Technical Report CSD TR #06-022

REALIZING PRIVACY-PRESERVING FEATURES IN HIPPOCRATIC DATABASES

By Yasin Laura-Silva and Walid G. Aref

Purdue University
Purdue University, West Lafayette, IN 47907

Realizing Privacy-Preserving Features in Hippocratic Databases

Yasin Laura-Silva Walid G. Aref
Purdue University
{ylaurasi, aref}@cs.purdue.edu

Abstract

Preserving privacy has become a crucial requirement for operating a business that manages personal data. Hippocratic databases have been proposed to answer this requirement through a database design that includes responsibility for the privacy of data as a founding tenet. We identify, study, and implement several privacy- preserving features that extend the previous work on Limiting Disclosure in Hippocratic databases. These features include the support of multiple policy versions, retention time, generalization hierarchies, and multiple SQL operations. The proposed features facilitate in making Hippocratic databases one step closer to fitting real-world scenarios. We present the design and implementation guidelines of each of the proposed features. The evaluation of the effect in performance shows that the cost of these extensions is small and scales well to large databases.

1. Introduction

Privacy preservation is an important requirement when personal data is collected, stored and published. One of the main challenges is to share information while complying with the data-owner privacy preferences. In recent years, several research directions have received substantial attention including Hippocratic databases, anonymization and generalization, privacy-preserving data mining, privacy rules languages, e.g., P3P and EPAL and fine-grained access control techniques in discretionary and mandatory access control.

The notion of Hippocratic databases was introduced to incorporate privacy protection as a founding tenet in relational database systems [1] [2] [3] [9]. Ten guiding principles of Hippocratic databases and initial designs to provide limited disclosure and compliance audition were introduced. One key element of the Hippocratic database architecture is that it makes use of a centralized and standardized definition of privacy rules via a privacy policy. A privacy policy usually is born outside the database system and is expressed using natural language. In order to process this policy more effectively it is expressed using a standard privacy specification language, e.g., P3P [10] or EPAL [11]. The resulting version is translated into its Hippocratic database equivalent, i.e., the policy rules tables inside the database. The great value of this policy-driven approach is that companies that use the Hippocratic database have at their disposal an important tool to comply with privacy laws and guidelines, e.g., the Health Insurance Portability and Accountability Act (HIPAA), or the EOCD Guidelines in Europe.

Even though the previous work in the area of limiting disclosure in Hippocratic databases has discussed the main guidelines and proposed an initial architecture, there are still several problems that need to be addressed before a Hippocratic database can support efficiently the requirements in real-world systems. Among these problems are the inadequate support of policy retention time, the lack of support of policy versions that could allow a company to use several versions of a policy simultaneously, the lack of an effective and flexible way to ensure that users only use purposes and recipients that they are supposed to use, and a way to restrict the access not only for the SELECT operation but also for all the DML operations.

Along with Hippocratic databases, there has been a significant amount of research in the area of anonymization and generalization [4] [5] [6] [7]. The main goal is to transform a database table into its anonymized form that allows users to get useful information that does not single out data about individuals (owners of the data) who want their data to remain private. Two main notions of anonymization that have been proposed are: k-anonymity [4] [5] and l-diversity [6]. Although, both Hippocratic databases and anonymization are important areas in the effort to achieve effective mechanisms to ensure privacy in database systems, to the best of our knowledge, no much work has been done to integrate their results.

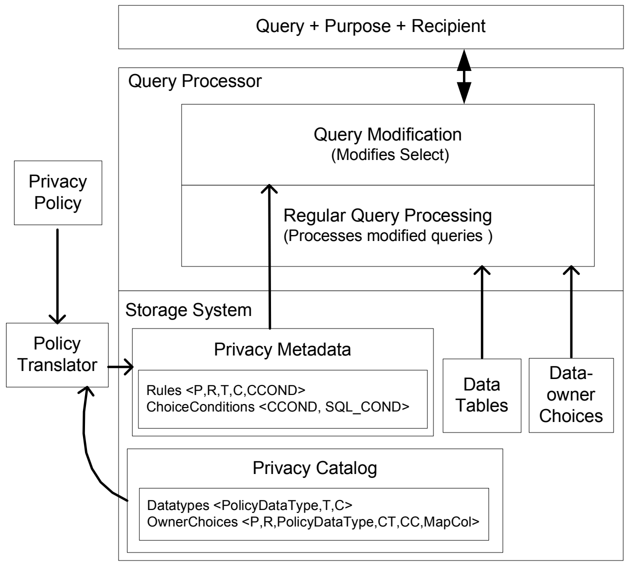

Figure 1: Unified original architecture for limiting disclosure

1.1. Contributions

We integrate the different design features related to limiting disclosure in Hippocratic databases proposed in previous work, and present a unified architecture to support limited disclosure. We take this unified architecture as our starting point to study various extensions. These extensions solve problems that are faced while implementing Hippocratic databases that support real-world privacy requirements.

The extensions covered are:

- Mapping purpose, recipient, and data type of a policy with database roles
- Support of multiple DML operations
- Support of retention time
- Support of policy versions
- Support of generalization hierarchies

We implement these extensions and present the study of their effect on database performance.

The rest of the paper is organized as follows. Section 2 presents the unified original architecture for limiting disclosure. Section 3 presents the realization of the various extensions cited above. Section 4 presents the evaluation of their effect in performance. Finally, Section 5 contains concluding remarks.

