

2013

A role-based access control schema for materialized views

Hassaan Yousafi
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Yousafi, Hassaan, "A role-based access control schema for materialized views" (2013). *Electronic Theses and Dissertations*. 4895.
<https://scholar.uwindsor.ca/etd/4895>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

A ROLE-BASED ACCESS CONTROL SCHEMA FOR MATERIALIZED VIEWS

By

Hassaan Yousafi

A Thesis
Submitted to the Faculty of Graduate Studies
Through the **School of Computer Science**
in Partial Fulfillment of the Requirements for
the Degree of **Master of Science**
at the University of Windsor

Windsor, Ontario, Canada

2013

© 2013 Hassaan Yousafi

A ROLE-BASED ACCESS CONTROL SCHEMA FOR MATERIALIZED VIEWS

by

Hassaan Yousafi

APPROVED BY:

Kemal Tepe
Department of Electrical Engineering

Dan Wu
School of Computer Science

Robert D. Kent, Advisor
School of Computer Science

May 06, 2013

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

This thesis research presents a framework that enhances security at the level of materialized views. Materialized views can be used for performance reasons in very large systems such as data warehouses or distributed systems, or for providing a filtered selection of data from a more general database. Existing proposed techniques provide rule-based access control for materialized views, however, the administration of such systems is time consuming and cumbersome in a large environment. This thesis presents a role-based access control schema for materialized views in which data authorization rules are associated with roles and defined in *Datalog* syntax in plain text files, a column level restriction is imposed on a materialized view based on a user assigned role, and a role conflict strategy is defined in which priority is given to each conflicting role in order to resolve role conflicts if a user is gaining authorization for permissions associated with conflicting roles at the same time.

KEYWORDS

Materialized Views, Authorization Views, Session Roles, Role Conflicts

DEDICATION

To my parents, my sister Munnaza, and my loving wife Mariam

ACKNOWLEDGEMENTS

I would like to thank all the people who have helped me during my thesis research. Special thanks to my supervisor Dr. Robert D. Kent for his immense support and guidance throughout my master's degree.

My sincere appreciation goes to my parents for their unconditional support, and also to my wife Mariam for her encouragement and support in all these years.

I express my deep appreciation to my elder sister Munazza for standing by me during these years.

I would also like to thank Dr. Dan Wu and Dr. Kamal Tepe for serving in my thesis committee.

Lastly, I would like to thank all my colleagues in Computer Science department who helped me through their inputs to improve my thesis work.

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY	iii
ABSTRACT.....	iv
DEDICATION	v
ACKNOWLEDGEMENTS	vi
LIST OF TABLES.....	ix
LIST OF FIGURES	x
CHAPTER I	1
INTRODUCTION.....	1
1.1 - Data Access Control Methods	2
1.1.1 - Mandatory Access Control (MAC)	2
1.1.2 - Discretionary Access Control (DAC).....	2
1.1.3 - Role-based Access Control (RBAC)	3
1.2 - Materialized View	6
1.3 - Authorization View	9
1.4 - Query Rewriting	11
1.5 - Problem Statement.....	12
1.6 - Thesis Contribution	13
1.7 - Organization of thesis.....	13
CHAPTER II.....	14
RELATED WORK	14
2.1 - Context Based Access Control (CBAC).....	14
2.2 - Attribute Based Access Control (ABAC).....	15
2.3 - XML Based Access Control	16
2.4 - Access Authorization for Relational Database	18
2.5 - Fine-grained Authorization Policies	19
2.6 - Authorization Views and Conditional Query Containment.....	20
2.7 - Access Control to Materialized Views	21

2.8 - Comparison of Various Research Works	24
CHAPTER III	25
PROPOSED FRAMEWORK	25
3.1 - Data Access Control to Materialized Views.....	25
3.2 - Proposed Role-Based Framework	26
3.2.1 - Role Assignment.....	28
3.2.2 - Role-Based Authorizations	30
3.2.3 - Session Roles.....	32
3.2.4 - Role Conflicts.....	33
CHAPTER IV	36
IMPLEMENTATION AND VERIFICATION.....	36
4.1 - Background.....	36
4.2 - Implementation.....	36
4.2.1 - Role Assignment Module	37
4.2.2 - Request Handler Module	38
4.3 - Verification.....	41
4.3.1 - Basic Requirements	42
4.3.2 - Additional Functions	46
4.4 - Scalability Test Results	49
4.5 - Summary Comments	49
CHAPTER V	50
CONCLUSION AND FUTURE WORK	50
5.1 - Conclusion.....	50
5.2 - Future Work.....	51
5.2.1 - Role Automation.....	51
5.2.2 - Workflow Management in RBAC.....	51
5.2.3 - Global vs. Local Authorizations	52
BIBLIOGRAPHY	53
VITA AUCTORIS	57

