MASTER

Query rewriting and stored queries for XACML policy enforcement

Martinelli, S.A.

Award date:
2017

Technische Universiteit Eindhoven University of Technology
Department of Mathematics and Computer Science
Web Engineering and Security Research Groups

Query Rewriting and Stored Queries for XACML Policy Enforcement

Master Thesis

Stefano Antonio Martinelli

Supervisors:
Dr. G.H.L Fletcher
Dr. N. Zannone
Dr. N. Yakovets

Eindhoven, July 2017

Abstract

In the last decades, the world has witnessed an exponential growth in data volumes. Thanks to the Internet, this information is exchanged everywhere and then saved in systems which storing capacities are increasing accordingly. A significant part of this data is classified as sensitive or concerning the privacy of a subject. For this reason, confidentiality, integrity, and availability have become essential properties that databases need to provide in order to protect users' information. Consequently, access control with regard to data and databases gained a central role as a mean to address the problem of data security. In this scenario, the OASIS (Organization for the Advancement of Structured Information Standards) proposed XACML (eXtensible Access Control Markup Language), a standard for the specification and enforcement of access control policies. XACML uses a declarative approach that separates the constraints of access control from the data and the database, in favor of compatibility with different systems and ease of policy management over time. On the other hand, the lack of direct integration with database systems entails lower efficiency when querying the database and enforcing access control. In particular, the implementation model of a system employing XACML for access control, as proposed by the OASIS, can be modified in order to improve it in terms of efficiency. The enhancements would be significant especially in situations where the data involved in the security enforcement is massive, since the current implementation performs badly with large datasets. In this context, search engines which employ access control using XACML policies are a scenario of application for an improved security enforcement mechanism.

In this work, we discuss possible alternatives to the model recommended by the OASIS and we propose a new solution that makes use of query rewriting and stored queries to enforce the security constraints expressed in XACML. All the alternative approaches aim to improve the performance of the current XACML implementation by enforcing the security restrictions in the database but still keeping the policy definition and management separated from both the database and the application querying it. We describe and compare possible options to design and implement the structure of the new solution we propose. Then, we evaluate the performances achieved by executing experiments using our approach, a simplified (and more efficient) version of the implementation suggested by the OASIS and other existing alternatives. In conclusion, we summarize the results achieved in the experiments. In particular, we describe how our solution performs better than the other alternatives especially when large volumes of data or complex security restrictions are involved during access control enforcement.

Acknowledgments

This thesis marks the conclusion of my Master project and my experience at the Eindhoven University of Technology. The two years of my Master have been a challenging but enjoyable journey that brought me to the generation of this document.
Firstly, I would like to thank my supervisors dr. George Fletcher, dr. Nicola Zannone and dr. Nikolay Yakovets. They provided me a really interesting topic, they patiently guided me during the project suggesting new ideas that stimulated and inspired me, and they gave me valuable feedback until the final days.
I also have to thank my the fellow travelers during these years: Robin, Igor, Daniyal, Nitin, Rodrigo, Alberto, Lorenzo, Giorgio, Marcin, Eduardo and all the other friends I met in Eindhoven. All the projects at the university, but also the talks with these people from different countries, with different cultures and with different visions of the world, have helped my personal growth but most of all have changed me and my life.
Last but not least, I would like to immensely thank my parents, my sisters and my whole family. Without them, nothing I have reached in these two years would have been possible. They encouraged all my decisions and they have always supported me.
This thesis is just a little accomplishment among the many other personal achievements I attained in these two years thanks to the people around me. Therefore, thank you again, thank you so much!

Stefano Antonio Martinelli
July 2017

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

Every information system which contains sensitive or private data should not be accessible to anyone that is able to connect to it. In this context, access control plays a fundamental role in order to guarantee the confidentiality and integrity of the data stored in a system.
In recent years, access control models and systems have evolved in order to cope with the increasing volume and complexity of data. In this context, Attribute-based Access Control (ABAC) is one of the newest access control models that allows the definition of fine-grained policies thanks to its broad expressiveness. As a consequence, security-policy languages like XACML (eXtensible Access Control Markup Language) [8] are spreading as they enable the definition of security policies attributable to ABAC. As stated in a NIST publication in 2013 [14], XACML provides great security and flexibility, hence it has become the current major standard for the definition of ABAC model policies.
Together with the creation of the language, the OASIS (Organization for the Advancement of Structured Information Standards) designed a non-normative model and data-flow for a system employing XACML policy enforcement [19]. This model has in its adaptability to different databases and applications its main strength. However, it is not very efficient as some processes involved in the data-flow of a system using this model are very expensive (especially when they have to deal with big volumes of data). In particular, the implementation model proposed by the OASIS applies policy enforcement on the result of the user's query, hence at a later stage, after the query is executed. In fact, for every entry resulting from the initial user query, a check is performed in order to verify if the retrieved element is compliant with the security restrictions expressed in the XACML policies.
This procedure entails two expensive operations (which cost may vary depending on their complexity): the retrieval of the elements in the database as specified by the user's query and the examination of each individual element retrieved to verify its compliance with the access control conditions. These tasks are particularly expensive when the number of results of the user's initial query is large. An example of a scenario where the current security enforcement implementation does not perform well is a search engine. This type of system deals with a huge amount of data and usually outputs to the user a lot of results. If access control policies, specified in XACML, are enforced on search engine results employing the OASIS's approach, then the performance of the whole system would be negatively affected. The reason is that the number of results to check for compliance would be typically very large as the results correspond to the information retrieved from the execution of the search query executed by the user on the search engine.
Hence, a modification in the current XACML implementation is needed to enhance the performance of the policy enforcement system, especially for big data.

In this work, after recalling the main concepts of XACML language, structure and policy en-

forcement, and after presenting some related work, we illustrate two query rewriting approaches which aim to improve the performance of the OASIS's implementation model. In particular, the first query rewriting approach we describe is a solution which was already proposed in similar manners in other works. Instead, the latter is an alternative policy enforcement approach that we propose. More specifically, this new query rewriting solution makes use of database objects (i.e. virtual or materialized views) generated employing stored queries. Both the query rewriting approaches avoid the expensive operation of checking each retrieved element to verify its compliance with the security policies. This is achieved by including the access control restrictions to the rewritten query. Consequently, the elements retrieved with the execution of the rewritten query correspond to the desired, security-enforced results. It is important to note that, with the query rewriting approaches, the security restrictions are enforced within the database but the policy definition and management is kept separated from both the database and the application querying it. This is the same of what currently happens with the OASIS's implementation.

1.2 Contributions

The main contribution of this work consists in the design and implementation of a new approach for the enforcement of XACML policies in the context of the OASIS model (Chapter 5).
In particular, to compare our solution with available alternatives, we implemented (Chapter 6):

- *OASIS' (simplified OASIS approach)*, a simplified and more efficient version of the implementation suggested by the OASIS;

- *APF (A-Poseriori Filter)*, an "a-posteriori filter", where the results of the subject's query are stored in a Java *List* and the security conditions are enforced by a Java component;

- *QR-TA (Query Rewriting - Trivial Approach)*, a query rewriting approach that consists in just adding the security restrictions to the user's query;

- *QR-SQ (Query Rewriting employing Stored Queries)*, a new query rewriting approach which makes use of database objects (i.e. virtual and materialized views) created and maintained by employing stored queries expressing the restrictions in the XACML policies.

The first step required by both query rewriting solutions consists in translating XACML policies to SQL conditions (Sections 3.3 and 6.1). This means flattening the hierarchical structure of XACML to a SQL formula. To achieve that, we employ a tool which extracts the XACML conditions that are necessary to permit the access of the user to the requested resource. Then, we translate these constraints to SQL conditions. At this point, for the second solution mentioned above, we just add the SQL conditions to the user's query. In our solution instead, we include these conditions separately in stored queries which are then used to generate database objects (i.e. virtual and materialized views). Then, the user's query will be rewritten in such a way that it interrogates these objects including the security restrictions. This approach avoids expensive operations like joins, Cartesian products and the use of SQL *DISTINCT* operator, which are needed in the other query rewriting approach.
Our solution, as well as the other alternatives to the OASIS's one, intends to improve the efficiency of the access control system without demanding much time and resources to maintain and manage the system and the database. For this reason, we analyze and evaluate the most suitable database object to store the results of the stored queries considering both performance and ease of maintenance (Chapter 5). We demonstrate with experiments executed on two different datasets and employing various queries (Chapter 7) that our solution performs better than the current XACML implementation especially when complex security conditions are involved and when the restrictions have to be applied on many tuples (in order to check their compliance with the access control policies).

1.3 Thesis Outline

Chapter 2 introduces the problem and presents query rewriting as a solution. In Chapter 3 we introduce basic concepts of access control including the main access control models, we describe the essential components of XACML policy structure and language, and finally, we describe how we transform XACML policies to SQL conditions. Chapter 4 presents related work in the context of access control, stored queries employment and query rewriting techniques. Available alternatives to the model proposed by the OASIS and our new solution are described and compared in Chapter 5. There, we also discuss possible design and implementation alternatives which can be adopted for our solution. In Chapter 6, we illustrate the structure of the system we implemented in order to execute experiments using the different approaches, described in the previous chapters, to enforce XACML policies. Then, in Chapter 7, after introducing the data used for the experiments, we show and evaluate the results achieved testing the different approaches. Finally, in Chapter 8, we draw conclusions on the alternatives we examined to enforce access control restrictions defined using XACML and we present possible future works and extensions.

Chapter 2

Problem Statement

2.1 Problem Definition

Access control is typically used to regulate the actions that an entity can perform on a single or a group of resources belonging to a database. In an ordinary access control setting, a user requests access to a given object and the access control mechanism determines whether the access should be granted based on the provided policies.
This approach is well accepted and widely used in organizational environments, however, as implemented in the model suggested by the OASIS to enforce XACML policies, it has drawbacks when applied to big data. To illustrate these drawbacks, consider a system comprising the following components:

- an application A that a user or another application uses to connect to a database and to execute queries;
- a generic DBMS (database management system) D;
- a generic access control system ACS that regulates the operations that a subject can perform on data and resources.

In the current setting, the security conditions specified using XACML are enforced after the query has been executed and the data retrieved from a database. Hence, some data which resulted from the database query is discarded afterward as a consequence of a security restriction.

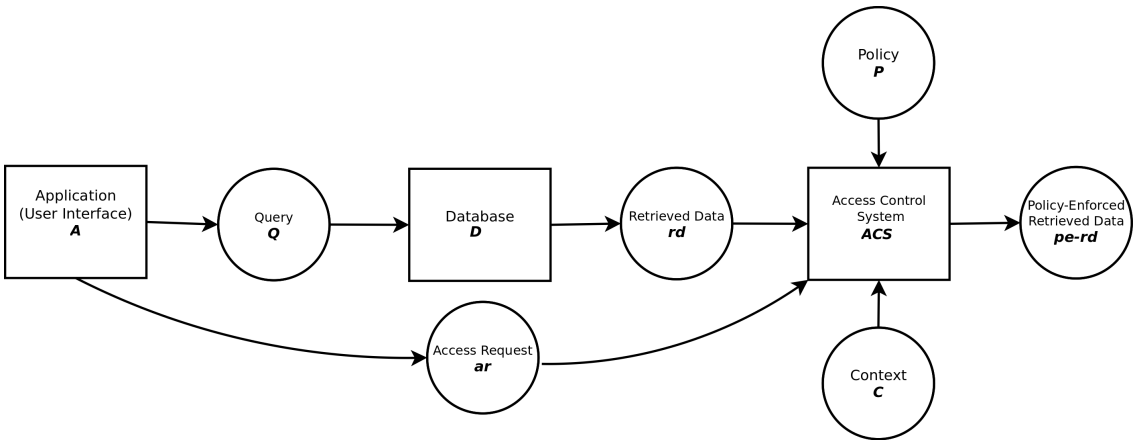

Figure 2.1: Current procedure for policy enforcement on query execution.

In the diagram above, the inputs and outputs to the components listed before are displayed. These elements are:

- a query Q that a user (or an application) wants to perform on a database;
- an access request ar which is generated by a component of A based on Q and on the user's identity;
- the resulting data rd derived from the execution of a query Q on the database D;
- one (or more) security policy P;
- a context C which corresponds to a set of parameters related to the environment, the action, the subject or the resources that are relevant to the policy P;
- the resulting data after applying access control restrictions, *pe-rd* (policy-enforced resulting data).

The main working principles underlying the current access decision-making process are shown in Figure 2.1. It is possible to notice from the diagram that, once the user launches a query, the database D, given the query Q, outputs the data rd. The records rd, together with the context C, the policy P and the access request ar are the inputs to the ACS which provides the policy-enforced data *pe-rd*.
Considering D and ACS as functions, we can summarize this system with the following compact formulation:

$$D(Q) = rd \tag{2.1}$$

$$ACS(rd, ar, P, C) = \textit{pe-rd} \tag{2.2}$$

Equations 2.1 and 2.2 can be combined leading to the compressed expression:

$$ACS(D(Q), ar, P, C) = \textit{pe-rd} \tag{2.3}$$

Given this procedure, it is important to highlight that the system analyzes the data two times along the process: firstly, in the database D to retrieve the records that satisfy the query Q. Then, at a later stage, the ACS examines the query results rd to apply the access control restrictions. These two operations make the current process inefficient especially if the database D and the records rd are composed of thousands or millions of entries (which is likely to happen in a real-world system). The second task is particularly expensive as the ACS checks the results one by one to verify that they comply with the given access control policies. On the contrary, rewriting the query Q including the access control restrictions would optimize the mechanism by limiting the system to only checking and retrieving the data in a single step in order to get the final result.

2.2 Our Solution

We propose a solution that rewrites queries in such a way that the access control requirements, defined by XACML policies, are included, and the query results are retrieved accordingly. As a consequence, we optimize the mechanism by limiting the system to only one verification over the data to get the final result. The core of this idea is to substitute the existent process with the one presented in Figure 2.2.

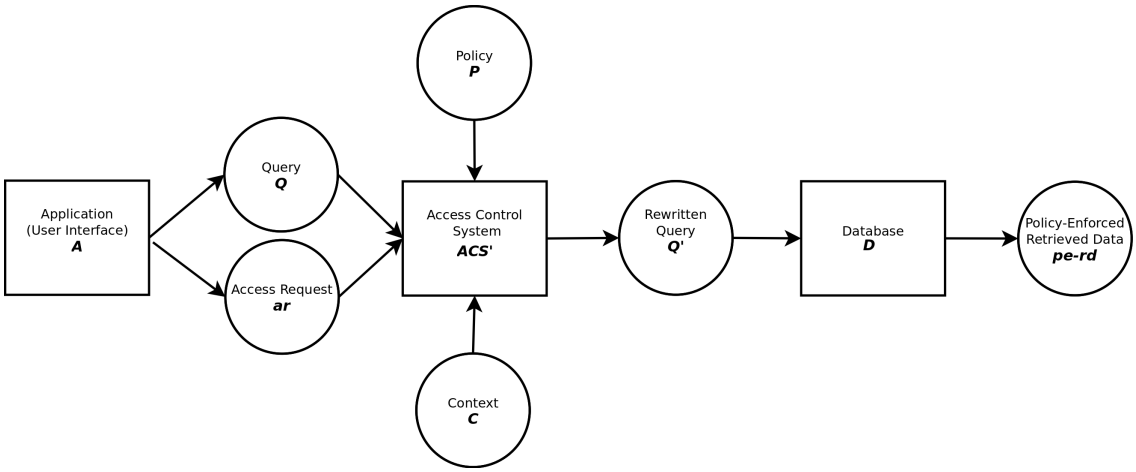

Figure 2.2: Proposed procedure for policy enforcement on query execution.

In this case, a modified version of access control system, ACS', receives the user's query Q and the access request ar from the user's application, the policy P and the context C. Then, it generates a rewritten query Q' which is executed on the database D which produces the desired output *pe-rd*. That is, in compact notation:

$$ACS'(Q, ar, P, C) = Q' \tag{2.4}$$

$$D(Q') = \textit{pe-rd} \tag{2.5}$$

That can be combined into:

$$D(ACS'(Q, ar, P, C)) = \textit{pe-rd} \tag{2.6}$$

Where *pe-rd* is the same output resulting from the current process, therefore, considering equations 2.3 and 2.6:

$$ACS(D(Q), ar, P, C) = D(ACS'(Q, ar, P, C)) = \textit{pe-rd} \tag{2.7}$$

Specifically, this new approach requires the translation of XACML policies to SQL conditions, which must have the same overall meaning of the original regulations in XACML. This is necessary because the transformed requirements will then be included in the query that the user is executing and that involves data which is subject to access control. However, we propose a new approach which does not just append this conditions to the user's query. In particular, we include in the user's query some conditions which make use of database objects (e.g., tables, virtual views, materialized views, etc.) generated through the execution of stored queries. These stored queries contain the SQL conditions translated from XACML, hence they express the security restrictions described in the XACML policies. As a consequence, the system using our approach needs the most suitable database objects to store the results of the stored queries. The choices we propose are the optimal solutions in terms of performance and ease of maintenance and update of the data which is stored in these database objects.
In addition, the whole process of security enforcement must remain invisible to the user who should not notice any difference with the new approach (compared to the one proposed by the OASIS).

2.3 Motivating Example

Suppose there is a relational database storing sensitive data of a company. In particular, there exists a table *orders*, which is composed of more than a million entries.
Since the company does not want to grant the access to all the information in the table to everyone, it employs an access control system using XACML policies. One of these policies contains a restriction which states that an employee identified with the role of "Intern" in any functional area of the company (e.g., marketing area, shipping area, supply chain area, etc.) cannot view the

entries in *orders* with total price greater than 100000 euro.

In this situation, consider a user in the marketing area with intern role that wants to retrieve all the orders in the database with urgent order priority. In order to do that, the user launches the following query:

```
SELECT *
FROM orders
WHERE o_orderpriority='1-URGENT'
```

Applying the current process for access control (the one displayed in Figure 2.1) the system would perform the following steps:

- the query is executed and the results are retrieved from *orders* table. Those would correspond to all the rows in *orders* with *o_orderpriority* = *'1-URGENT'*.
- These results, together with the policy (in this simplified example, just a single restriction), the access request and some information from the context, are fed into the access control system. If all the attributes required to evaluate the policy are already known, the system proceeds with the policy enforcement. On the contrary, if some attributes are missing, the access control system interrogates the database or other components of the system to obtain what is required. For example, suppose that the application provides with the access request only the id (e.g., an identification number like '1238') of the user but not his role (i.e. Intern) which is required for the policy evaluation. In this case, the access control system would query the database to get the user's role.
- When all the needed information is known to the access control system, it enforces the security restrictions by checking one by one the tuples resulting from query execution, so as to see if they respect the policy. In this case, it checks the employee's role and the value of the field *o_totalprice*. Since the role of the employee is "Intern", if *o_totalprice* is greater or equal than 100000, the tuple is discarded, otherwise it is kept. At the end of this process, the employee will get the policy-enforced data.