2. Unified original architecture for limiting disclosure

We integrate the design elements of previous work [2] [9] [1] into a unified architecture to support limited disclosure in Hippocratic databases presented in Figure 1. In this figure, P stands for purpose, R for recipient, PolicyDataType for data type of a P3P-like policy, T for table, C for column or attribute, CT for choice table, and CC for choice column. Furthermore, *data type* makes reference to the data categories used in a privacy policy, e.g., PatientDiseaseInfo, not to the regular database data types. The remaining part of this section explains the main components of this architecture.

- *Privacy policy*. The document that specifies how an organization, e.g., a company, can use data associated to the data owner. It states the purposes, recipients and retention time of the different pieces of data. A privacy policy is expressed using a privacy specification language, e.g., P3P [10] or EPAL [11]. In this work, we assume the use of a P3P-like language.

```
Select name, phone, address from PATIENT;
Purpose = Treatment; Recipient = Nurses
```

⇩

```
Select name, phone, address from
(Select pno,name, NULL AS phone,
        CASE WHEN
                EXISTS (select address_option from options_patient
                        where patient.pno=options_patient.pno
                        AND options_patient.addres_option=TRUE)
                THEN address ELSE NULL END
        AS address
From patient)
```

Figure 2: Example of query modification

Patient <pno, name, birth, sex, address, phone>
Drug <dno, drug_name>
DrugAdm <pno, dno, dosage, adm_period_begin,adm_period_end>
DiseasePatient <pno,dname>

Figure 3: Example database schema

- *Privacy catalog*. These tables drive the translation of the P3P-like policy into the database privacy policy. Table *Datatypes* stores the mapping between the data types used in the privacy policy and the database tables and attributes associated with them. Table *OwnerChoices* stores the table and attribute names where the individual opt-in/opt-out choices are stored for a combination of purpose-recipient-data type if a choice is available for this combination; this table is known as the choice table. The attribute *MapCol* in OwnerChoices is used to match each tuple in the table associated to the data type with the corresponding tuple in the choice table. For example, the attribute patient ID could be used to match each tuple in *DiseasePatient* (table associated to data type PatientDiseaseInfo) with the choice table *PatientChoices* that stores individual preferences.

- *Policy translator*. Translates the privacy policy expressed in the P3P-like language into the privacy metadata tables in the database.

- *Policy metadata*. It is the equivalent of the privacy policy inside the database. It contains the tables *Rules* and *ChoiceConditions*. Table Rules contains tuples of the form (P,R,T,C,CCOND); each tuple represents a rule that grants access to the table T and column C for the purpose P, and Recipient R. The optional condition CCOND restricts this access in case an opt-in/opt-out choice is available for that combination. Table ChoiceConditions stores the SQL statement (similar to WHERE condition statement) for each condition used in Rules.

- *Query modification*. Before execution, a query is modified into its privacy-preserving form: each table in the FROM clause is transformed into a privacy-preserving view that checks the privacy metadata rules and data-owner preferences. Figure 2 gives the result of modifying a query when the privacy policy does not allow access to the attribute Phone and only opt-in access over the attribute Address for the purpose Treatment and the recipient Nurses.

3. Extending the architecture for limiting disclosure

This section describes each of the extensions on the initial design for limited disclosure in Hippocratic databases introduced in section 2. The extensions are independent but are presented here incrementally. Figure 3 gives the database schema that is used in the examples.

3.1 Mapping purpose, recipient and data type of a policy with database roles

The initial design for limiting disclosure translates P3P-like rules of the form (purpose, recipient, data type, opt-in/opt-out condition) into database privacy rules of the form (purpose, recipient, table, column, choice condition). When a user issues a query we need to determine the *purpose* and *recipient* of this access. Purpose and recipient are elements used to specify privacy policies even in its natural language form; consequently, there is not necessarily a one-to-one mapping between recipients and database roles or users. The mapping will depend on the specific way users are organized and the relationships between the roles and the different entities that will receive the data.

There are different ways in which the purpose and recipient can be identified when a user issues a query: (1) The user could explicitly state the purpose and recipient along with the query; this requires trust on the users. (2) Dynamically infer the purpose and recipient from the context of the application [2]. A downside of this approach is that it is difficult to capture all possibilities. (3) Register every application or procedure with a purpose and recipient, which becomes a difficult task for complex applications and procedures. (4) The user specifies the purpose and the system validates it based on user attributes, e.g., active roles, job position and location [12].

We propose to use the relationship between purpose-recipient-data type and database roles during privacy policy translation. We accomplish this using an additional privacy catalog table *RoleAccess* that records this mapping. This approach is flexible enough to represent any relationship between the elements of a policy rule and the database roles associated to them. The mapping can be viewed as a way to specify the database roles that can access specific sections of the data using a particular combination of purpose and recipient. The policy translator gets the (purpose, recipient, data type) triplet from each P3P-like rule and creates a database privacy rule for each role associated with this triplet in RoleAccess. The database rule will have the following structure: (*DBRole*, purpose, recipient, table, column, choice condition). The query modification module considers only the rules defined for the roles of the user issuing the query and the purpose-recipient specified with this query. If a user is not allowed to use a certain combination of purpose-recipient, the query processing is terminated. This extension allows us to enforce the following example restrictions:

- User Mary should use only recipient Doctors while user Tom should use only recipient Nurses when accessing table Patients for the purpose Treatment.

- Given two database roles that are allowed to use purpose Treatment and recipient Doctors, e.g., doctors1 and sysadmin, allow sysadmin to access all the columns of table Patient, and doctors1 a subset of them.

