# Accelerating Multilevel Secure Database Queries using P-Tree Technology

**Imad Rahal**
**IACC 258**
**Computer Science Department**
**North Dakota State University**
**Fargo, ND 58105, USA**
**imad.rahal@ndsu.nodak.edu**

**William Perrizo**
**IACC 258 A10**
**Computer Science Department**
**North Dakota State University**
**Fargo, ND 58105, USA**
**william.perrizo@ndsu.nodak.edu**

**Abstract**

*In the early 1990s, a lot of research was conducted in the area of multi-level secure database systems. Most of the work was directed towards the security aspect without much concentration on query acceleration. In this paper, a P-tree [1] based algorithm using the Sea View model [5] for multilevel relations is presented to accelerate queries in multilevel secure database systems. This algorithm recovers query output from single level relations in a fast and very space-efficient manner. The algorithm does not employ temporary data structures heavily nor does it produce spurious tuples while recovering, which has always been a major problem in this area.*

**Keywords**

Multilevel secure, query acceleration, P-tree, multilevel relations, Sea View Model

## 1. INTRODUCTION

Security in database systems, especially when confidential data is involved, has always been a great concern to enterprises. While a lot of work in the area of multilevel security has been done to solve this problem, most of it was directed towards the security aspect without much regard to query efficiency.

The Sea View model [5], one of the initial attempts to model multilevel relations, presented two algorithms: a decomposition algorithm to generate single level relations from a multilevel relation and a recovery algorithm to regenerate a multilevel relation from a set of single level relations. The decomposition algorithm is very successful and efficient and will be used in our algorithm; however, we will devise a new recovery algorithm to overcome the inefficiency and incorrectness – in terms of creating spurious tuples – of the current recovery algorithm. Our approach uses a quadrant-based lossless tree representation, the Peano Count Tree (P-tree) [1], to represent this data and overcome the sparsity problem. In addition, we will not employ a lot of temporary data structures.

The rest of the paper is organized as follows. In section 2 we give a brief description of the MAC access control mechanism which is used to maintain data confidentiality in secure database systems. Then, the concept of polyinstantiation [2,4,6] and its effects on multilevel recovery algorithms is described, followed by an explanation of multilevel relations [5]. In section 3, a brief preview of the SEA VIEW model is provided to give way to our algorithm, which is presented with an example in section 4. In section 5 a performance analysis is presented and finally the paper ends with a conclusion, in section 6, which sums up the overall benefits of our algorithm.

[1] Patents are pending for Peano Trees (P-trees) and bSQ technology

## 2. SECURITY CONSIDERATIONS

### 2.1 The Security Policy

The Mandatory Access Control or MAC mechanism, developed by Bell and LaPadula, defines a database for its subjects and objects. A subject is an active entity such as a process and an object is a passive entity such as a data item or table. Subjects have clearance levels and objects are assigned sensitivity levels (i.e., Top Secret, Secret, Confidential and Unclassified). In order for a certain subject to access an object, one of following two conditions must be satisfied (depending on the type of access):

1- *The Simple Security Policy*: A subject *X* can read access object *Y*, if *X*'s clearance level dominates (is greater than or equal to) *Y*'s sensitivity level.
2- *The *-Policy*: A subject *X* can write access object *Y*, if *X*'s clearance level is dominated by (is less than or equal to) *Y*'s sensitivity level.

In short, the MAC policy states that reads should propagate downwards and writes should propagate upwards.

All security authentication functions are stored in a TCB (Trusted Computing Base) away from the DBMS. When a request for object *X* by subject *Y* is issued, it is first authenticated in the TCB. If *Y* can access *X* then the request is forwarded to the DBMS; if not, then the request is rejected.

### 2.2 Polyinstantiation

Due to security and access control mechanisms, some tuples in multilevel relations may be polyinstantiated. A polyinstantiated tuple is a tuple that exists more than once in a relation with the same apparent key (refer to section 2.3) but with some other attribute value(s) being changed. This is due to the fact that different subjects are authorized to update or view different data. For example, suppose that a subject *X* with clearance *C* (Classified) is attempting to write a new value to a data item *Y* with sensitivity *S* (Secret). The old value in item *Y* is not viewable by subject *X* but the new value written into *Y* by *X* is viewable (it has *X*'s clearance level which *C*). To preserve the old value of *Y*, a new tuple is inserted into the relation with same apparent key (and same attribute values except for *Y* in this case). Now *X* can view the new value inserted into *Y* and Secret and Top Secret users can view the old value with sensitivity *S*.