As expressed in general before, the last step, especially in case the resulting tuples after query execution are numerous, is very inefficient. In fact, the access control system needs to singularly verify the entries to check if they respect the policy specifications.
On the contrary, employing a revised procedure for access control (as shown in Figure 2.2) the steps would be:

- the user launches the query but, this time it is not executed on the database. The query, as the access request, the context, and the policy, is an input to the access control system that modifies it to make it compliant with the policy.
- In this case, the access control system would produce the following "raw" rewritten query, adding the access control conditions defined in the policy:

```
SELECT *
FROM orders,employees
WHERE
o_orderpriority='1-URGENT' and
e_userid=1238 and
(not ((100000 <= o_totalprice) and (e_role = 'Intern')))
```

- This query is then executed on the database and the results are retrieved from *orders*. The resulting records are going to correspond to the results obtained with the previous procedure and system.

In this kind of system, the inefficient step where the policy is enforced on the results after query execution on the database is absent. This is possible because the security restrictions are added to the query, hence the results of this modified query are already the final one, complying with the access control policies.

Chapter 3

Preliminaries on Access control

This chapter provides definitions, context, and information that serve as background to understand the contents in the following chapters. Firstly, we recall the definition of access control and we briefly present the main access control models and techniques. Then, we focus on XACML language and the procedures involved in the enforcement of an XACML policy. At the end of the chapter, we describe how we transform XACML policies into SQL conditions.

3.1 Access Control

Access control, in the context of computer security, is the component of a system that regulates the actions that an entity can perform on a single or a group of resources. In particular, the entities which are part of an access control process are typically called *subjects*, while the resources are named *objects*. The main goal of access control is to provide confidentiality and integrity for some data. These properties are among the most relevant characteristics that a system (e.g., a database management system or an operating system) should possess in order to be considered secure. The first property states that an object is not accessible or disclosed to unauthorized subjects (e.g., users, processes or other entities). Integrity ensures that an object cannot be deleted or modified by unauthorized subjects. Additionally, access control also indirectly contributes to protecting the system against attacks on availability, hence it guarantees reliable access to authorized subjects. In fact, if the access control system is properly set up, it prevents attackers from the misuse of the available resources of the system.
In general, an action that a subject can perform on an object corresponds to an access right that is granted to the subject. The basic access rights are *read*, that allows to look at the content of an object, and *write*, which authorizes an object modification. Then, depending on the system on which the access control is enforced and on the resources to protect, there exist other access rights like *execute*, *select*, *insert*, *update*, *delete*, etc.
There are many models to provide access control, the most well-known are:

- DAC (Discretionary Access Control): the subject that owns the object defines what operations the other subjects can perform on the object.

- MAC (Mandatory Access Control): given the subjects security clearance and the objects security classification, there are access control rules that regulate all the possible access scenarios.

- RBAC (Role-Based Access Control): every subject is assigned a role (that can correspond to the job title, tasks for users or other characteristics) and every role has different access rights on the objects. The subjects inherit the access rights of the role they are assigned, hence they can operate on the objects as stated for their role.

- ABAC (Attribute-Based Access Control): access control is regulated by rules that are defined on different attributes that can be related to subjects, objects, actions, and environment. In this way, it is possible to express flexible and fine-grained policy to regulate the operations of the subjects on the objects. Moreover, it is also important to highlight the fact that through ABAC, it is possible to define rules belonging to all the other models.

3.2 XACML

XACML, which stands for eXtensible Access Control Markup Language, is a language designed by the OASIS (The Organization for the Advancement of Structured Information Standards) to express access control policies, access requests, and responses. In particular, it is possible to express ABAC policies, but also rules attributable to other access control models like MAC, DAC and RBAC. The language is based on XML (eXtensible Markup Language) and is equipped with some default attributes and functions. The OASIS also defined a high-level structure and data-flow that should be the foundation of the access control system supporting XACML.

3.2.1 XACML Policy Structure and Language

In this work, we consider XACML version 3 (without Obligations) as detailed in [19]. The structure of an XACML policy is hierarchical, the main components (ordered from the leaves to the root of the hierarchy) are *Rules, Policies,* and *Policy Sets*. Each one of these elements may comprise a Target element that defines the applicability of the policy. Rule elements are the most elementary unit of a policy. They may contain, together with a Target, some additional conditions and they must contain an attribute, *Effect*, which defines the outcome of a Rule that can be either *Permit* or *Deny*.
The Policy is the minimal unit required to form a valid XACML policy and it is the most basic element evaluated in order to determine if a subject request is applicable. Each policy may contain multiple Rule and Target elements.
Two or more policies can be combined in a Policy Set. Both Policy and Policy Set elements have a mandatory attribute, *combining algorithm*, which is used to unify the evaluation of two or more Rules in a Policy element, otherwise to evaluate two or more Policies in a Policy Set element.
The Target is composed by one or more *AnyOf* elements which contain a disjunctive sequence of one or more *AllOf* elements. These, in turn, are formed by a conjunctive sequence of one or more *Match* elements that contain the actual condition on the attribute (of a subject, object, action or environment). Hence, the regulations and restrictions of access control are concretely defined in the conditions on the attributes in the Match elements.

The evaluation of a Target and its composing elements can result in a *match* (which should not be confused with the Match element seen before), *no match* or *indeterminate*. A Match element results in a match if the condition is true, in a no match if the condition is false or in an indeterminate if it is not possible to evaluate the condition (e.g., the attribute is missing in the request).

Condition	Match element
True	match
False	no match
Otherwise	indeterminate

Table 3.1: Match element evaluation

The AllOf is evaluated as match if all the Match elements resulted as a match, if there is one no match it is evaluated as no match, or it is evaluated as indeterminate in the other cases.

Match element(s)	AllOf Element
All match	match
At least one no match	no match
Otherwise	indeterminate

Table 3.2: AllOf element evaluation

The AnyOf is a match if at least one AllOf is a match, if every AllOf are no match then also the AnyOf is a no match, while in the remaining cases the AnyOf is indeterminate.

AllOf element(s)	AnyOf Element
At least one match	match
All no match	no match
Otherwise	indeterminate

Table 3.3: AnyOf element evaluation

Finally, if the Target is empty, it results in a match as it will authorize the request in any case since there are no conditions. If all the AnyOf resulted in a match then also the Target is a match, if there is at least one no match, the Target is a no match, otherwise the Target is indeterminate.

AnyOf element(s)	Target Element
No AnyOf	match
All match	match
At least one no match	no match
Otherwise	indeterminate

Table 3.4: Target element evaluation

At this point, it is important to say that every attribute in the Match elements has a flag *MustBePresent* (if it is absent, it means that it is set to the default value, that is $false$). This additional characteristic governs whether the Match element returns indeterminate (if $MustBePresent = true$) or an empty bag (if $MustBePresent = false$) in the event that an attribute in the condition is absent in the request.
As we have already said, the Target can be part of another of the main components of XACML policies: the Rule, which in turn is part of a Policy (possibly contained in a Policy Set). The evaluation of these elements may result in *Permit*, *Deny*, *NotApplicable* or *Indeterminate*. Permit, as opposed to Deny, allows the subject to access the object specified in the request. While if the decision is NotApplicable or Indeterminate, then the behavior is undefined. In this work, we talk about a four-dimensional decision set. However, it is possible to extend the Indeterminate to *Indeterminate*$\{P\}$ (Indeterminate Permit), *Indeterminate*$\{D\}$ (Indeterminate Deny) and *Indeterminate*$\{PD\}$ (Indeterminate Permit-Deny) leading to a six-dimensional decision set. In particular, *Indeterminate*$\{P\}$ stands for Indeterminate Permit and it is used on a Rule which could have evaluated to Permit, but not Deny; similarly, *Indeterminate*$\{D\}$ is used on a rule which could have evaluated to Deny, but not Permit. *Indeterminate*$\{PD\}$ may result when a Policy or Policy Set could have evaluated to both Deny or Permit.
In order to evaluate a Rule, if the Target is a match or there is no Target, then it is necessary to check the additional conditions in that Rule. In case the conditions are satisfied, the Rule results in Permit or Deny as stated in the Effect attribute. If the conditions are false, the Rule value is *NotApplicable* while if the conditions are indeterminate, the Rule value is *Indeterminate*$\{P\}$ if the Effect of the Rule is Permit or *Indeterminate*$\{D\}$ if the Effect is Deny. It is not possible that a Rule ends in an *Indeterminate*$\{PD\}$. On the other hand, if the Target is a no match, regardless of the conditions outcome, the Rule is evaluated as NotApplicable. Finally, if the Target is inde-

terminate, again the conditions are irrelevant, and the Rule is *Indeterminate*$\{P\}$ if the Effect is Permit, or *Indeterminate*$\{D\}$ if the Effect is Deny.

Contained Target element	Condition(s)	Result
match or no Target	True	Effect (Permit or Deny)
match or no Target	False	NotApplicable
match or no Target	indeterminate	Indeterminate{P} if the Effect is Permit, or Indeterminate{D} if the Effect is Deny
no match	irrelevant	NotApplicable
indeterminate	irrelevant	Indeterminate{P} if the Effect is Permit, or Indeterminate{D} if the Effect is Deny

Table 3.5: Rule element evaluation

The remaining structure of XACML policies is composed by Policy elements which contain one or more Rules, and PolicySet element(s) which contain one or more Policy elements. After the evaluation of every Rule singularly, the overall evaluation of Policy and PolicySet elements is done by the use of a Combining Algorithm. In XACML v3, the OASIS defines the following combining algorithms which work as described in Appendix A:

- Deny-overrides
- Ordered-deny-overrides
- Permit-overrides
- Ordered-permit-overrides
- Deny-unless-permit
- Permit-unless-deny
- First-applicable
- Only-one-applicable (only for Policy elements)

At the end, each evaluation leads to a Permit, Deny, NotApplicable, Indeterminate{P}, Indeterminate{D} or Indeterminate{PD} in case of a six-dimensional decision set. It leads only to Permit, Deny, NotApplicable or Indeterminate in case of a four-dimensional decision set.

3.2.2 XACML Policy Enforcement

Given an access request composed of subject, object, action to perform and environment-related values, systems that enforce XACML policies operate on the results obtained after running the subject's query by processing the retrieved data, the user request and the context as described in Figure 3.1.

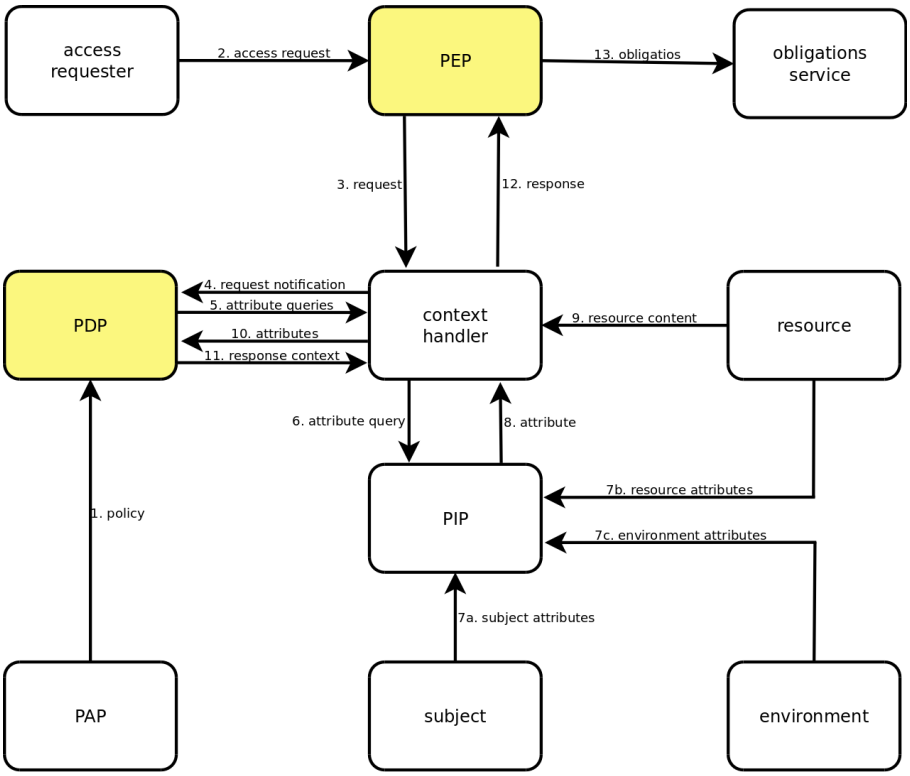

Figure 3.1: XACML data-flow diagram

The OASIS proposes the use of the data flow model in the figure above where:

1. policies and policy sets are generated in the policy administration point (PAP) and are accessible by the policy decision point (PDP).
2. The access request of a user is sent to the policy enforcement point (PEP).
3. The request, in its original format, is forwarded to the context handler together with other optional attributes regarding the subject, resource, action, and environment.
4. The context handler translates the request in XACML format and sends it to the PDP.
5. The PDP, while evaluating possible policies that apply to the request, may require some additional subject, resource, action or environment attributes to the context handler.
6. The context handler interrogates the policy information point (PIP) to get the required attributes.
7. The PIP gathers the attributes from the subject, resource or environment.
8. The context handler receives the requested attributes from the PIP.
9. The resource might be included in the context of the requested attributes by the context handler.
10. The information collected is sent to the PDP which evaluates the policy with regards to the whole request.
11. An authorization decision is returned and included in the response context which is delivered to the context handler.

12. The response is translated again into the native format of the request and sent to the PEP.
13. The PEP performs any existing obligation.

Finally, if the access is granted, the PEP allows the access to the resource otherwise it denies it.

The tasks performed by the PDP and PEP (which are highlighted in Figure 3.1) are accomplished by the database in the query rewriting approaches. In fact, the decision of granting or denying the access and the enforcement of the policy happens in the database. In particular, the security restrictions are enforced by including them in the subject's query, which is executed within the database.

3.3 Flattening XACML Policies

In order to transform XACML policies to SQL conditions, one step consists in flattening XACML policies.
As described previously, XACML policies have a hierarchical structure. Consequently, as query rewriting approaches for policy enforcement require the addition of these constraints in SQL queries, we need to flatten XACML policies to be able to include them.
To complete this task, we start by identifying and extracting from the XACML policies those conditions that are necessary to permit the access of the subject to the requested object. This set of conditions, together with the logic connectives that combine them, determine the permit decision space of the XACML policies, which identifies the requirements that need to be satisfied to be granted the access to one or more specific objects. One way to do this consists in transforming the policies to a first-order logic formula, as done in [28] where the formulas are used to analyze the policies and demonstrate whether they possess specific security properties.

Chapter 4

Related Work

Already in 1974, at the very first days of relational databases, a solution to combine queries and access control was proposed. The authors introduced a high-level query language, QUEL [25], that could also include access control constraints. This solution allows the enforcement of simple security conditions but it is not suitable for the current systems where finer-grained policies and more complex policies are required.
Another alternative is represented by Hippocratic Databases [1] which aim to incorporate data protection within the database. The concept is to use privacy metadata tables: a privacy-policies table and a privacy-authorization table. The first captures the privacy policy by storing whom the protected information can be given out to and how long the information is stored. While the second captures the access controls that support the privacy policy by storing the set of subjects who can access this protected information. Then, before each query is executed, these tables are checked so as to find possible correspondences between the user, attributes, and purpose (specified and sent to the database together with the query) for the access. While the query is executed, only the entries satisfying the matches between user, attributes and purpose for the access found previously are retrieved and presented as results. However, only design and implementation proposals exist because this solution presents some challenges deriving from its theoretical principles which prevented the realization of this type of system.

Our approach for security policy enforcement involves the use of a query rewriting technique. This methodology is widely adopted in the context of database systems for many purposes and applications. One example is the use of query rewriting for query optimization. In [13], it is described how query rewriting using a set of materialized views yields to a more efficient query execution plan. This solution was previously analyzed in [7], where the focus is on the optimization of select-project-join queries. The mechanism of query rewriting is also employed for data integration [31, 5, 9]. This allows the transformation of a query formulated over a specific schema to a query over various sources. Other examples of employments of query rewriting are in the context of semistructured data interrogation, where query rewriting using views is a fundamental query processing and optimization tool for semistructured queries [20]; sponsored search, where query rewriting includes elements in the original queries which allow the retrieval of other relevant ads [2] and distributed complex event processing, to detect complex patterns in high-rate event streams (e.g., all the Visa transactions in the United States) [23].

More recently, query rewriting started to be used in the context of access control. Oracle proposes the use of a proprietary system named "Virtual Private Database" using query rewriting to provide access control [30]. Other employments of query rewriting apply to specific domains: in [6] it is used to enforce access control over data streams, while in [12] it is used for EPC information services. Another solution is proposed in [22]. In particular, the authors suggest the use of relational views, called "Authorization views" and query rewriting, to enforce access control policies and make the constraints transparent to the user that runs the query (which they named

"Truman Model" in the paper). Specifically, the authorization views are employed to check the validity of the user's original query. This approach turned out to be too expensive to create and maintain as stated also in [16], where the authors focus their attention on access control for XML database. In this case, three solutions are proposed: merge access control policies and queries, apply access control rules on the results of the query or pre-prune parts of the query that violate the access control rules before query execution. The last option, called QFilter, is particularly related to our work as it presents a way to pre-process the user's query. Given a query Q, access control rules ACR, and an (optional) schema S, it returns a modified query $Q^{'}$ as output:

$$Q^{'} := \mathbf{QFilter}(Q, ACR, S)$$

In the process of security enforcement, the ACR is distinguished in ACR^{+} and ACR^{-}. The first identifies the rules which grant the access, while the second those which deny the access. QFilter has three types of operations: (1) Accept: If answers of Q are contained by that of ACR^{+} (i.e., Q asks for answers granted by ACR^{+}) and disjoint from that of ACR^{-} (i.e., Q does not ask for answers blocked by ACR^{-}), then QFilter accepts the query as it is: $Q^{'} = Q$; (2) Deny: If answers of Q are disjoint from that of ACR^{+} (i.e., no answers to Q are granted by ACR^{+}) or contained by that of ACR^{-} (i.e., all answers to Q are blocked by ACR^{-}), then QFilter rejects the query outright: $Q^{'} = \{\}$; (3) Rewrite: if only partial answer is granted by ACR^{+} or partial answer is blocked by ACR^{-}, QFilter rewrites Q into the ACR-obeying output query $Q^{'}$. Also in [18] the target are XML databases. In this case views, together with a query rewriting mechanisms are employed to enforce security restrictions over XML data. Solutions proposed in [16, 18] are applicable to XML database while ours is designed for the more widespread relational databases. Moreover, [16] support an ad-hoc access control model which corresponds to [10] extended with RBAC. This fact makes this solution only applicable to specific systems and scenarios.
Finally, in [3], a middleware architecture between database and access control system is designed to enforce access control. This component performs policy transformations and query rewriting for policies written in STAPL language [17] to optimize the access control process for SQL queries executed on large data sets. In a later paper [4], the same authors describe in details how it is possible to transform a (hierarchical) XACML policy to a (flat) condition in a form that is suitable to be included in a SQL query in order to apply some security restrictions. The use of STAPL is an interesting alternative since this recently created language allows the re-usability of policy patterns and modules as described in [11]. Nevertheless, the diffused and established knowledge of XACML still makes it a more desirable choice over the newly originated STAPL. Additionally, this solution has some limitations: the transformation is possible only for three XACML combining algorithms (*permit overrides*, *deny overrides* and *first applicable*) and in case during the evaluation process an *indeterminate* is obtained, it has to be handled as an error at the application level, so it is not considered as a possible evaluation result like in XACML.

