Auskunftsbeschränkung in Hippokratischen Datenbanken

Falk Husemann

Lehrstuhl 6 für Informationssysteme und Sicherheit

Zusammenfassung Diese Seminararbeit stellt zwei wissenschaftliche Arbeiten vor , die das Konzept der *hippokratischen Datenbanken* zur Sicherung von Datenschutz in Datenbanksystemen darstellen. Das Konzept der hippokratischen Datenbanken wird gegen existierende Ansätze zum Datenschutz im Datenbanksystem abgegrenzt. Es werden eine Meta-Sprache zur Speicherung der Datenschutzrichtlinien, zwei Semantiken und ein Anfragemodifikationsalgorithmus eingeführt. Dann wird die Tabellen-Semantik vorgestellt, die Sichten auf die Datentabellen erzeugt und die Anfrage-Semantik, die keine vollständigen Sichten erzeugt. Das Konzept verwendet einen Begriff aus der SQL-Semantik ($NULL$), um verbotene Felder abzubilden.

1 Einleitung

Wir leben in einem Zeitalter der Digitalisierung von Informationen. Datensammlung durch Harvesting oder Crawling gestaltet sich immer einfacher und die Weitergabe von persönlichen Daten zur Anmeldung an Informationssystemen ist in allen Bereichen des Lebens notwendig. Datenschutzskandale und Datendiebstahl sind heute aktuelle Themen. Die hier betrachteten Arbeiten stellen eine Erweiterung zur Sicherung dieser Daten in SQL-Datenbanken vor. Aber können Datenschutzprobleme überhaupt technisch in einem Datenbanksystem gelöst werden?

Die Grundvoraussetzung für eine prüfbare Zweckbindung ist die zuverlässige Kenntnis über den Anfragekontext. Wer fragt an und wofür sind die angefragten Daten notwendig? Nur wenn diese Fragen beantwortet werden kann ein technisches System anhand von Richtlinien entscheiden, ob der Zugriff legitim oder verboten ist. Die betrachteten Arbeiten [1] und [2] setzen diese Zuverlässigkeit des Anfragekontexts voraus und stellen darauf aufbauend Werkzeuge zur Garantie der Zweckbindung innerhalb des Datenbanksystems zur Verfügung.

2 Existierende Ansätze

2.1 Datenschutz auf Anwendungsebene

Die hinsichtlich der Datenverarbeitung am weitesten vom Datenbanksystem entfernte Methode, um Datenschutz sicherzustellen, ist der Datenschutz auf Anwendungsebene. Dabei wird das Einhalten der definierten Richtlinien in der dem

Datenbanksystem vorgeschalteten Anwendung sichergestellt.

Diese Anwendung verarbeitet die Anfragen des Benutzers und stellt dann eine Anfrage an die Datenbank. Die Anwendung erhält die Anwort der Datenbank. Daraufhin prüft die vorgeschaltete Anwendung die Antwort gegen eine vorher definierte Menge von Regeln (Datenschutzrichtlinie) und löscht verbotene Spalten, Zeilen, oder Felder aus der Antwort. Die so modifizierte Antwort wird an den Benutzer geliefert.

Problematisch ist bei diesem Ansatz, dass der Kontext der Daten unbekannt ist. Angenommen, es werden Patientendaten aus einer Datenbank abgefragt und ein Schutz auf Anwendungsebene soll diese schützen, indem verbotene Felder aus dem Ergebnis entfernt werden, so sind Rückschlüsse gegebenenfalls dennoch möglich.

Beispiel 1 *Krankendaten erschleichen*
SELECT *Name, Krankheit* ***FROM*** *Patienten* ***WHERE*** *Krankheit = "Hepatitis"*

Das Ergebnis des Beispiels enthält eine Liste aller Patienten, die an der Krankheit Hepatitis leiden, wobei nur das Feld *Krankheit*, das diese Diagnose enthält, entfernt wurde. Dadurch sind Rückschlüsse auf die Krankheit leicht möglich. Ein Patient in der Ergebnismenge der Anfrage aus dem Beispiel leidet an Hepatitis, da genau nach dieser Krankheit gefragt wurde.

2.2 Statistische Datenbanken

Dieser Ansatz dient dazu, statistische Daten, wie Summe oder arithmetisches Mittel auszugeben. Dabei soll die Vertraulichkeit der Daten und die Anonymität gewahrt werden. Um dieses Ziel zu erreichen existieren mehrere Ansätze.

