# Privacy Preservation and Protection by Extending Generalized Partial Indices

Guoqiang Zhan, Zude Li, Xiaojun Ye, and Jianmin Wang

School of Software, Tsinghua University, Beijing, 100084, China
{zhan-gq03, li-zd04}@mails.tsinghua.edu.cn,
{yexj, jimwang}@tsinghua.edu.cn

**Abstract.** Privacy[1] violation has attracted more and more attention from the public, and privacy preservation has become a hot topic in academic communities, industries and societies. Recent research has been focused on purpose-based techniques and models with little consideration on balancing privacy enhancement and performance. We propose an efficient Privacy Aware Partial Index (PAPI) mechanism based on both the concept of purposes and the theory of partial indices. In the PAPI mechanism, all purposes are independent from each other and organized in a flatten purpose tree($\mathcal{FPT}$). Thus, security administrators can update the flatten purpose tree by adding or deleting purposes. Intended purposes are maintained in PAPI directly. Furthermore, based on the PAPI mechanism, we extend the existing query optimizer and executor to enforce the privacy policies. Finally, the experimental results demonstrate the feasibility and efficiency of the PAPI mechanism.

## 1 Introduction

The privacy issue has currently become a critical one. Many privacy-aware access control models [1, 11, 13] and specifications [10, 5, 18] have been proposed. Especially the most recent Purpose-Base Access Control model (PBAC) [4, 2] and Micro views [3] have been developed as feasible models and experiments have demonstrated their efficiency. The core techniques, which are used in current models, include *query modification* and *privacy labelling relational* (PLR) data models derived from MLR [14]. However, the two approaches lead to lower performance essentially: PLR increases disk IO and requires extra computing resources for relevant labels; while query modification techniques rewrite a user's queries by appending extra predicates or nested queries, which increases optimizing time, and probably leads to an inefficient executing plan.

An ideal solution to the privacy preservation problem would flexibly protect donor sensitive information without privacy violation, and would incur minimal

[1] The work is supported by both National Basic Research Program of China, Project No.2002CB312006 and National Natural Science Foundation of China, Project No.60473077.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 102–114, 2006.

privacy enforcing overhead when processing queries. Motivated by this requirement, we propose a new technique, which avoids using both the query modification and PLR data models, to support privacy access control based on the concept of purposes and the theory of partial indices. In our mechanism, we develop a notion of Privacy Aware Partial Index (PAPI), by which privacy policies (i.e. intended purposes in PBAC) are stored and enforced efficiently. We also extend *Purpose Trees* in PBAC into flatten purpose trees with addressing restrictions on *Purpose Trees* in PBAC. Our experimental results verify the feasibility and efficiency of our model.

The rest of our paper is organized as follows. In Section 2, we summarize the recent achievements in the privacy enhancing techniques pertain to purposes. In Section 3, by extending the general partial index, we develop two notions which underlie our research work. In next section, we illustrate how to maintain PAPIs, and how to organize and manage intended purposes based on PAPI. In Section 5, based on PAPI, we extend the traditional query processing engine to provide privacy enhancement. In Section 6, we introduce how to implement our PAPI mechanism, and describe some experiments that demonstrate the efficiency and scalability of our approach. Finally, we conclude the paper and outline future work.

## 2 Related Work

Privacy protection is related to many different areas in secure data management. As described in Common Criteria (CC) [12], to implement a solid privacy preserving data management system, we have to support at least the following three security requirements: Access Control, Unobservability and linkability. According to the above privacy preserving requirements, privacy enhancing techniques can be classified into three categories: Privacy Aware Access Control (PAAC) [4, 2, 11], Private Information Retrieval (PIR) [8] and Privacy Information Inference Control (RIIC) [7]( *k-anonymity* technique [17] belongs to this category).

Our work focuses on the PAAC technique for access control. The most recent popular techniques in this field include privacy policy specification [10, 5, 18] , purpose specification and management [1, 4, 18] and privacy polices enforcement models [4, 3]. In addition, we have used the following three core concepts in our work: partial indices [16], query optimizer [15] and executing engines [9].

The Platform for Privacy Preferences(P3P) [18] by W3C enables users to gain more control over the use of their personal information on web sites they visit. And also, APPEL [5] by W3C and EPAL [10] by IBM provide a formal way to define the privacy policies or usage preferences, but without detailed specifications to enforce the policies in an information system or product, such as DBMS.