With the extension described in the next section, we will be able to enforce restrictions like:

- Allow user Mary, using purpose Treatment and recipient Doctors, to access the table Drugs only to perform SELECT but not UPDATE.
- Given two database roles that are allowed to use purpose Treatment and recipient Doctors, e.g., doctors1 and sysadmin, allow sysadmin to perform SELECT and UPDATE over table Patient but only SELECT to doctors1.

3.2 Support of multiple DML operations

The original architecture for limiting disclosure ensures that access using the SELECT command will respect the privacy rules and user preferences. In this section, we extend the ideas used for SELECT to other DML operations, i.e., INSERT, UPDATE, and DELETE.

To support privacy restrictions for other DML operations, we extend the structure of the privacy catalog table RoleAccess to (P,R,PolicyDataType,DBRole, *Operations*). Operations is a bitmap in which each bit is associated to each DML operation (bit0=SELECT, bit1=INSERT, bit2=UPDATE, bit3= DELETE). When the value of a bit is 1 the operation is allowed, otherwise it is restricted. For example the tuples (Treatment, Nurses,DrugAdm,nurse,0001) and (Treatment,Nurses, DrugAdm,nurse-practitioner,0111), mean that if the privacy policy contains rules that give access to drug administration data for purpose Treatment and recipient Nurses, the database roles that should receive this access are nurse and nurse-practitioner, additionally the role nurse will receive only access to view the data while the role nurse-practitioner will receive access to view and modify it.

The policy translator will produce privacy rules of the form (DBRole,P,R,T,c,CCOND,*Operations*) and this information will be used when processing DML operations. The processing of the SELECT operation is similar to the one implemented in the original design. The main difference is that when the process requires checking if a rule has been defined for purpose P, recipient R, table T and column C, it also ensures that the operations granted with this rule include SELECT. For other DML operations, a privacy checking process is performed based on the algorithms provided in Figure 4. An operation can be allowed, denied or allowed with *limited effect*; in this last case, the effect of an update operation is restricted to the subset of the data to which a user has access to. As in previous work in limiting disclosure in Hippocratic databases, we use NULL to represent a prohibited value; the advantages and disadvantages of this use are presented in [2]. For the INSERT operation, we treat NULL as a special value that users can always insert independently of the privacy restrictions; this will allow a user who only has access to insert on certain columns of a table, to insert a tuple with values for these columns and NULL for the remaining columns. Naturally, if there is a column that is NOT NULL and the user

```
INSERT
Input: INSERT INTO t1 (col_list) VALUES (value_list)
For each column in col_list in which value_list[i]≠NULL
    status =checkPermission(purpose, recipient, dbRole, t1, col_list[i],
                            Insert, out conditionChoice)
     case status //0=prohibited, 1=allowed without condition,
                 //2=allowed without condition
        0: return -1
        1: break //continue with the next column
        2: If conditionChoice does not depend on t1
              Check if conditionChoice is fulfilled
Execute (unmodified) INSERT command
If operation was successful
    We insert in the choice tables that depend on t1

UPDATE
Input: UPDATE t1 SET col_1=newValue_1 [,...] WHERE conditions
translatedCols=""
For each column in col_list
    status =checkPermission(purpose, recipient, dbRole, t1, col_list[i],
                            Update, out conditionChoice);
     case status //0=prohibited, 1=allowed without condition,
                 //2=allowed without condition
        0: break //update will not affect this col
        1: //update will affect all rows of this col
           translatedCols += col_i + "=" + newValue_i + ","
           break
        2: //update will affect the allowed rows of this col
           translatedCols += col_i  + "=" + "CASE WHEN " +
           conditionChoice + " THEN " + newValue_i +
                                   " ELSE " + col_i  + " END,";
Execute "UPDATE " + t1 + " SET " + translatedCols +conditions;

DELETE
Input: DELETE FROM t1 WHERE conditions
col_list = set of all columns in t1
newConditions=""
For each column in col_list
    status =checkPermission(purpose, recipient, dbRole, t1, col_list[i],
                            Delete, out conditionChoice);
     case status //0=prohibited, 1=allowed without condition,
                 //2=allowed without condition
         case 0: return -1;// abort
         case 1: break;//there is access to the whole column
         case 2: //delete will affect the allowed rows of this col
                newConditions += conditionChoice + " AND ";
Execute "DELETE FROM " + t1 + conditions + newConditions;
If operation was successful
    Remove rows in choice tables that depend on t1
```

Figure 4: Algorithms for other DML operations

does not have access to insert on it, he will be unable to insert in this table. For UPDATE, the user needs to have access to all the columns being updated independently of the new values; the modified command will apply the changes only to those columns that the user has access to according to the privacy rules, and the rows he has access to according to the data-owner preferences. For DELETE, the user needs to have permission over all the columns of the table; additionally, the translated command will delete only the rows that the user has access to according to data-owner preferences. The resulting architecture after applying the modifications introduced in the two first extensions is presented in Figure 5. The new or modified components are in bold.

3.3 Support of retention time

Limited retention is a principle of Hippocratic databases and a key element of privacy policies. It ensures that data is retained only as long as necessary for the fulfillment of the purposes for which it has been collected. The original architecture of the Hippocratic database [1] suggests the implementation of the *Data Retention Manager* which basically deletes all data items that have outlived their purpose. The same work recognizes that completely forgetting some information once it is stored in a database without affecting recovery is non-trivial. To the best of our knowledge no further mechanism to support retention time was proposed in the context of Hippocratic databases.