Der erste soll Vertraulichkeit durch Anfragebeschränkung auf Basis von vorherigen Sitzungen und dem daraus resultierenden Anfrageverhalten sicherstellen. Der zweite soll Anonymität durch Schlüsseltausch[1] in den Tabellen sicherstellen. Dabei wird die Annahme gemacht, dass sich die Summe über eine Spalte nicht ändert, wenn die Zuordnung der Schlüssel[2] zufällig permutiert wird. Der dritte Ansatz soll Anonymität durch Hinzufügen von sich aufhebenden zufälligen Datensätzen herstellen. Dabei wird zu einem zufällig erzeugten Datensatz der inverse Datensatz gebildet und ebenfalls eingefügt. Das muss nicht immer möglich sein[3] .

Problem am dritten Ansatz ist, dass entweder starke Anonymität oder präzise

[1] Beispiel: Umordnung des Zusammenhangs zwischen Gehalt und Mitarbeiternummern. Summe der Gehälter ändert sich nicht.
[2] Teilmenge der Attribute eines Relationenschemas. *Beispiel: Die ISBN-Nummer eines Buches kann als Schlüssel zur eindeutigen Indentifikation dienen.*
[3] Es muss ein inverses Element in der Relation definiert sein.

Ergebnisse erreicht werden. Anonymität ist genau dann erreicht, wenn zufällige Datensätze ohne ihr Inverses eingefügt werden. Eine Zweckbindung wird nicht garantiert.

2.3 Sichere Datenbanken

In sicheren Datenbanken werden die Daten verschlüsselt abgelegt, und es findet eine Kontrolle mittels Sicherheitsmodellen statt. Dabei ist zwischen *Discretionary Access Control*, bei dem Zugriffsrechte auf Basis der Identität des Akteurs vergeben werden und vom Eigentümer bzw. den Eigentümern eines Objekts festgelegt werden, und *Mandatory Access Control*[4] zu unterscheiden. Mandatory Access Control nutzt Schutzstufen[5], die Objekten zugeordnet werden. Ein Subjekt muss dabei mindestens über dieselbe Schutzstufe, wie das Objekt auf das lesend zugegriffen wird, verfügen. Die Schutzstufen werden vom Systemadministrator festgelegt. Beide Modelle haben gemeinsam, dass die Festlegung der Zugriffsrechte nicht den Zweck des Zugriffs miteinbezieht.

3 Hippokratische Datenbanken

3.1 Motivation

Über alles, was ich während oder außerhalb der Behandlung im Leben der Menschen sehe oder höre und das man nicht nach draußen tragen darf, werde ich schweigen und es geheimhalten.(Hippokrates von Kos, ca. 400 v.Chr.)

Diese antiken Überlegungen sind die Motivation zur Spezifikation von hippokratischen Datenbanken.

3.2 Prinzipien hippokratischer Datenbanken

Die hippokratischen Datenbanken sollen Datenschutz im Datenbanksystem leisten. Dabei sollen sie die Prinzipien des Datenschutzes garantieren: Zweckbindung, Verhältnismäßigkeit, Datensparsamkeit und Transparenz.

Ergänzend sollen auch Methoden der sicheren Datenbanken verfügbar gemacht werden und die Prinzipien-Konformität automatisch nachprüfbar sein. Dabei wird die Zweckbindung als zentraler Aspekt festgelegt und Vertraulichkeit in Abhängigkeit dazu modelliert.

3.3 Systemübersicht

Wie können die eingangs geforderten Eigenschaften ohne umfangreiche Anpassungen bestehender Anwendungen umgesetzt werden? Die hier behandelten Arbeiten wählen SQL als Grundlage für hippokratische Datenbanken und bilden

[4] Ein Beispiel für ein so konzipiertes Sicherheitsmodell ist das Bell LaPadula-Modell.
[5] Wie z.B.: *Öffentlich, Vertraulich, Geheim, Streng Geheim.*