LIST OF TABLES

2.1 - Comparison of various researches works.....	24
4.1 - Scalability test results.....	49

LIST OF FIGURES

1.1 - RBAC Architecture (NIST solution).....	4
1.2 - RBAC Hierarchal Structure (NIST solution).....	5
1.3 - RBAC Separation of Duties (NIST solution).....	5
1.4 - RBAC Elements (NIST solution).....	6
1.5 - Materialized view site architecture.....	7
2.1 - XACML data flow.....	17
2.2 - RBAC Model and X-RBAC Policy Components.....	17
2.3 - Authorization policies for materialized views.....	22
3.1 - Proposed RBAC Architecture for Materialized Views.....	27
3.2 - Workflow of Role Assignment process.....	28
3.3 - Architecture of Role-Based Authorization.....	30
3.4 - Role and Authorization Views.....	32
3.5 - Role Conflict Strategies.....	34
4.1 - Proposed Architecture.....	36
4.2 - Role Assignment module pseudo code.....	37
4.3 - Active Sessions pseudo code.....	38
4.4 - View Selection pseudo code.....	40
4.5 - Query Construction.....	41
4.6 - Login page.....	42
4.7 - Homepage.....	43

4.8 - Simple query execution and results.....	44
4.9 - Authorization Views.....	44
4.10 - Conflict Strategy.....	45
4.11 - Role conflict detected.....	45
4.12 - Additional SQL statement clauses.....	46
4.13 - Query Generation using SQL statement clauses.....	47
4.14 - Results of query with additional SQL clauses.....	47
4.15 - Query Generation using Display None function.....	48
4.16 - Results of query using Display None function.....	48

CHAPTER I

INTRODUCTION

Data Access Control has been considered a major issue in the information technology community. The main focus of researcher is to provide a mechanism to secure data, and provide the access of data based on identity and attributes of a known users or a process by using a reference monitor and specialized authorization rules.

The authorization policies are defined to limit the access of data based on user attributes or role. Several different approaches have been proposed based on the requirements of different domains. There are some research works which are generic, and can be applied to any domain. Role-Based Access Control (RBAC) [24] is the leading access control model due to its flexible nature and ease of maintenance.

Currently, data amount and availability is increasing rapidly; much of the data is stored on remote file systems. Materialized Views is another addition in Relational Database Management Systems. A materialized view takes a different approach in which the query result is cached as a concrete table that may be updated from the original base tables from time to time. This enables much more efficient access of data. It may be a local copy of data located remotely or a subset of the rows or columns of a table or join result, or may be a summary based on aggregations of a table's data.

Existing proposed techniques provide rule-based access control for materialized views [2][3], but to the best of our knowledge, the administration of such systems is time consuming and cumbersome in a large environment as administrators define rules for each user to control the access to materialized views.

In this chapter, we give an introduction of existing data access control methods, Materialized View, Authorization View, and Query Rewriting. In last two sections, we describe the problem statement, and thesis contribution.

1.1- Data Access Control Methods

Several Data Access Control methods have been introduced by keeping in view the requirements of an organization, and the sensitivity of the data. In this section, we discuss different data access control methods.

1.1.1 - Mandatory Access Control (MAC)

Mandatory Access Control utilizes hard coded security rules. Rules are coded into an application or operating system. The security policy is centrally administered and can be override by the users, and it is applied to various resources, objects, and applications. The data classification of MAC security policy begins with sensitive, secret, and confidential, and next the classification of resources that will be making requests for data. MAC concept is incorporated mostly in military and governmental applications where high level security is required.

The benefit of this model is that the rules are hard coded into software so there are very less chances of an administrative error or social engineering.

The shortcoming of this model is that the rules are hard coded so it takes time to review and modify the rules as the requirements are changed. MAC is best suited for a group of users with similar needs.