Our approach to support retention time is similar to the one used to support opt-in/opt-out preferences. The advantage of this approach is that it does not require deleting the information after the allowed retention time. Additionally, using SQL conditions constitutes a flexible mechanism to express complex retention restrictions. P3P defines the element *Retention* as part of privacy rules. This element can have several predefined values: no-retention, stated-purpose, legal-requirement, business-practices, and indefinitely [10]. The time length associated to each of these values depends on the specific privacy policy and organization. Furthermore, for values, e.g., stated-purpose or legal-requirement, the time length can depend also on the purpose associated to each privacy rule. We store this mapping between P3P retention value, purpose and actual time length in the privacy catalog table *Retention.*

We assume there is a table, referred to as primary table, which stores basic information of the data owner and where each row is associated with exactly one data owner. Our support of retention time makes use of the *Signature-Date* table in which we store the policy signature date for each data owner. During policy translation, if the retention element is included in a P3P rule, the values of the retention and purpose elements are used to determine the retention time length *tl*. The translator also builds a condition that ensures that the date in which a command is executed falls in the period between the privacy signature date *sd*, which will probably be different for each data owner, and *sd+tl*. We store the reference to this condition in the new column *DCOND* of the table Rules and the actual condition in the table *DateConditions*. Figure 6

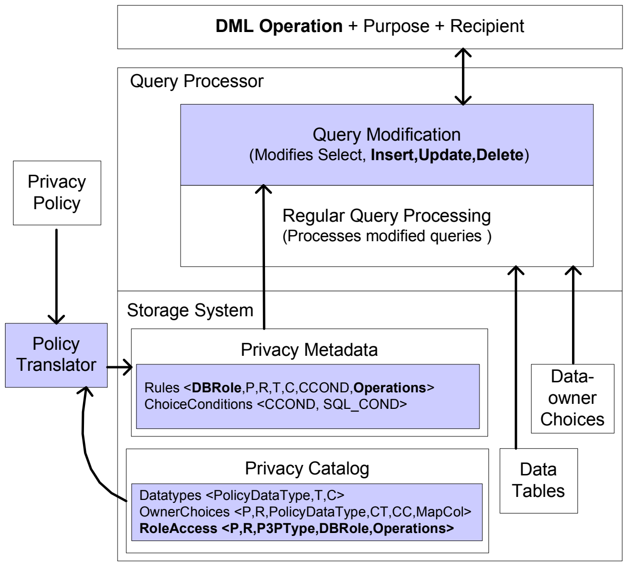
DML Operation + Purpose + Recipient
Query Processor
Query Modification
(Modifies Select, Insert,Update,Delete)
Regular Query Processing
(Processes modified queries)
Privacy Policy
Storage System
Policy Translator
Privacy Metadata
Rules <DBRole,P,R,T,C,CCOND,Operations>
ChoiceConditions <CCOND, SQL_COND>
Data-owner Choices
Privacy Catalog
Datatypes <PolicyDataType,T,C>
OwnerChoices <P,R,PolicyDataType,CT,CC,MapCol>
RoleAccess <P,R,P3PType,DBRole,Operations>
Data Tables

Figure 5: Architecture after first two extensions

```
Select name, phone, address from PATIENT;
Purpose = Treatment; Recipient = Nurses
```

```
Select name, phone, address from
(Select pno,name, NULL AS phone,
    CASE WHEN
        EXISTS (select address_option from options_patient where
        patient.pno=options_patient.pno and
        options_patient.addres_option=TRUE) AND
        current_date<=((Select signature_date from
        patient_signature_date where
        patient_signature_date.pno=patient.pno) + integer '90')
    THEN address ELSE NULL END
        AS address
From patient)
```

Figure 6: Example of limited retention

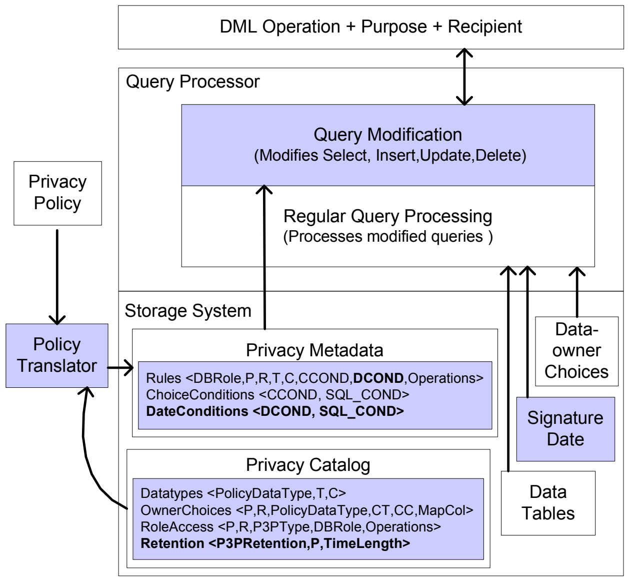
DML Operation + Purpose + Recipient
Query Processor
Query Modification
(Modifies Select, Insert,Update,Delete)
Regular Query Processing
(Processes modified queries)
Privacy Policy
Policy Translator
Storage System
Privacy Metadata
Rules <DBRole,P,R,T,C,CCOND,DCOND,Operations>
ChoiceConditions <CCOND, SQL_COND>
DateConditions <DCOND, SQL_COND>
Privacy Catalog
Datatypes <PolicyDataType,T,C>
OwnerChoices <P,R,PolicyDataType,CT,CC,MapCol>
RoleAccess <P,R,P3PType,DBRole,Operations>
Retention <P3PRetention,P,TimeLength>
Data-owner Choices
Signature Date
Data Tables

Figure 7: Architecture after adding support for limited retention

gives a query and its modified form that ensures limited disclosure and limited retention. Figure 7 gives the incremental architecture after adding this feature.

