Extending Relational Database Systems to Automatically Enforce Privacy Policies

Rakesh Agrawal† Paul Bird‡ Tyrone Grandison† Jerry Kiernan† Scott Logan‡ Walid Rjaibi‡

† IBM Almaden Research Center
650 Harry Road, San Jose, CA, USA
{ragrawal, tyroneg, jkiernan}@us.ibm.com

‡ IBM Toronto Lab
8200 Warden Ave.,Markham, ON, Canada
{pbird, silogan, wrjaibi}@ca.ibm.com

Abstract

Databases are at the core of successful businesses. Due to the voluminous stores of personal data being held by companies today, preserving privacy has become a crucial requirement for operating a business. This paper proposes how current relational database management systems can be transformed into their privacy-preserving equivalents. Specifically, we present language constructs and implementation design for fine-grained access control to realize this goal.

1. Introduction

The pervasive use of computing technology and the increased reliance on information systems have created a heightened awareness and concern about the storage and use of private information. This worldwide phenomenon has ushered in a plethora of privacy-related guidelines and legislations, e.g. the OECD Privacy Guidelines in Europe, the Canadian Privacy Act, the Australian Privacy Amendment Act, the Japanese Privacy Code, the Health Insurance Portability and Accountability Act (HIPAA), and Gramm-Leach-Bliley Consumer Privacy Rule. Compliance with these legislation has become an important corporate concern. The current methods employed to address the disclosure compliance problem involve training individuals to be cognizant of the various regulations and changing organizational processes and procedures. However, these approaches are only a partial solution and need to be augmented with technology support.

We present constructs for imbuing relational database systems with fine grained access control (FGAC) and show how they can be used to enforce disclosure control enunciated in the vision for Hippocratic databases [1]. These constructs have been designed to be integrated with the rest of the infrastructure of a relational database system. We also discuss the implementation of the proposed FGAC constructs, building upon the ideas from [6]. Finally, we show how privacy policies written in a higher-level specification language such as P3P [3] can be algorithmically translated into the proposed constructs.

The users of relational databases are requiring that an FGAC implementation meets the following desiderata:

- The implementation must solve the problem within the database itself without application changes or application awareness of the implementation.
- The implementation must ensure that all users of the data are covered, regardless of how the data is accessed.
- The implementation must minimize the complexity and maintenance of the FGAC policies.
- The implementation must provide the ability to control access to rows, columns, or cells as desired.

Traditional methods of database access control have relied upon the use of statically defined views, which are logical constructs imposed over database tables that can alter or restrict the data seen by a user. Using predefined views as the method for FGAC works well only when the number of different restrictions is few or the granularity of the restrictions is such that it affects large, easily identified groups of users. When these conditions are not true, view definitions may become complex in an effort to accommodate all the restrictions in one view. This complexity can strain system limits and can make maintenance of the views difficult.

Consider the use of a large number of views, each one implementing restrictions for a specific set of users. One issue that arises immediately is how to correctly route user requests to the view that is appropriate to them. Often, the solution chosen is to resolve the request in the application,

not in the database. Moreover, if a user can bypass the view when accessing data, for example by having direct access to the underlying tables, then the restrictions are not enforced.

Given the shortcomings of the traditional methods of implementing FGAC, some database vendors have proposed solutions that do not rely on the use of views to control access to tabular data. For instance, the Oracle Virtual Private Database [5, 7] solution allows users to define a security policy, which is a function written in PL/SQL that returns a string representing a predicate, and to attach the security policy to a table. When that table is accessed, the security policy is automatically enforced. Sybase Row Level Access Control [9] allows users to define access rules that apply restrictions to retrieved data. The related work section further discusses the Oracle and Sybase approaches. Microsoft SQL Server primarily supports traditional view based access control, though they have a feature called row level permissions. However, row level permissions seem to be usable only with table hierarchies. In DB2, support for FGAC is currently provided through traditional mechanisms based on views, triggers and special registers.

The remainder of this paper is organized as follows. Section 2 proposes FGAC constructs that allow restrictions to be expressed on database accesses. Aside from row and column level restrictions that respectively restrict the set of rows and columns of a table, cell level restrictions can be specified to limit access to specific fields of a row. Section 3 describes how restrictions expressed in terms of the proposed constructs can be enforced using dynamic views. Section 4 presents an algorithm for translating a P3P privacy policy into the proposed FGAC constructs. Section 5 discusses related work, and Section 6 presents concluding remarks. Appendix A argues for extending the functionality of current relational database systems with cell level access control.