1.1.2 - Discretionary Access Control (DAC)

Discretionary Access Control can be utilized as a centralized and distributed model. DAC centralized model is administered by an administrator or a team of administrators, who are responsible to make security policies and assign privileges as per policy, but this approach is time consuming, especially if the administrator is off or outsourced. In distributed approach, the data access is distributed to some responsible personnel such as managers, supervisors, or team.

This approach provides a way to avoid delays as the administration of accounts is dispersed.

The shortcoming of this model is that the uniformity of data access for end-user with same job functions can be diminished as access of data is distributed at the discretion of the owner.

1.1.3 - Role-based Access Control (RBAC)

In this competitive environment, the risk of losing information is more for leading organization. MAC and DAC model secure data, but they have limitations. To overcome their shortcomings, RBAC has been proposed [24]. As Role Based Access controls are in existence in last 20 years, especially in UNIX and mainframe environments, but they lack some standards as each system use its own propriety elements. There was a need to design such a system which is standardized, scalable, logical in design, and non-system dependent.

National Institute of Standards and Technology (NIST) started a project for unified standards of RBAC by integrating the existing model.

In 1992 a model was introduced by David Ferraiolo and Rick Kuhn that attempted to meet the requirements of the scope and created a full-fledged RBAC solution.

RBAC0 is the first proposed method in this series, this model consist of separation of duties and providing minimum privileges to each role. It doesn't have the hierarchy mechanism so the permissions were assigned directly to the users within a certain role or function.

By considering the need of hierarchy as it exists in any organization such as Manager, Supervisor, and team members RBAC1 is introduced based on RBAC0. It provides a natural distribution of responsibilities within an organization that is usually layered as senior and junior roles. This layered security distribution method is suitable for large environments.

Constraints are introduced in RBAC2 which offer more control over any network in large environments. Constraints help to enforce the policies while not having the hierarchy. Constraints work as limiters and ensure that the policies are being enforced. For instance,

if an organization wants to give administrative rights to one user or role, the constraints ensures that only one user has the system administration rights.

Another purpose of constraints is to regulate the access by ensuring certain criteria is met. For instance, to gain the permissions of Senior Analyst role one must have the memberships of Junior Analyst role.

Finally, constraints serve the purpose of separation of duties by limiting the users in a certain domain.

RBAC3 is complete model of RBAC, and it consists of both hierarchy and constraints. Figure 1.1 represents RBAC architecture.

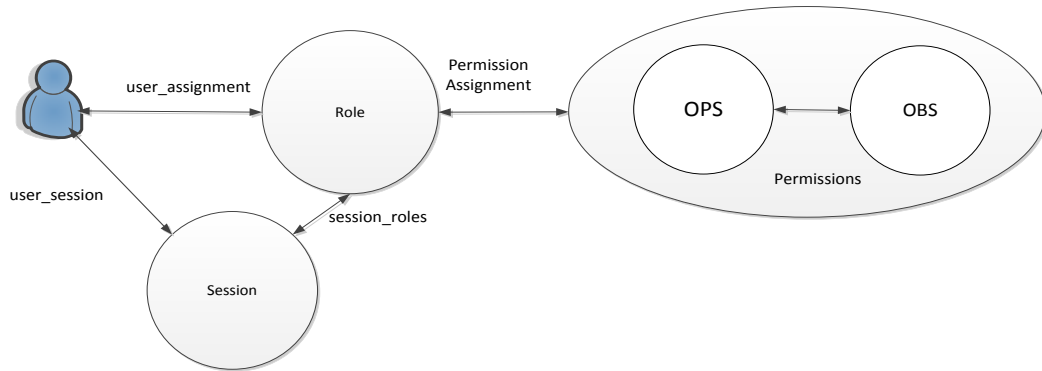


Fig 1.1: RBAC Architecture (NIST solution)

In RBAC3 constraints are used to regulate access on hierarchal structure. For instance, a programmer role is associated with a senior programmer role, and there are several senior programmers but a programmer role is associated only a specific senior programmer. As RBAC3 supports a hierarchal design, it provides ease of administration by allowing rights to flow down to subordinate objects. Figure 1.2 represents RBAC Hierarchal Structure.