3.4 Support of policy versions

The study in [8] found that 80% of organizations use a different privacy policy for employees and clients, 42% have multiple policies for clients, and 75% require support of policy versions. To the best of our knowledge, there is little work about how we could support multiple versions and multiple policies in the context of Hippocratic databases. The different cases of multiple versions and multiple policies requirements can be analyzed as follows:

- *Multiple policies.* Company ABC needs to support two policies, P1 for patients and P2 for doctors. *Solution*: We translate P1 and P2 independently. The metadata will contain the rules of both policies and we will have two primary tables.

- *Single policy, multiple data owners*. Company ABC uses policy P for patients and doctors. Patients and doctors are different entities in the database. *Solution*: We translate P twice. During the first time, the privacy catalog considers the tables associated to patients; during the second one, the tables associated to doctors.

- *Multiple policies over time.* A policy is updated for old and new patients. *Solution*: We translate initially the original policy. When it is updated, we delete the metadata and translate the updated policy. We have one primary table.

- *Multiple versions.* a) The policy for patients is updated only for new patients. b) Two policy versions for different groups of patients are simultaneously used. *Solution*: Since this case will require the use of two policies associated with the same database entity Patient, this case is not directly supported by the frameworks for limiting disclosure proposed in previous work. The remaining part of this section presents our extension to support multiple policy versions.

In our approach we get the policy ID and version from the P3P-like policy. We assume that the version of a policy is part of its ID. Also, each row of a primary table being used by more than one policy will have a label i.e., extra column, with the ID of the active policy for this row. Each data owner has one active policy at any time, but different data owners can have different policy versions. The new privacy catalog table *Policies* contains information of the policies supported by the system, and the primary and signature-date tables they should use. This information is used during policy translation and each generated rule is stored with its corresponding Policy ID. During query modification, the system performs the regular test to determine if there is access for the specific combination of database role, purpose, recipient, data table and attribute. In the presence of multiple versions, there will be more than one rule for this combination and the system will add another level of CASE statement to process the versions accordingly. Figure 8 shows an example of this query modification. We need to propagate the association with policy versions to other tables that store information about data owners; we could add another column to store the version, or we could implement the query modification module such that each privacy-preserving view joins the corresponding primary table and consequently uses its version information. In this work, we use the first approach. Figure 9 shows the incremental design with support for multiple policy versions.

3.5 Support of generalization hierarchies in Hippocratic databases

Hippocratic databases and anonymization are two important areas in the effort to achieve effective mechanisms to ensure privacy in database systems. Unfortunately, little work has been done to integrate their results. In the design for limited disclosure presented so far the support of opt-in/opt-out choices is very limited; data owners can only give either full access to the data or deny it completely; there is not the option to give access to a generalized version of the data. We propose the study of the integration of Hippocratic databases and anonymization/generalization techniques. The ideas we present in this section represent only the first step in this integration path. We present here a design to introduce generalization hierarchies into the limiting disclosure framework for Hippocratic databases.

The first step is to identify the data elements that will be generalized and build a generalization hierarchy for each of them. The number of levels of a generalization tree could be different for different elements. The first level represents the actual value of the data element; level two represents the first degree of generalization, and so on. The information of the tree is loaded by the DBA into the metadata table *Generalization*. Figure 10 gives an example of a generalization tree and some tuples of the table Generalization corresponding to this tree. The content of the choice tables for the data elements

```
Select name, phone, address from PATIENT;
Purpose = Treatment; Recipient = Nurses
```

$\Downarrow$

```
Select name, phone, address from
(Select pno,name, NULL AS phone,
    CASE  WHEN policyversion=01 THEN address
             WHEN policyversion=02 THEN
                CASE WHEN
                     EXISTS (select address_option from options_patient
                          where patient.pno=options_patient.pno
                          AND options_patient.addres_option=TRUE)
                     THEN address ELSE NULL END
    END AS address
From patient)
```