2. Language Constructs

We provide constructs that allow restrictions to be specified on access to data in a table at the level of a row, a column, or a cell (i.e., individual column-row intersections). Privacy policies specified in high-level languages such as P3P can be translated into these constructs, or one could specify the policy directly using these constructs.

The proposed facility is complimentary to the current table level authorization mechanisms provided by commercial database systems using the **grant** command [2]. While grant controls whether a user can access a table at all, the proposed constructs define the subset of the data within a table that the user is allowed to access. Conceptually, a restriction defines a view of the table in which inaccessible data has been replaced by null values. As discussed in [6], it is possible to use either "table semantics" or "query semantics". With query semantics, if all the values in a row are hidden by a restriction, then the row is omitted altogether from the view. With table semantics, the row would instead be retained.

```
create restriction restriction-name
on table-x
for auth-name-1 [ except auth-name-2]
( (    (to columns column-name-list)
     | (to rows [ where search-condition ] )
     | (to cells (column-name-list [ where search-condition ] )+ )
  )
  [ for purpose purpose-list ]
  [ for recipient recipient-list ]
)+
command-restriction
```

Figure 1. Fine grained restriction syntax

Figure 1 gives the syntax of a fine grained restriction command. It states that those in auth-name-1 except those in auth-name-2 are allowed only restricted access to table-x. The keywords **public** (i.e., all users), **group**, **role**, and **user** can be used to qualify the authorized names. Table-x can be any table expression.

A restriction can be specified at the level of a column (Section 2.1), a row (Section 2.2), or a cell (Section 2.3). More than one restriction can be specified on a table for the same user (Section 2.4).

A restriction may additionally specify purposes and/or recipients [1, 3, 6] for which the access is allowed. If no purpose or recipient is specified, then the restriction applies to all purposes and recipients respectively. If a purpose or recipient is specified, the user's access is limited to only the specified purpose-recipient combinations.

Akin to the database system variable **user** that can be referenced in queries and returns the id of the user issuing the query, the new system variables **purpose** and **recipient** return the list of purposes and recipients from the current query context [6]. These values in turn determine the restrictions for the current query.

The command-restriction that appears as the last element of the syntax has the following form and states that access can be restricted to any combination of select, delete, insert, or update commands:

restricting access to (**all** | (**select** | **delete** | **insert** | **update**)+)

The discussion below will use, for illustration, the Customer table with the following schema: Customer (id **integer**, name **char**(32), phone **char**(32)).

2.1 Column Restriction

A column restriction specifies a subset of the columns in table-x that auth-name-1 is allowed to access. The following restriction, named r1, ensures that only the id column of Customer is accessed by any database user:

create restriction r1
on Customer
for public
to columns id
restricting access to all

The restriction r2 below ensures that members of the account group and user Bob have only select access to columns name and phone.

create restriction r2
on Customer
for group acct, **user** Bob
to columns name, phone
restricting access to select

2.2 Row Restriction

A row restriction gives the subset of rows in table-x that auth-name-1 is allowed to access. This subset is specified using a search-condition over table-x. The restriction r3 below ensures that every access to Customer is qualified by the predicate, name = **user**.

create restriction r3
on Customer
for public
to rows where name = **user**
restricting access to all

If user Bob issues **select** * **from** Customer, he would see id, name and phone for those rows where name equaled Bob.

2.3 Cell Restriction

A cell restriction defines the row-column intersections that auth-name-1 is allowed to access. It is possible to specify multiple column-name lists, each possibly annotated with a search-condition. A search-condition is a correlated subquery with an implicit correlation variable t defined over the tuples of table-x. Access to the columns in column-name-list for each individual row identified by t is conditionally granted depending upon the result of the search condition. If no search-condition is given, then access is granted to all column values in column-name-list in table-x. If the search condition ignores correlation variable, then access is granted or denied to all columns values in column-name-list in table-x, depending upon the result of the search-condition.