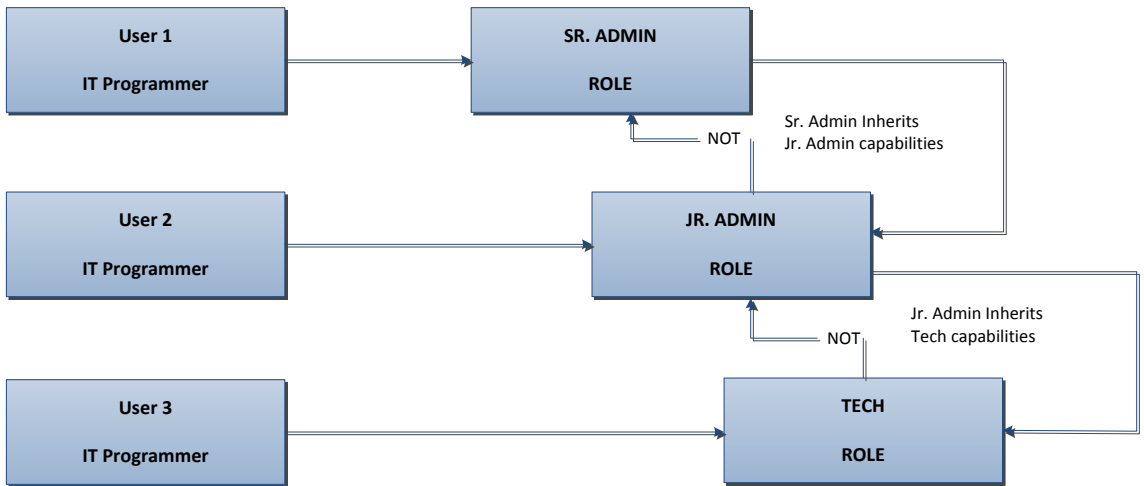


Fig 1.2: RBAC Hierarchical Structure (NIST solution)

Another benefit is the multiple roles which are associated with each other and provide better functionality to the users. RBAC3 constraints enforce separation of duties; it restricts the users to perform other tasks which are not in his job function. Figure 1.3 represents RBAC Separation of Duties (SoD).

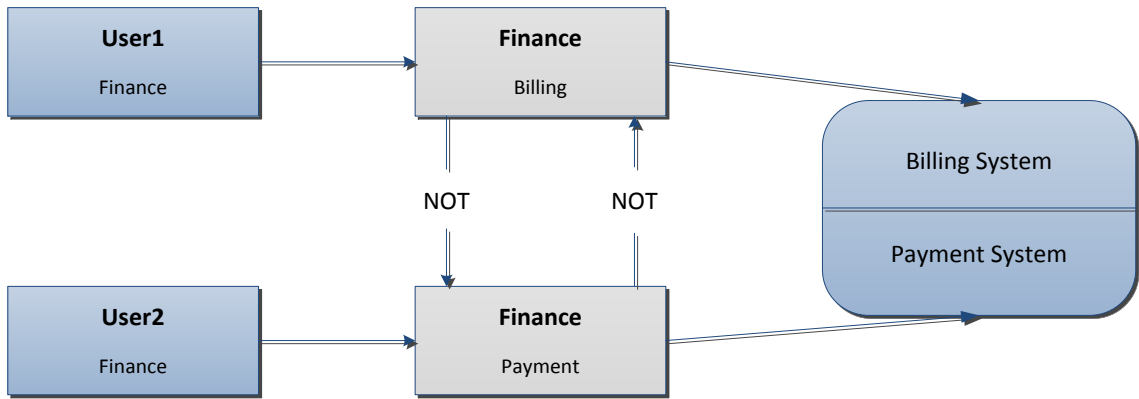


Fig 1.3: RBAC Separation of Duties (NIST solution)

In Figure 1.3, 'NOT' represents that both the roles cannot be activated at the same time by the same user.

RBAC3 is a complex method, but it is suitable for large environments where implementation and maintenance of other security models is time consuming with high cost.

RBAC3 provides all functionalities that a large organization requires, but implementation is complex due to its structure. Before implementing RBAC3, the organizational structure or roles must be well documented, otherwise, it can turn into a nightmare for an organization.

RBAC3 has five elements as represented in Figure 1.4; they are related to each other in order to create level of permissions and constraints.