Figure 8: Example of limiting disclosure with multiple policy versions

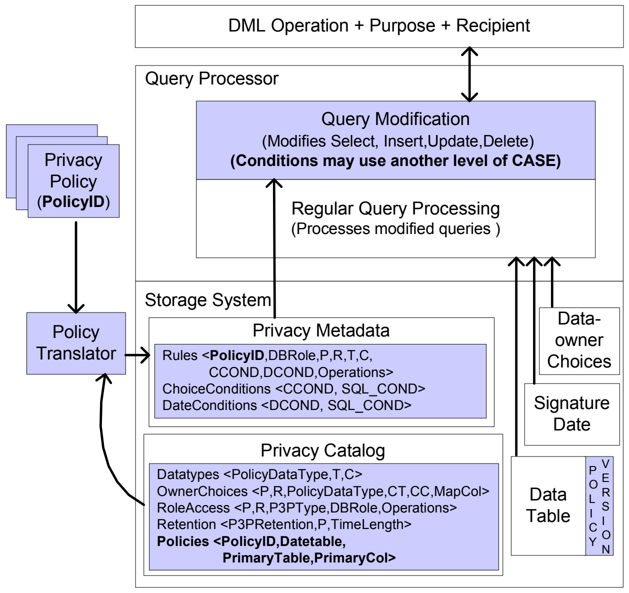
DML Operation + Purpose + Recipient
Query Processor
Query Modification
(Modifies Select, Insert,Update,Delete)
(Conditions may use another level of CASE)
Regular Query Processing
(Processes modified queries)
Privacy Policy (PolicyID)
Policy Translator
Storage System
Privacy Metadata
Rules <PolicyID,DBRole,P,R,T,C, CCOND,DCOND,Operations>
ChoiceConditions <CCOND, SQL_COND>
DateConditions <DCOND, SQL_COND>
Data-owner Choices
Signature Date
Privacy Catalog
Datatypes <PolicyDataType,T,C>
OwnerChoices <P,R,PolicyDataType,CT,CC,MapCol>
RoleAccess <P,R,P3PType,DBRole,Operations>
Retention <P3PRetention,P,TimeLength>
Policies <PolicyID,Datetable, PrimaryTable,PrimaryCol>
Data Table
POLICY VERSION

Figure 9: Architecture after adding support for multiple policy versions

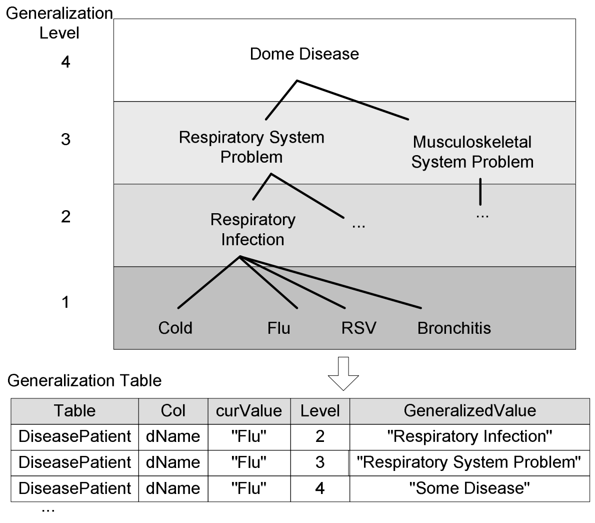

Table	Col	curValue	Level	GeneralizedValue
DiseasePatient	dName	"Flu"	2	"Respiratory Infection"
DiseasePatient	dName	"Flu"	3	"Respiratory System Problem"
DiseasePatient	dName	"Flu"	4	"Some Disease"
...				

Figure 10: Example of a generalization hierarchy

that can be generalized will not be Boolean anymore. They will store instead, the level of generalization that the data owner wants for the element. A value of 0 means that access is not allowed, 1 means full access and values greater than 1 will allow the disclosure of generalized values. The query modification module will use a generalization function that will convert a data value into its generalized form. The form of the CASE statement will change to process each possible choice value. Figure 11 gives an example of query modification with support of generalization hierarchies. Figure 12 shows the incremental design after adding support of generalization hierarchies.

4. Experiments

We implement the extensions presented in Section 3 as a middleware application that performs the functionality of the SQL modification module. In this section, we present the results of the performance study of the various extensions, analyzing the overhead, scalability, and effect of record filtering associated to them. The cost considered for selection queries is the query execution and retrieval time. We ignore the cost of query rewriting. For update queries, we consider

```
Select P.Name, DP.dName from Patient P, DiseasePatient DP where
P.pid=DP.pid
Purpose = Research; Recipient = Lab
```

```
Select P.Name, D.dName from
   (Select pno from Patient) AS P
   (Select pno,
        CASE (select diseaseName_option from options_disease
                 where DiseasePatient.pno=options_disease.pno) as level
        WHEN 0 THEN NULL
        WHEN 1 THEN dname
        ELSE  generalize("DiseasePatient","dName",dname,level)
        END AS dname
   From DiseasePatient) AS DP
where P.pno=DP.pno and DP.dno=D.dno
```