The following is an example of a cell restriction used to enforce individual user's privacy preferences expressed as opt-in/out choices. Assume that for the purpose of marketing, Bob is allowed to see name, but his access to phone is allowed only if the user has opted-in to revealing her phone number.

create restriction r4
on Customer **for user** Bob,
to cells name,
(phone **where exists** (
select 1
from SysCat.Choices_Customer c
where c.ID = Customer.ID **and** c.C1 = 1))
for purpose marketing
for recipient others
restricting access to select

The above restriction specifies cell restrictions for two column-name-lists: The first list contains the name column, and the second contains the phone column. The restriction allows Bob access to name, only if the variable **purpose** includes marketing, and **recipient** includes others. Otherwise, all values of the name column will be null for Bob.

The second list of columns has a search-condition associated with it since access to phone is dependent upon individual user choices. The search-condition comprises an existential subquery that uses the implicit correlation variable Customer. For each row in Customer, the subquery verifies, using the SysCat.Choices_Customer table that stores individual opt-in/out choices, whether the user has opted-in for the disclosure of her phone number (represented by a column value of 1).

2.4 Combining Multiple Restrictions

If multiple restrictions have been defined for a user u and a table T, then u's access to T is governed by the combination of these restrictions.

Assume initially that a user associates with a query a single purpose and a single recipient. We consider two design choices for combining multiple restrictions:

- *Intersection* — User u is allowed access to data defined by the intersection of all applicable restrictions. The details are shown in Table 1.
- *Union* — User u is allowed access to data defined by the union of all applicable restrictions. The details are shown in Table 2.

If the commands specified in the command-restriction clauses of the restrictions being combined are different, they

	row	column	cell
row	The search-conditions of individual row restrictions are and'ed together to define the intersection of rows accessible to a user.	The row restriction limits the rows accessible to the user. The column restriction further limits the columns within the rows accessible to the user.	The row restriction limits the rows accessible to the user. Within each row, the cell restriction further limits the access to the cells that qualify the cells' search-condition.
column		The user's access is limited to those columns that appear in both of the column restrictions.	Column and cell restrictions intersect to limit access to only those columns that appear in both the restrictions. In addition, the cell restriction's search-condition further limits accessible cells within a column.
cell			The search-conditions are and'ed together and the user is allowed access to a cell if the composite condition is satisfied for the cell. The value of the composite condition for a cell that does not appear in both the restrictions is false.

Table 1. Combining restrictions with intersection

	row	column	cell
row	The search-conditions of individual row restrictions are or'ed together to define the union of rows accessible to a user.	The user is given access to all the cells for any row that satisfies the row restriction. Additionally, the user is allowed access to all the cells in any of the columns that satisfies the column restriction, irrespective of whether the corresponding rows satisfy the row restriction.	The user is given access to all the cells in any of the rows that satisfy the row restriction. Additionally, the user is allowed access to all other cells that satisfy the cell restriction's search-condition, irrespective of whether the corresponding rows satisfy the row restriction.
column		The user is allowed access to a column if it appears in either of the two column restrictions.	The user is given access to all the cells in any column appearing in the column restriction, regardless of whether the cell restriction is satisfied for these cells. For cells in a column for which the column restriction does not apply, access is given if the cell restriction is satisfied.
cell			The search-conditions are or'ed together and the user is allowed access to a cell if the composite condition is satisfied for the cell.

Table 2. Combining restrictions with union

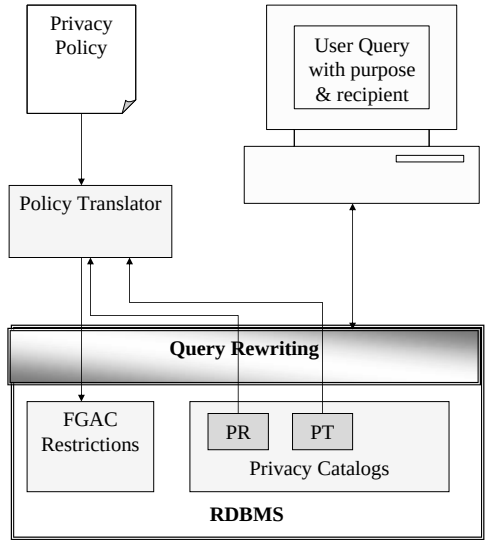

Figure 2. Implementation architecture

are respectively and'ed or or'ed depending upon the choice of intersection or union semantics.

Multiple restrictions can be combined in any order, both with intersection and union semantics. With the intersection semantics, the user's access to data decreases as additional restrictions are applied. Conversely, with union semantics, access to data increases as additional restrictions are applied.