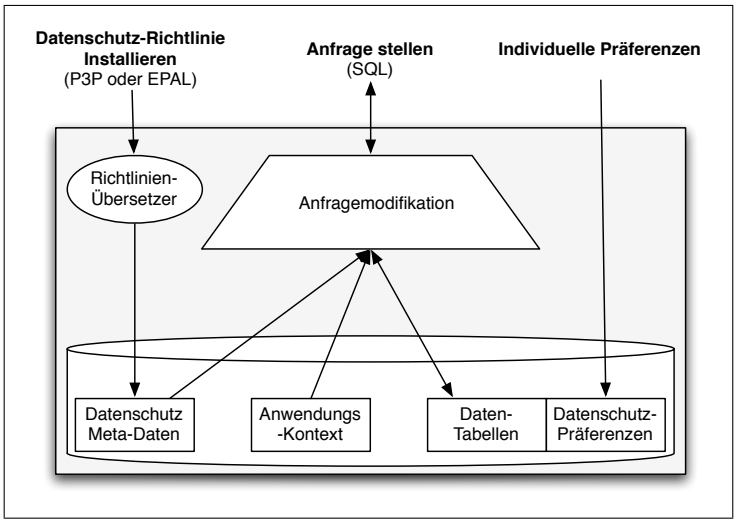

Abbildung 1. Systemübersicht

die entwickelten Strukturen (s. Systemübersicht) auf die SQL-Spache ab. Dadurch sollen bereits produktiv eingesetzte Anwendungen ohne Anpassungen mit hippokratischen Datenbanken funktionieren.

Das Konzept erfordert keine Neuimplementierung eines Datenbanksystems, vielmehr können erprobte SQL-Implementierungen, wie DB2 oder Oracle, die seit über 20 Jahren existieren, erweitert werden. Von Optimierungen in diesen Datenbanksystemen profitieren hippokratische Datenbanken direkt. Die Architektur soll wie in *Abbildung 1* konstruiert werden. Ein Richtlinienübersetzer führt die notwendigen Modifikationen der Richtliniensyntax durch, ohne die Semantik der SQL-Sprache zu verändern. In Datentabellen werden die Datenschutzpräferenzen der Anwender gespeichert. Anfragen werden durch eine Vorverarbeitung[6] anhand der im Datenbanksystem gespeicherten Richtlinien und Präferenzen modifiziert. Dadurch wird sichergestellt, dass keine Anfragen von Benutzern oder Anwendungen direkt an das Datenbanksystem gestellt werden.

4 Auskunftsbeschränkung

4.1 Anforderungen

Um hippokratische Datenbanken in ein bestehendes Datenbanksystem einzubinden sollen die existierenden Sprachen zur Datenschutzrichtlinienspezifikation automatisch semantisch äquivalent umgewandelt werden, sodass keine Neuformulierung der Datenschutzrichtlinien notwendig ist. Die so gewonnenen Richtlinien

[6] Ein denkbarer Mechanismus hierfür ist ein vorgeschalteter Proxyserver.

sollen dann in einfachen Datenstrukturen gespeichert werden und auch benutzerspezifische Datenschutzpräferenzen berücksichtigen können. Dabei soll diese Datenstruktur möglichst sparsam hinsichtlich zusätzlich notwendigem Speicherplatz und zusätzlich notwendigen Datenbankoperationen sein. Dafür wird eine Meta-Sprache vorgestellt, die diese Forderungen erfüllt.

Weiter wird von den Autoren gefordert, dass jede Anfrage an das Datenbanksystem auf Feldebene[7] geprüft werden kann.

4.2 Meta-Sprache

Die in der Arbeit eingeführte Meta-Sprache dient als Mittler zwischen bestehenden Datenschutzrichtlinenspezifikationen in EPAL[8] und P3P[9]. Diese Meta-Sprache kann P3P komplett abbilden. EPAL wird mit kleinen Einschränkungen hinsichtlich tief verschachtelter Richtlinien unterstützt.

4.3 Richtlinien der Meta-Sprache

Definition 1 (Format einer Regel der Meta-Sprache).
< daten, zweck-empfänger paar, bedingung >

Gegeben Daten, ein Paar von Zweck und Empfänger, sowie eine Bedingung, können aus obigen Regeln der Meta-Sprache Datenschutzrichtlinien (eine Menge von Regeln) erzeugt werden. Die informelle Bedeutung ist, dass die Daten für Zweck unter Bedingung an Empfänger weitergegeben werden dürfen. Hierbei wird der Angabe des Zwecks vom Benutzer oder Frontends vertraut. Die Bedingung ist ein in SQL auswertbares Prädikat, das nach Auswertung entweder den Wahrheitswert *wahr* oder *falsch* ergibt.