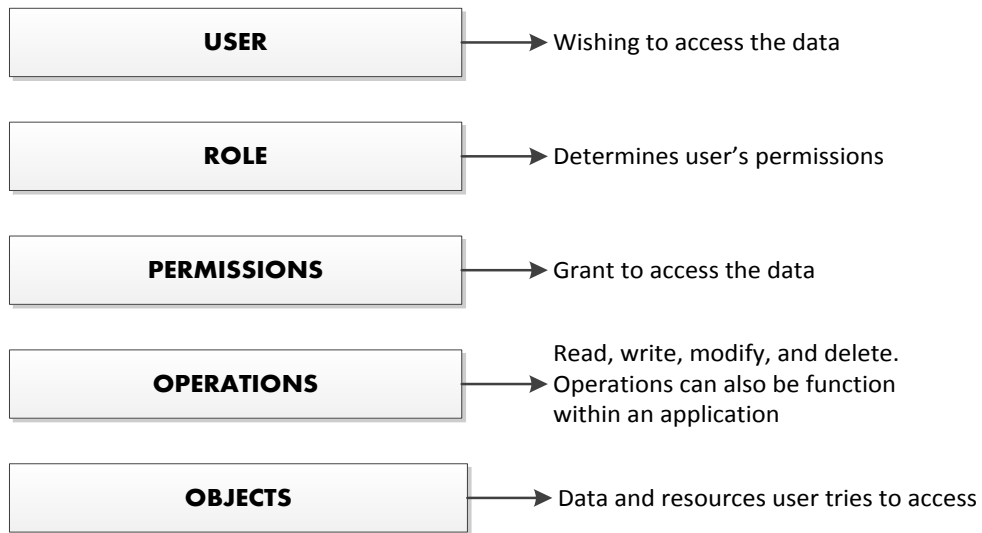


Fig 1.4: RBAC Elements (NIST solution)

1.2- Materialized View

Materialized view is a replica of a table or tables which is created in a distributed environment where master table is located in a main database. Materialized views are created and deployed at remote locations in a distributed environment in order to ease the network load, and provide uninterrupted data extraction as it doesn't require a dedicated network connection to the main database server.

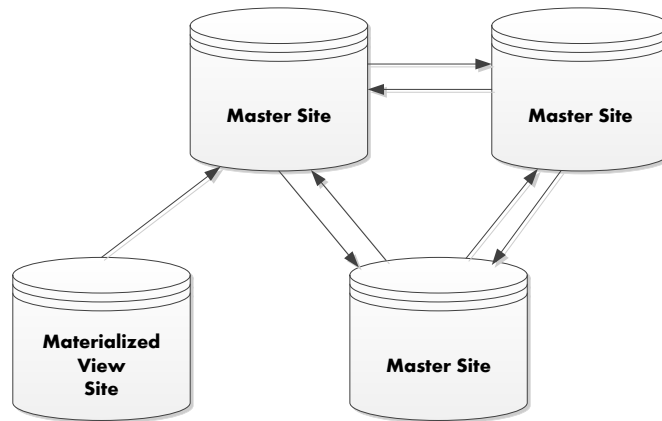


Fig 1.5: Materialized view site architecture <http://docs.oracle.com>

Materialized views can increase query execution performance in the following ways:

- Pre-computed aggregations can be created to minimize expensive computations during query executions.
- Tables can be pre-joined and resulting data can be stored in a materialized view.
- Combinations of joins and aggregations can be stored for analysis purpose.

Application that benefits from the creation/deployment of the materialized views:

- Decision Support Systems
- Data Marts
- Data Warehouses
- Online Analytical Processing (OLAP)
- Data Mining Workloads

A view is a temporary table that is created during run time, and discarded when the session is closed. A materialized view is a physical table that is stored in a database such as the traditional database tables. The creation process of the materialized view is different in each Database Management System (DBMS).

Oracle supports read only, updatable, and writable materialized views [27]. Users cannot manipulate read only materialized views, but they can perform actions on writable and updatable materialized views.

Oracle offers several different types of materialized views in order to meet the different replication situation.

- a) Primary Key Materialized Views
- b) Object Materialized Views
- c) ROWID Materialized Views
- d) Complex Materialized Views

a. Primary Key Materialized Views

Primary Key Materialized Views are the default type materialized views. They are updatable and created as part of a materialized view group. The updatable materialized views must reside in a different database instead of the master replication group. Changes are occurred at row level as identified by the primary key value of the row.

b. Object Materialized Views

Object Materialized View is based on the object table and created using *OF* type clause. The structure is same as the object table and composed of row objects, and each row object is identified by the object identifier (OID) column.

c. ROWID Materialized Views

A ROWID materialized view is based on the physical row identifiers (rowids) of the rows in a master table. They are used with Oracle7 database.

d. Complex Materialized Views

There are certain restrictions which need to be observed during the creation of a materialized view, and if these restrictions are not followed then a materialized view is considered a complex materialized view, and it cannot be fast refreshed.