Based on HDB [1], LeFevre etc. [11] presented a database architecture for enforcing limited disclosure expressed by privacy polices based on their proposed ten principles. They also suggested an implementation based on query modification techniques. By extending the concepts of purposes in [18], Ji-Won Byun

etc. [4, 2] presented a comprehensive approach to purpose management (called purpose-based access control, PBAC for short), in which all the purposes are organized in a hierarchical way. In the PBAC model [4], they developed three basic concepts: intended purposes, access purposes and purpose compliance. Based on these concepts, they suggested an implementation for PBAC based on the query modification technique and PLR derived from MLR [14]. In [2], extended their previous work in [4] to the XML-oriented information system and Object-oriented system, and proposed a systematical model to determine the access purposes based on RBAC. In [3], Ji-Won Byun etc. go even further on the purpose-based access control (PBAC) with incorporating generalization techniques to enhance the privacy preservation.

As drawn from the above models, some open issues are listed below, which have motivated us to seek more feasible and efficient solutions to enforce privacy policies.

• **Query Modification Techniques.** Query rewriting always changes the original queries by introducing some extra predicates or the nested queries, which increase optimizing time and require extra computing time.

• **Privacy Labelling Relational Data Model (PLR).** PLR is adopted in many models, which changes the standard relational data model by adding the privacy labelling attributes [4] or choice columns [11]. This strategy will actually increase the I/O cost and involve considerable extra computation against the labels. Especially if there are many privacy-sensitive attributes in a relation, the performance will be degraded drastically.

• **Hierarchy Relationship Among Purposes.** In [4], it is assumed that all the purposes are predetermined. However, this assumption does not always hold, especially in small organizations (which are project-oriented). The organizational structures will change frequently in these project-oriented organizations, leading the purpose tree being reshaped repeatedly because of the hierarchical relationships among purposes. As a result, all intended purposes in the relation have to be re-evaluated too. In addition, a non-leaf purpose is combinational purpose, which consists of multiple nested purposes. If a donator allows her/his information accessed for a purpose, then the information can also be used for its nested purposes. So donators should have knowledge of the purpose tree or the organizational structure. Obviously, it is very inconvenient for users to protecting their sensitive data.

To address the above challenges, we have developed a privacy-aware query processing mechanism based on partial indices to provide an efficient solution to the above problems. In our model, we first adopt from [4] three concepts, namely *purpose*,*intended purpose* and *access purpose*. We then avoid using the query modification technique and discard the PLR data model. Finally, by transforming a purpose tree $\mathcal{PT}$ [4] to a flatten purpose tree ($\mathcal{FPT}$) as shown in Fig.1, all purposes in $\mathcal{FPT}$ are peers without hierarchical relationships. Thus DBA can easily change purposes and reshape $\mathcal{FPT}$ without re-evaluating all intended purposes.

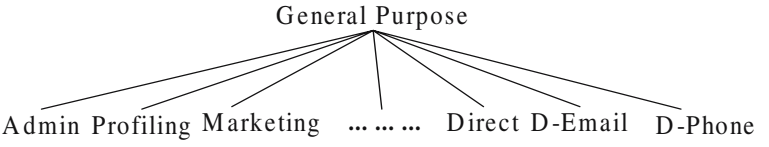


**Fig. 1.** Flatten Purpose Tree

Based on the concept of $\mathcal{FPT}$, relevant concepts in [4], including purpose & purpose tree, intended purpose, and access purpose, are redefined.

**Definition 1.** *(Purpose and Flatten Purpose Tree) A purpose describes the reasons for what data is collected or used. Purpose are organized in a flatten hierarchical structure, referred to Flatten Purpose Tree ($\mathcal{FPT}$). Actually, all the purposes are peer except the root purpose which is a virtual purpose.*

**Definition 2.** *(Intended Purpose) Let $\mathcal{FPT}$ be a flatten purpose tree and $\mathcal{P}$ be the set of all purposes in $\mathcal{FPT}$. An intended purpose, denoted by* IP*, is used to describe usage granted by the donators, and it is $\{AIP - PIP\}$, where $AIP \subseteq \mathbf{P}$ is a set of allowed intended purpose, and $PIP \subseteq \mathcal{P}$ is a set of prohibited purpose.*

**Example 1.** Suppose $AIP = \{Admin, Direct\}, PIP = \{D\_Email\}$ is defined over $\mathcal{FPT}$ given in Fig.1. Then, intended purposes (IP) is evaluated as: $IP = AIP - PIP = \{Admin, Direct\}$.

**Definition 3.** *(*Access Purpose*) Let $\mathcal{FPT}$ be a flatten purpose tree. An access purpose(**AP**), is the purpose for accessing data, and it is included in $\mathcal{FPT}$.*

**Definition 4.** *(Access Purpose Compliance) Let $\mathcal{FPT}$ be a flatten purpose tree, Let* IP *and* AP *be intended purpose and access purpose respectively. AP is said to be compliant with IP, only if the following condition is satisfied: $AP \in IP$.*

## 3 Purpose Aware Partial Indices

The concept and performance of partial indices are surveyed in [16], which illustrated that partial indices would lead to a great improvement of performance in many scenarios, especially in distributed system. In traditional indexing schemes, some of the columns are not indexed. A partial indexing scheme takes this one step further, and some of the tuples are not indexing into the indexes.

