

密文数据库检索方法的研究

宋俊洪, 汪 冰

(广东工业大学 计算机学院, 广东 广州 510090)

摘要: 在密文索引的基础上, 针对实际密文数据库的应用, 将二叉树密文索引和 B⁺ 树索引相结合, 设计了一种新的密文索引, 并介绍了其查询方法, 分析了其查询效率及安全性.

关键词: B⁺ 树索引; 密文二叉树索引; 数据库

中图分类号: TP391 文献标识码: A 文章编号: 1007-7162(2007) 02-0093-03

对于加密后的数据库, 由于密文数据不再保持原有明文数据的顺序关系, 这就给查询操作带来了极大的困难. 为了提高密文数据库的查询效率, 有人提出了能实施密文查询的同态加密算法^[1-2]. 这种方法虽然提高了查询效率, 但密码强度不高, 致使安全性降低. 文献[3]中提出了一种基于密文索引的查询方法, 本文在此基础上作了进一步的探索.

1 二叉树密文索引

顺序表索引结构, 其结构简单, 修改很不方便, 每次修改都要重组索引, 因此在实际应用中不多使用, 而普遍采用基于树表结构的动态索引. 通过建立二叉树来为密文数据库建立密文索引时可用二叉排序树(或二叉平衡树)的形式. 二叉树密文索引结点的结构如图 1^[3]所示.

加密粒度	加密方式
行号	列号
安全类标记	密文数据
左指针密文	右指针密文

图 1 二叉树密文索引结构

图 1 中加密粒度为 0 时, 表示不加密; 为 1 时, 表示粒度为数据项级; 为 2 时, 表示粒度为属性段级; 为 3 时, 表示粒度为记录级. 加密方式表示用来对数据和物理记录号进行加密的加密算法类型. 行号和列号表示数据在密表中的位置. 安全类标记为 0 时, 表示公开数据; 为 1, 2, .. 时分别表示相应的安全级别. 密文数据、左指针密文、右指针密文分别表示该结点的明文数据加密后的密文、该结点的左儿子结点的地址被加密后的密文以及该结点的右儿子结点的地址被加密后的密文.

2 B⁺ 树密文索引

由于密文数据库与明文数据库的差别, 因此采用 B⁺ 建立的密文索引与明文树索引会有所不同. 在明文 B⁺ 中, 重点考虑的是如何提高效率, 主要把如何降低 I/O 次数作为考虑对象; 而在密文数据库中, 考虑的是如何在确保安全的前提下提高系统效率. 现行的一些商用数据库系统普遍支持的聚簇存储技术, 其中隐藏有密文对应明文的大小顺序关系, 所以在明文数据库中采用的聚簇存储技术在密文数据库中不适用, 必须采用非聚簇存储技术来建立索引. B⁺ 树密文索引结构如图 2^[4]所示.

加密粒度	加密方式
行号	列号
安全类标记	结点标记
关键字数目	关键字数据密文
指向父结点指针的密文	指向儿子结点的全部指针的密文
指向后继结点的指针的密文	指向记录的全部指针的密文

图 2 B⁺ 树密文索引结构

图 2 中的加密粒度、加密方式、行号、列号、安全类标记的意义与前面的二叉树密文索引结构相同. 结点标记为 no_leaf 表示该结点是非叶子结点; 为 leaf 表示该结点是叶子结点. 关键字数目表示结点中关键字的个数. 关键字数据密文表示结点中各关键字加密后的密文. 指向父结点指针的密文表示指向该结点父结点的指针加密后的密文. 指向儿子结点的全部指针的密文表示当结点标记为 no_leaf 时, 指向该结点的全部儿子结点的指针加密后的密文. 指向后继结点的指针的密文和指向记录的全部指针的密文分别表示当结点标记为 leaf 时, 指向该结点后继结点的指针加密后的密文和该结点中指向对应

记录的全部指针加密后的密文.

3 检索方法的探索

3 1 索引的设计

通常的数据库中, 需要保密的数据, 其数量不大, 绝大部分仍是明文数据. 由于明文数据量大, 其索引表也将很大, 通常将它放在外存, 为减少 I/O 次数, 采用 m 叉的 B^+ 树作为其索引表, 现行的关系型数据库管理系统都支持这种索引, 因此设计中不在这里下工夫. 而对于少量的密文数据, 其索引表很小, 在设计中考虑到它可以一次全部放入内存, 对这部分数据我们为它设计了二叉树密文索引. 它的索引结构描述如下.