Beispiel 2 *< adresse, werbung-caritas, optin=yes >*
Die Adresse darf zum Zweck der Werbung nur dann an die Caritas weitergegeben werden, wenn der Benutzer explizit zugestimmt hat.

Beispiel 3 *< VISANummer, rechnung-rechnungsstelle, offeneRechnungen >= 1 >*
Die VISA-Nummer darf zum Zweck der Rechnungsstellung nur dann an die Rechnungsstelle weitergegeben werden, wenn es mindestens eine offene Rechnung gibt.

---

[7] Es soll der Zugriff auf jedes einzelne Feld gegen die Datenschutzrichtlinien geprüft werden und einzeln verboten oder gestattet werden können.

[8] Enterprise Privacy Authorization Language, XML-basierte Beschreibungssprache für Unternehmen und [5].

[9] Platform for Privacy Preferences Project, Standard des W3C zur Speicherung von Datenschutzinformationen: `http://www.w3.org/TR/2002/REC-P3P-20020416/`

4.4 Tabellen-Semantik

4.5 Modell

Mit der Tabellen-Semantik soll jedes Zweck-Empfänger Paar eine eigene Sicht auf jede Tabelle erzeugen. Dabei sollen durch eine Regel erlaubte Felder in der Sicht den Inhalt der Felder aus der Datentabelle enthalten. Verbotene Felder sollen in der Sicht den Wert $NULL$ enthalten. Das Verwenden eines schon existierenden Begriffs aus der SQL-Semantik, nämlich $NULL$, bringt einige Vorteile mit sich. Eine Anwendung, die $NULL$ als Ergebnis der Anfragemodifikation erhält, kann dies gegebenenfalls als *false negative* Fehler, und nicht als *false positive* Fehler behandeln [10]. Weiter sind viele Anwendungen bereits darauf eingestellt, $NULL$ als Fehlerfall zu erkennen, sollte eine Anfrage nach einem verbotenen Feld $NULL$ als Ergebnis liefern. Durch Verwendung dieses Begriffs aus der SQL-Sprache kann außerdem die Neuentwicklung eines Datenbanksystems vermieden werden. Für die folgende formelle Einführung der Sichten mittels Tabellen-Semantik werden einige vorbereitende Festlegungen benötigt.

Definition 2. *Tabellen-Semantik*
Sei T_{P_j} die Sicht (view) des Zweck-Empfänger Paar P_j auf Tabelle T.
Sei K die Menge von Spalten, die den Primärschlüssel von Tabelle T bilden.
Sei $eval(t[i,j])$ die Funktion, um das Zweck-Empfänger Paar j, das für Zeile t angegeben ist, auszuwerten. Die Funktion eval liefert als Ergebnis $false$, falls der Zugriff für das Paar verboten ist, und $true$ sonst.

$$T_{P_j} = \{r \mid \exists t \in T \wedge \forall i, 1 \leq i \leq n \tag{1}$$

$$(r[i] = t[i] \ if \ eval(t[i,j]) = true, \tag{2}$$

$$r[i] = null \ otherwise) \tag{3}$$

$$\wedge \ r[K] nonenull\} \tag{4}$$

Die Sichten werden also durch Betrachtung der Regeln für jede Zeile und den Primärschlüssel erzeugt, wobei der Primärschlüssel doppelt betrachtet wird. Die Bedingung, dass der Zugriff auf den Primärschlüssel erlaubt sein muss, um überhaupt Felder in der Ergebnismenge anzuzeigen, führt dazu, dass es komplett mit $NULL$ gefüllte Datensätze geben kann.

4.6 Anfrage-Semantik

Um dieses Problem zu lösen und kleinere Ergebnismengen zu erhalten, indem die doppelte Prüfung des Primärschlüssels ausgelassen wird, führen die Autoren die Anfrage-Semantik ein. Die Anfrage-Semantik soll nicht nur kleinere Ergebnismengen liefern, sondern erzeugt keine Sichten für jedes Zweck-Empfänger-Paar

[10] Im Beispiel zur Patientendatenbank führt das dazu, dass der Patient nicht fälschlich als erkrankt, sondern als nicht erkrankt erkannt wird.

mehr. Es soll eine konkrete Anfrage bearbeitet werden. Darum kann das Ergebnis auch eine unvollständige Abbildung der geschützten Tabellen sein.

