余的可能性就越大,因此其性能就越好。

结束语 通过对 DVR 算法分析,得知 DVR 算法没有考虑用户提交的 SQL 语句和细粒度访问控制策略的特性,使得最终生成的查询语句中存在冗余,从而影响了查询语句的执行性能。本文给出了两类冗余,并提出了移除冗余的方法。最终提出了 ADVR 算法,该算法移除了 DVR 算法所产生的冗余。通过实验得知,ADVR 算法优于 DVR 算法。

参 考 文 献

[1] Agrawal R, Kiernan J, Srikant R, et al. Hippocratic Database [C]// Proceedings of VLDB. Hong Kong, China, 2002: 563-574

[2] Agrawal R, Bird P, Grandison T, et al. Extending relational database systems to automatically enforce privacy policies [C]// Proceedings of ICDE 2005. Tokyo, Japan: IEEE, 2005: 1013-1022

[3] Bertino E, Sandbu R. Database Security-Concepts, Approaches, and Challenges [J]. IEEE Transactions on Dependable and Secure Computing, 2005, 2(1): 2-19

[4] Byun J W, Bertino E, Li N. Purpose Based Access Control of Complex Data for Privacy Protection [C]// Proceedings of SACMAT 2005. Stockholm, Sweden: ACM, 2005: 102-110

[5] Dwivedi S, Menezes B, Singh A. Database Access Control for E-business-A Case Study [C]// Proceedings of Int. Conf. on Management of Data. Bombay, India: IEEE, 2005: 168-175

[6] Rizvi S, Mendelzon A, Sudarshan S, et al. Extending Query Rewriting Techniques for Fine-grained Access Control [C]// SIGMOD 2004. Paris, France: ACM, 2004: 551-562

[7] Chaudhuri S, Dutta T, Sudarshan S. Fine Grained Authorization Through Predicated Grants [C]// Proceedings of ICDE 2007. Istanbul, Turkey: ACM, 2007: 1174-1183

[8] Wang Q, Yu T, Li N, et al. On the Correctness Criteria of Fine-grained Access Control in Relational Databases [C]// Proceedings of VLDB 2007. Vienna, Austria, 2007: 555-566

[9] Stonebraker M, Wong E. Access control in a relational database *management system by query modification* [C]// *Proceedings of* the 1974 ACM/CSC-ER. 1974: 180-186

[10] Corporation O. Oracle Virtual Private Database [EB/OL]. http://www.oralce.com/technology/deploy/security/db_security/virtual-private-database/index.html

[11] LeFevre K, Agrawal R, Ercegovac R, et al. Limiting disclosure in Hippocratic databases [C]// Proceedings of VLDB 2004. Roronto, Canada, 2004108-119

[12] Zhu H, Shi J, Wang Y, et al. Controlling Information Leakage of Fine-grained Access Control Model in DBMSs [C]// Proceedings of WAIM 2008. Zhangjiajie, China: IEEE, 2008: 583-590

[13] Shi J, Zhu H, Fu G, et al. On the Soundness Property for SQL Queries of Fine-grained Access Control in DBMs [C]// Proceedings of ICIS/ACIS 2009. Shanghai, China: IEEE, 2009: 469-474

[14] DeWitt D J. The Wisconsin benchmark: Past, present, and future [M]. The Benchmark Handbook, 1993