首先将待加密的明文属性值进行排序: $a_1, a_2, \dots, a_b, a_p, \dots, a_n$ ($a_i \leq a_j, 1 \leq i < j \leq n$), 然后假设加密算法一次可以对 w ($w = 4$) 个明文属性值进行加密. 则可将它们按 4 个一组进行划分:

$$\begin{array}{cccccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ & \underbrace{\hspace{1.5cm}} & & \underbrace{\hspace{1.5cm}} & & & & \\ & b_1 & & b_2 & & & & \\ & & & & & & & \\ \dots & \underbrace{a_{4m-3} \quad a_{4m-2} \quad a_{4m-1} \quad a_{4m}}_{b_m}, & & & & & & \end{array}$$

若 $n = 4m$, 则正好划分为 m 块; 若 $4m - 3 = n < 4m$, 则在最后一块内进行填充, 使其也划分为 m 块. 然后对每组分别加密成密文 $b_1, b_2, \dots, b_m$. 对这 m 块密文 $b_1, b_2, \dots, b_m$, 按其序列建立一棵折半查找过程的判定树, 其密文判定树形式如图 3 所示.

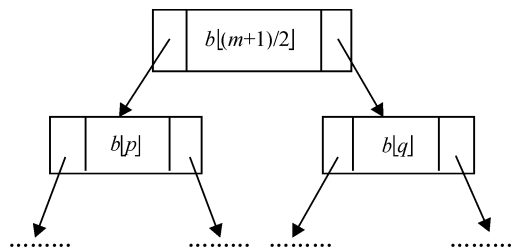


图 3 二叉密文索引树

其中: $p = \lfloor (m + 1) / 2 \rfloor$, $q = \lfloor (m + 1) / 2 \rfloor + 1$.

在这棵树中对全部指针也进行加密, 这样就对密文类数据构造了一棵全密文索引树. 在实际应用中将明文数据和密文数据集中放在一起, 这样在一个属性段上就有明文和密文二类数据. 结合系统为明文数据建立的 B^+ 树索引, 于是在一个属性段上就有了 2 个索引: 明文 B^+ 树索引和密文二叉树索引.

3 2 查询操作

3 2 1 已知查找

若事先已知查询结果为明文, 则只需查明文 B^+ 索引树. 假设每个节点占据一个物理块, 则先将根节

点调入内存与待查值比较, 由此决定将下一个节点调入内存, 直到找到满足条件的所有节点; 若事先已知查询结果为密文, 则只需将密文索引树调入内存, 从根节点解密与查询条件进行比较, 由此决定下一个将检索的节点, 直到找到满足条件的所有节点; 若事先已知查询结果中既有明文又有密文, 则先查密文索引树, 再查明文 B^+ 索引树, 或颠倒查找次序.

3 2 2 随机查找

即事先不知道查询结果中是明文、密文或二者都有, 则先将密文索引树调入内存进行查找, 若查到所有满足条件的结果, 则无需再查明文 B^+ 索引树. 若查到只有部分满足条件或无法确定是否满足条件及没有查到满足条件的结果, 则继续对明文 B^+ 树进行查找, 直到找到满足条件的全部结果.

3 2 3 试验数据

测试表 test 中有 10 万条记录, 其中在某一列上有 1 万条是密文数据, 9 万条是明文数据; 这里建立 3 棵索引树, 第 1 棵树 Tree1 将 10 万条记录在此列上采用图 1 的二叉树密文索引结构形式, 按其明文关键字以二叉排序树的形式建立; 第 2 棵树 Tree2 将全部 1 万条密文也在此列上采用图 1 的二叉树密文索引结构形式, 按其明文关键字以二叉排序树的形式建立 (Tree2 和 Tree1 在安全类标记设置中会有不同); 第 3 棵树 Tree3 将其余 9 万条明文记录在此列上建立普通的 B^+ 树索引. 测试所用系统配置为: CPU - Intel Celeron 1.5 GHz, 内存 - 512M, 硬盘 - 40G, OS - XP (sp2), DBMS - Oracle9i 针对已知查找, SQL 语句: select column from test where column = value 针对随机查找, SQL 语句: select column from test where column $\leq$ value (其中: column - 索引列, value - 查询条件值). 测试结果如图 4 图 5 所示.