Für die Definition der Anfrage-Semantik werden wieder die Begriffe der Tabellen-Semantik benötigt.

Definition 3. *Anfrage-Semantik*

$$T_{P_j} = \{r \mid \exists t \in T \wedge \forall i, 1 \leq i \leq n \tag{5}$$

$$(r[i] = t[i] \ if \ eval(t[i,j]) = true, \tag{6}$$

$$r[i] = null \ otherwise\} \tag{7}$$

Durch das Auslassen der Bedingung $r[K]$ *nonenull* wird die doppelte Prüfung des Primärschlüssels nicht mehr durchgeführt.

4.7 Anfragemodifikation

Die beiden vorgestellten Semantiken, sowie die Meta-Regeln zur Spezifizierung von Datenschutzrichtlinien, sollen jetzt dazu eingesetzt werden, SQL-Anfragen zu modifizieren und die Zweckbindung sicherzustellen. Dafür werden zusätzlich zur zu schützenden SQL-Tabelle oder -Datenbank drei weitere Komponenten benötigt. Eine zusätzliche Tabelle, um die globalen Datenschutzrichtlinien zu speichern, eine weitere Tabelle für die Bedingungen der Meta-Sprache und einen der zwei Anfragemodifikationsalgorithmen.

4.8 Datenschutzrichtlinien

Die Regeln der Meta-Sprache werden in zwei Tabellen *policy* und *conditions* aufgeteilt. Hier exemplarisch für das Beispiel zur Meta-Sprachen Spezifikation.

Beispiel 4 *< visanummer, rechnung−rechnungsstelle, offeneRechnungen >= 1 >*

Hier soll das Feld visanummer *nur zum Erstellen von Rechnungen (Zweck hier* rechnung*) dem Empfänger, der* rechnungsstelle *zugänglich gemacht werden, wenn es mindestens eine offene Rechnung gibt.*

Die Regel in Meta-Sprache wird zur policy *Tabelle:*

RuleID	*PID*	*Purpose*	*Recipient*	*Table*	*Column*	*CondID*
R1	*P1*	*Rechnung*	*Rechnungsstelle*	*Kunden*	*VISANummer*	*C1*

Die Spalten der obigen Tabelle stellen das Zweck-Empfänger-Paar mit den zusätzlich geforderten Spalten dar. Die Spalten $RuleID$ und PID dienen der Numerierung und der Zuordnung einer Regel zu einer Datenschutzrichtlinie. Die Spalten *Purpose* und *Recipient* repräsentieren das Tupel und die Spalten *Table* und *Column* die zu schützende Tabelle und Spalte, auf die sich diese Regel bezieht.

CondID gibt die eindeutige Identifizierungsnummer der Regel in SQL-Sprache an.

Die Tabelle *conditions* wird von den Regeln getrennt. Das spart Speicherplatz bei Mehrfachnennung von Bedingungen. Die Bedingung zum obigen Beispiel lässt sich damit in einer Tabelle ablegen.

Beispiel 5

CondID	*Predicate*
C1	*EXISTS (SELECT offeneRechnungen FROM Kunden WHERE offeneRechnungen >= 1)*

4.9 CASE-Anweisung

Um mit den Datenschutzrichtlinien nun Anfragen zu modifizieren, sodass die Zweckbindung garantiert werden kann, soll zuerst die Anfrage mit einer CASE-Anweisung umschlossen werden. Das CASE-Konstrukt dient dazu, Bedingungen in einer SQL-Anfrage auszuwerten[11]. Dafür wird die SQL-Anfrage wie folgt um eine CASE-Anweisung erweitert:

Definition 4. *CASE-Anweisungsmodifikation*
SELECT CASE WHEN *BedingungFeld* ***Then*** *Feld* ***ELSE null END***
FROM *Tabelle* ***WHERE EXISTS*** *BedingungPK*

Dabei sei $Feld$ das Feld, auf das zugegriffen werden soll. $BedingungFeld$ sei die Bedingung für dieses Feld und $BedingungPK$ die Bedingung für die Spalten, aus denen der Primärschlüssel für die anzufragende Tabelle besteht. Da wie eingangs dargestellt, die Nähe zu SQL vorausgesetzt wurde, deckt die Definition so auch mehrere Felder und zusammengesetzte Primärschlüssel ab. Dafür werden die booleschen Operatoren der SQL-Syntax benutzt. Soll jetzt auf das Feld VISANummer zugegriffen werden, für das im vorherigen Beispiel eine Datenschutzrichtlinie in SQL-Tabellen übertragen wurde, kann nun die Anfrage modifiziert werden.