We prefer intersection semantics over union since additional restrictions should intuitively decrease a user's access to information.[1]

Finally, if a query is annotated with multiple purpose-recipient pairs, instead of a single pair, then restrictions governing access to any of the pairs become relevant for the query. These restrictions are then combined as above. Note that once a user's access to a table has been restricted, the user can only access the data allowed for the purposes and recipients specified in the restrictions.

3. Implementation

We next present a design for implementing the proposed constructs, building upon the ideas presented in [1, 6]. In this and the remainder of the paper, we focus on cell restrictions limited to select statement access. Figure 2 shows the overview of the design. The policy translator accepts a privacy policy (written in, for example, P3P) and metadata stored in privacy catalogs and generates cell restrictions that implement the policy. The schema of the privacy metadata catalogs shown in Figure 2 used to drive the translation of P3P privacy policies into cell level restrictions are given below.

PR (purp-recip **char**(18),
p3ptype **char**(32),
choice_tabname **char**(32),
choice_colname **char**(32))

PT (p3ptype **char** (32), tabname **char**(32), colname **char**(32))

Table PR stores, for each purpose, recipient and p3p data type pair, the (table name-column name) pair that records individual user opt-in/out choice, should any choice be available for that combination. Table PT stores, for each P3P data type, the table names and column names which store values of these P3P types.

Figure 3 gives the algorithm used for enforcing the fine grain restrictions. For ease of exposition, we assume there is a single purpose-recipient pair associated with a query and there is at most a single restriction which is relevant for the query. The enforcement algorithm combines the restrictions relevant to individual queries annotated with purpose and recipient information and transforms the user's query into an equivalent query over a dynamic view that implements the restriction.

In detail, Line 1 iterates over each table reference t in a query Q. Line 2 accesses metadata to determine if there is a restriction r governing the usage of t by user u who is submitting the query Q. If no such restriction exists, then t remains unmodified in Q. Otherwise, Lines 3 and 4 replace each reference to table t in query Q with a reference to a dynamic view v.

The generation of the dynamic view v is implemented in Lines 5 through 25. The view v is a select statement which conditionally projects each column $c \in t$. Line 7 searches for a column reference to $c \in r$. If no such reference exists with the purpose/recipient of query Q, then the user u is not allowed access to c and Line 8 thus projects a null value for all values of c. Otherwise, Line 10 searches for a where clause associated with $c \in r$. If no such clause exists, then u is granted unconditional access to c. Otherwise, Line 15 outputs the condition of the where clause into a SQL case statement which verifies the condition before outputting the value of c (on Line 18). If the condition is false, access to the column value is denied and Line 19 outputs a null value for c.

4. Translating Privacy Policies

It is expected that the privacy policies will likely be written in some high-level policy language. The following illus-

[1] It is conceivable to use mixed modes for combining restrictions. For example, intersection could be used to combine multiple row restrictions while union could be used to combine multiple column or cell restrictions. However, the semantics of such combinations can become quite complex as the restriction imposed by a combination may no longer be order-independent.

```
1  for each table reference t in query Q do begin
2    if (exists a restriction r pertaining to t for Q) then begin
3      create a dynamic view v ∈ Q over t
4      replace each reference to t ∈ Q with a reference to v ∈ Q

       // create the dynamic view v using
       // the following print statements
       //
5      print "select"
6      for each column c ∈ t do begin
         // c_p, c_r are the purposes, recipients
         // of column c in restriction r
         // Q_p, Q_r are the purpose, recipient of query Q
         //
7        if (c ∉ r|Q_p ∈ c_p ∧ Q_r ∈ c_r
           // c isn't included in the restriction r
           // access to c is thus prohibited
           //
8          print "null"
9        else begin
           // The whereClause function returns
           // the predicate associated with c
           // that is specified in the restriction
           //
10         let w = whereClause(c)
11         if w = null then
             // There is no "where" condition
             // governing the use of c ∈ r, thus access
             // to all column values is granted unconditionally
             //
12           print c.colname
13         else begin
             // Implement the "where" condition
             // using a SQL case statement to grant
             // only conditional access to the column c
             //
14           print "case when exists ("
15           print w.condition
16           print ")"
17           print "then"
18           print c.colname
19           print "else null end as"
20           print c.colname
21         end
22       end
23     end
24     print "from"
25     print t.tablename
26   end
```