	结果为密文		结果为明文	
	调 Tree1	调 Tree2	调 Tree1	调 Tree3
查询时间 / s	0.16	0.10	0.14	0.08

图 4 已知查找

	结果中既有明文又有密文	
	调 Tree1	先调 Tree2 接着调 Tree3
查询时间 / s	0.62	0.48

图 5 随机查找

3 3 效率分析

这里针对一般情况, 对查找关键字 $value = \{u | u \leq key\}$ 进行分析. 首先查密文二叉树索引, 解密得到其根节点关键字数据为: $a_{\lfloor (q+1)/2 \rfloor - 3}, a_{\lfloor (q+1)/2 \rfloor - 2}, a_{\lfloor (q+1)/2 \rfloor - 1}, a_{\lfloor (q+1)/2 \rfloor}$.

1)若 $key > a_{\lfloor (q+1)/2 \rfloor}$, 则对该节点的解密操作是必要操作, 其节点中关键字数据满足查询结果集, 然后解密其右指针进入下一节点的操作, 并且其左子树节点均满足条件.

2) 若 $a_{\lfloor m(q+1)/2 \rfloor - 3} \leq key \leq a_{\lfloor (q+1)/2 \rfloor}$, 则 $a_{\lfloor (q+1)/2 \rfloor - 3}, \dots, key$ 都属于查询结果集, 然后解密其左指针进入下一节点的操作, 并且以后的解密操作均是必要操作.

3)若 $key < a_{\lfloor (q+1)/2 \rfloor - 3}$, 则对该节点的解密操作是额外操作, 然后解密其左指针进入下一节点操作.

以上 3 个步骤实际上是一个递归过程. 待对密文二叉树索引查找完后, 进入明文 B^+ 树进行常规 B^+ 树的查询操作, 最后将从 2 个索引中查到的结果求并集, 即得到所求结果集. 不难看出, 查找过程中最坏情况是, 每次对节点的解密都有 $key < \min(\min(\text{节点中最小关键字}), \text{共需进行额外的解密节点} \times \log_2 q) + 1$ 次及解密指针 $\log_2 q$ 次操作. 不过这些额外操作均在内存进行, 不需要 I/O.

3.4 安全性分析

该方法虽然将全部密文索引集中存放, 缩小了明文空间, 但由于对每一数据块及其指针均采用加

密形式, 密文索引树中的节点对于敌手来说都是孤立的密文信息, 因此只要加密算法具有足够的强度, 敌手是难以对其实施攻击的.

4 结论

采用基于这种密文索引的查询方法, 其效率与加/解密算法的速度、加密率 α 的大小及密文数据未加密之前在整个明文索引中的分布状况 F 有很大的关系. 当 α 较小且 F 呈均匀分布时, 以上讨论的额外操作会降低到最低程度. 同时这种查询方法安全性能较高.

参考文献:

[1] Niv Ahituv, Yeheskel Lapid, Segev Neumann. Processing encrypted data[J]. Communications of the ACM, 1987, 30(9): 777-785.
[2] Rivest R L, Adleman L. On data bank and privacy homomorphisms[J]. Computer and Security, 1993, 21(2): 169-178.
[3] 崔国华, 洪帆, 付小青, 等. 数据库系统中一种更安全的加密机制[J]. 华中理工大学学报, 2000, 28(7): 29-31.
[4] 余祥宣, 刘伟. 数据库的密文索引机制[J]. 华中科技大学学报, 2002, 30(3): 16-18.

Research of Retrieval Method on Encrypted Database

SONG Jun-hong WANG Bing

(Faculty of Computer, Guangdong University of Technology, Guangzhou 510090, China)

Abstract Based on ciphertext index and aiming at practical application on encrypted database, the paper designs a new ciphertext index by combining binary tree ciphertext index with B^+ tree index. Meanwhile, it also introduces query method and analyzes its query efficiency and security performance.

Key words B^+ tree index, binary tree ciphertext index, database