Beispiel 6 *Anfrage nach VISANummer*
SQL-Anfrage nach VISANummer vor der Modifikation:
SELECT *VISANummer* ***FROM*** *Kunden* ***WHERE*** *Name = 'Bob'*

SQL-Anfrage nach VISANummer nach der Modifikation:
SELECT CASE WHEN EXISTS (SELECT *offeneRechnungen* ***FROM*** *Kunden*
WHERE *offeneRechnungen >= 1)* ***THEN*** *VISANummer*
ELSE null END FROM *Kunden*

[11] *Beispiel für eingebettete Bedingung: SELECT Produkt,Verfuegbarkeit CASE WHEN anzahl > 0 THEN 'Auf Lager' ELSE 'Muss bestellt werden' END FROM Artikel*
Bedeutet: Wenn ein Produkt mehr als null mal verfügbar ist, gib *Auf Lager* aus, sonst *Muss bestellt werden.*

4.10 OUTER JOIN-Anweisung

Jetzt soll, ähnlich wie im IEEE SeaView Datenbankmodell [3], die SQL-Anfrage in eine OUTER JOIN-Anfrage eingebettet werden. Diese Alternative ist nach [2] notwendig, da das vorgestellte Verfahren zur Anfragemodifikation mittels CASE-Konstrukt die Annahme macht, dass für eine Spalte entweder gilt, dass sie vollständig sichtbar oder unsichtbar ist. Das grundsätzliche Format zur Ergänzung der Anfragen sieht wie folgt aus:

Definition 5. *OUTER JOIN-Anweisungsmodifikation*
SELECT *Feld* ***FROM***
*(****SELECT*** *Pk* ***FROM*** *Tabelle* ***WHERE*** *BedingungPK)* ***AS*** *t1(Pk)*
LEFT OUTER JOIN
*(****SELECT*** *Pk, Feld* ***FROM*** *Tabelle* ***WHERE*** *BedingungFeld)* ***AS*** *t2(Pk,Feld)*
ON *t1.Pk = t2.PK*

Aus dem schon von der Modifikation mittels CASE-Anweisung bekannten Beispiel lässt sich jetzt eine entsprechend angepasste SQL-Anfrage gewinnen.

Beispiel 7 *Anfrage nach VISANummer*
Vor der Modifikation
SELECT *VISANummer* ***FROM*** *Kunden* ***WHERE*** *Name = 'Bob'*

Nach der Modifikation
SELECT *VISANummer* ***FROM***
*(****SELECT*** *Name* ***FROM*** *Kunden* ***WHERE*** *Bedingung)* ***AS*** *t1(Name)*
LEFT OUTER JOIN
*(****SELECT*** *Name, VISANummer* ***FROM*** *Kunden*
WHERE *offeneRechnungen >=1)* ***AS*** *t2(Name,VISANummer)*
ON *t1.Name = t2.Name*

4.11 Algorithmus Tabellen-Semantik

Bisher blieb offen, wie die Anfragemodifikation algorithmisch funktioniert. Der Algorithmus zur Modifikation von Anfragen mittels Tabellen-Semantik soll jetzt eingeführt werden. Dafür muss die Eingabe definiert werden. Weiter sind drei Hilfsfunktionen notwendig, die dazu dienen, die Richtlinie für eine Spalte einer Tabelle abzurufen und die dazugehörende Bedingung zu berechnen. Es wird noch eine Auswertungsfunktion benötigt, die die Bedingung für einen Datensatz prüfen kann.

Definition 6. *Anfragemodifikationsalgorithmus Tabellen-Semantik*
Sei Q die zu modifizierende Anfrage.
Sei PID die Datenschutzrichtlinienidentifikationsnummer.
Sei P der Zweck und R der Empfänger der Anfrage.
Sei $C[x]$ eine Spalte der Tabelle C.
Sei t eine Tabelle, die von Q referenziert wird.