In Microsoft SQL Server, materialized views are called indexed views, and the creation of materialized views is available in tutorials.

In open source database systems such as MySQL that doesn't provide materialized view by itself. But it is easy to build materialized views manually or using some available APIs such as *flexviews* [28] which helps to create the materialized views and refresh them.

1.3 - Authorization View

Authorization view is a well-known database technique that provides a fine-grained access control [2][3][20]. It provides a way to restrict users to view only authorized data which is specified in the available authorization views for that particular user. Authorization views are logical tables that don't contain any records. The structure of these views consists of the table name and the number of columns which are mentioned during the authorized view creation process.

There are different approaches to define an authorization view. Motro [15] proposed a model in which database access is specified in terms of views. According to the authors, "a set of views is defined, and each user is granted permission to access one or more views". Users query the database and database system derives views of the request that are views of the views to which the user has access permission, and then presents the available views.

Motro [15] defines a view creation in following way:

```
view ELP (EMPLOYEE.NAME, EMPLOYEE.TITLE,  
          PROJECT.NUMBER, PROJECT.BUDGET)  
where EMPLOYEE.NAME = ASSIGNMENT.E_NAME  
       And PROJECT.NUMBER = ASSIGNMENT.P_NO  
       And PROJECT.BUDGET ≥ 250,000
```

Rizvi et al. [20] introduced parameterized authorization views. According to the authors, it is impractical to create and maintain the authorization views for each user when the database has thousands of users. Also, if there is a slight change in the authorization policy then a large number of views will be affected. Rizvi et al. [20] states that the

parameterized views are like the normal views, but there are some additional parameters such as *user-id*, *time* and *user-location* appearing in its definition.

Rizvi et al. [20] defines a view creation in the following way:

```
create authorization view Co-studentGrades as
select Grades.*
from Grades, Registered
where Registered.student-id = $user-id
and Grades.course-id = Registered.course-id
```

Bahloul et al. [3] proposed an inference-based approach in order to control the access of Materialized Views. According to Bahloul et al. [3], their approach “facilitates the administration of access control rules to ensure the confidentiality of data at the level of materialized views”. The authors use authorization views to provide a fine-grained access control over the materialized views. In this approach, the authors propose the use of *Datalog* as a formal framework for expressing the access control rules. Datalog is a declarative programming language that is syntactically a subset of Prolog. It is used as a query language for deductive databases. Datalog is more expressive than SQL; it can perform multi-database queries with a cleaner syntax, and it facilitates re-use of SQL code snippets for frequent joins and formulas. In recent years, Datalog is used for data integration, data extraction, cloud computing, data analysis, and security.

The authors [3] assume the existence of three types of symbols: variables, constants and predicate names. $p(t_1 \dots t_n)$ is a literal where p is a predicate name with arity n and each t_i for $1 \leq i \leq n$ is either a constant or a variable.

According to the authors, the logical sentence associated with the Datalog rule

$P(u) \leftarrow q_1(u_1), \dots, q_n(u_n)$ is:

$\forall x_1 \dots x_n (p(u) \leftarrow q_1(u_1) \wedge \dots \wedge q_n(u_n)).$

Bahloul et al. [3] states that the relations defined by deductive rules are called intentional relations. For example,

```
Info-Doc (Id-D, Dname, Dfname, Dspeciality) ←  
doctor (Id-D, Dname, Dfname, Dadr, Dphone, Dspeciality,  
Dsalary)
```

is a rule that defines the (intentional) Info-Doc relation in terms of the (extensional) doctor relation.

In the above example, the user is allowed to view *doctor ID, last-name, first-name, and specialty* from the available materialized view.

1.4 - Query Rewriting

The problem of answering queries using the views has recently received a significant attention due to data management problems. In order to solve such problems, query rewriting algorithms has been introduced. These algorithms rewrite the user query using the defined authorization views which are associated with traditional database tables or materialized views. A query is rewritten to take best advantage of summaries, joins or aggregations of base table that are found in materialized views

There are several proposed query rewriting algorithms, we discuss Bucket algorithm [18] to understand the query rewriting process in the following paragraph.