In this paper, we propose a new approach to take privacy preservation techniques further, called privacy-aware query processing based on the privacy aware partial index (PAPI) which is an extension to the generalized partial index [16] and is mainly used to store intended purpose and enforce privacy policies. For complicated features in SQL (like, conjunctive or disjunctive *Selection*), we propose two kinds of PAPI to enforce the privacy policies efficiently: attribute-oriented PAPI (APAPI) and tuple-oriented PAPI (TPAPI). And TPAPI is mainly used to process non-conjunctive queries. Based on PAPI, our model can support

both tuple-level and element-level privacy access control, and partial results for queries are supported,too.

**Definition 5.** (Attribute oriented Privacy Aware Partial Index, APAPI) *Let* $\mathbb{R} = \{A_1, A_2, \ldots, A_n\}$ *be a relation,* $\mathcal{P}$ *be the set of all purposes in* $\mathcal{FPT}$*, and* $P_k(\in \mathcal{P})$ *be a purpose. An* **APAPI(**$A_i, P_k$**)** *is a partial index defined on* $A_i$*, and index the tuple* $t_j(a_{j1}, a_{j2}, ..., a_{ji}, ..., a_{jn})$ $(t_j \in \mathbb{R})$ *if and only if* $P_k \in IP_{ji}$ *is held* ($IP_{ji}$ is the intended purpose on $a_{ji}$ in $t_j$).
*An APAPI*$(A_i, P_k)$*is defined as* $< key, CI, TID >$*, where:*

- ***key*** *is the element* $a_{ji}$ *for* $A_i$ *which APAPI*$(A_i, P_k)$ *is defined on.*
- ***Compliance Indicator****(CI) is a bit string* $\{b_1 b_2 \ldots b_n\}$*,the size of CI is determined by the number of attributes in* $\mathbb{R}$*. Given a tuple* $t_j$*,* $a_{jh}$ *is an element in* $t_j$ *with the intended purpose* $IP_{jh}$*, then* $b_h(j = 1, \ldots, n)$ *is assigned according to the following rules:*
  - $b_h = 1$*, if and only if* $P_k \in IP_{jh}$ *is true, or there is no privacy protection requirement for* $a_{jh}$*;*
  - $b_h = 0$*, otherwise.*

  *CI is mainly used to support the partial result of a query, which is different from PBAC which filters out the whole tuples if any of its elements violate the privacy policies. In CI, there are some reserved bits used for new appending attributes in future.*
- ***TID*** *is the physical address locating a tuple uniquely and directly.*

**Table 1.** Personal Information Table : PI_Table

| TID | Name | N_L | Gender | G_L | Age | A_L |
|---|---|---|---|---|---|---|
| 1 | Jone | Marketing | Male | Marketing | 18 | Analysis |
| 2 | Smith | Profiling | Male | Analysis | 34 | Marketing |
| 3 | Alice | Marketing | Female | Analysis | 18 | Marketing |
| 4 | Vincent | Third Party | Male | Marketing | 29 | Third Party |

**Example 2.** Given (PI_Table) which records the personal information and will be as an example throughout this paper. An APAPI, denoted as **Name_APAPI**, is created on the attribute *Name* for the purpose *Marketing*, and the last five bits in its CI are the reserved bits. The result is shown in the Table 2.

**Table 2.** APAPI on PI_Table

| Name | N_CI | TID |
|---|---|---|
| Jone | 110 00000 | 1 |
| Alice | 001 00000 | 2 |

**Table 3.** TPAPI on PI_Table

| N_CI | TID |
|---|---|
| 110 00000 | 1 |
| 101 00000 | 3 |
| 010 00000 | 4 |

As we know, in traditional DBMS, it is difficult for an optimizer to choose an index access method to access the relation on which has disjunctive selection predicates, except few combinational indices. And in our model, the privacy

policies are enforced in privacy aware query optimizer firstly by choosing optimal and suitable PAPIs to access the relevant relation. So, when there exists an disjunctive conditions on the base relation, the model probably fail to enforce the privacy policies because the optimizer can not choose APAPIs to access this relation under this scenarios. Fortunately, we introduce TPAPI, whenever we can choose the TPAPI as the access method to a relation. Because a TPAPI is independent from any attributes.