Funktion $GetPolicy(PID, P, R, t, C[x])$*, die entweder* verboten *zurückgibt, falls der Zugriff verboten ist, oder* erlaubt*, falls der Zugriff erlaubt ist.*
Funktion $GetCondition(PID, P, R, t, C[x])$*, die aus der Bedingungstabelle das Feld* Predicate *liest.*
Auswertungsfunktion $Eval(Condition, Row)$*, die den Wahrheitswert* wahr *oder* falsch *für eine Bedingung und einen Datensatz liefert.*
$Q' = Q$
for all *Tabellen t, die von Q referenziert werden* ***do***
    $C[] \leftarrow$ *Spalten-Liste von t*
    for $i = 0$ *bis Länge von* C ***do***
        if $GetPolicy(PID, P, R, t, C[i])) = erlaubt$ ***then***
            $C'[i] \leftarrow C[i]$
        else if $GetPolicy(PID, P, R, t, C[i] = verboten$ ***then***
            $C'[i] \leftarrow NULL$
        else
            $condition \leftarrow GetCondition(PID, P, R, t, C[i])$
            $C'[i] \leftarrow <$ ***if*** $eval(condition, row)$ ***then*** $C[i]$ ***else*** $NULL$ ***endif*** $>$
        end if
    end for
    $K[] \leftarrow$ *Die Menge der Spalten aus t, die Primärschlässel sind*
    $predicate \leftarrow < \neg(\exists k \in K$ *sodass* $Policy(PID, P, R, t, k)$ *verboten ergibt* $)\wedge$
    $\neg(\exists k \in K$ *sodass* $\neg eval(GetCondition(PID, P, R, t, k), row)) >$

    Erstelle Unteranfrage t' *mit Projektionsliste* C'
    Hänge Bedingungsfunktion predicate an t' *an*
    Ersetze die Referenz auf t *in* Q' *mit* t'
end for
return *Q'*

Der Algorithmus dient dazu, die Sichten durch Modifikation der Anfragen für die Empfänger mit deren Zweck herzustellen. Dabei wird eine Anfrage Q in eine modifizierte Anfrage *Q'* überführt. Schrittweise wird für jede Tabelle t, die in der Anfrage auftaucht, jedes Feld in jeder Spalte aus t betrachtet. Ist der Zugriff für den Empfänger mit angegebenem Zweck verboten, wird der Inhalt des Feldes durch $NULL$ ersetzt. Darauf folgt die Betrachtung aller Spalten der Tabelle, die den Primärschlüssel bilden. Für den Primärschlüssel wird eine Bedingungsfunktion [12] gebildet, die in die Anfrage eingebettet und durch das Datenbanksystem ausgewertet wird.

5 Kritik an dem vorgestellten Ansatz

Leider lassen die beiden betrachteten Arbeiten ([1] und [2]) einige Fragen offen.

[12] SQL stored procedures, siehe [4]

Korrektheit: [2] stellt zwei Semantiken vor, mit denen es möglich sein soll Auskunftsbeschränkung herzustellen. Den Recherchen zufolge wurde weder die Korrektheit der beiden Semantiken auf der SQL-Sprache, noch die des Anfragemodifikationsalgorithmus bewiesen. Eine induktive Definition der den Semantiken und des Modifikationsalgorithmus zugrundeliegenden Mengen würde den Nachweis der Korrektheit erleichtern.

Gesamtkontext: Die Annahme, dass dem angegebenen Zweck eines Empfängers und auch dessen Identität vertraut werden kann, geht weit über beweisbare Annahmen hinaus. Es ist schwer, sich eine Situation vorzustellen, in der es möglich ist, die Absichten einer Person in einem mathematischen Modell nachzuweisen. Auch der Identität eines Empfängers zu vertrauen ist mindestens schwierig. Aus diesen Gründen ist es fraglich, wie die Wissenschaftler ihr Modell in der Praxis einsetzen.

Literatur

1. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB*, pages 143–154. Morgan Kaufmann, 2002.
2. Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David J. DeWitt. Limiting disclosure in hippocratic databases. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 108–119. Morgan Kaufmann, 2004.
3. Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley. The seaview security model. *IEEE Trans. Software Eng.*, 16(6):593–607, 1990.
4. Jim Melton. *Understanding SQL Stored Procedures: A Complete Guide to SQL/PSM*. Morgan Kaufmann, 1998.
5. William H. Stufflebeam, Annie I. Antón, Qingfeng He, and Neha Jain. Specifying privacy policies with p3p and epal: lessons learned. In Vijay Atluri, Paul F. Syverson, and Sabrina De Capitani di Vimercati, editors, *WPES*, page 35. ACM, 2004.