Another peculiarity of our work is the use of stored queries to generate, maintain and manage database objects (i.e. virtual views or materialized views) which are interrogated by the rewritten subject's query to enforce access control regulations. Stored queries have been used in [21] to support the use of intensional attributes for efficiently storing and querying ontologies and in [24] to model data, metadata and their association. The main concepts of these two papers are described more in details in Chapter 5 where we discuss their relations with our work. A further work related to the topic of stored queries is [29]. In particular, a meta-querying system for databases containing stored query (defined in XML) is described in this paper. In this instance, the idea is to still use a relational database but extending standard SQL with specific meta-querying features. More specifically, XML variables and XSLT (the standard XML transformation language) function calls can be included in SQL queries to allow the interrogation of the XML queries stored in the database.

The solution we propose in this work makes use of query rewriting and stored queries to enforce security constraints. It aims to optimize the process expressed in the data-flow diagram in

Section 3.2.2 by improving the efficiency of the current XACML implementation suggested by the OASIS. This means having a system which supports fine-grained ABAC policies (defined in the standardized and diffused XACML) that can be easily managed and maintained. As a matter of fact, in order to manage the access control, the administrator will only have to operate on the XACML policies without being concerned, to some extent, about modifying the database structure. This is possible because the access control component in our solution works together with the DBMS but it is separated from it.

Chapter 5

Query Rewriting for XACML Policy Enforcement

In this chapter, we introduce definitions and restrictions essential to outline alternative approaches to enforce XACML policies. Then, we illustrate a straightforward way to include the access control condition in the user's initial query. After that, we describe our solution and we present different design alternatives and implementation options that can be adopted. One of these options might be more suitable than the others depending on the needs and objectives of the system enforcing security policies.

5.1 Notation

Given an XACML policy, it is possible to identify and extract restrictions, consisting of atomic conditions (on subject, object, action or environment), which regulate the access to a specific resource. This was done in [28], where first-order logic formulas are derived from XACML policies.

An atomic condition is made of a predicate and two terms. In order to give a definition of "term", we have to introduce the concept of entity and attribute in the context of an XACML policy.
In access control and in an XACML policy, the word entity can be used to refer to one of the following:

- the subject who wants to access the system,
- the object (or resource) that the subject is trying to access,
- the action that the subject wants to perform on the object or
- the environment which is relevant to the access control evaluation.

An attribute in XACML is a characteristic of an entity (i.e. subject, object, environment or action) that may be referenced in a predicate or target [19]. We extend this definition considering attributes also the characteristics of attributes themselves. It is important to note that at the beginning of a "chain of attributes" (which are attributes of attributes) there is always an entity.

Based on the above concepts, which we consider in the context of atomic conditions extracted from an XACML policy, we define the notion of term.
A term is an attribute (of an entity or an attribute of another attribute) or a constant representing an attribute value.

Definition 1. *A term t is defined as follows:*

$$t := constant \mid entity.attribute \mid entity.attribute.attribute \mid \ldots$$

An atomic condition c is a logic expression that can be either true or false. The expression contains a predicate that indicates a relation (e.g., equality, inequality, inclusion, etc.) between two terms t_1 and t_2.

Definition 2. *An atomic condition c is defined as follows:*

$$c := t_1 = t_2 \mid t_1 > t_2 \mid t_1 < t_2 \mid t_1 \in t_2$$

The atomic conditions extracted from the policy can then be combined into a formula f with the use of logical connectives.

Definition 3. *A formula f is defined as follow:*

$$f := c \mid \neg f \mid f_1 \wedge f_2 \mid f_1 \vee f_2$$

where c denotes an atomic condition as defined previously.

5.2 Considerations on Query Rewriting Approaches

To optimize the current procedure to enforce XACML policies, we propose to add the conditions extracted from XACML policies to the query executed by the subject who requests access to a resource which is under the regulation of an access control policy. By applying this method, there are some specific cases where the final results are different than the ones achieved with the implementation proposed by the OASIS. In particular, the results can be different if the initial subject's query contains a *LIMIT*, *ROWNUM* or *TOP* operator. Consider, for example, that *LIMIT 5* is specified in a query. Then, in a system enforcing access control using the current XACML implementation, the query would be executed on the database and it would retrieve five results. After that, if any of these is not compliant with the policy, it would be discarded and the subject that launched the query would obtain less than the desired five results. On the contrary, if the access control conditions are included in the subject's initial query the security restrictions are enforced before the *LIMIT 5*, which is the final operation executed in the query. As a consequence, the subject will always obtain the number of results specified with the *LIMIT* operator. A similar situation occurs when using *ROWNUM* or *TOP*.
Another important aspect that should be taken into account is that the conditions or formula, expressing the security restrictions of an XACML policy, is appended to the WHERE clause of the subject's query. Hence, possible complications may arise when the subject's query contains nested queries. In fact, the security conditions should be enforced also to the inner queries, requiring to add them more than once in the subject's query. In this way, the performance would be negatively affected but this would avoid possible leakage of information that may derive from the sub-queries on which no access control would be performed. In that case, a malicious user could adopt a nested query as a covert channel to gather information about data for which he is not granted the access. For example, if a user is able to generate a query containing a nested query that interrogates a "top secret" table for which he has no access right, he might be able to gather some information from it. The following simplified queries represent three different examples where nested query can be used to collect unauthorized information if the security conditions are appended only to the WHERE clause of the main query:

```
SELECT A.attribute1, (SELECT FIRST(U.attribute1)
                      FROM UnathorizedTable U
                      WHERE U.attribute3 = 1)
FROM AuthorizedTable A
WHERE A.attribute2 = True AND f
```

where f is a formula representing an XACML policy.

```
SELECT A.attribute1
FROM AuthorizedTable A,   (SELECT U.attribute1
                           FROM UnathorizedTable U
                           WHERE U.attribute3 = 1) B
WHERE A.attribute2 = B.attribute1 AND f
```

where f is a formula representing an XACML policy.

```
SELECT A.attribute1
FROM AuthorizedTable A
WHERE A.attribute2 =  (SELECT MAX(U.attribute1)
                       FROM UnathorizedTable U
                       WHERE U.attribute3 = 1) AND f
```

where f is a formula representing an XACML policy.

In the first example, the leaked information will be in the second column for each result. In the second, there is going to be a result for each element in the *AuthorizedTable* where the value of *attribute2* matches the value of an *attribute1* in the *UnauthorizedTable*. Finally, in the third query, a result is retrieved if an attribute in the *AuthorizedTable* matches the result of a function (in this case MAX) applied to the results of a query on the *UnauthorizedTable*.
The problem is that the conditions deriving from the security policies are only going to be added to the main query. As previously mentioned, a solution would be to apply the security conditions also to the nested query but that would impact the performance significantly as the access control restrictions would be applied more than once, one time in the main query and one time in every nested query. A specific case that may be accepted is when the object evaluated in the nested query is the same of the main query. In these circumstances, the security policies applied would lead to the same evaluation, hence the inclusion of the security restrictions in the main query would be enough to correctly enforce access control.
In addition, in case multiple queries results are combined with the use of SQL set operators (i.e. UNION, INTERSECT, EXCEPT), the access control policies must be enforced on every single query.

5.3 Reference Example

In order to better understand the concepts introduced previously and to present the query rewriting approaches, next we illustrate an example based on one of the two database schemas and on policies, which will be also part of our experiments described in Chapter 6 and 7.
Consider the following tables which are part of the schema generated through the tool used to populate the database of TPC-H benchmark [26]:

$customer(c\_custkey, c\_name, c\_address, c\_nationkey, c\_phone, c\_acctbal, c\_mktsegment, c\_comment, resource\_c\_table)$

$employees(e\_userID, e\_firstname, e\_lastname, e\_area, e\_role, e\_nationkey, resource\_e\_table)$

$conditions(cn\_id, cn\_name, cn\_definition, resource\_cn\_table)$

With the tables *employees* and *conditions* and the attributes *resource_c_table*, *resource_e_table* and *resource_cn_table* added to the original TPC-H schema in order to execute our experiments supporting access control.

The table *customer* is subject to access control regulations. In particular, there is an XACML policy containing the following restriction: an employee of a company, belonging to the *Marketing* functional area and with the role of *Analyst*, is allowed to see the entries of a table *customer* (listing the clients of the company) in the DB of the firm only if he is from a nation which is in the same continent of the customer's nation.

This restriction is composed of three atomic conditions c_i:
c_1 : the employee has to be in the marketing area;
c_2 : the employee needs to have the role of analyst;
c_3 : the nation of the employee must be in the same continent of the customer's nation.

The above requirements can be expressed in compact notation as:
$c_1 : employee.area =$ 'Marketing'
$c_2 : employee.role =$ 'Analyst'
$c_3 : employee.nation.continent = customer.nation.continent$
where *employee* corresponds to the subject and *customer* to the object of the access request.
In the atomic condition c_1 (and similarly for c_2) *employee.area* and *'Marketing'* are the terms. In particular, *area* is an attribute of the subject entity *employee*, while *'Marketing'* is a constant. Differently, the atomic condition c_3 contains two "attribute of attribute" terms: *employee.nation.continent* and *customer.nation.continent*. In fact, the attribute *continent* is related to the attribute *nation*, which is in turn regarding the subject entity *employee* for the left-hand side of the equation, and the object entity *customer* for the right side.

We can summarize the policy with the following formula:

$$f : c_1 \wedge c_2 \wedge c_3$$

5.4 Trivial Approach

The main principle at the basis of some available alternatives [3, 24], which do not employ XACML to define security restrictions, is to include these conditions to the subject's initial query. A straightforward way to incorporate the conditions expressed in an XACML policy into the query executed by the subject is to:

- transform the XACML formula f with atomic conditions $c_1, \ldots, c_n$ into an equivalent SQL formula f^{sql} with atomic conditions $c_1^{sql}, \ldots, c_m^{sql}$,
- append f^{sql} to the user's query (in the WHERE clause and add the necessary tables in the FROM clause).

Note that c_i^{sql} are conditions expressed using the SQL syntax for both the terms $t_i^{1,sql}$, $t_i^{2,sql}$ and the predicate expressing their relation. So an atomic SQL condition containing an equality predicate would be of the form: $c_i^{sql} : t_i^{1,sql} = t_i^{2,sql}$, similarly to the conditions extracted from the XACML policy.
With this procedure, the equalities/inequalities expressed in the WHERE clause in SQL could be more than the ones in XACML because of the necessity of operations like joins or because the conditions in XACML are defined on attributes of attribute (which result in more equalities/inequalities in SQL). Hence, it is possible that

$$c_1 \Rightarrow c_1^{sql}, \ldots, c_k^{sql}$$

So it might be the case that there is not a one-to-one correspondence between the condition extracted from the XACML policy and the same constraints expressed in SQL. However, the overall meaning of the conditions $c_1^{sql}, \ldots, c_m^{sql}$ must be the same of the one expressed with the conditions $c_1, \ldots, c_n$. In other words, the meaning and logic of the formula f composed of $c_1, \ldots, c_n$ should be the same of f^{sql} consisting of $c_1^{sql}, \ldots, c_m^{sql}$.
Another aspect to highlight is that, in order to avoid errors in the execution of the query, it is necessary to include in the FROM clause all the tables required by the attributes composing the security conditions added in the WHERE clause. In this way, if there is no condition on any relational key-pair of different tables in the FROM clause, it occurs the necessity of Cartesian products between tables. This is the most significant drawback of this solution as the Cartesian product is a very expensive operation which affects substantially the efficiency of the system.

5.4.1 Example (Continued)

Consider the scenario introduced in Section 5.3. Assume there is an employee named Bob Smith who wants to see the list of all the clients of his company with the account balance greater or equal than 5000. Supposing that at query time Bob is logged in, so his identity (e_userID) is known to the system, the query would be the following:

```
Subject query:
SELECT c.*
FROM customer c, employees e
WHERE e_userID = 1234 AND c_acctbal >=5000
```

In the query above, it is important to highlight that one Cartesian product is performed between *customer* and *employees* tables. In this case, the operation is not very expensive because, thanks to the condition $e\_userID = 1234$, the Cartesian product is not on the whole *employees* table but just on one of its rows. However, when other conditions are added, the risk of having more expensive Cartesian products increases.

The conditions defined in XACML and presented before can be expressed in SQL as:
$c_1 \Rightarrow c_1^{sql} : e\_area = \text{'Marketing'}$
$c_2 \Rightarrow c_2^{sql} : e\_role = \text{'Analyst'}$
$c_3 \Rightarrow c_3^{sql} : e\_nationkey = n1.n\_nationkey$
$c_4^{sql} : c\_nationkey = n2.n\_nationkey$
$c_5^{sql} : n1.n\_regionkey = n2.n\_regionkey$

As a consequence, after appending these conditions with the necessary connectives, the query becomes:

```
Subject query:
SELECT c.*
FROM customer c, employees e, nation n1, nation n2
WHERE e_userID = 1234 AND c_acctbal >=5000 AND
e_area = 'Marketing' AND e_role = 'Analyst' AND
e_nationkey=n1.n_nationkey AND c_nationkey=n2.n_nationkey AND n1.n_regionkey=n2.
    n_regionkey
```

From the example, it is clear that this approach requires the addition of some tables in the FROM clause (i.e. *nation n1* and *nation n2*) necessary to include the security conditions.

5.5 Employing Stored Queries

An alternative approach consists in defining a query for each condition $c_1, \ldots, c_n$ along with a label to identify the query. Then, the label and the query corresponding to a condition are stored separately in a relational table. Hence, for the atomic conditions $c_1, \ldots, c_n$ there are going to be n labels $L_1, \ldots, L_n$ and n queries $Q_1, \ldots, Q_n$. Each couple L_i - Q_i, corresponding to a condition extracted from an XACML policy, would be a row stored in a relational table. The stored queries, differently from the subjects' queries, can also contain nested queries and can involve other stored queries.

This idea of storing queries was proposed in [24] where queries are considered as data value in order to model data, metadata and their association (also for security purposes). In our solution, we will employ a mechanism similar to the one proposed in [24]. More specifically, the queries correspond to a new user-defined atomic type *Q-type* which extends the common SQL type system. The authors of the paper also introduce a new predicate (i.e. conditional expression) to evaluate the stored queries at runtime. Such a predicate is represented using the operator $\doteq$. Together with this new symbol, the notation presented in [24] includes the use of square brackets in order to identify the terms in a condition involving a stored query. In particular, the terms can be

attributes resulting from the execution of a stored query or constants. For example, if there is a table T with a column Col storing a stored query generating an attribute $Attr$ which should be equals to one, the SQL condition would be:

$$T.col[Attr] \doteq [1]$$

With a procedure detailed in [24], it is possible to rewrite the subject's query containing conditions with this notation to a more complex query with the same overall meaning but that can be executed on a database system. At the end, the final query is similar to one achievable with the Trivial Approach presented before because the conditions are added to the WHERE clause and more tables are included to the FROM clause.
In our solution, we use the stored query differently. In fact, stored queries are employed to create and manage database structures (like views or materialized views) which contain elements of the database that comply a condition c_i. This data is then used to extend the user's query in a more efficient and elegant way compared to the solution proposed previously. The final query will include conditions on this generated database structure in the WHERE clause but it will not need the addition of any further tables in the FROM clause of the original subject's query. As a result, this approach reduces the number of joins between tables and other operations required with the straightforward technique presented previously; as a consequence, the query execution (and policy enforcement) is faster and more efficient.

5.5.1 Example (Continued)

Suppose again that a subject named Bob Smith wants to execute a query to see the list of all the clients of his company with the account balance greater or equal than 5000.
When employing stored queries, the query is

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000
```

It is important to notice that, in this case, there is no need to insert the table *employees* in the FROM clause and the related condition in the WHERE clause. In fact, those are substituted by just a parametrized condition in the rewritten query, as we will see later. This avoids the Cartesian product which was performed in the Trivial Approach.
Regarding the security requirements, the atomic conditions c_1 and c_2 are combined into a single formula and are labeled *isMarketingAnalyst* (in the next Section, 5.5.2, we will explain the reason of this merger) and are specified by the following query:

```
Q1 (named isMarketingAnalyst):
SELECT e_userid as userid
FROM employees
WHERE e_area='Marketing' and e_role='Analyst'
```

The three atomic SQL conditions composing condition c_3 are again combined and they are specified in the following query named *sameregioncustomer*:

```
Q2 (named sameregioncustomer):
SELECT e_userid as userid, c_custkey as custkey
FROM employees, customer, nation n1, nation n2
WHERE employees.e_nationkey = n1.n_nationkey AND customer.c_nationkey = n2.
    n_nationkey AND n1.n_regionkey = n2.n_regionkey
```

Below an example of how this information would be stored is displayed:

cn_id integer	cn_name character varying(255)	cn_definition character varying(255)	resource_cn_table character varying(255)
4	isMarketingAnalyst	select e userid from employees where e area='Marketing' and e role='Analyst'	conditions
7	sameregioncustomer	SELECT e userid as userid, c custkey as custkey	conditions

Figure 5.1: Names and queries stored in a relational table *conditions*.

The stored queries *Q1* and *Q2* are then used to generate a view (or table) with the name corresponding to their label. Then, using the stored information in this database instances, the final query becomes:

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000 AND
EXISTS (select * from isMarketingAnalyst where userid=?) and EXISTS (select * from
    sameRegionCustomer where userid=? and c_custkey=custkey)
```

After instantiating the value of *userid* for the specific subject who is requesting the access (in this example Bob's id is *1234*), the query becomes:

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000 AND
EXISTS (select * from isMarketingAnalyst where userid=1234) and EXISTS (select *
    from sameRegionCustomer where userid=1234 and c_custkey=custkey)
```

In the above query, it is checked that the user with id equals to 1234 (who is the subject issuing the query) is included in *isMarketingAnalyst* which contains all the analyst in the marketing area. In addition, condition c_3 is checked by verifying that there is a tuple in *sameregioncustomer* where the user id is 1234 and the identifier of the nation correspond to the one of the entry of *customer* table which is being evaluated (before retrieval).
By doing so, there is no need to add further tables in the FROM clause and possible expensive join operations are avoided.

5.5.2 Considerations on Stored Queries

The adoption of stored queries as shown in the example above leads to many possibilities for the database administrator. Depending on the queries that the users are allowed to execute, on the policies and the conditions they include, it is possible to define the stored query in many different ways. We can take conditions c_1 : *employee.area* = *'Marketing'* and c_2 : *employee.role* = *'Analyst'* as an example to present some of the choices that can be made in order to improve the efficiency of the system. Given these atomic conditions, we have seen previously an example of how they can be combined in a single stored query which is later used to enforce these restrictions. However there are other options. The first one is to have two stored queries, one for each atomic condition, defined as:

```
Q3 (named inMarketingArea):
SELECT e_userid as userid
FROM employees
WHERE e_area='Marketing'
```

```
Q4 (named isAnalyst):
SELECT e_userid as userid
FROM employees
WHERE e_role='Analyst'
```

Having two stored queries implies the creation of an additional view, materialized view or table to store the data. The results are two of these database instances named *inMarketingArea* and

isAnalyst for *Q3* and *Q4* respectively. This means that the final query would be more complex and with worse performance because of the additional condition:

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000 AND
EXISTS (select * from inMarketingArea where userid=1234) AND
EXISTS (select * from isAnalyst where userid=1234) AND EXISTS (select * from
    sameRegionCustomer where userid=1234 and c_custkey=custkey)
```

The advantage of this choice is that, having a single stored query for every atomic condition, allows more flexibility. In fact, by combining with logical connectives the results of the single stored queries, it is possible to create more complex formulas without modifying the stored queries themselves. For example, supposing that the role *Analyst* exists not only in the marketing area, then it would make sense to keep a separate stored query for it, so that it can be used in combination with the store query of another area specified in the security restriction. On the contrary, if the role exists only in one area (e.g., *Marketing*) than it is better to consider the performance and merge the atomic conditions in a single stored query as done previously.