**Definition 6.** (Tuple oriented Privacy Aware Partial Index, TPAPI) *Let* $\mathbb{R} = \{A_1, A_2, \ldots, A_n\}$ *be a relation,* $\mathcal{P}$ *be a set of purposes in* $\mathcal{FPT}$*, and* $P_k (\in \mathcal{P})$ *be a purpose. A* **TPAPI(**$P_k$**)** *is a partial index to index the tuple* $t_j(a_{j1}, a_{j2}, ..., a_{ji}, ..., a_{jn})$ $(t_j \in \mathbb{R})$ *if and only if there exists any element* $a_{ji}$ *satisfying* $P_k \in IP_{ji}$*(*$IP_{ji}$ is the intended purpose on $a_{ji}$ in $t_j$*),at least. A TPAPI(*$P_k$*)is defined as* $< CI, TID >$*, where:*

- ***Compliance Indicator****(CI) is same to the counterpart defined in APAPI*
- ***TID*** *is the physical address locating a tuple uniquely and directly.*

**Example 3.** The TPAPI, denoted as ***Name*_TPAPI**, is created on PI_Table for the purpose *Marketing*, and the result is shown in the Table 3.

## 4 PAPI Maintenance and Intended Purpose Management

This section focuses on how to collect and store user data and designated privacy policies (i.e. purposes) by donators. When users request some services, necessary data is collected by terminals, like Web Browsers, according to P3P [18]. Users provide the necessary personal information, and specify the usage type (intended purpose, short for **IP**), thus the tuple $< data, IP >$ is formed and transported into the back-end privacy aware DBMS. Then, *data* is inserted into the relation, and update the PAPIs according to *IP*.

### 4.1 Establishing Basic PAPIs

The objectives of PAPI include: 1) store the privacy policies **completely**; 2) improve the processing performance of privacy-oriented query (in which users designate access purposes) based on the feature of indices. And the former is a fundamental and indispensable objective. We should ensure the completeness for the privacy policies in PAPIs. It is equal to the question: how many PAPIs should be created to store the privacy policies completely, or at least?

• **PAPI Completeness.** Given $\mathcal{P}$ is the set of purposes in $\mathcal{FPT}, \mathcal{P} = \{P_1, P_2, \ldots, P_m\}$, and a relation $\mathbb{R}$ defined as: $\mathbb{R} = \{A_1, A_2, \ldots, A_n\}$.

**Theorem 1.** *Given* $\mathcal{P}$ *and* $\mathbb{R}$*, and considering the* APAPI *only. For an attributes* $A_i$*, at least* $\boldsymbol{m}$ APAPIs *have to be defined on it for m purposes respectively to ensure the completeness.*

**Proof**. All the possible purposes are defined in $\mathcal{FPT}$. Give an element $elem_i$ in the tuple $\mathcal{T}$ for $A_i$ , if its intended purpose ($IP_i$) satisfies: $IP_i \in \mathcal{P}$, there must exist an entry for $\mathcal{T}$ in the APAPI($A_i, P_k$). If there is only (m-1) APAPIs are defined, then there must exists a purpose $P_h$ on which an APAPI ($A_k, P_h$) is not defined. So if there certainly exists a tuple $\mathcal{T}'$ in which $elem'_i$ for $A_i$ can used for $P_h$, then $elem'_i$ will lose the privacy policy for $P_h$ because of absenting APAPI($A_k, P_h$). So, we need to define m APAPIs on each purpose, at least.

Therefore, according to the Theorem 1, for a relation $\mathbb{R}$ with $n$ attributes, we have to create $n * m$ APAPIs totally to ensure the privacy completeness.

**Theorem 2.** *Given $\mathcal{P}$ and $\mathbb{R}$, and considering the TPAPI only. For $\mathbb{R}$, **m** TPAPIs need to be defined for m purposes respectively to ensure the completeness.*

**Proof**. According to the definition for TPAPI, and given a TPAPI($P_i$), if a tuple $\mathcal{T}$ including any element used for $P_i$ at least, then there must exist an entry for $\mathcal{T}$ in TPAPI($P_i$). Obviously, if TPAPI($P_k$) is not created, then tuple for $P_k$ cannot be located by TPAPIs. So $m$ TPAPIs have to be maintained for m purposes respectively.

For considering the complexity of SQL syntax and diversity of applications, we maintain APAPI and TPAPI simultaneously to facilitate different kinds of queries and applications.