Bucket algorithm creates subgoal g in the original query, and then each subgoal g contains the views that include subgoals to which g can be mapped in a rewriting of the query. In second step, the algorithm consider conjunctive query rewriting, each consists of one conjunct from every bucket. The algorithm further checks whether it is contained or can be made to be contained with join predicates in the query. The result of the Bucket algorithm is the result of conjunctive query rewritings.

Example:

```
create authorization view MyGrades as  
select * from Grades where student-id = $user-id
```

Let q be the query posed by the user.

q : select avg(grade) from Grades

The system-modified query

q' : select avg(grade) from MyGrades

1.5 - Problem Statement

In distributed environment, materialized views can be used to replicate data at distributed sites. Materialized views provide local access to data instead of accessing data from remote sites.

In large organizations, materialized view site are created at remote destinations in order to reduce the network load and the load from the main database server. A user query is sent to the local materialized view database instead of main database server. Existing proposed techniques provide rule-based access control for materialized views; however, the administration of such systems is time consuming and cumbersome in a large environment where administrators define rules for each user to control the access to materialized views.

In order to control the access of materialized view, we need a framework which provides a fine-grained access control to materialized view using the authorization view [3], and provides a *Role-Based* access instead of defining policies for each user. In our thesis research, we identified the problem of assigning authorization views to roles instead of defining for each user. We also noticed that if our framework allows the users to activate multiple roles in same session with different permissions, then we need to provide a strategy to resolve role conflicts that occur if a user is gaining authorization for permissions associated with conflicting roles at the same time. In our thesis research, we have identified the following problems:

- How can we assign the authorization views to roles?
- How can we impose a column level restriction on a materialized view based on rules associated with roles?

- How can we avoid role conflicts if user is gaining authorization for permissions associated with conflicting roles?

1.6 - Thesis Contribution

My contributions in this thesis are:

- A fine-grained access control framework to prevent unauthorized access of materialized views
- Users are assigned to roles, and users acquire permissions through authorization views
- Provides a column level restriction based on an assigned role
- Enforces constraints to avoid role conflicts
- The proposed framework can be implemented for materialized view sites in distributed environments and for data warehouses

1.7- Organization of thesis

The rest of the thesis is organized as follows. Chapter II provides the background literature review of different proposed models which extend Role-Based Access Control approach and data authorization techniques using authorization views. The details of this thesis research are provided in Chapter III that includes proposed framework, workflows diagrams, and the steps that each module performs in the entire role-based data authorization process. The details of implementation and verification of proposed framework along with the final results are provided in Chapter IV. Finally, Chapter V concludes our contribution and provides recommendations for future work.

CHAPTER II

RELATED WORK

In this chapter, we will discuss about the approaches and models proposed to control the access of data. In first three sections, we discuss some existing access control models which are the extension of Role-Based Access Control (RBAC). In next three sections, we discuss the proposed techniques to build authorization views at the database level instead of specifying the authorization policies in the application code, which has numerous drawbacks. In the last section, we discuss recent papers [2][3] which introduce *Datalog* based syntax to build the authorization views to control the access of materialized views.

2.1 - Context Based Access Control (CBAC)

RBAC grants access on the basis of the role regardless of the context of the request. In CBAC the request is granted by verifying the context of the request.

Toninelli et al. [22] propose a “Semantic Context-Aware Access Control Policy Model”. The authors state that they adopt a resource-centric approach to context modeling; contexts are associated with resources to be controlled and represent only those conditions that enable access to the resources. According to the authors, access control policies define for each context how to operate on the associated resource. The authors describe that the context consist of characterization information which is considered relevant for access control such as load, the entity operation on the resource, roles, identities and security credentials, and surrounding environment conditions, such as time, location, and other available resources.

Kulkarni et al. [13] introduce a Context-aware RBAC (CRBAC) model that is an extension of RBAC. According to the authors, in NIST RBAC model roles are assigned by the administrator, and in such systems roles have a long life time. In CRBAC model, roles are defined as part of application’s design, and roles come to in existence only when

the application is deployed and executed. A person's context is defined in terms of his/her current physical location, devices being used, network on which the devices are connected, and the activities in which the user is currently engaged.

Feng et al. [8] propose TCAC “A Trust and Context based Access Control Model for open and distributed systems”. According to the authors, role assignment is based on the trustworthiness and context information of the requestors. The authors state that if the trust value is not less than the set threshold value defined by the system policies, and the user context information satisfies the context constraints, the user is assigned a role and can perform the operations associated with assigned role. The authors state that context constraints which are considered in their model such as time and location of the user.