Another alternative would be to have again two queries but defined as follows:

```
Q3 (named inMarketingArea):
SELECT *
FROM employees
WHERE e_area='Marketing'
```

```
Q4 (named isMarketingAnalyst):
SELECT e_userid as userid
FROM inMarketingArea
WHERE e_role='Analyst'
```

Query *Q3* is similar to before, the only difference is in the SELECT clause where now the projection is on all the attributes of *employees* table. The main change is in *Q4* which uses in the FROM clause *inMarketingArea* instead of the table *employees*. With this option, two database structures are created: *inMarketingArea* from *Q3* and *isMarketingAnalyst* from *Q4*. In this case, *isMarketingAnalyst* has to be generated first, as *Q4* interrogates it. This solution allows the best performance because the final query, shown below, would have just a condition on *isMarketingAnalyst* (for the specific case of the subject's query we are using in this example).

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000 AND
EXISTS (select * from isMarketingAnalyst where userid=1234) AND EXISTS (select *
    from sameRegionCustomer where userid=1234 and c_custkey=custkey)
```

Additionally, the condition *inMarketingArea* is more generic and can be reused/combined to generate other, more flexible security formulas. On the other hand, the fact of having the result of a stored query in another stored query may cause some management issues. If *inMarketingArea* is deleted or modified, the changes will affect the result of *Q4*. Hence, every time *Q3* is modified, the database administrator must check if the result of *Q4* is still the desired one.

These are just three alternative possibilities that apply to this particular example. However, depending on the structure of the database, the conditions in the security policies and the stored queries derived from these restrictions, there might be other options. The choice on which one to use should be based on the specific database, queries, and policies.

5.5.3 Design Alternatives

In [21], three alternatives are presented in order to realize a system that makes use of stored queries: the materialized, the lazy and the indexed approach. These approaches were originally employed to support the use of intensional attributes in the context of efficiently storing and querying ontologies in a database.
One of the objectives in [21] is to identify all the intensional attributes related to a desired individual using stored queries. Each individual has some attribute-value pairs which characterize it (e.g., the individual *Canada* has the following pairs: *type=Country*, *group=non-EU*, *funding=70M* and *population=32M*). The intensional attributes are defined using a triple $\langle q\_d, m, q\_r \rangle$ where q_d is a stored query which defines the domain, m is a name (or label) of the attribute, and q_r is another stored query which specifies the range. If the attribute-value pairs of an individual correspond or satisfy the conditions expressed in q_d, the concept expressed with the intensional attribute is valid for that individual. For example, suppose that every EU country is governed by a regulation named AG/345. To represent it in a database, for each country belonging to EU (suppose there is a attribute-value pair *group=EU* for the individuals to express it) the alternatives are: add another attribute-value pair *governedBy=AG/345* for each individual or create an intensional attribute $\langle q\_d, m, q\_r \rangle$ employing stored queries where q_d is a query identifying nations in EU and q_r represents the regulation AG/345. The second alternative offers space saving and requires less maintenance as there is no need to repeat the insertion of the new attribute for all the individuals. Moreover, if new countries become part of EU or leave EU, there is no need for modifications.
Each one of the approaches described in [21] makes this functionality invisible to the user who does not know if any of the stored queries is executed or which one of them is evaluated. However, these alternatives are fundamentally different when considering overhead at query execution, database maintenance, and storage utilization.

Materialized Approach The principle of this option consists in materializing the stored queries q_d and q_r in order to be able to treat the resulting attributes and tuples in the same way of the rest of the data in the database. This is done in [21] by evaluating q_d and q_r and adding all the resulting triplets $\langle d, m, r \rangle$ to the knowledge base (where d and r are single elements resulting from the evaluation of the queries). Hence, with this approach, finding the intensional attributes of an individual corresponds to finding its regular attributes. Differently, if a new individual is added in the system, all the queries defining the intensional attributes are executed in order to find possible related attributes.

Lazy Approach The lazy approach is the opposite of the materialized approach. The idea is to just keep the definition of the stored queries q_d and q_r and avoid any kind of materialization. As a consequence, storage space is saved and there is no need to take any further action when an individual is added to the database. The disadvantage is that, when a subject interrogates the database and the results of the stored queries are needed, those queries must be executed at runtime leading to a high cost in terms of performance and execution time of the subject's query.

Indexed Approach This alternative was designed in [21] as a custom solution that could avoid the disadvantages deriving from the opposite solutions already presented. In fact, the indexed approach aims to mitigate the downsides of the materialized and lazy approach. The main idea is to create a special index structure that allows evaluating the stored queries in a smart way. This is done employing four relational tables: *DTable*, *MTable*, *ETable* and *ITable*.
DTable consists of three column: the first containing q_d, the second m and the third q_r.
MTable has an entry for each stored query (both q_d and q_r), the first column contains the identifier of the stored query, the second the number of conditions (equalities and inequalities) that the stored query contains and the third a counter (set to 0 every time before a user query is executed).
ETable stores the equality conditions of the stored queries. They are saved in a three column table where the first specifies the attribute name, the second the attribute value and the third the

identifier of the stored query which contains that condition. If a condition is in more than one stored query, there will be an entry for each stored query containing it.
ITable is similar to ETable, the only difference is that it stores the inequality conditions of the stored queries. It is made of two columns, the first one contains the whole condition (e.g., $population < 20M$) while the second has the identifier of the stored query that contains the inequality.
When a user wants to retrieve all the intensional attributes related to an individual, the conditions stored in ETable and ITable are checked against the attribute-value pairs characterizing that individual. For every condition that matches an attribute-value pair of the individual, the counter in MTable of the stored query containing that condition is increased by one. At the end of this process, the counters in MTable that reach the number stored in the adjacent column (which specify the total number of condition in a stored query) identify the stored queries which are satisfied. Then, for the stored queries which are fulfilled, using DTable, it is possible to identify the intensional attributes that involve the verified stored queries as domain or range.

Considerations In this work, we consider only the materialized and the lazy approach since the indexed approach would not be beneficial in the scenario of security policy enforcement. In fact, this option was designed to avoid the execution of stored queries which were not relevant for the correct execution of the initial user query. In particular, in the scenario presented in [21], the advantage of the indexed approach consists in evaluating (through the use of special index structures) only the stored queries which contain one or more specific conditions that are in the initial user query.
Since this improvement cannot happen in our context because all the conditions (i.e. all the stored queries) have to be evaluated, we excluded the indexed approach. The remaining two approaches represent two opposite alternatives to employ stored queries for access control. Depending on the requirements and purposes of the application that uses data which is subject to security regulations, one option might be more suitable than the other. In particular, when the performances and response time are fundamental for the application, the materialized approach is the optimal choice. On the contrary, when efficiency is secondary while the focus of the application is on having up-to-date information, the best solution is the lazy approach. Other benefits of this alternative are detectable in terms of space saving, in avoiding any additional modification to the database in favour of a less complex database structure and in the possibility to immediately use the stored queries.
Another option is to materialize only some stored queries, while leaving the rest as proposed in the lazy approach. This solution would be optimal in case the data involved in the materialized stored queries is not modified frequently or it is not important for the security policy if the data is not totally up-to-date when a user tries the access. In this way, the evaluation of these stored queries is faster. On the other hand, the rest of the stored queries, which employ the lazy approach, would be less efficient when evaluated but they would always use updated information at the time the user attempts to access the system.

5.5.4 Implementation Alternatives

Before, we have described different approaches to design a system that involves stored queries and, in particular, we have seen how it could be done in a scenario like the one in [21]. In this section, we present some concrete alternatives to implement such a system using the materialized or lazy approach for security policy enforcement. These options utilize query rewriting, virtual views, materialized views or relational tables. The first two are related to the lazy approach while the lasts are the actualization of the materialized approach.

Query Rewriting This implementation is presented in [24]. The stored queries definitions are saved in dedicated column(s). Then, starting from a query with the notation introduced in Section 5.5, the idea is to rewrite it in such a way that the database can interpret it correctly.

The procedure described in [24] produces a final query which strongly resembles the one shown in the Trivial Approach (Section 5.4). More specifically, the result of query rewriting includes the tables and the conditions of the stored queries in the FROM and WHERE clauses of the initial user's query. Depending on the number and complexity of the stored queries, the final query might become extremely long, complicated and inefficient.

Virtual Views In general, when talking about a "view" in the context of DBMS, people refer to virtual views. These are the result set of a query on the data available in the database. They can be simply generated in any relational DBMS with the SQL command " *CREATE VIEW view_name AS* " followed by a normal SQL query (which result will be the data populating the view). The only stored information in the database is the definition of the view, hence the data composing it is not stored in the disks, saving storage. However, virtual views perform worse than materialized views because the queries that define them are executed each time they are accessed. The difference in performance is influenced by the query which retrieves the data of the view. If the query contains expensive operations (e.g., many joins involving big tables or on non-indexed columns) the generation of the view will be affected negatively.

Materialized Views Materialized views are generated similarly to virtual views. In fact, they are also based on the result of a query. The basic SQL command to run in the DBMS is " *CREATE MATERIALIZED VIEW view_name AS* " followed by a query. However, materialized views do not have any overhead when their data is interrogated because it is written on the disks like all the rest of the information in the database. This aspect has the disadvantage of occupying more storage and it requires that the stored query is executed with a specific frequency (defined by the database administrator) in order to update the data contained in the materialized view. Therefore, the information in the materialized views are not always up-to-date.

Relational Tables As done also in [21], one option is to generate an entry in the database for each result of the stored queries. In this case, there is the need of a table that stores all the resulting tuples of all the stored queries, if the projection in the stored queries is done on the same attributes. In particular, a column with labels is needed to identify the entries of the different originating queries. Otherwise, a new component would be needed to automate the generation and management of the different tables resulting from the stored queries execution. This option, as described in [21], requires that every time a new element is introduced (or removed) in the database, all the stored queries that populate the dedicated relational table have to be executed to check if the new element is involved or is a result of one of them. Moreover, if there is the necessity to separate the output of the stored queries in multiple tables, the overhead to automate the whole procedure would be even more inefficient.

Considerations The implementations of the materialization approach, either using relational tables or materialized views, are more efficient at execution time but requires periodic maintenance, require more storage and increment the complexity of the structure of the database. Among the two implementation solutions, the choice has to be made considering the requirements of the application. However, the automated update with a fixed periodicity of the materialized views seems to be the more reasonable solution compared to the update every time a new element is introduced in the database to refresh the relational tables. Moreover, as mentioned in the dedicated paragraph, the choice for the relational tables might require an additional component to generate and manage the tables resulting from the execution of the stored queries. As a consequence, by adopting this choice, the system could become more complex and less efficient than one employing materialized views.
On the contrary, the implementations deriving from the lazy approach (i.e. query rewriting and virtual views) are less efficient at execution time but have updated information, save storage and do not require modification when elements are introduced or removed from the database. Virtual

views are preferable over query rewriting as the final query might become very complex and inefficient using the latter.

Chapter 6

System Design

This chapter illustrates the system designed for the experiments. It includes a description of every component comprising the system and all their inputs and outputs. In particular, it is described the role of these components in the distinct procedures of the different policy enforcement alternatives.

The system we designed was devised to test the following access control enforcement mechanisms:

- *OASIS' (simplified OASIS approach)*, a simplified (and more efficient) version of the original implementation suggested by the OASIS, which checks the compliance of the retrieved results of the subject's query by querying again the database;
- *APF (A-Poseriori Filter)*, an "a-posteriori filter" which applies the security conditions by checking the resulting tuples of the subject's query, which are stored in a Java *List*, within the component that enforces the access control (hence, without connecting to the database anymore);
- *QR-TA (Query Rewriting - Trivial Approach)*, a query rewriting approach that consists in just adding the SQL security restrictions to the subject's query, which we named Trivial Approach in Chapter 5;
- *QR-SQ (Query Rewriting employing Stored Queries)*, our new query rewriting approach which makes use of database objects (e.g., tables, virtual views, materialized views, etc.) created and maintained by employing stored queries expressing the restrictions in the XACML policies.

In particular, we tested the two best implementation alternatives for our query rewriting approach: virtual (*QR-SQvv*) and materialized views (*QR-SQmv*).
The approaches listed above requires two components which are not directly involved during the procedure of XACML policy enforcement: one to transform XACML policies into SQL formulas and one to generate database objects (i.e. virtual or materialized views) based on stored queries results, used for the last approach mentioned. These parts of the system are described in the following two sections.

6.1 XACML to SQL Transformation

The first phase which is common for the query rewriting approaches, and also to *APF* and *OASIS'* approaches we implemented (as we will describe later), consists in the transformation of an XACML policy to SQL conditions. These conditions, combined with the necessary logic connectives, form the SQL formula which is later employed to enforce the access control regulations.

In our implementation, the following steps are performed to achieve the translation of XACML policies into SQL formulas:

- the XACML policy is fed into *X2S*, a tool presented in [27] which transforms XACML policies into SMT instances (i.e. formulas in first-order logic). This tool outputs the formulas identifying the permit, deny and indeterminate decision spaces.
- Then, a Java component we designed extracts just the SMT instance characterizing the permit decision space from the *X2S* tool output. This formula in first-order logic is parsed by our component. In particular, XACML attributes are extracted from their XML definition (e.g., e_role is extracted from $urn : oasis : names : tc : xacml : 1.0 : subject : e\_role$), XACML predicates are translated to the equivalent SQL predicates and the prefix notation, used in SMT instances, is converted to infix.
- Lastly, the attributes in the formula, which are named as in the XACML policy, are mapped to the corresponding attributes in the database, when necessary (i.e. when the names are different). In particular, in case of an "attribute of an attribute" (as described in Chapter 5), the mapping of a condition will generate more than one, final conditions.

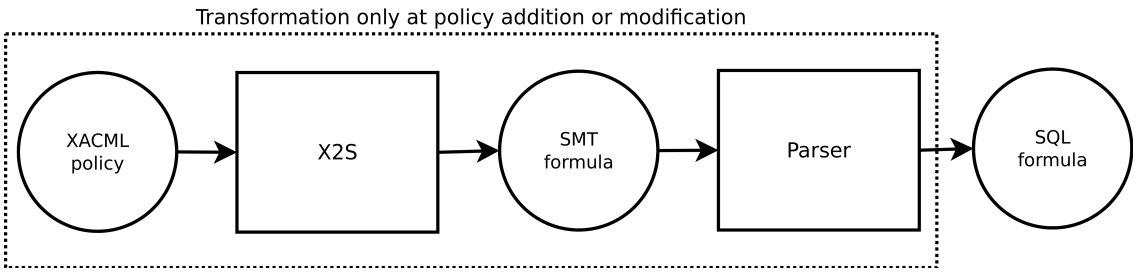

Figure 6.1: Transformation of an XACML policy to a SQL formula

This procedure, which is illustrated in Figure 6.1, must be executed by the system only when a new XACML policy is added or when an existing one is modified. In this way, the system always have up-to-date SQL formulas to include in the subject's query.
The parser we created generates the SQL formulas (from formulas in first-order logic) which will be used in *QR-TA* and *APF* implementations. The SQL formulas employed in *OASIS'* and in *QR-SQ* implementations are generated manually. In addition, in the system we tested, we mapped the "attributes of attribute" manually.

Example (Continued) Consider again the example described in Chapter 5. The part of the XACML policy defining those conditions is the following:

```
<Policy PolicyId="P1"
xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-
    overrides">
    <Target>
        <AnyOf>
          <AllOf>
            <Match
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
            >'customer'</AttributeValue>
            <AttributeDesignator
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:1.0:attribute-category:resource"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource_c_table"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
```

```
            </Match>
          </AllOf>
        </AnyOf>
    </Target>

  <Rule Effect="Permit" RuleId="R1">
    <Target>
      <AnyOf>
        <AllOf>
          <Match
          MatchId="urn:oasis:names:tc:xacml:1.0:function:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
          >'Marketing'</AttributeValue>
          <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:e_area"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </Match>
          <Match
          MatchId="urn:oasis:names:tc:xacml:1.0:function:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
          >'Analyst'</AttributeValue>
          <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:e_role"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </Match>

          <Match
          MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
          <AttributeDesignator MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:c_regionkey"
          DataType="http://www.w3.org/2001/XMLSchema#integer"/>
          <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:e_regionkey"
          DataType="http://www.w3.org/2001/XMLSchema#integer"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  </Rule>

</Policy>
```

The relevant part of the output of the *X2S* tool for this XACML policy is:

```
Spaces for policyP1
fin_rule_P1
C_p :(and (select |urn:oasis:names:tc:xacml:1.0:resource:resource_c_table|
              |'customer'|)
     (select |urn:oasis:names:tc:xacml:1.0:subject:e_area| |'Marketing'|)
     (select |urn:oasis:names:tc:xacml:1.0:subject:e_role| |'Analyst'|)
     (integer-equal |urn:oasis:names:tc:xacml:1.0:resource:c_regionkey|
                    |urn:oasis:names:tc:xacml:1.0:subject:e_regionkey|))
C_d :false
C_in :false
```

where *C_p* identify the permit decision space that we will transform into a SQL formula through our parser. In particular, the output of the parser after the mapping is:

```
((resource_c_table = 'customer') and ((e_area = 'Marketing') and (e_role = 'Analyst
    ') and (e_nationkey=n1.n_nationkey and c_nationkey=n2.n_nationkey and n1.
    n_regionkey=n2.n_regionkey)))
```

The above condition is used for the Trivial Approach and for the A-Posteriori Filter. Instead, the SQL formula employed in the simplified OASIS implementation and in the Query Rewriting Using Stored Queries implementation is:

```
(resource_c_table = 'customer' and (exists (select * from ismarketinganalyst where
    userid=?) and exists (select * from sameregioncustomer where userid=? and
    c_custkey=custkey)))
```

which, instantiated for the user Bob with id equals to *1234*, becomes:

```
(resource_c_table = 'customer' and (exists (select * from ismarketinganalyst where
    userid=1234) and exists (select * from sameregioncustomer where userid=1234 and
     c_custkey=custkey)))
```

6.2 Views Generation and Management Employing Stored Queries

In order to use *QR-SQvv* and *QR-SQmv* approaches and *OASIS'* approach (as we will explain later in this chapter), the system needs the existence of database views (virtual or materialized) that store the results of the stored queries. We recall that the stored queries reside in a dedicated table and represent one or more conditions defined in XACML policies.
As represented in Figure 6.2, a component (named *Stored Queries Manager* in the diagram) queries the database in order to obtain the names and definitions of all the existing stored queries. The database provides the Stored Queries Manager with the information needed to create SQL statements in order to generate in the database a view for each stored query. These statements are sent to the database which takes care of the creation of the specified views.

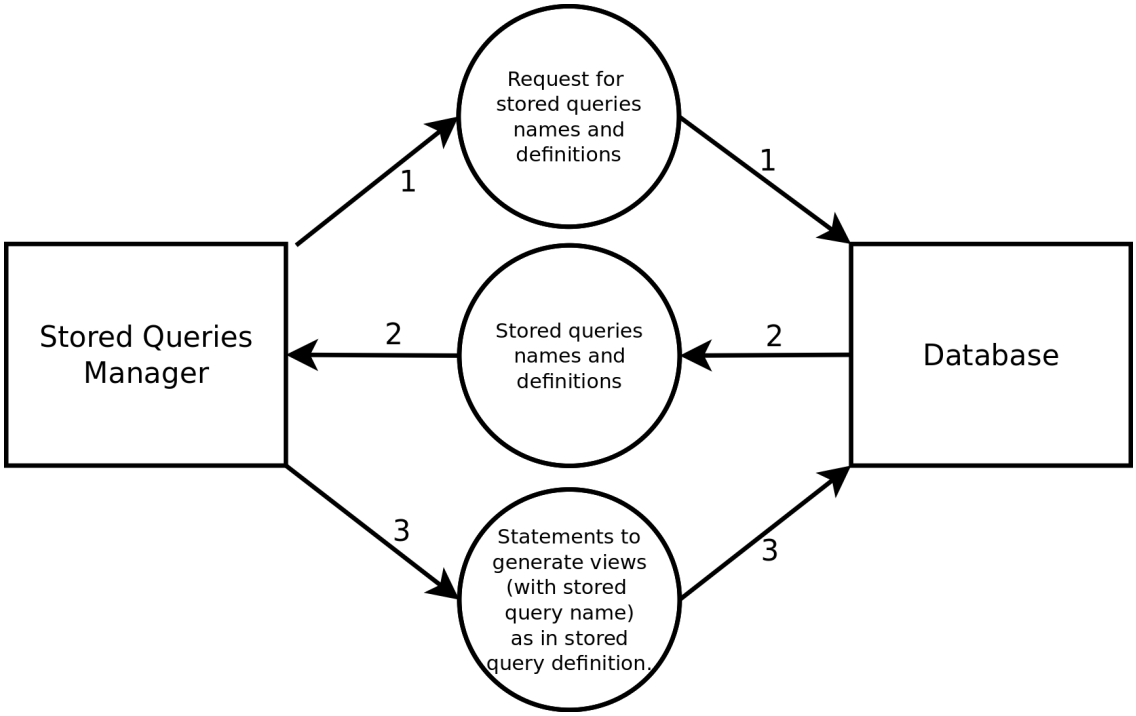

Figure 6.2: Procedure for the generation and management of views from stored queries

Similarly to the XACML to SQL transformation presented previously, also this process needs to be executed every time a policy is modified or a new policy is added to the existing ones, since the stored queries would be affected by these alterations.

6.3 Implementation of Security Enforcement Approaches

All the security enforcement approaches and the related functionalities are implemented in a Java component which resides between the application used by the subject to launch the query and the database. For the rest of this work, we will refer to this component as *middleware*.

6.3.1 Simplified OASIS

For our experiments, we implemented a simplified, more efficient version of the suggested system design by the OASIS. In this way, we can compare the performances of the query rewriting approaches with an enhanced version of the current XACML implementation.
The data-flow of this implementation is shown in Figure 6.3. The middleware receives the subject's query as input and it outputs a statement to create a temporary materialized view storing the results of the subject's query. Then, each tuple stored in the temporary view is checked in order to see if it is compliant with the security restrictions. This is done by executing the following query:

```
SELECT temp.*
FROM (select * from temporayView limit 1 offset i ) as temp
WHERE f
```

with i an integer identifying the iteration over the tuples of $temporaryView$ and f the SQL formula with the access control conditions which apply to the specific object(s).

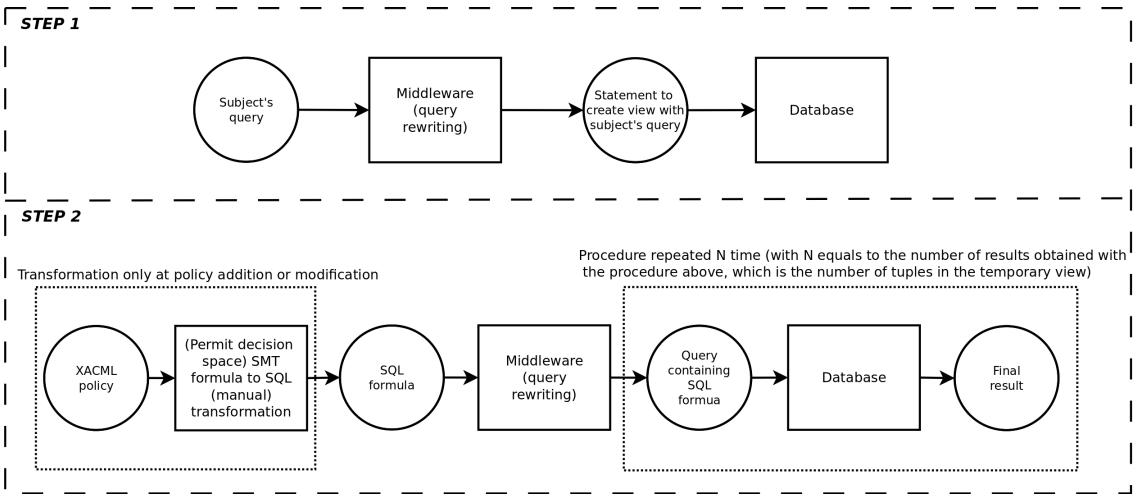

Figure 6.3: Procedure of policy enforcement for the Simplified OASIS implementation

In *OASIS'* approach, the tasks executed by the PDP and PEP in the original process (described in Chapter 3) are performed within the database, which executes the queries containing the security restrictions in the security enforcement step. Compared to the original design by the OASIS (described in Section 3.2.2), this implementation is more efficient because, even if the iteration over the results of the initial subject's query still exists, there is no need to translate any access request or policy from XACML and the PIP function of retrieving missing attribute can be performed together with the security enforcement step. In fact, as the middleware is already querying the database, it can include the request of missing attribute to the security enforcement step by modifying the query.
In order to obtain the best performance from this approach, we used a materialized view, instead of a virtual view, to store the values resulting from the execution of the initial subject's query. Additionally, the security formulas employed to check the compliance of the results with the security

regulations are those which we will also use for the *QR-SQmv* approach employing materialized views. These formulas are the ones which allow the best performance.

Example (Continued) For the example presented in Chapter 5, the statement to create the materialized view to temporarily store the results of the original subject's query is:

```
create materialized view temporayView as
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000
```

which is iterated in order to check the compliance of each entry through the query below

```
SELECT temp.*
FROM (select * from temporayView limit 1 offset i ) as temp
WHERE EXISTS (select * from m_isMarketingAnalyst where userid=?) and EXISTS (select
     * from m_sameRegionCustomer where userid=? and c_custkey=custkey)
```

Note that the temporary view is dropped after the above query is executed. In this way, a new temporary view can be created for the next subject's query.

6.3.2 A-Posteriori Filter

This approach was created with the idea of applying a filter to the results of the subject's original query, within the middleware, based on the security restrictions. In particular, the regulations are applied without using any other connection to the database after the execution of the subject's query. In fact, as illustrated in Figure 6.4, once the results of the subject's query are retrieved, together with some additional attributes (i.e. attribute in the security conditions but not in the query results), unlike what happens with the simplified OASIS implementation, the database is not accessed anymore. This is possible because the retrieved data is stored in a list of hashmap values by the middleware where the hashmap keys are the attribute names and the hashmap values are the attribute values. Then, using a *for loop*, the list is iterated and the elements are kept only if they comply with the restrictions specified in the security formulas (which are adapted from SQL to Java) that apply to the particular object(s).

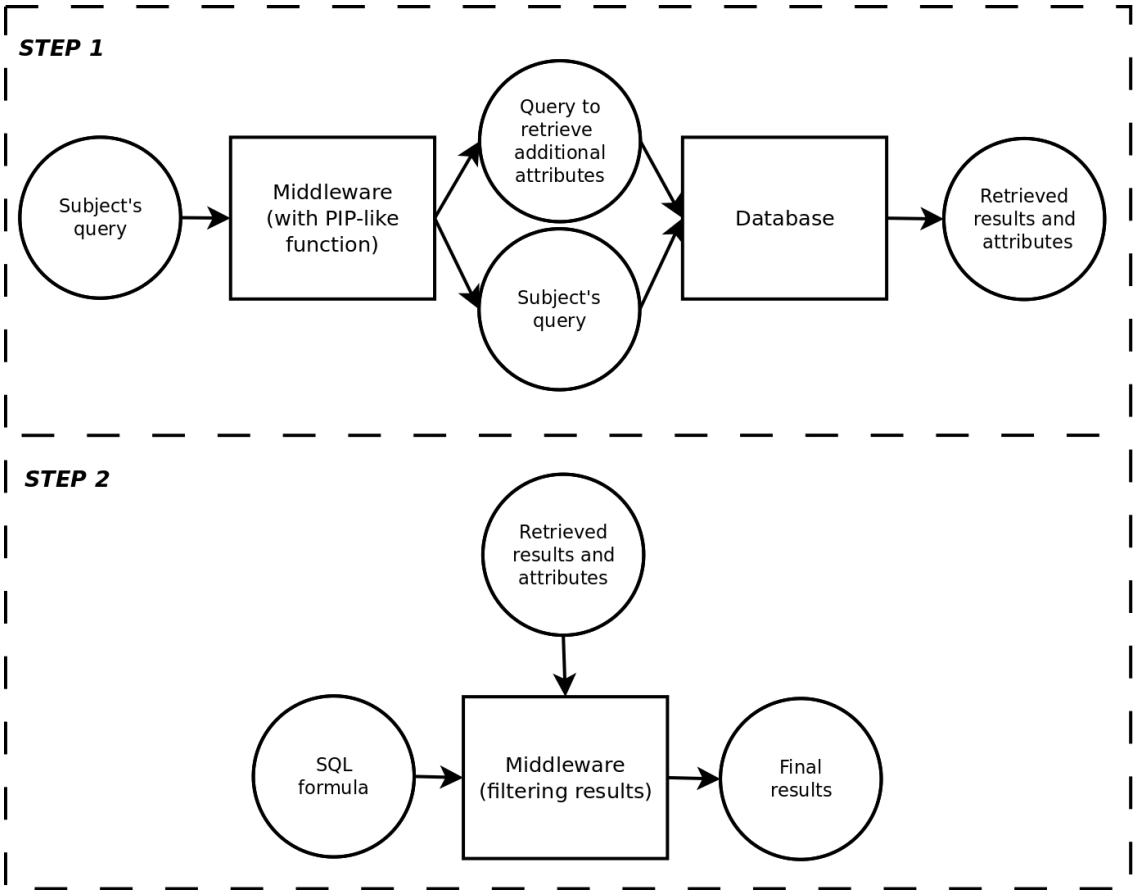

Figure 6.4: Procedure of policy enforcement for the A-Posteriori Filter implementation

The PDP and PEP functions are performed by the middleware which filters out the results which do not comply with the security conditions.
All missing attributes, except those related to or belonging to the object of access control, can be retrieved and appended to the results of the subject's query by the middleware which execute also a PIP-like function. However, if any attribute related to the object in the SQL security formula is missing in the results of the subject's query, it cannot be included at a later stage. In fact, once the query is executed, it is hardly possible or impossible (depending on the subject's query) to reconstruct the relationship between the missing attribute and the retrieved results. This is especially true if the subject's query is complex or the resulting tuples were subject to a SQL function (e.g., *SUM*, *AVG*, *MAX*, etc.).
Another drawback of this solution is that the retrieved results from the database are iterated two times to enforce access control: the first time to store them in the hashmap list and the second time to check their compliance with the security restrictions. This makes this approach inefficient when the results of the subject's query are numerous.

6.3.3 Query Rewriting - Trivial Approach

This implementation reflects the Trivial Approach described in Section 5.4.
In fact, the middleware gets as inputs the initial subject's query and the SQL security formulas obtained with the procedure described previously. Then, it appends the SQL security formulas, related to the object(s) which is being accessed, to the WHERE clause of the subject's query. This rewritten query is sent to the database which outputs the final results, compliant with the access control policies. This procedure is shown in Figure 6.5.

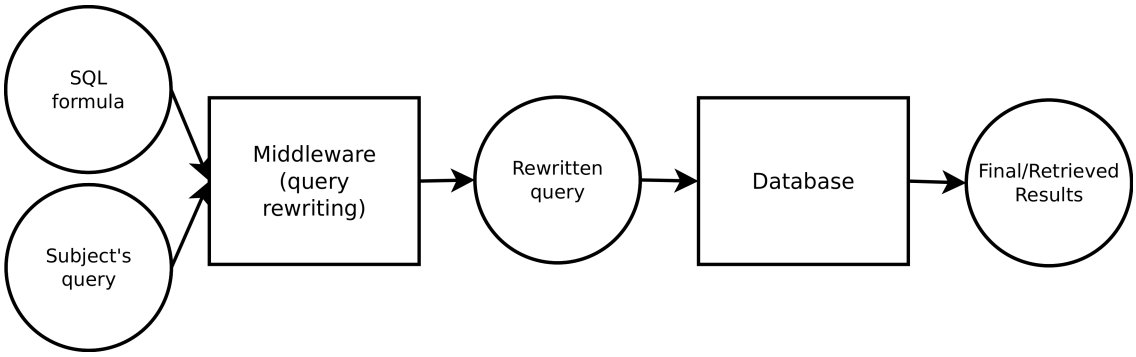

Figure 6.5: Procedure of policy enforcement for the Trivial Approach implementation

It is important to specify that for this approach, the subject's initial query is modified before appending the SQL security formula in the WHERE clause. In particular, all the tables required by the SQL security formula are added to the FROM clause and a condition that identifies the subject requesting the access is included in the WHERE clause.

Example (Continued) Consider the example introduced in Chapter 5, the following original query

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000
```

after the inclusion of the SQL security formula and the elements mentioned before becomes:

```
Subject query:
SELECT c.*
FROM customer c, employees e, nation n1, nation n2
WHERE e_userID = 1234 AND c_acctbal >=5000 AND
e_area = 'Marketing' AND e_role = 'Analyst' AND
e_nationkey=n1.n_nationkey AND c_nationkey=n2.n_nationkey AND n1.n_regionkey=n2.
    n_regionkey
```

The tasks done by the PDP and PEP in the current XACML implementation, in the Trivial Approach are performed by the database which does not retrieve the results which do not comply with the security conditions.

6.3.4 Query Rewriting Employing Stored Queries

Similarly to what happens in *QR-TA*, also for the *QR-SQ* implementations (*QR-SQvv* and *QR-SQmv*), the middleware is fed with the initial subject's query and the SQL security formulas. However, this time the SQL formulas are different from the ones used in the Trivial Approach since they interrogate the database objects generated by the stored queries, as described previously in this chapter. The procedure is shown in Figure 6.6.

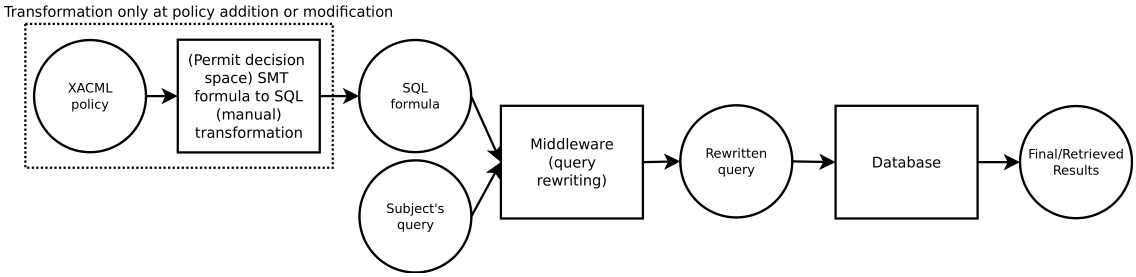

Figure 6.6: Procedure of policy enforcement for the Query Rewriting Employing Stored Queries implementation

We implemented two options diversified by the type of objects generated by the stored queries: one using virtual views *QR-SQvv* and one with materialized views *QR-SQmv*. The advantages and disadvantages of these different alternatives are described in Section 5.5.4. The SQL security formulas will be the same for these two options except for the name of the views they are interrogating. In fact, virtual and materialized views are differentiated with distinct names.

Example (Continued) Consider again the example of Chapter 5. The user initial query is

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000
```

The rewritten query using virtual views is the following

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000 AND
EXISTS (select * from isMarketingAnalyst where userid=?) and EXISTS (select * from
    sameRegionCustomer where userid=? and c_custkey=custkey)
```

Similarly, the query employing materialized views is

```
Subject query:
SELECT c.*
FROM customer c
WHERE c_acctbal >=5000 AND
EXISTS (select * from m_isMarketingAnalyst where userid=?) and EXISTS (select *
    from m_sameRegionCustomer where userid=? and c_custkey=custkey)
```

In both cases, what is done by the PDP and PEP in the current XACML implementation, in this solution is performed by the database which does not retrieve the results which do not comply with the security conditions.

Chapter 7

Experiments

In this chapter, we first introduce the data, queries, and policies we have used to test the different policy enforcement approaches. Then, the results achieved using the different datasets are evaluated and compared.
All the experiments were executed on a machine with 8GB of RAM, an Intel Core i7-3632QM CPU @ 2.20GHz × 4 and a 1TB 5400 RPM 8MB Cache hard drive. The data is stored in PostgreSQL 9.4.12 databases in a Linux Mint 17.3 64-bit environment.
All the queries are executed 10 times, one after the other, using the same JDBC connection to the database. This is done simulating the access of different subjects. Before the execution of the queries for a new user, a *VACUUM* operation is executed on the database. This whole procedure is repeated again 10 times, for a total of 100 executions per query, per user.

7.1 Datasets, Queries, and Policies

In order to test the security enforcement approaches described previously, we used two datasets, hence two different sets of subject's queries and XACML policies. In the experiments, we only assessed the scenario of a read request access performed through the use of a SELECT query. We did not consider INSERT, DELETE, UPDATE or other clauses related to other types of access requests.