So, given $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathcal{FPT}$, and a relation $\mathbb{R}$ with n attributes, we have to create n*m APAPIs, and m TPAPIs. These are demonstrated in a matrix (X-coordinate for attributes, Y-coordinate for purposes) in Fig.2.

| | | $A_1$ | $A_2$ | $A_3$ | ... | $A_{N-1}$ | $A_N$ |
|---|---|---|---|---|---|---|---|
| PAPI | TPAPI | APAPI | | | | | |
| $P_1$ | $TPAPI_1$ | $APAPI_{1,1}$ | $APAPI_{1,2}$ | $APAPI_{1,3}$ | ... | $APAPI_{1,N-1}$ | $APAPI_{1,N}$ |
| $P_2$ | $TPAPI_2$ | $APAPI_{2,1}$ | $APAPI_{2,2}$ | $APAPI_{2,3}$ | ... | $APAPI_{2,N-1}$ | $APAPI_{2,N}$ |
| $P_3$ | $TPAPI_3$ | $APAPI_{3,1}$ | $APAPI_{3,2}$ | $APAPI_{3,3}$ | ... | $APAPI_{3,N-1}$ | $APAPI_{3,N}$ |
| ... | ... | ... | ... | ... | ... | ... | ... |
| $P_{M-1}$ | $TPAPI_{M-1}$ | $APAPI_{M-1,1}$ | $APAPI_{M-1,2}$ | $APAPI_{M-1,3}$ | ... | $APAPI_{M-1,N-1}$ | $APAPI_{M-1,N}$ |
| $P_M$ | $TPAPI_M$ | $APAPI_{M,1}$ | $APAPI_{M,2}$ | $APAPI_{M,3}$ | ... | $APAPI_{M,N-1}$ | $APAPI_{M,N}$ |

**Fig. 2.** PAPI matrix for all basic PAPIs

### 4.2 Updating PAPIs

Updating PAPIs occurs in the following three cases: 1) update the relation $\mathbb{R}$, including insertion, update, and delete; 2)update the privacy policies on data by donators; 3) update the flatten purpose tree $\mathcal{FPT}$. We discuss how to adjust the PAPIs to these changes.

Assuming that $\mathbb{R}$ contains $n$ attributes, $\mathcal{P}$ includes $m$ purposes, and each data item can only used for one purpose (actually, one intended purpose may contain several purpose).

- **Updating Relation.** When a relation $\mathbb{R}$ is updated, associated PAPIs are updated simultaneously, as general indices do.
- **Updating Privacy Policies on Data.** As a flexible privacy preserving model, it has to facilitate donators to modify their privacy policies efficiently and conveniently. In our model, when donators change privacy policies on an element, four PAPIs have to be updated,(including two TPAPIs and two APAPIs respectively).
- **Updating $\mathcal{FPT}$.** If security administrator has to remove a purpose, they just need to remove the relevant row in the matrix shown in Fig.2. Taking the purpose $P_2$ for example, if $P_2$ is deleted, only the second row in the matrix is deleted without any influence to the rest PAPIs. When a new purpose is appended, only a new row is appended into the matrix.

### 4.3 Analysis of PAPI

The analysis focuses on PAPI selectivity which is defined as the number of entries in each PAPI and its storage cost. The smaller the selectivity is,the faster it will accelerate query processing. The analysis is based on the following assumptions:

- Probability of an element used for a purpose is equal.
- Average number of intended purposes for each cell is denoted as $\boldsymbol{l}$.
- $\mathbb{R}$ contains $\boldsymbol{n}$ attributes and $\boldsymbol{c}$ tuples, and $\mathcal{P}$ includes $\boldsymbol{m}$ purposes.

**Selectivity**
According to the above assumptions, given APAPI($A_i, P_i$) and a tuple $t_j(a_{j1}, a_{j2}, \ldots, a_{jn})$ , the probability of $a_{ji}$ used for $P_i$ is $l/m$. So the probability of indexing $t$ in APAPI($A_i, P_i$) is also $l/m$, and there are $(l \times c)/m$ entries on average for APAPI($A_i, P_i$).

As we know that the probability of $a_{ji}$ used for $P_i$ is $l/m$, while there are $n$ elements in a tuple $t$, and if all the attributes are independent from each other, thus the probability of indexing $t$ in TPAPI($P_i$) is $(n \times l)/m$ according to the definition of TPAPI. So, there are $(n \times l) \times c/m$ entries for TPAPI($P_i$) on average.

**Storage Overhead**
According to the above analysis, for an APAPI, there are $(l \times c)/m$ (c is cardinality of $\mathbb{R}$) entries. So, for $n \times m$ APAPIs, there are $(n \times m) \times (l \times c)/m$ entries in all. And also there are $(n \times l) \times c/m$ for each TPAPI, thus for $m$ TPAPIs, there are $m \times (n \times l) \times c/m$ entries in all. So, in a privacy-aware DBMS based on PAPI mechanism, the number of all the indices entries in PAPIs (both APAPI and TPAPI) sums to $(n \times m) \times (l \times c)/m + m \times (n \times l) \times c/m = 2(n \times c) \times l$. While, in $\mathbb{R}$, there are $n * c$ data cells, so the whole extra storage overhead is $2 \times l$ times as the original table. So, the storage overhead is attributed to $l$, while $l$ is due to the specific applications.

From the above analysis, the selectivity for TPAPI on average is n times lower than it for APAPI. So, the retrieval performance of TPAPI may be greatly lower than that of APAPI theoretically.