## 2.3 Multilevel relations

A multilevel relation is a relation of the form R (*A1*, *C1*,…, *An*, *Cn*, *TC*) where *Ai* is any attribute and Ci is its classification (or sensitivity level). *TC* is the classification of the tuple. *Ci* belongs to the domain of classifications of the data items. We denote *A1* to be the apparent key of R. The concept of a key is a little bit different in multilevel relations because keys can be duplicated; this is why we refer to them as *apparent keys* instead of just keys. The reason behind this duplication of keys is polyinstantiation, (refer to section 2.2).

## 3. THE SEA VIEW (Secure VIEW) Model

The Sea View model [5] – a joint effort by the Stanford Research Institute and Gemini Computers – is considered as one of the most important moves towards multilevel security. It consists of two algorithms: a decomposition algorithm and a recovery algorithm. The decomposition algorithm divides a multilevel relation R into a set of single level relations. The multilevel relation exists only at logical level; single level relations are stored physically. For every query, an output multilevel relation is reconstructed from the single-level relations using the recovery algorithm.

Unfortunately, the recovery algorithm of the Sea View model suffers from the following [6]:

1- Creation of spurious tuples in the output (due to polyinstatiation).
2- Space inefficiency due to the use of many temporary data structures.
3- Time inefficiency due to unions and joins, which are two of the most expensive database operations.

## 4. OUR APPROACH

We will assume that the decomposition algorithm of the Sea View Model [4,5] was used to decompose the multilevel relation (presented in Figure 1), which represents all missiles deployed in Iraq during the gulf war, into single level relations. Our algorithm will be applied to recover the multilevel relation from those single level relations.

| R = Iraqi Missiles | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Name | | Developed by (devby) | | Length (M) | | Range (KM) | | TC |
| AS30L | *U* | France | *U* | 3.65 | *U* | 10 | *U* | *U* |
| AS-9 Kyle | *U* | Russia | *U* | 6 | *U* | 90 | *C* | *C* |
| Al Hussein | *U* | Iraq | *U* | 12.2 | *C* | 650 | *C* | *C* |
| Aspide | *U* | Italy | *C* | 3.7 | *C* | 35 | *C* | *C* |
| Roland1/2/3 | *C* | France | *C* | 2.4 | *C* | 6.3 | *C* | *C* |
| Roland1/2/3 | *C* | Germany | *S* | NIL | *S* | 8 | *S* | *S* |

**Figure 1. A multilevel relation**

| Rname,*u* |
|---|
| AS30L |
| AS-9 Kyle |
| Al Hussein |
| Aspide |

| Rname,*c* |
|---|
| Roland1/2/3 |

| Rdevby,*u*,*u* | |
|---|---|
| AS30L | France |
| AS-9 Kyle | Russia |
| Al Hussein | Iraq |

| Rdevby,*u*,*c* | |
|---|---|
| Aspide | Italy |

| Rdevby,*c*,*c* | |
|---|---|
| Roland 1/2/3 | France |

| Rdevby,*c*,*s* | |
|---|---|
| Roland 1/2/3 | Germany |

| Rlength,*u*,*u* | |
|---|---|
| AS30L | 3.65 |
| AS-9 Kyle | 6 |

| Rlength,*u*,*c* | |
|---|---|
| Al Hussein | 12.2 |
| Aspide | 3.7 |

| Rlength,*c*,*c* | |
|---|---|
| Roland 1/2/3 | 2.4 |

| Rrange,*u*,*u* | |
|---|---|
| AS30L | 10 |

| Rrange,*u*,*c* | |
|---|---|
| AS-9 Kyle | 90 |
| Al Hussein | 650 |
| Aspide | 35 |

| Rrange,*c*,*c* | |
|---|---|
| Roland 1/2/3 | 6.3 |

| Rrange,*c*,*s* | |
|---|---|
| Roland 1/2/3 | 8 |

**Figure 2. Single level relations**

Figure 2 represents all the single level relations created by the Sea view model's decomposition algorithm. All relations containing the key only – i.e., Rname,*u* and Rname,*c* – are referred to as base relations. *Ai* denotes any attribute and *A1* denotes the apparent key. We say that a classification *x* is greater than classification *y* (i.e. *x*>*y*) iff x dominates y in the classification hierarchy. For simplicity, assume that all relations sort their entries in an order consistent with base relations – otherwise we would need indexes to keep track of key entries. Suppose we want to recover the output of the query *"Select name, devby, length from R where range<>35"*