The first dataset is the one provided for the TPC-H benchmark [26]. The tool supplied to generate the data allows the specification of the size of the dataset which will be automatically created and which will keep some predefined proportion among the dimensions of the composing table. For our experiments, we opted for a dataset with the overall size of 1GB.
Besides the tool to populate the database, an automatic query generator is provided. This tool outputs 22 queries containing some random parameter chosen during the creation. We tested all the queries for the different security enforcement approaches but we will report only the results for six of them which present some peculiarities and which are representative for all the 22 queries. In addition to these six queries, we created a custom query (identified with the number 23) to experiment the behavior of the approaches when dealing with a large number of tuples.
TPC-H benchmark is a decision support benchmark, hence it does not include any security policy. As a consequence, we designed six XACML policies based on the available data and considering that they must also make sense for the available queries. Additionally, we included in the database two tables: *conditions* and *employees*. The first is used to save the stored queries, while the second contains information regarding pretended subjects. The *employees* table is necessary to simulate the access of a user to the database by running one of the available queries. Moreover, in order to define policies both at a table and row level, we added to every table a column *resource_i_table* (where i is the letter used as the prefix for every attribute name in the table) which is employed to identify the object in the policy.

The second dataset is the one employed in the SAFAX project [15]. SAFAX is an XACML-based framework tailored to the development of extensible authorization services for distributed and collaborative systems. The dataset is made of 37 tables with an overall dimension of approximately 180MB.
The data was generated and designed to be part of an access control system. Hence, it contains all the necessary tables to simulate the access to its resources. Moreover, SAFAX project includes six XACML policies that we also used for our experiments.
For this dataset, we designed four queries, involving different tables and different policies, to analyze the performances of the security enforcement approaches.

7.2 Results on TPC-H

XACML to SQL Transformation As described in Section 6.1, the procedure to transform an XACML policy to a SQL formula comprises two steps: the translation of the policy to a formula in first-order logic, performed by the *X2S* tool [28], and the conversion from formula in first-order logic to SQL, executed by a Java parser we implemented. The execution times measured for these two steps are displayed in Tables 7.1 and Table 7.2.

XACMLtoSMT
1.49 seconds
1.515 seconds
1.53 seconds
1.559 seconds
1.546 seconds
1.543 seconds
1.533 seconds
1.492 seconds
1.522 seconds
1.52 seconds
Average = 1.525

Table 7.1: Execution times in seconds for the transformation of an XACML policy to a formula in first-order logic

SMTtoSQL
0.085 seconds
0.104 seconds
0.087 seconds
0.087 seconds
0.09 seconds
0.082 seconds
0.086 seconds
0.098 seconds
0.104 seconds
0.083 seconds
Average = 0.0906

Table 7.2: Execution times in seconds for the transformation of a formula in first-order logic to a SQL formula

Considering the averages of the two phases, the whole transformation takes 1.6156 seconds.

Views Generation and Management Employing Stored Queries The measurements in Table 7.3 refer to the time required to generate the views from stored queries as described in Section 6.2:

Virtual Views Generation	Materialized Views Generation
0.1 seconds	0.736 seconds
0.102 seconds	0.781 seconds
0.099 seconds	0.687 seconds
0.104 seconds	0.677 seconds
0.106 seconds	0.598 seconds
0.098 seconds	0.64 seconds
0.098 seconds	0.676 seconds
0.103 seconds	0.776 seconds
0.1 seconds	0.611 seconds
0.108 seconds	0.706 seconds
Average = 0.1018	**Average = 0.6888**

Table 7.3: Execution times in seconds for the generation of views from stored queries in TPC-H database

In particular, 7 views (virtual or materialized) are generated in order to enforce access control employing *QR-SQ* (and *OASIS'*) on TPC-H data.

Queries execution times We executed queries 3,6,9,11,16,18 (generated automatically with the tool provided for TPC-H benchmark) and our custom query 23, to test the different security enforcement approaches on TPC-H data. In particular, queries 3,9,18 and 23, because of the objects they access, involve more complex security conditions. While queries 16 and 23 have the largest outputs after the execution of the original query with 18716 and 19325 tuples respectively. Three different user ids were used to simulate the execution of these queries from different subjects, having different access privileges. In Figure 7.1, a matrix displays for which queries the subject's access is granted, considering the interrogated objects.

	Access to the objects involved in the query						
User	**3**	**6**	**9**	**11**	**16**	**18**	**23**
1234	yes*	yes	yes	yes	yes	yes*	yes*
1239	yes**	yes	no	no	no	yes**	yes
9876	no	no	no	no	no	no	no

*access only to rows where customer is in the same continent of the subject

**access only to rows where the total price of the order is lower than 100000

Figure 7.1: Matrix summarizing the access permissions of the subjects on the objects interrogated in the queries

In regard to the *APF*, since this approach requires that all the attributes in the security conditions are present in the tuples resulting from the execution of the original query, it was not always applicable. In particular, it could not be executed for queries 3,9,18 and 23. A similar problem arises for the *OASIS'* approach for queries 3 and 9. In this case, the execution of this approach generates an error because some attributes which appear in the SQL security formula are not in the results of the original query. In these instances, it is not feasible to retrieve them with a PIP-like component because it is impossible to trace back their values (given the results of the original subject's query).

In Figure 7.2, the area and role of the different users is shown. Additionally, the number of results, before and after security enforcement, for the different users is illustrated. Finally, in the last column, some relevant notes about the queries are presented.

Query	User (Area - Role)	Number of Results (Original Query)	Number of Results (After Policy Enforcement)	Important Notes
3	1234 (Marketing - Analyst)	10	10	Query containing LIMIT operator (in particular LIMIT 10).
	1239 (Shipping - Intern)		10	
	9876 (Legal Department - Lawyer)		0	
6	1234 (Marketing - Analyst)	1	1	
	1239 (Shipping - Intern)		1	
	9876 (Legal Department - Lawyer)		0	
9	1234 (Marketing - Analyst)	175	175	
	1239 (Shipping - Intern)		0	
	9876 (Legal Department - Lawyer)		0	
11	1234 (Marketing - Analyst)	1031	1031	Nested query in HAVING clause (with same object in main query)
	1239 (Shipping - Intern)		0	
	9876 (Legal Department - Lawyer)		0	
16	1234 (Marketing - Analyst)	18316	18316	Nested query in WHERE clause. RIsk of leakage of information stored in table 'supplier' interrogated in the nested query (and not in the main query)
	1239 (Shipping - Intern)		0	
	9876 (Legal Department - Lawyer)		0	
18	1234 (Marketing - Analyst)	9	1	Query containing LIMIT operator (in particular LIMIT 100). Nested query in WHERE clause (with same object in main query)
	1239 (Shipping - Intern)		0	
	9876 (Legal Department - Lawyer)		0	
23	1234 (Marketing - Analyst)	19325	3871	Custom query created manually. Not available in the original TPC-H benchmark
	1239 (Shipping - Intern)		19325	
	9876 (Legal Department - Lawyer)		0	

Figure 7.2: Matrix illustrating users and queries additional information for TPC-H experiments

It is important to highlight that none of the notes illustrated in Figure 7.2 affected the security enforcement in any way. In fact, for the queries we chose, even when the *LIMIT* operator is present, there is no difference in the final results achieved with the different security enforcement approaches. Specifically, these queries contain a LIMIT operator that maintains the number of the results lower than the specified bound. Because of this operator, as described in Section 5.2, the results of the *APF* and the *OASIS'* approaches can be different than the others. In fact, in these two approaches, the LIMIT is applied on the results of the original subject's query before the security conditions are enforced. On the contrary, in the other approaches, the LIMIT is applied after the security restrictions are enforced (in the WHERE clause). However, in our experiments, for query 18 the results retrieved executing the original query are already less than the specified limit, hence the scenario described before can not happen. While in query 3, the *APF* and the *OASIS'* approaches cannot be applied as explained previously.

In the heat map shown in Figure 7.3 the average of the several execution times is displayed. The color scale is based on the following rules which are applied independently on each row:

- the minimum value has a green background;
- any value exceeding the minimum of 50% have a yellow background;
- any value doubling (or more) the minimum have a red background.

From these results we can infer the following considerations:

- when applicable, the *APF* and the *OASIS'* approaches almost always have the worst performance. Query rewriting approaches have lower execution times. This is especially true for queries with many tuples as results of the original queries (i.e. for queries 16 and 23), and when the security conditions are more complex (i.e. in queries 18 and 23).
- When the subject is granted the access to all or a subset of the objects, like *user 1234*, *QR-SQ* performs better than *QR-TA*. In particular, the execution times are worse for *QR-TA* when more complex security conditions are involved (i.e. for queries 3,9,18 and 23).
- *QR-TA* performs better, in terms of tenths of a second at most, when the user does not have access to the resource, like for *user 9876*.

User	Query	Original Query	APF	OASIS'	QR-SQmv	QR-SQvv	QR-TA
1234	3	4.62879			3.58703	3.45511	9.35984
	6	1.63375	1.67908	1.74526	1.63077	1.63118	1.62315
	9	20.09037			20.92499	20.83403	20.95446
	11	1.13223	3.39897	1.69314	1.19535	1.20971	1.18905
	16	3.87631	25.57419	19.34713	3.96031	3.96297	3.96006
	18	8.8651		9.23921	9.5079	9.5098	38.78425
	23	0.10063		484.59058	0.10187	0.11612	4.00872
1239	3	2.98878			3.42007	3.39872	84.42435
	6	1.63235	1.63411	1.74999	1.63475	1.63737	1.63633
	9	19.7499			0.01243	0.01541	0.01281
	11	1.13614	2.50483	1.62915	0.00352	0.00448	0.00342
	16	3.8742	26.84427	9.42257	0.00322	0.00443	0.00371
	18	9.00383		9.09488	10.41403	10.17481	526.22987
	23	0.49666		19.81647	0.11181	0.10968	11.8214
9876	3	2.97515			0.00457	0.00812	0.63252
	6	1.63512	1.63485	1.74026	0.00294	0.00397	0.00277
	9	19.63974			0.01058	0.015	0.01364
	11	1.12705	2.44976	1.62279	0.00358	0.0046	0.00369
	16	3.88248	26.5692	9.20646	0.00337	0.00522	0.00306
	18	9.18332		9.03663	0.00491	0.00841	6.74416
	23	0.09836		19.00954	0.05356	0.05894	0.0557
Total Result		**5.607155238**	**10.25436222**	**39.929604**	**2.694836191**	**2.674670476**	**33.87937905**

Figure 7.3: Heat Map with the execution times (in seconds) achieved with TPC-H dataset

- In general, *QR-SQmv* performs better than *QR-SQvv*. However, there are a few cases when the execution time of queries employing virtual views is slightly better than the one obtained with materialized views.
- The worst results are achieved by query 23 for *user 1234* using the simplified OASIS approach, and by queries 3 and 18 for *user 1239* using *QR-TA*. In the first case, the reason is that the number of the entries returned by the original query is very large and the security conditions to apply are complex, hence the iteration over the data required by this approach is very expensive. In the second case, the reason is more specific to the combination of security conditions, query, and subject. Consider, for example, query 3: the rewritten query that the database needs to execute is the following

```
SELECT
  l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) as revenue,
  o_orderdate,
  o_shippriority,
  resource_l_table,
  resource_o_table,
  resource_c_table,
  c_custkey
FROM
  customer,
  orders,
  lineitem,
  employees,
  nation n1,
  nation n2
WHERE
    ((resource_l_table = 'lineitem') and ((e_area = 'Marketing') or (e_area =
        'Shipping') or ((e_area = 'Supply_chain') and (e_role = 'Manager'))))
    and ((resource_o_table = 'orders') and ((e_area = 'Shipping') or ((e_area
        = 'Supply_chain') and (e_role = 'Manager')) or ((e_area = 'Marketing')
         and (e_role = 'Analyst'))) and (not ((100000 <= o_totalprice) and (
        e_role = 'Intern'))))
    and ((resource_c_table = 'customer') and ((e_area = 'Shipping') or ((
        e_area = 'Supply_chain') and (e_role = 'Manager')) or ((e_area = '
        Marketing') and (e_role = 'Analyst') and e_nationkey=n1.n_nationkey
        AND c_nationkey=n2.n_nationkey AND n1.n_regionkey=n2.n_regionkey )))
    and e_userID = 1239
```

```
  and c_mktsegment = 'MACHINERY'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < date '1995-03-29'
  and l_shipdate > date '1995-03-29'
GROUP BY
  l_orderkey,
  o_orderdate,
  o_shippriority,
  resource_l_table,
  resource_o_table,
  resource_c_table,
  c_custkey
ORDER BY
  revenue desc,
  o_orderdate
LIMIT 10;
```

User 1239 is an intern in the Shipping area. The problem in this query is that the tables *nation n1* and *nation n2* were added for the condition which specifies that a subject which is a marketing analyst can see only tuples of customers in the same continent. However, in this instance, since the user is in the Shipping area, the conditions on *nation* tables are not applied. As a consequence, a Cartesian product is performed with the whole *nation n1* and *nation n2* tables. This operation is extremely expensive, for this reason the execution time is so high. A similar Cartesian product is also performed in query 18.

7.3 Results on SAFAX

XACML to SQL Transformation The transformation from XACML to SQL was done manually for SAFAX policies because of unsolved errors generated by *X2S* tool when receiving these policies as input.

Views Generation and Management Employing Stored Queries Below, the measurements of time required to generate the views, as described in Section 6.2, are reported:

Virtual Views Generation	Materialized Views Generation
0.139	0.638
0.135	0.539
0.136	0.559
0.125	0.554
0.144	0.569
0.126	0.546
0.147	0.545
0.127	0.509
0.13	0.554
0.125	0.575
Average = 0.1334	**Average = 0.5588**

Table 7.4: Execution times in seconds for the generation of views from stored queries in SAFAX database

In particular, 9 views (virtual or materialized) are generated in order to enforce access control employing *QR-SQ* (and *OASIS'*) on SAFAX data.

Queries execution times The experiments on SAFAX dataset were performed by executing four queries using three different subjects. The four queries were designed to interrogate the

different objects in the database which are protected by security constraints. The users were chosen in order to have a representative from different groups, since most of the regulations are based on the membership to one or more groups. *User 1* is in the *Admin* group, *user 3* is in *Registered* and *Student* groups, while *user 16* is in *Registered* and *Staff* groups. In Figure 7.4, this information regarding the users is shown. Additionally, the number of results, before and after security enforcement, for the different subjects, is illustrated.

Query	User (Groups)	Number of Results (Original Query)	Number of Results (After Policy Enforcement)
1	1 (Admin)	279	279
	3 (Student, Registered)		1
	16 (Staff, Registered)		0
2	1 (Admin)	8	8
	3 (Student, Registered)		8
	16 (Staff, Registered)		8
3	1 (Admin)	114	114
	3 (Student, Registered)		1
	16 (Staff, Registered)		1
4	1 (Admin)	85	85
	3 (Student, Registered)		1
	16 (Staff, Registered)		85

Figure 7.4: Matrix illustrating users and queries additional information for SAFAX experiments

APF was not used for SAFAX dataset since the restrictions of the policies would have required complex queries with more tables in the FROM clauses and the addition of many different attributes in the SELECT clauses just to check the access control policies after the execution of the original query.
The execution times obtained in the experiments are reported in Figure 7.5 with the same rules applying for the color scale of the heatmap:

- the minimum value has a green background;
- any value exceeding the minimum of 50% have a yellow background;
- any value doubling (or more) the minimum have a red background.

User	Query	Original Query	OASIS'	QR-SQmv	QR-SQvv	QR-TA
1	1	0.003	0.2457	0.00292	0.00526	6.04426
	2	0.00097	0.1164	0.00267	0.00451	0.00523
	3	0.00107	0.17095	0.00268	0.0037	0.02696
	4	0.00143	0.18187	0.00329	0.0064	3.03539
3	1	0.00135	0.26067	0.00266	0.00478	3.34746
	2	0.00081	0.11846	0.00233	0.00431	0.0042
	3	0.00102	0.16984	0.00257	0.00399	0.02845
	4	0.0011	0.18987	0.00326	0.00667	1.99701
16	1	0.00117	0.2638	0.00254	0.00508	3.35119
	2	0.00082	0.11877	0.00264	0.00482	0.00436
	3	0.00116	0.186	0.00255	0.00408	0.02997
	4	0.00102	0.18547	0.00327	0.00638	2.00194
Total Result		**0.0012433333**	**0.1839833333**	**0.0027816667**	**0.0049983333**	**1.656368333**

Figure 7.5: Heat Map with the execution times (in seconds) achieved with SAFAX dataset

With SAFAX dataset, the results are more uniform. The approaches using stored queries always perform better than the rest. In particular, the option employing materialized views is constantly the best solution. *QR-TA* have the worst execution times for queries 1 and 4. This happens again because of Cartesian products are required during the execution of these queries. Thus, similarly to what we have described in the results of TPC-H dataset, the performances of this approach are severely affected. *OASIS'* is the worst solution for queries 2 and 3. Nevertheless, the longest execution time using *OASIS'* is obtained running query 1 which is the one with the largest number of tuples resulting from the original query. In general, compared to the other approaches, *OASIS'* is not always the worse as *QR-TA* performs even worse when Cartesian products are involved.

7.4 Discussion

The management of the elements and structures required for the alternative approaches and, in particular, for *QR-SQ* does not require much time and resources. As we have seen for the policies used for TPC-H dataset, the overall time required to get SQL formulas from six XACML policies is less than 2 seconds. Moreover, this transformation is not required every time a subject requests the access to the database, but it needs to be executed only when one or more XACML policies are added to the system or modified. The other task that must be performed before using *QR-SQ* is the generation of the necessary views (either virtual or materialized) from the stored queries. In any case, the time to generate the views is less than 1 second, as we have experimented with both the datasets. In particular, the time to create the materialized views is longer than the one to generate virtual views because, in the first case, the results of the queries that define the views are stored in the database. On the contrary, when employing virtual views, the results of the queries defining the views are not stored, in fact these queries are executed every time the virtual views are interrogated. As already discussed in Section 5.5.4, this fact entails better performance with the materialized views but the possibility of having information which is not up-to-date. On the other hand, with virtual views the performances are worse (and depends on the queries which define the views) but the data is always up-to-date.

In the results reported in this chapter, we never mentioned the time required to modify the queries in the query rewriting approaches and in the security enforcement step of *OASIS'*. In fact, the time we observed in all the experiments was always lower than 0.002 seconds so we considered it negligible when evaluating the results of the different approaches.