## 5 Privacy-Aware Query Processing Engine

This section mainly focuses on how to enforce privacy policies in query processing engine. And we assume that user's access purposes are authenticated. As we known, a general query processing engine consists of two modules: optimizer and executor. So, our efforts focus on extending existing optimizer and executor to enhance privacy policies with incorporating PAPI mechanism.

### 5.1 Privacy Aware Query Optimizer (PAQO)

As described in [15, 9], the mechanism for a general query optimizer facility can formulated briefly as: 1) choose two optimal access methods for each base relation firstly. One is the cheapest access method returning tuples in an interesting order, another one is the cheapest access method without considering order; 2) choose an overall optimal path (mainly considering the join orders), which is generated by combining these optimal access methods based on dynamic planning algorithm, greedy algorithm, or genetic algorithm etc. Then, the optimal path is passed into the executor in which tuples are processed one by one.

According to principles of the optimizer, in PAQO, the optimal access methods on each relation are restricted within these PAPIs which are created for the access purpose indicated in users' queries. So, when a plan is determined, privacy policies are enforced at tuple level, because accessible tuples are pre-determined by PAPIs. However, we don't know which attributes can be accessed. This problem can be solved by the compliance indicator (**CI**) in PAPIs, **CI** need to be checked in executor when accessing cells in tuples. That's why we extend the existing query executor.

### 5.2 Privacy Aware Query Executor (PAQE)

Through PAQO, the coarse privacy policies (**PP**) have been enforced. The fine-grained PP enforcement is left for executor to check the compliance indicator (CI) further and return the suitable partially incomplete result.

Without considering the inference violation, the fine-grained PP is enforced according to below rules(called Loose Rules):

- Let $t_j\{a_{j1}, a_{j2}, \ldots, a_{jn}\}$ be a tuple, and $CI_j(b_{j1}b_{j2}\ldots b_{jn})$ is the **CI**.
- If $b_{ji}(CI) = 1(i = 1, \ldots, n)$, output the $a_{ji}$ if necessary;
- If $b_{ji} = 0$, replace $a_{ji}$ by NULL.

**Example 4.** Display the personal information used for *Marketing*.
$Q_2$: SELECT * FROM PI_Table WHERE Name='Jone' And Age=18;
The result for $Q_2$ is displayed as in the **Table 4**.

**Table 4.** Result under Loose Rules

| Name | Gender | Age |
|---|---|---|
| Jone | Male | NULL |
| Jone | NULL | 18 |

**Table 5.** Result under Strict Rules

| Name | Gender | Age |
|---|---|---|
| Jone | NULL | 18 |

There is an inference violation from the result of $Q_2$ in Table 4. The $Q_2$'owner can easily infer that age for Jone in the first row is also 18 and its intended purposes do not include *Marketing*.

To avoid this inference disclosure, we find out selective attributes (SA) which are used to filter out unqualified tuples in the relation $\mathbb{R}$. With considering the inference control, based on *Loose Rules*, **Strict Rules** for the fine-grained PP enforcement are developed:

- Let $t_j\{a_{j1}, a_{j2}, \ldots, a_{jn}\}$ be a tuple, and $CI_j(b_{j1}b_{j2}\ldots b_{jn})$ is the **CI**.
- If $b_{ji}(CI) = 1(i = 1, \ldots, n)$, output the $a_{ji}$ if necessary;
- If $b_{ji} = 0 \bigwedge A_i \notin SA$, replace $a_{ji}$ by NULL.
- If $b_{ji} = 0 \bigwedge A_i \in SA$, $t_j$ is discarded directly.

According the strict PP enforcing rules, $SA$ for $Q_2$ is { Name, Age}, and the result for $Q_2$ is shown in **Table 5**.

To summarize, privacy policies are enforced through two steps: coarse PP enforcement which is fixed in privacy aware query optimizer (PAQO) and fine-grained PP enforcement which is fixed in privacy aware query executor (PAQE).

## 6 Implementation and Experiments

### 6.1 Implementation on PostgreSQL

The implementation mainly involves the below aspects: purpose management, PAPI maintenance, and extending query optimizer and executor.

Purposes are stored in a system catalog (called *Pg_Purpose*) which is created as a part of data dictionary (DD) when database is installed initially. DBAs can create new purpose by extended DDL. And each purpose has a unique code.

In reality, purposes, in our implementation, can be considered as special privileges, and it will be granted or revoked as general privileges do. The access purposes are granted to users firstly. If a user intends to access data for a purpose, DBMS checks the access purpose against allowed access purposes in **ACL**. If the access purpose is allowed, then the request is processed; else, rejected directly.