1. For every relation R*Ai*,*x*,*y* (single level relations containing all entries from multilevel relation having keys at level *x* and *Ai* attribute values at level *y*), excluding base relations, create a P-tree, P*Ai*,*x*,*y*, denoting the presence or absence of keys at level *x*. The created P-trees are represented in Figure 3. Keys at level *u* = {AS30L, AS-9 Kyle, Al Hussein, Aspide} and Keys a level *c* = {Roland 1/2/3}

| | | | | | |
|---|---|---|---|---|---|
| Prange,*u*,*u* = | 1 1000 | Pdevby,*u*,*u* = | 3 1110 | Plength,*u*,*u* = | 2 1100 |
| Prange,*u*,*c* = | 3 0111 | Pdevby,*u*,*c* = | 1 0001 | Plength,*u*,*c* = | 2 0011 |
| Prange,*c*,*c* = | 1 | Pdevby,*c*,*c* = | 1 | Plength,*c*,*c* = | 1 |
| Prange,*c*,*s* = | 1 | Pdevby,*c*,*s* = | 1 | | |

**Figure 3**. **P-trees for length and range attributes**

2. Create the output P-trees, Pout,*x*, at every level x (contains all keys with classification *x* that will appear in the output table) as follows:

a. Read all relations having an attribute participating in the selection criteria of the query (range <> 35). We should read Rrange,*u*,*u* and Rrange,*u*,*c* at level *u* and Rrange,*c*,*c* and Rrange,*c*,*s* at level *c*.

b. Get all entries from those relations satisfying the selection criteria at each level *x*. At level *u* we have AS30L, AS-9 Kyle, and Al Hussein; and at level *c* we have Roland1/2/3

c. Create Pout,*x* at level x where a 1 value is given to those keys succeeding from step b and 0 otherwise. Results are presented in Figure 4.

```
Pout,u =    3    (the first 3 keys in Rname,u will
         1110   appear in the output)
Pout,c =    1    (the first and only  key in Rname,c
                 will appear in the output)
```

**Figure 4. Output P-trees**

3. Create the polyinstantiated P-trees, Ppoly,*x*,*y*, for all *x*, *y* combinations such that *x*<*y* (contains all polyinstantiated keys at level *x* by subjects at level *y*).

a. For all attributes *Ai* requested in the output of the query (name, devby and length) except for the key (name) create the following temporary p-trees: Temp-Ai,*x*,*y*= AND all PAi,*x*,*y* where *x*<=*y*. Results are presented in Figure 5.

b. To get Ppoly,*x*,*y*, OR all Temp-Ai,*x*,*y*. Results are shown in Figure 6.

```
Temp-devby,u,c =   0  Temp-devby,c,s =   1
Temp-devby,u,s =   0  Temp-length,u,c =  0
Temp-length,c,s =  0  Temp-length,u,s =  0
```

**Figure 5**. **Temporary P-trees**

```
Ppoly,u,c = 0   Ppoly,c,s = 1   Ppolyu,s = 0
```

**Figure 6**. **Polyinstantiated P-trees**

If we view the leaf nodes as a bit vector, a 1-bit in position n in any Ppoly,*x*,*y* signifies that the nth entry in Rkey,*x* is polyinstantiated . Therefore, entry 1 in Rname,*c* which is *Roland1/2/3* is polyinstantiated.

4. Create the polyinstantiated output P-trees, Ppoly,out,*x*,*y*, by ANDing Ppoly,*x*,*y* and Pout,*x* (where *x*<*y*). Results are presented in Figure 7.

```
Ppoly,out,u,c= Ppoly,u,c AND Pout,u
             = 0  AND   1
                     1000
             = 0
Ppoly,out,c,s = Ppoly,c,s AND Pout,c
              = 1  AND   1
              = 1
Ppoly,out,u,s= Ppoly,u,s AND Pout,u
             = 0  AND   1
                     1000
              = 0
```

**Figure 7**. **Polyinstantiated output P-trees**

A 1-bit in position n in any Ppoly,out,*x*,*y* signifies that the nth entry in Rkey,*x* is polyinstantiated and appears in the output of the query . Therefore, entry 1 in Rname,*c* which is *Roland1/2/3* is polyinstantiated and appears in the output.