Figure 3. Algorithm for enforcing fine grained cell level restrictions using a Hippocratic database system

trates the basic syntax of the P3P policy specification language [3].

```
<POLICIES> ...
 <POLICY name = "Policy_Name1"  > ...
  <STATEMENT>
   ...
   <PURPOSE>
     stated-purpose
      [ required = ("always"|"opt-in"|"opt-out") ]
   </PURPOSE>
   <RECIPIENT>
     stated-recip
      [ required = ("always"|"opt-in"|"opt-out") ]
   </RECIPIENT>
   <RETENTION>  retention_val  </RETENTION>
   <DATA GROUP>
    <DATA ref = data-ref-val>
     ...
    </DATA GROUP>
  </STATEMENT>
 </POLICY>
 <POLICY>
  ...
 </POLICY>
 ...
</POLICIES>
```

The process of transforming a policy like the one above into fine grained restrictions involves: (1) parsing the policy to extract the list of statements, (2) mapping data abstractions into their implementation specific equivalents, e.g. in the above specification this would mean mapping data-ref-val to its corresponding table name(s) and column name(s), (3) structuring the choice tables which record individual user opt-in/out choices (in some cases, this may not be necessary since there may be no such choices), and (4) generating the restriction statements. Assuming that data-ref-val maps to columns A and B of table T, the above abstract specification would lead to the following restriction being constructed:

create restriction Policy_Name1
on T
for public
to cells A,B
  [**where** opt-in-out-conditions]
  for purpose stated-purpose
  for recipient stated-recip
restricting access to select

Figure 4 is a detailed example of a privacy policy, for a fictional Healthcare provider.

The metadata contains the information needed to associate "#personal" (personal information) and "#medical" (medical information) with database tables which store this information. Personal information maps to the name, SSN, address, email and DOB fields of the Patients table, while medical information maps to the xray, pharmacy, family, appointment and lifestyle fields of the Patients table. Thus,

```
...
<!-- Statement1 -->
<STATEMENT>
 <CONSEQUENCE>
  Encodes that personal and medical information
  can be accessed for emergency purposes
  by ourselves
 </CONSEQUENCE>
 <PURPOSE>
  <other-purpose>
    Emergency
  </other-purpose>
 </PURPOSE>
 <RECIPIENT><ours/></RECIPIENT>
 <RETENTION><stated-purpose/></RETENTION>
 <DATA-GROUP>
  <DATA ref = "#personal"/>
  <DATA ref = "#medical">
   <CATEGORIES>
    <health/>
   </CATEGORIES>
  </DATA>
 </DATA-GROUP>
</STATEMENT>

<!-- Statement2 -->
 <STATEMENT>
  <CONSEQUENCE>
   Encodes that we and drug companies
   with the same data usage policies
   can access personal and medical information
   for new_drug_research on an opt-out basis
  </CONSEQUENCE>
  <PURPOSE><develop/></PURPOSE>
  <RECIPIENT>
   <ours required="opt-out"/>
   <same required="opt-out"/>
  </RECIPIENT>
  <RETENTION><stated-purpose/></RETENTION>
  <DATA-GROUP>
   <DATA ref = "#personal"/>
   <DATA ref = "#medical">
    <CATEGORIES>
     <health/>
    </CATEGORIES>
   </DATA>
  </DATA-GROUP>
 </STATEMENT>
 ...
```

Figure 4. A sample privacy policy written for a health care provider

```
create restriction Statement1
on Patients
for public
to cells          Name, SSN, Address, Email, DOB,
                  XRay, Pharmacy, Family,
                  Appointment, Lifestyle
    purpose Emergency
    recipient ours
restricting access to select

create restriction Statement2.1
on Patients
for public
to cells          Name, SSN, Address, Email, DOB,
                  XRay, Pharmacy, Family,
                  Appointment, Lifestyle
      where
        exists (
          select 1
          from SysCat.Choices_Patients cp
          where cp.ID = Patients.ID
            and cp.C1 = 1 )
    for purpose develop
    for recipient ours
restricting access to select

create restriction Statement2.2
on Patients
for public
to cells          Name, SSN, Address, Email, DOB,
                  XRay, Pharmacy, Family,
                  Appointment, Lifestyle
      where
        exists (
          select 1
          from SysCat.Choices_Patients cp
          where cp.ID = Patients.ID
            and cp.C2 = 1 )
    for purpose develop
    for recipient same
restricting access to select
```

Figure 5. Translation of the privacy policy in Figure 4 into fine grained cell level restrictions

the P3P healthcare policy given in Figure 4 is translated into the restrictions given in Figure 5.