Basic PAPIs are created automatically, when a relation is created, and automatically establish dependency with target relations. The command for PAPIs is extended from the standard command for the index. Maintenance for PAPIs follows the paradigm for general indices. Besides, all PAPIs have to establish the dependency on *Pg_Purpose*, too. When any updates on $\mathcal{FPT}$, cascading actions will take effect on the corresponding PAPIs.

For the privacy aware query optimizer (PAQO), we only need to add a new branch to support the privacy feature without influencing its original function. When users submit privacy-oriented queries, the optimizer chooses optimal path from those PAPIs defined for the access purpose.

According to PAQO, all tuples, which are indexed in PAPIs, can be accessed for privacy-oriented queries. And PAQE enforces **strict rules** on cell level by filtering out the unqualified tuples or suppressing tuple cells.

### 6.2 Experimental Evaluation

The goal of our experiments is to investigate the feasibility and performance of our mechanism. We mainly focus on comparing the performance against the two traditional techniques, namely query modification techniques, labelling schema (such as element-based labelling PBAC model), varying the number of attributes accessed and privacy policies selectivity. Sequentially, we demonstrate the relationship between the performance of our model and different cardinalities. Finally, we analyze the storage overhead of our model.

**Experimental Setup.** The main experimental environment is configured as below: CPU is Intel P4C with 1G DDR-400, Redhat Linux AS 4.0 is installed on the machine, and PostgreSQL-7.43 is used as the RDBMS. Our PAPI mechanism is implemented by extending the PostgreSQL-7.43, while the element-base PBAC model is simulated: each attribute is entailed with two extra labels with the type *smallint*, and the privacy policies enforcement function (i.e. Comp_check(AP, AIP, PIP)) is substituted by a simple predicate involving the two corresponding labels. The tested data set is a version of large size tuples schema used in [6] .

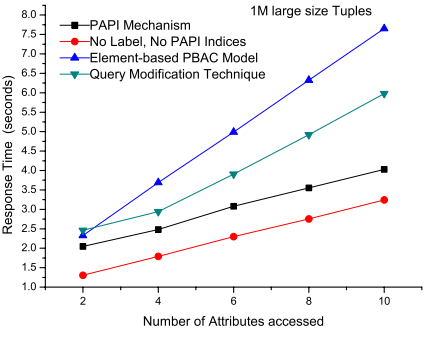


**Fig. 3.** Mechanisms VS Performance

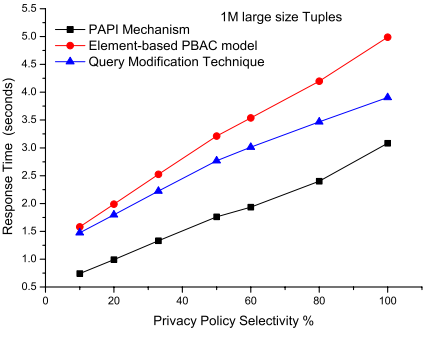


**Fig. 4.** Selectivity and Performance

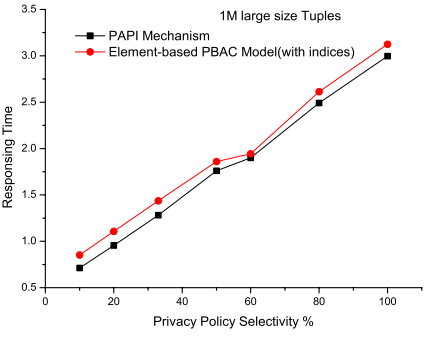


**Fig. 5.** PAPI and Element-based PBAC

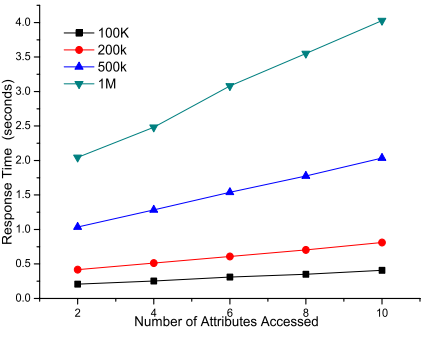


**Fig. 6.** Cardinality and Performance

Response time is used as the metric to measure the performance. In our experiments, the response time is referred to the retrieval time and a trivial counting time by evaluating the standard aggregate function COUNT(). A sample of queries used in the experiments is:

SELECT COUNT(unique1),COUNT(two),COUNT(unique2),
COUNT(four) FROM mtuples5;
Before we execute the target testing cases, we load the data set into memory as possible as we can, and each testing case is run for ten times.