In the results illustrated in the heatmaps in Figures 7.3 and 7.5, the performances of the different approaches are not uniform in both cases, in particular, for TPC-H dataset, they are more variable. The main difference is that, in the experiments executed on TPC-H, there are queries, like query 6,11 and 16, where the security conditions involve only attributes in the tables which are

interrogated in the initial subject's query, without the need of any Cartesian product or join operations. In this instances, the approaches, especially the query rewriting ones, perform similarly. Moreover, the number of tuples resulting from the execution of the initial subject's query varies a lot, this fact also entails further variability in the performances of the different security enforcement implementations. Differently, *QR-SQmv* is constantly the best solution for SAFAX dataset because of the more complex security restrictions (which involve joins or Cartesian products) and because of the fact that the number of tuples involved in the security enforcement is not as variable as in TPC-H.
Nevertheless, *QR-SQ*, especially with the *QR-SQmv* variant, achieved promising results in terms of performance. Considering the overall experiments, it can be identified as the best solution. In particular, we noticed that *QR-SQ* (more specifically *QR-SQmv*) outperforms the other approaches in the following situations:

- when the security conditions are more complex and require the addition of a Cartesian product or join operation to the original query. In this case, *QR-SQ* execution times are substantially lower than the ones obtained with the other approaches and particularly with *QR-TA*.
- When the tuples resulting from the execution of the initial subject's query are numerous. In this instance, *QR-SQ* execution times are considerably lower than the ones achieved with *OASIS'* and *APF*.

Since the expensive operations like joins and Cartesian products are even more expensive when many tuples are involved, the combination of the two situations mentioned above affects significantly the performances of *QR-TA*. Similarly, with *OASIS'* and *APF*, when there are many tuples to check for compliance, and complex security conditions are involved, the execution times increase.
In general, another trend we observed is that, when the access is denied to the object(s), the execution times are lower and the differences among the different approaches are inferior.

Chapter 8

Conclusions

XACML is gaining significant attention and is becoming the de-facto standard for the specification and enforcement of ABAC policies. In this work, we recalled the current implementation for XACML policy enforcement as proposed by the OASIS and we presented alternative implementations which are designed to be more efficient.
With the aim of improving the current process in terms of performance, we focused our attention on query rewriting techniques. We also identified stored queries as a possible mean to enhance the efficiency of the access control system. Hence, after the study of related works in the context of access control, stored queries and query rewriting, we presented two query rewriting approaches that can be adopted for XACML policy enforcement. The first one, which we named Trivial Approach, was already introduced in similar manners in other works. While the second, which employs stored queries, is a new technique we propose that also aims to overcome the inefficiencies and complexities introduced in the Trivial Approach. Both alternatives require the transformation of an XACML policy to a set of SQL conditions which will be included in the subject's initial query. To achieve this, we proposed a method which consists in transforming the policy into an SMT instance which identifies the permit decision space of the policy expressed in XACML. Since the access would be granted if the subject respects these conditions, we just translate them into SQL conditions and include them to the initial subject's query.
Regarding the new approach, we analyzed the possible design alternatives and implementation options that can be adopted to concretely realize this system. Then, we illustrated the results we achieved testing on two datasets (TPC-H and SAFAX) the different security enforcement approaches: a simplified version of the OASIS's system, an A-Posteriori Filter, the Trivial Approach, and our new technique employing virtual and materialized views.

In conclusion, we achieved promising results with our new proposed approach. Even if in some instances the Trivial Approach performs similarly or slightly better (in terms of tenth of a second) than our new system employing stored queries, the overall behavior of our technique is better in terms of execution times than the other approaches. In particular, the efficiency of our solution makes the difference compared to the current XACML implementation when the results of the initial subject's query are numerous. Additionally, our solution employing stored queries outperforms the other query rewriting approach (which we named Trivial Approach), when the security restrictions are complex and involve different attributes in various tables in the database, as they can cause the execution of very expensive Cartesian products.
Another aspect which is not mentioned in the available literature on the topics and that we encountered in this work is the behavior of the current XACML security enforcement when *LIMIT*, *ROWNUM* or *TOP* operators are included in the subject's initial query. As described in Section 5.2, the results can be different to the one achieved with query rewriting approaches. From the point of view of the subject who launches the query, the results retrieved with the query rewriting approaches are always numerically corresponding to the number specified with one of these SQL operators. On the contrary, employing the current XACML implementation, the final results

obtained by the user could be less than the desired ones (if the implementation is not extended to handle these operators in a different way). The available literature about XACML does not mention any specific way to manage these operators differently or whether the subject's query should not include them. As a consequence, the final results achieved with the query rewriting approaches seem to be more coherent to the specifications defined with *LIMIT*, *ROWNUM* or *TOP* in the initial query.

8.1 Future Work

The new security enforcement approach employing query rewriting and stored query, as well as all the other described approaches, offers perspectives for further analysis and extensions.
First of all, the approaches we presented could be tested with more datasets, queries, and policies possibly involving some peculiarity or extreme cases like a dataset with a very large size or a highly complex policy.
Additionally, in this work we have examined only read access requests, corresponding to SELECT queries in the experiments we executed. Hence, the analysis of the different approaches can be expanded considering other types of access requests.
From the point of view of the implementation, there is one component in particular that can be improved. In fact, the parser that transforms formulas in first-order logic to SQL formulas can be extended in order to support the automatic mapping of XACML attributes to database attributes and to automatically get the SQL formula from SMT also for the query rewriting approach employing stored queries. These two tasks were performed manually in the experiments. On the other hand, a completely different procedure to transform XACML policies into SQL conditions could be designed, without the use of SMT instances identifying the permit decision space of the policies.
Further extensions could include the study of how to efficiently apply query rewriting when the subject's initial query contains nested queries. Additionally, the behavior of the current XACML implementation should be analyzed in more details for the instances when *LIMIT*, *ROWNUM* or *TOP* operators are present in the subject's initial query.
Finally, an extensive research on the topics introduced in the thesis and other relevant work could be the starting point to the creation of new security enforcement techniques designed to improve the current procedure suggested by the OASIS.

Bibliography

[1] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 143–154. VLDB Endowment, 2002. 17

[2] Ioannis Antonellis, Hector Garcia Molina, and Chi Chao Chang. Simrank++: query rewriting through link analysis of the click graph. *Proceedings of the VLDB Endowment*, 1(1):408–421, 2008. 17

[3] Jasper Bogaerts, Bert Lagaisse, and Wouter Joosen. Idea: Supporting policy-based access control on database systems. In *International Symposium on Engineering Secure Software and Systems*, pages 251–259. Springer, 2016. 18, 24

[4] Jasper Bogaerts, Bert Lagaisse, and Wouter Joosen. Transforming xacml policies into database search queries. 2017. 18

[5] Andrea Calı, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 16–21, 2003. 17

[6] Barbara Carminati, Elena Ferrari, Jianneng Cao, and Kian Lee Tan. A framework to enforce access control over data streams. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):28, 2010. 17

[7] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 190–200. IEEE, 1995. 17

[8] OASIS XACML Technical Committee. eXtensible Access Control Markup Language (XACML), 2013. 1

[9] Gianluca Correndo, Manuel Salvadores, Ian Millard, Hugh Glaser, and Nigel Shadbolt. Sparql query rewriting for implementing data integration over linked data. In *Proceedings of the 2010 EDBT/ICDT Workshops*, page 4. ACM, 2010. 17

[10] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for xml documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202, 2002. 18

[11] Maarten Decat, Jasper Moeys, Bert Lagaisse, and Wouter Joosen. Improving reuse of attribute-based access control policies using policy templates. In *International Symposium on Engineering Secure Software and Systems*, pages 196–210. Springer, 2015. 18

[12] Eberhard Grummt and Markus Müller. Fine-grained access control for epc information services. *The internet of things*, pages 35–49, 2008. 17

[13] Alon Y Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001. 17

[14] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013. 1

[15] Samuel Paul Kaluvuri, Alexandru Ionut Egner, Jerry den Hartog, and Nicola Zannone. Safax–an extensible authorization service for cloud environments. *Frontiers in ICT*, 2:9, 2015. 44

[16] Bo Luo, Dongwon Lee, Wang-Chien Lee, and Peng Liu. Qfilter: fine-grained run-time xml access control via nfa-based query rewriting. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 543–552. ACM, 2004. 18

[17] J Moeys and M Decat. Simple tree-structured attribute-based policy language (stapl). "https://github.com/stapl-dsl", 2015. 18

[18] Sriram Mohan, Arijit Sengupta, and Yuqing Wu. Access control for xml: a dynamic query rewriting approach. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 251–252. ACM, 2005. 18

[19] OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0. "http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html", 2013. 1, 12, 21, 59

[20] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting for semistructured data. In *ACM SIGMOD Record*, volume 28, pages 455–466. ACM, 1999. 17

[21] Andrea Presa, Yannis Velegrakis, Flavio Rizzolo, and Siarhei Bykau. Modeling associations through intensional attributes. In *International Conference on Conceptual Modeling*, pages 315–330. Springer, 2009. 18, 29, 30, 31

[22] Shariq Rizvi, Alberto Mendelzon, Sundararajarao Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 551–562. ACM, 2004. 17

[23] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, page 4. ACM, 2009. 17

[24] Divesh Srivastava and Yannis Velegrakis. Intensional associations between data and metadata. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 401–412. ACM, 2007. 18, 24, 25, 26, 30, 31

[25] Michael Stonebraker and Eugene Wong. Access control in a relational data base management system by query modification. In *Proceedings of the 1974 annual conference-Volume 1*, pages 180–186. ACM, 1974. 17

[26] TPC. Tpc benchmark h. "http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.2.pdf". 23, 43

[27] F Turkmen, J Den Hartog, and N Zannone. Analyzing access control policies with smt. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1508–1510, 2014. 34

[28] Fatih Turkmen, Jerry den Hartog, Silvio Ranise, and Nicola Zannone. Analysis of xacml policies with smt. In *International Conference on Principles of Security and Trust*, pages 115–134. Springer, 2015. 16, 21, 44

[29] Jan Van den Bussche, Stijn Vansummeren, and Gottfried Vossen. Towards practical meta-querying. *Information Systems*, 30(4):317–332, 2005. 18

[30] Oracle Virtual Private Database (VPD). Using oracle virtual private database to control data access. "http://docs.oracle.com/cd/B28359_01/network.111/b28531/vpd.htm#DBSEG007". 17

[31] Cong Yu and Lucian Popa. Constraint-based xml query rewriting for data integration. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 371–382. ACM, 2004. 17

Appendix A

Combining Algorithms

The following non-normative descriptions of the Combining Algorithms available in XACML v3 are taken from [19].

A.0.1 Combining Algorithm with six-dimensional decision set

The following Combining Algorithms use the six-dimensional decision set, that is

$$\mathcal{D}_6 = \{P, D, NA, I\{P\}, I\{D\}, I\{PD\}\}$$

where P stands for Permit, D for Deny, NA for NotApplicable and $I\{P\}$, $I\{D\}$, $I\{PD\}$ stand for Indeterminate Permit, Deny and PermitDeny respectively.

Deny-overrides The algorithm has the following behaviour:

1. If any decision is "Deny", the result is "Deny".
2. Otherwise, if any decision is "IndeterminateDP", the result is "IndeterminateDP".
3. Otherwise, if any decision is "IndeterminateD" and another decision is IndeterminateP or Permit, the result is "IndeterminateDP".
4. Otherwise, if any decision is "IndeterminateD", the result is "IndeterminateD".
5. Otherwise, if any decision is "Permit", the result is "Permit".
6. Otherwise, if any decision is "IndeterminateP", the result is "IndeterminateP".
7. Otherwise, the result is "NotApplicable".

Ordered-deny-overrides The behavior of the algorithm is identical to that of the Deny-overrides with one exception. The order in which the collection of rules is evaluated shall match the order as listed in the policy.

Permit-overrides The algorithm has the following behaviour:

1. If any decision is "Permit", the result is "Permit".
2. Otherwise, if any decision is "IndeterminateDP", the result is "IndeterminateDP".
3. Otherwise, if any decision is "IndeterminateP" and another decision is IndeterminateD or Deny, the result is "IndeterminateDP".
4. Otherwise, if any decision is "IndeterminateP", the result is "IndeterminateP".

5. Otherwise, if any decision is "Deny", the result is "Deny".
6. Otherwise, if any decision is "IndeterminateD", the result is "IndeterminateD".
7. Otherwise, the result is "NotApplicable".

Ordered-permit-overrides The behavior of the algorithm is identical to that of the Permit-overrides with one exception. The order in which the collection of rules is evaluated shall match the order as listed in the policy.

A.0.2 Combining Algorithm with four-dimensional decision set

The following Combining Algorithms use the four-dimensional decision set, that is

$$\mathcal{D}_4 = \{P, D, NA, I\}$$

First-applicable Each rule shall be evaluated in the order in which it is listed in the policy. For a particular rule, if the target matches and the condition evaluates to "True", then the evaluation of the policy shall halt and the corresponding effect of the rule shall be the result of the evaluation of the policy (i.e. "Permit" or "Deny"). For a particular rule selected in the evaluation, if the target evaluates to "False" or the condition evaluates to "False", then the next rule in the order shall be evaluated. If no further rule in the order exists, then the policy shall evaluate to "NotApplicable".

Only-one-applicable The algorithm, which is applicable only to Policy elements, has the following behaviour: in the entire set of policies in the policy set, if no policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm shall be "NotApplicable". If more than one policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm shall be "Indeterminate". If only one policy is considered applicable by evaluation of its target, then the result of the policy-combining algorithm shall be the result of evaluating the policy.

Deny-unless-permit The algorithm has the following behaviour:

1. If any decision is "Permit", the result is "Permit".
2. Otherwise, the result is "Deny".

Permit-unless-deny The algorithm has the following behaviour:

1. If any decision is "Deny", the result is "Deny".
2. Otherwise, the result is "Permit".

Appendix B

TPC-H Data

B.1 Dataset Schema

We list below the elements which are part of the dataset. The ones in blue are those which were added for this work but were not in the original dataset.

table_name	column_name	ordinal_position	data_type	character_maximum_length	numeric_precision	numeric_precision_radix	numeric_scale
customer	c_custkey	1	integer		32	2	0
	c_name	2	character varying	25			
	c_address	3	character varying	40			
	c_nationkey	4	integer		32	2	0
	c_phone	5	character	15			
	c_acctbal	6	numeric		15	10	2
	c_mktsegment	7	character	10			
	c_comment	8	character varying	117			
	resource_c_table	9	character varying	255			
nation	n_nationkey	1	integer		32	2	0
	n_name	2	character	25			
	n_regionkey	3	integer		32	2	0
	n_comment	4	character varying	152			
	resource_n_table	5	character varying	255			
part	p_partkey	1	integer		32	2	0
	p_name	2	character varying	55			
	p_mfgr	3	character	25			
	p_brand	4	character	10			
	p_type	5	character varying	25			
	p_size	6	integer		32	2	0
	p_container	7	character	10			
	p_retailprice	8	numeric		15	10	2
	p_comment	9	character varying	23			
	resource_p_table	10	character varying	255			
region	r_regionkey	1	integer		32	2	0
	r_name	2	character	25			
	r_comment	3	character varying	152			
	resource_r_table	4	character varying	255			
supplier	s_suppkey	1	integer		32	2	0
	s_name	2	character	25			
	s_address	3	character varying	40			
	s_nationkey	4	integer		32	2	0
	s_phone	5	character	15			
	s_acctbal	6	numeric		15	10	2
	s_comment	7	character varying	101			
	resource_s_table	8	character varying	255			
lineitem	l_orderkey	1	integer		32	2	0
	l_partkey	2	integer		32	2	0
	l_suppkey	3	integer		32	2	0
	l_linenumber	4	integer		32	2	0
	l_quantity	5	numeric		15	10	2
	l_extendedprice	6	numeric		15	10	2
	l_discount	7	numeric		15	10	2
	l_tax	8	numeric		15	10	2
	l_returnflag	9	character	1			

	l_linestatus	10	character	1			
	l_shipdate	11	date				
	l_commitdate	12	date				
	l_receiptdate	13	date				
	l_shipinstruct	14	character	25			
	l_shipmode	15	character	10			
	l_comment	16	character varying	44			
	resource_l_table	17	character varying	255			
orders	o_orderkey	1	integer		32	2	0
	o_custkey	2	integer		32	2	0
	o_orderstatus	3	character	1			
	o_totalprice	4	numeric		15	10	2
	o_orderdate	5	date				
	o_orderpriority	6	character	15			
	o_clerk	7	character	15			
	o_shippriority	8	integer		32	2	0
	o_comment	9	character varying	79			
	resource_o_table	10	character varying	255			
partsupp	ps_partkey	1	integer		32	2	0
	ps_suppkey	2	integer		32	2	0
	ps_availqty	3	integer		32	2	0
	ps_supplycost	4	numeric		15	10	2
	ps_comment	5	character varying	199			
	resource_ps_table	6	character varying	255			
conditions	cn_id	1	integer		32	2	0
	cn_name	2	character varying	255			
	cn_definition	3	character varying	255			
	resource_cn_table	4	character varying	255			
employees	e_userid	1	integer		32	2	0
	e_firstname	2	character varying	255			
	e_lastname	3	character varying	255			
	e_area	4	character varying	255			
	e_role	5	character varying	255			
	resource_e_table	6	character varying	255			
	e_nationkey	7	integer		32	2	0
ismarketinganalyst	userid	1	integer		32	2	0
issupply_chainmanager	userid	1	integer		32	2	0
sameregioncustomer	userid	1	integer		32	2	0
	custkey	2	integer		32	2	0
inrule_orders	userid	1	integer		32	2	0
	limit_price	2	integer		32	2	0
inshippingarea	userid	1	integer		32	2	0
inmarketingarea	userid	1	integer		32	2	0
insupply_chainarea	userid	1	integer		32	2	0

B.2 Original Queries

B.2.1 Query 3

```
-- using 1493740174 as a seed to the RNG


select
  l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) as revenue,
```

```
  o_orderdate,
  o_shippriority,
  resource_l_table,
  resource_o_table,
  resource_c_table
from
  customer,
  orders,
  lineitem
where
  c_mktsegment = 'MACHINERY'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < date '1995-03-29'
  and l_shipdate > date '1995-03-29'
group by
  l_orderkey,
  o_orderdate,
  o_shippriority,
  resource_l_table,
  resource_o_table,
  resource_c_table
order by
  revenue desc,
  o_orderdate
limit 10;
```

B.2.2 Query 6

```
-- using 1493740174 as a seed to the RNG


select
  sum(l_extendedprice * l_discount) as revenue, resource_l_table
from
  lineitem
where
  l_shipdate >= date '1994-01-01'
  and l_shipdate < date '1994-01-01' + interval '1' year
  and l_discount between 0.04 - 0.01 and 0.04 + 0.01
  and l_quantity < 24
group by
  resource_l_table
```

B.2.3 Query 9

```
-- using 1493740174 as a seed to the RNG


select
  nation,
  o_year,
  sum(amount) as sum_profit,
  resource_l_table,
  resource_o_table,
  resource_p_table,
  resource_s_table,
  resource_ps_table
from
  (
    select
      n_name as nation,
      extract(year from o_orderdate) as o_year,
```

```
      l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount,
      resource_l_table,
      resource_o_table,
      resource_p_table,
      resource_s_table,
      resource_ps_table

    from
      part,
      supplier,
      lineitem,
      partsupp,
      orders,
      nation
    where
      s_suppkey = l_suppkey
      and ps_suppkey = l_suppkey
      and ps_partkey = l_partkey
      and p_partkey = l_partkey
      and o_orderkey = l_orderkey
      and s_nationkey = n_nationkey
      and p_name like '%antique%'
  ) as profit
group by
  nation,
  o_year,
  resource_l_table,
  resource_o_table,
  resource_p_table,
  resource_s_table,
  resource_ps_table
order by
  nation,
  o_year desc;
```

B.2.4 Query 11

```
-- using 1493740174 as a seed to the RNG


select
  ps_partkey,
  sum(ps_supplycost * ps_availqty) as value,
  resource_ps_table,
  resource_s_table
from
  partsupp,
  supplier,
  nation
where
  ps_suppkey = s_suppkey
  and s_nationkey = n_nationkey
  and n_name = 'UNITED STATES'
group by
  resource_ps_table,
  resource_s_table,
  ps_partkey having
    sum(ps_supplycost * ps_availqty) > (
      select
        sum(ps_supplycost * ps_availqty) * 0.0001000000
      from
        partsupp,
        supplier,
        nation
      where
```

```
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'UNITED STATES'
    )

order by
  value desc;
```

B.2.5 Query 16

```
-- using 1493740174 as a seed to the RNG


select
  p_brand,
  p_type,
  p_size,
  count(distinct ps_suppkey) as supplier_cnt,
  resource_ps_table,
  resource_p_table
from
  partsupp,
  part
where
  p_partkey = ps_partkey
  and p_brand <> 'Brand#41'
  and p_type not like 'PROMO BURNISHED%'
  and p_size in (26, 8, 44, 50, 29, 17, 47, 7)
  and ps_suppkey not in (
    select
      s_suppkey
    from
      supplier
    where
      s_comment like '%Customer%Complaints%'
  )
group by
  p_brand,
  p_type,
  p_size,
  resource_ps_table,
  resource_p_table
order by
  supplier_cnt desc,
  p_brand,
  p_type,
  p_size;
```

B.2.6 Query 18

```
-- using 1493740174 as a seed to the RNG


select
  c_name,
  c_custkey,
  o_orderkey,
  o_orderdate,
  o_totalprice,
  sum(l_quantity),
  resource_l_table,
  resource_o_table,
  resource_c_table
```

```
from
  customer,
  orders,
  lineitem
where
  o_orderkey in (
    select
      l_orderkey
    from
      lineitem
    group by
      l_orderkey having
        sum(l_quantity) > 315
  )
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by
  c_name,
  c_custkey,
  o_orderkey,
  o_orderdate,
  o_totalprice,
  resource_l_table,
  resource_o_table,
  resource_c_table
order by
  o_totalprice desc,
  o_orderdate
limit 100;
```

B.2.7 Query 23

```
select customer.*
from customer
where c_acctbal >8578
```

B.3 SQL Security Formulas

Note that the formulas reported are the one interrogating the virtual views. The formulas used when materialized views are employed are the same but the views are named differently, in particular they only have a prefix (i.e. *m_*) which denotes that they are materialized.