For simplicity, the restrictions in Figure 5 assume that all data types in a P3P statement are contained in a single table.

The Choices_Patients table is created by the database administrator to record individual opt-in/out decisions described in the privacy policy. In Figure 5, C1 represents the choice to allow Drug_Research to see personal and medical data if the drug research is being conducted by the healthcare company itself. Choice C2 is the option to allow usage of the personal and medical data for drug research by other healthcare companies having the same privacy policy as this company. The example illustrates the basic steps involved in the translation process.

Figure 6 gives the pseudo-code showing the steps involved in transforming P3P policy into our proposed constructs. A unique restriction name, needed for the command, is generated on Line 5. Line 7 uses the **mapP3PStatementToTable** function to recover the table name which stores the information described by the data types in the P3P statement. This metadata has been populated by the database administrator. On Line 8, the the restriction is set to **public** to apply to all users. Line 10 uses the **mapP3PDataTypeToColumns** function to retrieve the column names that store information described by the P3P data types in the statement. Again, this information has been prepared and supplied by the database administrator and stored in metadata tables.

The function **mapP3PPurposeToChoiceTable** accepts a statement id and returns the table storing individual user choices for this statement. The function **mapP3PPurposeToChoiceColumn** accepts a statement-purpose pair and returns the column in the choice table which records the corresponding users' choices. Both these functions are driven from metadata.

```
1  for each statement s in policy do begin
2    for each purpose p in s do begin
3      for each recipient r in s do begin

4        print "create restriction "
5        print generate-unique-restriction-name()
6        print " on table "
7        print mapP3PStatementToTable(s)
8        print " for public "
9        print " to cells "
10       print mapP3PDataTypeToColumns(s)

11       if (p.required != always) then
12         print "where exists (select 1 from "
13               + mapP3PPurposeToChoiceTable(s)
14               + " p where p.ID = "+ mapP3PStatementToTable(s) +".ID
15               and "+ mapP3PPurposeToChoiceColumn(s,p) + "= 1))"

16       if (r.required != always) then
17         print "and exists (select 1 from "
18               + mapP3PRecipientToChoiceTable(s)
19               + " r where r.ID = "+ mapP3PStatementToTable(s) +".ID"
20               + "and "+ mapP3PRecipientToChoiceColumn(s, r) + "= 1))"
21       print "for purpose" + p.name
22       print "for recipient" + r.name
23     end
24   end
25 end
26 print "restricting access to select"
```

Figure 6. Algorithm for translating a P3P privacy policy into fine grained cell level restrictions

5. Related Work

5.1 Oracle

Oracle has introduced a fine-grained access control capability via their security policy concept [5, 7] which, once defined on a table or view, modifies any future query against that table by adding a predicate into the query. In essence, they have allowed row restrictions traditionally handled by views to be dynamically added to queries [8].

The fundamental difference between the Oracle approach and the one in this paper is that Oracle modifies the query by adding predicates while the approach in this paper leaves the query alone and effectively modifies the table being accessed by injecting a dynamically created view of the table between the query and the target table.

The Oracle approach shares the following advantages with our design:

- It is pervasive to all users of the table.
- It does not require application modification.
- It does not require a large number of statically defined views.

Its primary disadvantages are:

- It requires user programming of a strictly defined "predicate producing" procedure in order to implement a security policy.
- It does not address column or cell restrictions.

5.2 Sybase

Sybase Adaptive Server version 12.5 has introduced a feature called row level access control [9] that enables the database owner or table owner to restrict access to a table's

rows by defining access rules and binding those rules to the table. Access to data can be further controlled by setting application contexts and creating login triggers.

Access rules apply restrictions to retrieved data, enforced on select, update and delete operations. Adaptive Server enforces the access rules on all columns that are read by a query, even if the columns are not included in the select list. Using access rules is similar to using views, or using an ad hoc query with where clauses. The query is compiled and optimized after the access rules are attached, so it does not cause performance degradation. Access rules provide a virtual view of the table data, the view depending on the specific access rules bound to the columns.

Our proposal differs from the Sybase row level access control solution as follows:

- It allows restrictions to be defined on columns and cells in addition to rows.
- A restriction can contain as many predicates as desired and this is done in a single statement (i.e., create restriction). Sybase would need to create a separate access rule for each predicate, and'ing them, and then binding them to the appropriate columns.