Figure 11: Example of limiting disclosure with generalization hierarchies

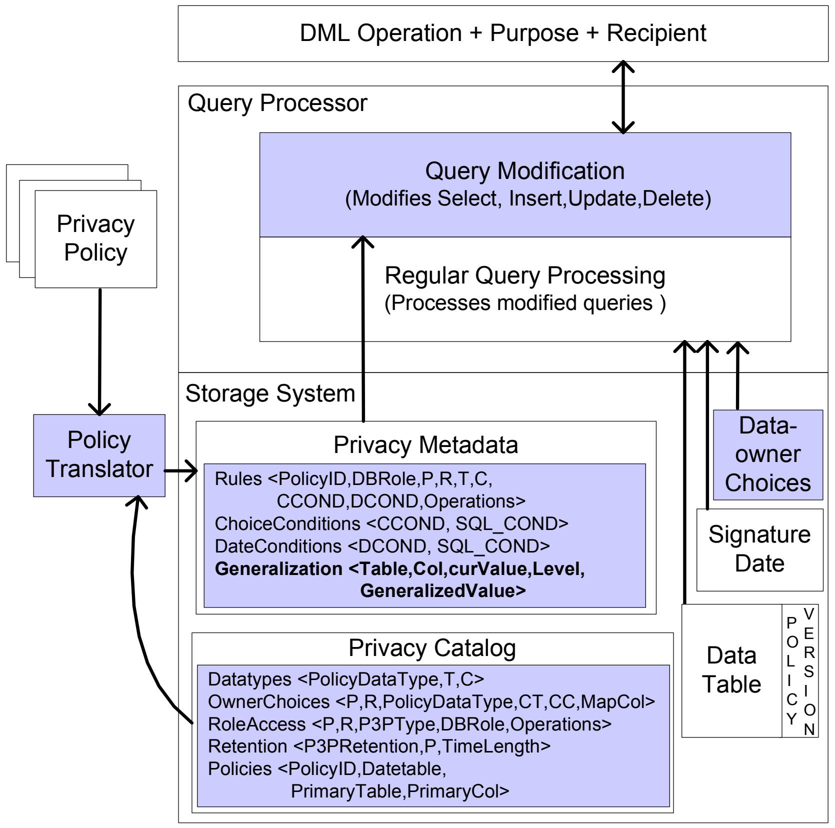
DML Operation + Purpose + Recipient
Query Processor
Query Modification
(Modifies Select, Insert,Update,Delete)
Regular Query Processing
(Processes modified queries)
Privacy Policy
Policy Translator
Storage System
Privacy Metadata
Rules <PolicyID,DBRole,P,R,T,C, CCOND,DCOND,Operations>
ChoiceConditions <CCOND, SQL_COND>
DateConditions <DCOND, SQL_COND>
Generalization <Table,Col,curValue,Level, GeneralizedValue>
Privacy Catalog
Datatypes <PolicyDataType,T,C>
OwnerChoices <P,R,PolicyDataType,CT,CC,MapCol>
RoleAccess <P,R,P3PType,DBRole,Operations>
Retention <P3PRetention,P,TimeLength>
Policies <PolicyID,Datetable, PrimaryTable,PrimaryCol>
Data-owner Choices
Signature Date
Data Table
POLICY VERSION

Figure 12: Architecture after adding support of generalization hierarchies

both the privacy-checking and execution times. We do not include the evaluation of generalization hierarchies because this extension is part of an ongoing work whose results will be presented in the future.

4.1. Tests configuration

We use a synthetic database based on the Wisconsin Benchmark [13] with attributes presented in Table 1. We use PostgreSQL 8.1, set the shared buffer to 25MB and leave all the other configuration parameters with their default values. The tests are performed in a 3.2 GHz Pentium IV machine with 1.5GB of memory and running Microsoft Windows XP as operating system. The results presented in this section consider the average of the warm performance numbers having 95%

confidence and an error margin less than ±5%. As discussed in [2], there are several ways in which we could store the choice columns. We use the external single approach since it was found to be an effective compromise. With this approach, we store all the choice columns in a single external table. We also use an external table to store the policy signature dates and assume the use of two versions in the experiments that use multiple version support.

4.2. Performance evaluation

4.2.1. Overhead and scalability of Select queries. To measure the overhead cost of the different extensions we consider a worst-case scenario to run simple Select queries. The queries select all the records of the data table i.e., application selectivity = 100%, and nothing is filtered by the data-owner preferences or the retention time restrictions, i.e., choice selectivity and retention selectivity are 100%. This scenario incurs in all the cost of privacy checking but does not get any benefit from record filtering. Figure 13 gives the overhead cost of the various extensions for different sizes of the data tables. We can observe that the costs of the extensions and combinations of them are small and scale well when the database size increases.

4.2.2. Effect of record filtering on select queries. When the choice selectivity and retention selectivity are less than 100%, select queries perform significantly better than in the worst case scenario and in several cases even better than the original queries. Figure 14 and 15 give the execution time of select queries when we change the choice selectivity and the retention selectivity, respectively. The results are presented for different combinations of the implemented extensions. For this experiment, we use tables with one million records and application selectivity of 100%. The performance improvement is significant for values of selectivities smaller than 50%. We expect even better results when choice and retention filtering are considered simultaneously.

The performance results for update queries follow those for select queries. The cost of privacy checking is relatively more significant in the case of update queries because of the reduced cost of update operations when modifying few tuples, and the extra cost of maintaining the choice and signature-date tables. For example, inserting a tuple in the primary table requires also inserting the corresponding tuples in choice and signature-date tables. This cost is compensated by the performance gains associated with the operations that do not need to be executed because their privacy check fails.

5. Conclusions and future work

We identified, studied and implemented several privacy-preserving features that extend the previous work on Limiting Disclosure in Hippocratic databases. The features studied in detail are: mapping purpose, recipient, and data type of a policy with database roles, support of multiple DML operations, support of retention time, support of policy versions, and support of generalization hierarchies. We discussed why we need these extensions and the limited or non-existing support of these features in previous work. Our performance analysis showed that the overhead of the implemented extensions is

Table 1: Benchmark attributes specification and choice columns

Column	Datatype	Description
Unique2	Int	Primary key, Sequential order
Unique1	Int	Candidate key, random order
Onepercent	Int	Values 0-99, random order
Tenpercent	Int	Values 0-9, random order
Twentypercent	Int	Values 0-4, random order
Fiftypercent	Int	Values 0-1, random order
stringu1	52-byte str	Unique character string
stringu2	52-byte str	Unique character string
Choice0	Int	Values 0-1 (1% = 1), indexed
Choice1	Int	Values 0-1 (10% = 1), indexed
Choice2	Int	Values 0-1 (50% = 1), indexed
Choice3	Int	Values 0-1 (90% = 1), indexed
Choice4	Int	Values 0-1 (100% = 1), indexed
SignatureDate	Date	Values d-d+99, random order