B.3.1 Security Formula on *lineitem* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
((resource_l_table = 'lineitem') and ((e_area = 'Marketing') or (e_area = 'Shipping
    ') or ((e_area = 'Supply_chain') and (e_role = 'Manager'))))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(resource_l_table = 'lineitem' and (exists (select * from inMarketingArea where
    userid=?) or exists (select * from inShippingArea where userid=?) or exists (
    select * from isSupply_chainManager where userid=?)))
```

B.3.2 Security Formula on *order* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
((resource_o_table = 'orders') and ((e_area = 'Shipping') or ((e_area = '
    Supply_chain') and (e_role = 'Manager')) or ((e_area = 'Marketing') and (e_role
     = 'Analyst'))) and (not ((100000 <= o_totalprice) and (e_role = 'Intern'))))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(resource_o_table = 'orders' and (exists (select * from inShippingArea where userid
    =?) or exists (select * from isSupply_chainManager where userid=?) or exists (
    select * from ismarketinganalyst where userid=?)) and (not (exists (select *
    from inRule_orders where userid=? and limit_price<=o_totalprice))) )
```

B.3.3 Security Formula on *customer* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
((resource_c_table = 'customer') and ((e_area = 'Shipping') or ((e_area = '
    Supply_chain') and (e_role = 'Manager')) or ((e_area = 'Marketing') and (e_role
     = 'Analyst') and e_nationkey=n1.n_nationkey AND c_nationkey=n2.n_nationkey AND
     n1.n_regionkey=n2.n_regionkey )))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(resource_c_table = 'customer' and (exists (select * from inShippingArea where
    userid=?) or exists (select * from isSupply_chainManager where userid=?) or (
    exists (select * from ismarketinganalyst where userid=?) and exists (select *
    from sameRegionCustomer where userid=? and c_custkey=custkey))))
```

B.3.4 Security Formula on *supplier* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
((resource_s_table = 'supplier') and ((e_area = 'Supply_chain') or ((e_area = '
    Marketing') and (e_role = 'Analyst'))))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(resource_s_table = 'supplier' and (exists (select * from inSupply_chainArea where
    userid=?) or (exists (select * from ismarketinganalyst where userid=?))))
```

B.3.5 Security Formula on *partsupp* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
((resource_ps_table = 'partsupp') and ((e_area = 'Supply_chain') or ((e_area = '
    Marketing') and (e_role = 'Analyst'))))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(resource_ps_table = 'partsupp' and (exists (select * from inSupply_chainArea where
     userid=?) or (exists (select * from ismarketinganalyst where userid=?))))
```

B.3.6 Security Formula on *part* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
((resource_p_table = 'part') and ((e_area = 'Supply_chain') or (e_area = 'Marketing
    '')))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(resource_p_table = 'part' and (exists (select * from inSupply_chainArea where
    userid=?) or (exists (select * from inMarketingArea where userid=?))))
```

Appendix C

SAFAX Data

C.1 Dataset Schema

We list below the elements which are part of the dataset. The ones in blue are those which were added for this work but were not in the original dataset.

table_name	column_name	ordinal_ position	data_type	character_ maximum _length	numeric_ precision	numeric_ precision _radix	numeric _scale
setting	sid	1	bigint		64	2	0
	skey	2	character varying	255			
	svalue	3	character varying	255			
	scomponent	4	character varying	255			
	stype	5	character varying	255			
sfx_attribute	attid	1	bigint		64	2	0
	attributeid	2	character varying	255			
	attributedatatype	3	character varying	255			
	attributevalue	4	character varying	10000			
	relid	5	bigint		64	2	0
	demoid	6	bigint		64	2	0
sfx_configuration	configid	1	bigint		64	2	0
	configkey	2	character varying	255			
	configvalue	3	character varying	255			
	demoid	4	bigint		64	2	0
sfx_authorizedclient	acid	1	bigint		64	2	0
	clientname	2	character varying	255			
	clientkey	3	text				
	isactive	4	boolean				
	uid	5	bigint		64	2	0
sfx_action	uaid	1	bigint		64	2	0
	actionid	2	character varying	45			
	action	3	character varying	255			
	actiondesc	4	text				
	actionresource	5	character varying	255			
	actionresourcetype	6	character varying	255			
	starttime	7	numeric			10	
	endtime	8	numeric			10	
	isactive	9	boolean				
	usid	10	bigint		64	2	0
sfx_action_details	uaid	1	bigint		64	2	0
	actionid	2	character varying	45			
	action	3	character varying	255			
	actiondesc	4	text				
	actionresource	5	character varying	255			
	actionresourcetype	6	character varying	255			
	starttime	7	numeric			10	
	endtime	8	numeric			10	
	isactive	9	boolean				
	usid	10	bigint		64	2	0
	serviceid	11	character varying	255			
	isdown	12	boolean				

sfx_conflict	conflictid	1	bigint		64	2	0
	conflictname	2	character varying	45			
sfx_errorlog	uelid	1	bigint		64	2	0
	transactionid	2	character varying	45			
	errorheader	3	text				
	errorlog	4	text				
	logtype	5	smallint		16	2	0
	utimestamp	6	numeric			10	
	component	7	character varying	255			
sfx_log	ulid	1	bigint		64	2	0
	transactionid	2	character varying	45			
	log	3	text				
	logtype	4	smallint		16	2	0
	utimestamp	5	numeric			10	
	component	6	character varying	255			
sfx_serviceinterface	siid	1	bigint		64	2	0
	siname	2	character varying	255			
	endpoint	3	character varying	255			
	interfacedesc	4	text				
	httpmethod	5	character varying	20			
	serviceparams	6	bigint		64	2	0
	returntype	7	character varying	255			
	isactive	8	boolean				
	srid	9	bigint		64	2	0
sfx_pdp	pdpid	1	bigint		64	2	0
	pdpcode	2	character varying	45			
	pdpurl	3	character varying	255			
	rcalgorithm	4	character varying	45			
	ispersistent	5	boolean				
	isactive	6	boolean				
	haspip	7	boolean				
	demoid	8	bigint		64	2	0
sfx_demouser	demoid	1	bigint		64	2	0
	uid	2	bigint		64	2	0
	accesslevel	3	smallint		16	2	0
sfx_projectuser	prid	1	bigint		64	2	0
	uid	2	bigint		64	2	0
	accesslevel	3	smallint		16	2	0
sfx_democonflicts	transparencyid	1	bigint		64	2	0
	conflictid	2	bigint		64	2	0
sfx_servicedependency	srid	1	bigint		64	2	0
	sdid	2	bigint		64	2	0
sfx_demoudf	demoid	1	bigint		64	2	0
	udfid	2	bigint		64	2	0

sfx_group	gid	1	bigint		64	2	0
	groupname	2	character varying	255			
	isactive	3	boolean				
sfx_demo	demoid	1	bigint		64	2	0
	demoname	2	character varying	255			
	demodesc	3	text				
	isactive	4	boolean				
	prid	5	bigint		64	2	0
	demoowner	6	bigint		64	2	0
	resource_d_table	7	character varying	255			
sfx_projectsetting	settingid	1	bigint		64	2	0
	settingkey	2	character varying	255			
	settingvalue	3	character varying	255			
	projectid	4	bigint		64	2	0
sfx_issue	issid	1	bigint		64	2	0
	issueheader	2	character varying	255			
	issuedesc	3	text				
	isfeature	4	boolean				
	isactive	5	integer		32	2	0
	uid	6	bigint		64	2	0
	resource_i_table	7	character varying	255			
sfx_project	prid	1	bigint		64	2	0
	projectname	2	character varying	255			
	projecturl	3	character varying	255			
	projectdesc	4	text				
	isactive	5	boolean				
	ispublic	6	boolean				
	projectowner	7	bigint		64	2	0
	resource_p_table	8	character varying	255			
sfx_statistics_log	uaid	1	bigint		64	2	0
	actionid	2	character varying	45			
	action	3	character varying	255			
	actiondesc	4	text				
	evalstarttime	5	numeric			10	
	evalendtime	6	numeric			10	
	evaltotaltime	7	numeric			10	
	pepserviceid	8	character varying	255			
	pepevaltime	9	numeric			10	
	chserviceid	10	character varying	255			
	chevaltime	11	numeric			10	
	pipserviceid	12	character varying	255			
	pipevaltime	13	numeric			10	
	pdpserviceid	14	character varying	255			
	pdpevaltime	15	numeric			10	

	papserviceid	16	character varying	255			
	papevaltime	17	numeric			10	
	udfserviceid	18	character varying	255			
	udfevaltime	19	numeric			10	
	actionresource	20	character varying	255			
	actionresourcetype	21	character varying	255			
	isactive	22	boolean				
	usid	23	bigint		64	2	0
sfx_transaction	utid	1	bigint		64	2	0
	transactionid	2	character varying	45			
	utimestamp	3	numeric			10	
	isactive	4	boolean				
	clientcode	5	character varying	255			
	pdpcode	6	character varying	255			
sfx_ucon_action	uconactionid	1	bigint		64	2	0
	ucon_sessionid	2	character varying	255			
	starttime	3	numeric			10	
	endtime	4	numeric			10	
	isdown	5	boolean				
sfx_ucon_log	ulid	1	bigint		64	2	0
	log	2	text				
	ucon_sessionid	3	character varying	255			
	utimestamp	4	numeric			10	
sfx_ucon_request	id	1	bigint		64	2	0
	request_sessionid	2	character varying	255			
	xrid	3	bigint		64	2	0
	trigger_interval	4	character varying	255			
	updatetime	5	numeric			10	
	transactionid	6	character varying	255			
	ucon_sessionid	7	character varying	255			
	ucon_display	8	boolean				
	has_trigger	9	boolean				
sfx_ucon_update	id	1	bigint		64	2	0
	request_sessionid	2	character varying	255			
	updatetype	3	character varying	255			
	xrid	4	bigint		64	2	0
	updateinterval	5	bigint		64	2	0
	updatetime	6	numeric			10	
	transactionid	7	character varying	255			
	ucon_sessionid	8	character varying	255			
	ucon_display	9	boolean				
sfx_userrequest	urid	1	bigint		64	2	0
	requestheader	2	character varying	255			
	requestmessage	3	text				

	status	4	smallint		16	2	0
sfx_transparency	transparencyid	1	bigint		64	2	0
	pdpid	2	bigint		64	2	0
	uid	3	bigint		64	2	0
	viewpoint	4	character varying	45			
sfx_serviceregistry	srid	1	bigint		64	2	0
	serviceid	2	character varying	255			
	servicename	3	character varying	255			
	servicecomponent	4	character varying	45			
	serviceprovider	5	character varying	255			
	serviceurl	6	character varying	255			
	servicedesc	7	text				
	isactive	8	boolean				
	isroot	9	boolean				
	issafax	10	boolean				
sfx_usergroup	gid	1	bigint		64	2	0
	uid	2	bigint		64	2	0
sfx_user	uid	1	bigint		64	2	0
	uname	2	character varying	255			
	upwd	3	character	64			
	fullname	4	character varying	100			
	email	5	character varying	255			
	isactive	6	boolean				
	resource_u_table	7	character varying	255			
sfx_xacmlrequest	xrid	1	bigint		64	2	0
	requestid	2	character varying	100			
	request	3	text				
	isactive	4	boolean				
	demoid	5	bigint		64	2	0
ext_trust_policy	tpid	1	bigint		64	2	0
	policyid	2	character varying	255			
	policy	3	text				
	policypath	4	character varying	255			
	policytype	5	character varying	255			
	isactive	6	boolean				
	demoid	7	bigint		64	2	0
sfx_accessedattribute	id	1	bigint		64	2	0
	attid	2	bigint		64	2	0
	attributeid	3	character varying	255			
	attributedatatype	4	character varying	255			
	attributevalue	5	character varying	10000			
	updatetime	6	numeric			10	
	relid	7	bigint		64	2	0
	demoid	8	bigint		64	2	0

	transactionid	9	character varying	255			
	ucon_sessionid	10	character varying	255			
	ucon_display	11	boolean				
sfx_xacmlpolicy	xpid	1	bigint		64	2	0
	policyid	2	character varying	255			
	policy	3	text				
	policypath	4	character varying	255			
	isactive	5	boolean				
	demoid	6	bigint		64	2	0
sfx_usersession	usid	1	bigint		64	2	0
	uid	2	bigint		64	2	0
	ip	3	character varying	255			
	useragent	4	character varying	255			
	sessionid	5	character varying	255			
	sessionstart	6	numeric			10	
	lasttransaction	7	numeric			10	
	sessionend	8	numeric			10	
	isactive	9	boolean				
	isguest	10	boolean				
conditions	cn_id	1	integer		32	2	0
	cn_name	2	character varying	255			
	cn_definition	3	character varying	255			
	resource_cn_table	4	character varying	255			
isregistered	userid	1	bigint		64	2	0
isstaff	userid	1	bigint		64	2	0
demoisinproject	userid	1	bigint		64	2	0
	demoid	2	bigint		64	2	0
	prid	3	bigint		64	2	0
registerednumproject	userid	1	bigint		64	2	0
	numproj	2	bigint		64	2	0
isprofileowner	userid	1	bigint		64	2	0
isadmin	userid	1	bigint		64	2	0
isguest	userid	1	bigint		64	2	0
isinproject	userid	1	bigint		64	2	0
	prid	2	bigint		64	2	0
isprojectowner	userid	1	bigint		64	2	0
	prid	2	bigint		64	2	0

C.2 Original Queries

C.2.1 Query 1

```
select D.*
from sfx_demo D
```

C.2.2 Query 2

```
select I.*
from sfx_issue I
```

C.2.3 Query 3

```
select U2.*
from sfx_user U2
```

C.2.4 Query 4

```
select P.*
from sfx_project P
```

C.3 SQL Security Formulas

Note that the formulas reported are the one interrogating the virtual views. The formulas used when materialized views are employed are the same but the views are named differently, in particular they only have a prefix (i.e. *m_*) which denotes that they are materialized.

C.3.1 Security Formula on *sfx_demo* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
(D.resource_d_table='sfx_demo' and(
(U.uid=UG.uid and UG.gid=G.gid and g.groupname='admin') or
(U.uid=UP.uid and UP.prid=D.prid)
))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(D.resource_d_table='sfx_demo' and(
(exists (select * from isAdmin where userid=?)) or
(exists (select * from demoIsInProject m where m.userid=? and m.demoid=D.demoid))
))
```

C.3.2 Security Formula on *sfx_issue* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
(I.resource_i_table='sfx_issue' and(
(U.uid=UG.uid and UG.gid=G.gid and g.groupname='admin') or
(U.uid=UG.uid and UG.gid=G.gid and (g.groupname='registered' or g.groupname='guest'
    ))
))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(I.resource_i_table='sfx_issue' and(
(exists (select * from isAdmin where userid=?)) or
(exists (select * from isRegistered where userid=?)) or
(exists (select * from isGuest where userid=?))
))
```

C.3.3 Security Formula on *sfx_user* tuples

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
(U.resource_u_table='sfx_user' and(
(U.uid=U2.uid) or
(U.uid=UG.uid and UG.gid=G.gid and g.groupname='admin')
))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(U2.resource_u_table='sfx_user' and(
(exists (select * from isAdmin where userid=?)) or
(exists (select * from isProfileOwner where uid=?))
))
```

C.3.4 Security Formula on *sfx_project* tuples

As the security conditions in the XACML policy where particularly interesting, in this case we included the restrictions that in the policy apply also to actions different to *read*.

Formula employed for the Trivial Approach and the A-Posteriori Filter

```
(P.resource_p_table='sfx_project' and(
(U.uid=UG.uid and UG.gid=G.gid and g.groupname='admin') or
(U.uid=UG.uid and UG.gid=G.gid and (G.groupname='staff' or (G.groupname='registered
    ' and (select count(*) from sfx_projectuser where uid=?)<1)))
or (U.uid=P.projectowner)
or (U.uid=UP.uid and UP.prid=P.prid)
))
```

Formula employed for the simplified OASIS approach and the Query Rewriting Employing Stored Queries

```
(P.resource_p_table='sfx_project' and(
(exists (select * from isAdmin where userid=?)) or
(exists (select * from isStaff where userid=?)) or
(exists (select * from registeredNumProject where userid=? and numProj<1)) or
(exists (select * from isProjectOwner where userid=? and prid=P.prid)) or
(exists (select * from isInProject where userid=? and prid=P.prid))
))
```