5. Create Output table (OT) as follows (the output table is presented in Figure 8):

a. OT has number of columns equal to the number of fields, *Ai*, requested in the output of the query Select *name*, *devby* and *length* (3 columns)

b. Scan Pout,*x* for 1-bit entries. If a 1 bit appears in position n do the following for all *Ai* attributes requested in the output:

i. If *Ai* is the key (i=1) then get the nth record from Rkey,*x* and store it under *Ai* column in output table. This entry has classification *x*.

ii. Else, go to the nth entry in P*Ai*,*x*,*z* where *z* = *x*. If a 1 bit is found in position n then get the value of the nth entry from R*Ai*,*x*,*z* and store it under *Ai* column in output table. This entry has classification *z*. Else (a 1 bit is not found in position n of P*Ai*,*x*,*z* then) increment *z* to the next higher level and repeat this step.

c. Scan Ppoly,out,*x*,*y* for 1 bit entries. If a 1 bit appears in position n do the following for all *Ai* attributes requested in the output:

i. If *Ai* is the key (i=1) then get the nth record from R*key*,*x* and store it under *Ai* column in output table. This entry has classification *x*.

ii. Else, go to the nth entry in P*Ai*,*x*,*z* where *z* = *y* initially . If a 1 bit is found in position n then get the value of the nth entry from R*Ai*,*x*,*z* and store it under *Ai* column in output table. This entry has classification *z*. Else (if 1 bit is not found in position n of P*Ai*,*x*,*z* then) decrement *z* to the next lower level and repeat this step.

| **Output multilevel relation** | | | | | |
|---|---|---|---|---|---|
| ***Name** | | **Developed by (devby)** | | **Length (M)** | |
| AS30L | *U* | France | *U* | 3.65 | *U* |
| AS-9 Kyle | *U* | Russia | *U* | 6 | *U* |
| Al Hussein | *U* | Iraq | *U* | 12.2 | *C* |
| Roland1/2/3 | *C* | France | *C* | 2.4 | *C* |
| Roland1/2/3 | *C* | Germany | *S* | 2.4 | *S* |

**Figure 8. Final output multilevel relation**

## 5. PERFORMANCE ANALYSIS

For the purpose of performance analysis, we compare the cost associated in answering a query with two different techniques: one with our algorithm and the other without any join accelerator – the P-tree is a join accelerator. We only consider the secondary storage page I/O cost while ignoring CPU cost since the latter is negligible when compared to the former.

The total number of page accesses needed to retrieve two, four, and eight attributes of the multilevel relation is calculated and presented in Figures 10 and 11 (*Unacc* stands for an algorithm that does not use accelerators and *P-Acc* stands for our P-tree based accelerator. The number after *Unacc* or *P-Acc* represents the number of attributes accessed).

These cost figures were calculated using the Yao's formula [3]:

$$\text{Yao}(k, m, n) = m - m * \prod_{i=1}^{k}((n-(n/m)-i+1)/(n-i+1))$$

Given that a page is accessed at most once, this formula gives the number of page accesses needed to get k records randomly distributed in a file of n records in m pages. To retrieve records from the base relations at different levels, an algorithm that does not use an accelerator accesses the index of each base relation whether the tuples exist in them or not. But our algorithm does not access the index unless a record exists in the corresponding base relation. The cost of page access associated with reading the index of a relation having n records in m pages, when k records are retrieved is calculated by *Yao (Yao (k, m, n), m/FO, m)* where FO is the average fan out of an index node in a B+ tree. Figure 9 shows the parameters used in the cost model.