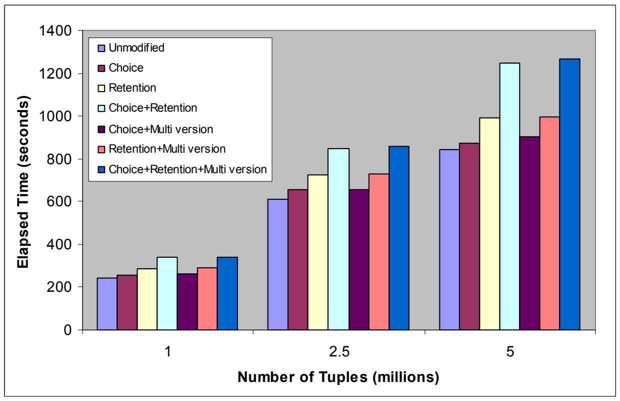

Figure 13: Overhead and scalability of select queries for different extensions

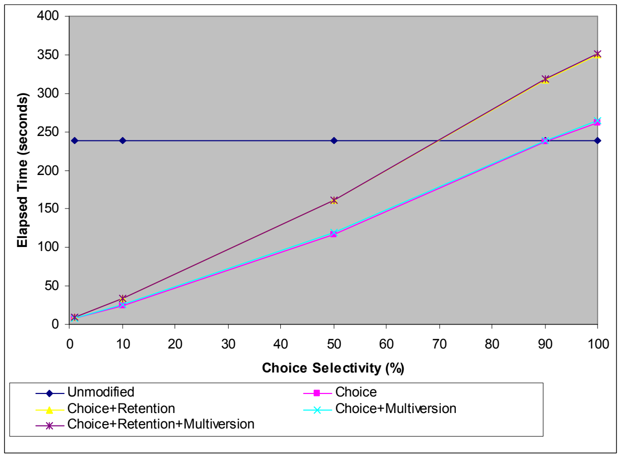

Figure 14: Effect of record filtering by choice restrictions

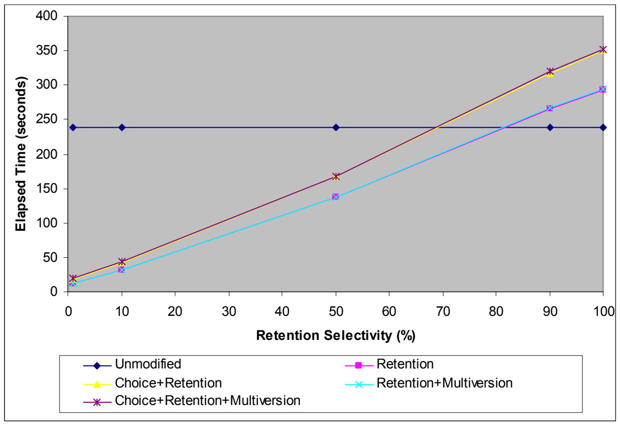

Figure 15: Effect of record filtering by retention restrictions

small and scale well to large databases. We believe that our contribution in this work represents a step in the challenging path of finding efficient ways to engineer the Hippocratic database and answer real world privacy requirements. Some paths for future work are: the integration of results in the area of anonymization into the Hippocratic database, the design of privacy-preserving mechanisms to support Export and Import operations maintaining privacy definitions, the support of Mandatory Access Control via Hippocratic databases, and the study of performance of different ways to organize the metadata (normalized versus de-normalized tables, storing conditions as strings versus storing the components used in conditions in compact attributes and building the conditions on-the-fly, indexes over privacy catalog and metadata, etc.).

6. References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. “Hippocratic databases”. VLDB 2002.

[2] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan and Y. Xu. “Limiting disclosure in Hippocratic databases”. VLDB 2004.

[3] R. Agrawal, R. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. “Auditing compliance with a Hippocratic database”. VLDB 2004.

[4] L. Sweeney. “K-anonymity: A model for protecting privacy”. International Journal on Uncertainty, Fuzziness, and Knowledge Based Systems, 2002, pp. 557-570.

[5] P. Samaraty, L. Sweeney. “Generalizing data to provide anonymity when disclosing information”. PODS 1998.

[6] A. Machanavajjhala, J. Gehrke, and D. Kifer. “l-diversity: Privacy beyond k-anonymity”. ICDE 2006.

[7] X. Xiao, and Y. Tao. “Personalized Privacy Preservation”. SIGMOD 2006.

[8] “Cross-National study of Canadian and U.S. corporate privacy practices”. Ponemon Institute & Ontario Information & Privacy Commissioner, 2004.

[9] R. Agrawal, P. Bird, T. Grandison, J. Kiernan, S. Logan, W. Rjaib. “Extending Relational Database Systems to Automatically Enforce Privacy Policies”. Industrial paper. ICDE 2005

[10] L. Cranor, M. Langheinrich, M. Marchiori, M. Pressler-Marshall, and J. Reagle. “The platform for privacy preferences 1.0 (P3P1.0) specification”. W3C Recommendation. April 2002.

[11] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. “Enterprise privacy authorization language 1.2 (EPAL 1.2)”. W3C Member Submission, November 2003.

[12] J. Byun, E. Bertino, N. Li. “Purpose-based access control for privacy protection in relational database systems”. Technical Report 2004-52, Purdue University, 2004.

[13] D. DeWitt. “The Wisconsin benchmark: Past, present, and future”. *The Benchmark Handbook*. Morgan Kauffmann, 1993.