**Experimental Results.** In order to compare the performance of different techniques, firstly we assume the selectivity of all data elements (for both actual attributes and purpose labels) to be 100 percent, but vary the number of attributes accessed in each query. The result shown in Fig. 3 demonstrates that PAPI gains the best performance, and has a significant improvement against the query modification technique and element-based PBAC model. Actually PAPI mechanism introduces some extra overhead against the standard relational data model (without any labels and PAPI's indices), because the in PAPI mechanism, it has to access the PAPI's indices to locate every tuples whatever method the tuples are accessed. Fig. 4 shows the performance of different techniques in case of different privacy policy selectivity (i.e. the selectivity of data elements for actual attributes is still 100 percent, but the selectivity of data elements for purpose labels is variable), varying from 10 percent to 100 percent, and all testing queries access the same six attributes. We learn that PAPI mechanism gains the best performance in any selectivity, too.

As we know that element-base PBAC model will achieve a drastic improvement on performance based on the functional index by pre-computing the given function which is used to enforce the privacy policies. Fig.5 shows the comparison between PAPI mechanism and element-base PBAC model(with indices). And from the experimental results, PAPI mechanism also has a little better performance, because in element-base PBAC model, each attribute are entailed with two labels for AIP and PIP respectively, and thus the size of each tuple is enlarged, directly leading to the longer accessing time in both memory and disks. If using the functional index to accelerate PBAC model, it will achieve a significant improvement; unfortunately, it will not support the partially incomplete result which is acceptable in some application environment.

Also, we investigate the scalability of our PAPI mechanism by considering different cardinalities. Fig.6 shows that the query processing cost merely increases in a linear way against the cardinality.

## 7 Conclusions

In this paper, we summarize the recent achievements on purpose-based privacy enhancing techniques. Motivated by some problems in these techniques, we propose a PAPI mechanism to facilitate the privacy policies enforcement by extending the traditional query processing engine based on two concepts: at-

tribute oriented privacy-aware partial index and tuple oriented privacy-aware partial index. And intended purposes are efficiently maintained by PAPIs. Finally, through experiments, the feasibility and performance of PAPI mechanism are demonstrated. We plan to design a privacy-aware DBMS based on PAPI mechanism with incorporating other techniques, such as suppression, generalization and so on.

## References


1. R. Agrawal, J. Keirnan, R. Srikant, and Y. Xu. Hippocratic database. *In Proceedings of the 28th VLDB Conference.*, 2002.
2. J.-W. Byun, E. Berino, and N. Li. Purpose based access control of complex data for privacy protection. *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT'05)*, pages 102–110, June 2005.
3. J.-W. Byun and E. Bertino. Vision paper: Micro-views, or on how to protect privacy while enhancing data usability. *To be published in SIGMOD Record.*, 2005.
4. J.-W. Byun, E. Bertino, and N. Li. Purpose based access control for privacy protection in relational dtabase systems. *Technical Report 2004-52,Purdue Univ,2004.*
5. W. W. W. Consortium(W3C). A p3p preference exchange language 1.0 (appel 1.0). available at. *www.zurich.ibm.com/security/enterprise-privacy/epal.*
6. D.Bitton, D.J.DeWitt, and C.Turbyfill. Benchmarking database: system a systematic approach. *In Ninth International Conference on Very Large Data Bases*, pages 8–19, Oct 1983.
7. D.Bitton, D.J.DeWitt, and C.Turbyfill. Benchmarking database system a systematic approach. *In Proceeding of CCS'04.*, pages 25–29, Oct 2004.
8. W. Gasarch. A survey on private information retrieval. *The Bulletin of the EATCS*, 82:72–107, 2004.
9. G. Graef. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73 – 169, June 1993.
10. IBM. The enterprise privacy authorization language (epal). available at. *www.w3.org/TR/P3P-preferences.*
11. K.LeFevre, R.Agrawal, V.Ercegovac, R.Ramakrishnan, Y.Xu, and D.DeWitt. Limiting disclosure in hippocratic database. *In The 30th International Conference on Very Large Databases*, Aug. 2004.
12. T. C. C. P. S. Organisations. Common criteria for information technology security evaluation, part 2, draft version 3 and version 2.1-2.3, august. 2005.
13. P.Ashley, C. Powers, and M.Schunter. Privacy, access control, and privacy management. *In Third International Symposium on Electronic Commerce*, 2002.
14. R. Sandhu and F. Chen. The multilevel relational(mlr) data model. *ACM Transactions on Information and System Security.*, 1(1):93–132, November 1998.
15. P. Selinger, M.M.Astrahan, D.d.Chamberlin, R.A.Lorie, and T.G.Price. Access path selection in a relational dababase management system. *In Proceedings of the 1979 ACM SIGMOD Conference on the Management of Data.*, May-June 1979.
16. P. Seshadri and A. Swami. Generalized partial indexes. *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, pages 420 – 427, Mar. 1995.
17. L. SWEENEY. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty,Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.
18. W. W. W. C. (W3C). Platform for privacy preferences (p3p). *Available at www.w3.org/P3P.*