6. Conclusion

Databases of the future must ensure the privacy of the data subjects that they store information on. The security functionality offered by current commercial database products is not adequate to enforce privacy compliance. The main contributions of this paper are:

- Language constructs for specifying restrictions at the level of a row, a column, or a cell that integrate well with the rest of the relational database infrastructure.
- Semantics of combining multiple restrictions.
- Design for implementing the proposed constructs.
- Algorithm for translating a P3P privacy policy into the proposed constructs.

Our fond hope is that this paper will serve to create dialog on the right functionality that the database systems must support and the efficient ways of its implementation.

Acknowledgments We wish to thank Alvin Cheung for useful comments on the paper.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *28th Int'l Conference on Very Large Databases*, Hong Kong, China, August 2002.

[2] D. Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann, 1998.

[3] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. W3C Recommendation, April 2002.

[4] US Department of Health and Human Services. http://www.hhs.gov/ocr/hipaa.

[5] T. Kyte. Fine-grained access control. Technical report, Oracle Corporation, 1999.

[6] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting disclosure in Hippocratic databases. In *30th Int'l Conf. on Very Large Data Bases*, Toronto, Canada, August 2004.

[7] A. Nanda and D. K. Burleson. *Oracle Privacy Security Auditing*. Rampant, 2003.

[8] M. Stonebraker and E. Wong. Access control in a relational data base management system by query modification. In *ACM/CSC-ER*, 1974.

[9] Sybase. *Sybase Adaptive Server Enterprise 12.5, System Administration Guide, Row Level Access Control*. http://sybooks.sybase.com/onlinebooks.

ID	Name	HomePhone	WorkPhone	Salary
1	Alicia Campbell	408-418-5198	408-419-9111	10,000
2	Bob Bobbett	408-418-5198	408-419-9112	20,000
3	Carl Abrahams	408-333-6633	408-419-9113	30,000
4	Dan Charmer	408-432-8644	408-419-9114	40,000
5	Ellen Generous	408-555-1235	408-419-9115	50,000

Table 3. Table of BlueCo's clients

Name	HomePhone	OfficePhone
Alicia Campbell	-	408-419-9111
Bob Bobbett	408-418-5198	-
Carl Abrahams	408-333-6633	408-419-9113

Table 4. Cell level enforcement

A. The Case for Cell Level Enforcement

Compliance with current privacy legislation mandates that the user's consent be obtained for the use/disclosure of the personal information stored about them. Row or column level restriction are not adequate for modeling scenarios where individuals may make opt-in/out choices with different aspects of their information. To achieve this goal of minimal disclosure while allowing useful tasks to be performed on relevant information, cell level enforcement is key. A similar case for cell level enforcement has been made in [6].

Consider a scenario requiring adherence to the HIPAA regulation [4]. BlueCo is a healthcare provider that stores personal data on individuals who enroll in its plans. BlueCo has affiliations with a number of hospitals, research institutions, and marketing companies. Under HIPAA, any individually identifiable healthcare information held or transmitted by BlueCo is considered protected information. For any use or disclosure of protected health information that is not for treatment, payment, or health care operation and that is not otherwise permitted (e.g. law enforcement), BlueCo must get the data subject's consent.

Assume a simplified version of BlueCo's database given in Table 3. ResearchCo is an epidemiological research institute that periodically harvests BlueCo's data. Under HIPAA, all clients must give their consent for release of their home and office numbers.

Alicia Campbell opts out of having her home phone number, but does not mind if BlueCo discloses her office number. Suppose John Seeker, a researcher at ResearchCo issues the following query:

select name, homephone, officephone
from clients **where** salary $\leq$ 30000

Given the choices that Alicia has made, only her name and office phone number should be displayed as shown in Table 4.

Name	HomePhone	OfficePhone
Carl Abrahams	408-333-6633	408-419-9113

Table 5. Row level enforcement

Database systems employing row level controls restrict disclosure to all information in a particular row, when a restriction is only on particular columns in that row. Thus, using conventional row level controls, the results for the query are those shown in Table 5. Both Alicia and Bob are no longer present in the result, even though they have agreed that one of their two phone numbers can be disclosed.

This simple example illustrates the inadequacy of row level restrictions. Similar arguments can be made for column level restrictions. They are not flexible enough to allow disclosure of non-sensitive data and suppression of sensitive data on a subject by subject basis.