Page size = 4 KB
Number of tuples in each base relation = 100,000
Number of pages in main memory = 1,000
Size of an attribute = 8 bytes
Size of a value identifier = 4 bytes
Average fan out of an index B+ tree = 287
Average page occupancy factor = 0.7

**Figure 9. Parameters for the cost model**

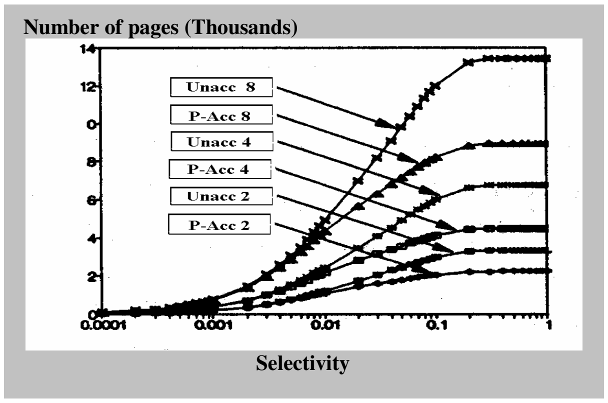

**Figure 10. Two classification levels accessed**

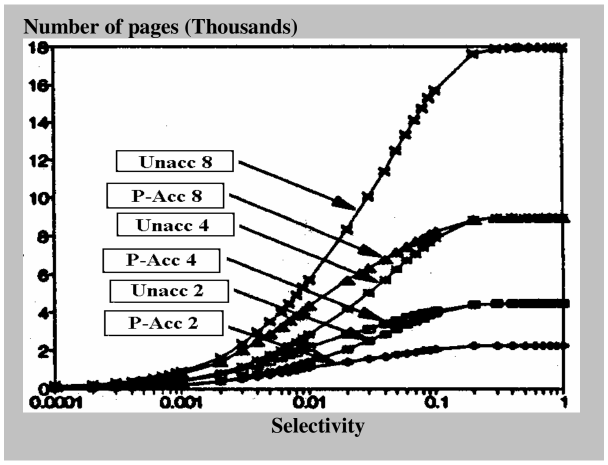

**Figure 11. Three classification levels accessed**

Figures 10 and 11 show the results when the query retrieves ecords from two and three different security levels respectively. The polyinstantiation rate is taken as 10 percent and the probability that a record has a null value in the query result (the value may exist at a higher level than accessed) is taken as 0.1. In all shown cases, our algorithm retrieves a smaller number of pages, thus, indicating that it is faster. The savings in using our algorithm increase as the number of levels increases, as the number of categories increases, and also as the number of attributes required for output increases. As the selectivity reaches 0.2, the number of pages required to retrieve the records exceeds the total number of pages in the base relations. The reason for this is that there are many page accesses to the indices.

## 6. CONCLUSION

Compared to the recovery algorithm of the Sea View model, which is one of the earliest and most important attempts towards multilevel database security, our algorithm has the following advantages:

1. No spurious tuples in the output table due of polyinstantiation.
2. No time inefficiency because our algorithm does not depend on the use of joins and unions like the Sea View model algorithm to create the output table.
3. Better efficiency in terms of time and space.

In short, our algorithm accelerates queries in multilevel secure databases without using a lot of temporary data structures. It is entirely based on a lossless space efficient representation, the P-tree data structure, where logical operations are very easy and efficient to perform.

## 7. REFERENCES

[1] Qin Ding, Maleq Khan, Amalendu Roy and William Perrizo, "The P-Tree Algebra", Proceedings of the ACM Symposium on Applied Computing (SAC'02), pp. 426-431, 2002.

[2] Ravi Sandhu and Sushile Jajodia, "Restricted polyinstatntiation or how to close signaling channels without duplicity", Proceedings of the 3rd RADC Workshop on Multilevel Database Security, pp. 7-12, 1990.

[3] S. B. Yao, "Approximating Block Accesses in Database Organizations", Communications of the ACM, vol. 20, No. 4, pp. 260-261, 1977.

[4] Sushile Jajodia and Ravi Sandhu, "Toward a Multilevel Secure Relational Data Model", Proceedings of the ACM SIGMOND International Conference on Management Data, pp. 50-59, 1991.

[5] T. F. Lunt, D. Dennis, R. R. Schell, M. Heckman and V. R. Shockleg, "The Sea View Security Model", IEEE Trans on Software Engineering (TOSE) vol. 16, No. 6, pp. 593-607, 1990.

[6] William Perrizo and Brajenda Panda, "Query Acceleration in Multilevel Secure Distributed Database Systems", Proceedings of the 16th National Computer Security Conference, pp. 53-62, 1993.