2.2 - Attribute Based Access Control (ABAC)

In ABAC, data access is granted on the basis of three attributes: Subject Attributes, Resource Attributes, and Environment Attributes.

Cruz et al. [5] propose “A Location Aware Role and Attribute Based Access Control System” by extending the role-based access control (RBAC) model for the dynamic association of roles with users. The authors state that in their framework privileges associated with resources are assigned depending on the attribute values of the resources, attribute values associated with the users determine the association of users with privileges, and a location mapping function between physical and logical locations allows to enable/disable roles depending on the logical location of the users and thus preserve the privacy of the location. The authors define their Access Model in which constraints can be defined on the attribute values of resources or users.

Finin et al. [7] introduce a model *ROWLBAC* “Representing Role Based Access Control in OWL”. The authors state that their work defines the relationship between Web Ontology Language (OWL) and the Role Based Access Control (RBAC). In addition, they further examine and assess OWL’s suitability for two other access control problems: supporting attribute based access control and performing security analysis in a trust-management framework. According to authors, in their approach role hierarchies are

represented by OWL class hierarchies, member get more privileges as one moves up the hierarchy, while in class hierarchies, classes get more attributes as user move down. The authors state that access constraints are based on general attributes of an action, including constraints on its subject and object. The authors further state that this provides general support to a more general model of attribute-based access control.

2.3 - XML Based Access Control

EXtensible Access Control Markup Language (XACML) is one of the well know XML formats for Access control. XACML is a general purpose access control policy language, and it provides syntax to enforce access control policies which help in managing authorization decisions. The policies defined by XACML decide whether to authorize access to data and at what extend. In addition, it provides an architecture that supports the services with the help of two enforcement entities or modules: Policy Enforcement Point (PEP) and Policy Decision Point (PDP).

XACML architecture consists of four components: The Policy Administration Point (PAP), Policy Decision Point (PDP), Policy Enforcement Point (PEP), and Policy Information Point (PIP). The PAP creates polices evaluated by the PDP. The PDP evaluates the policies against the incoming requests and sends the results to PEP. The PEP performs access controls on the basis of the authorization decision provided by the PDP. Finally, the PIP provides attributes values provided by the PDP during the policy evaluation process. Component that receives and dispatches all the information between these components is called Context Handler that performs as a mediator. Figure 2.1 represents XACML data flow.

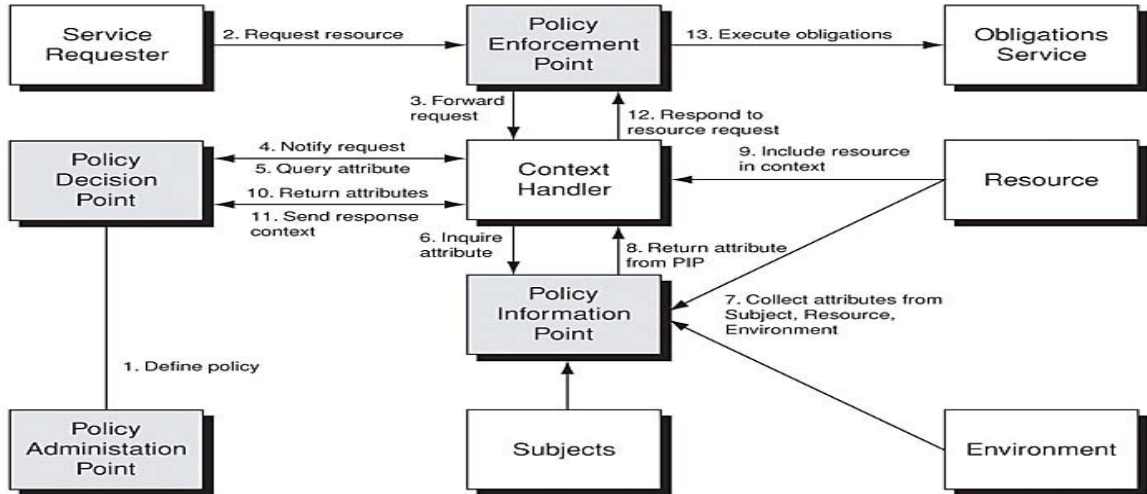


Fig 2.1: XACML data flow (<http://www.informit.com>)

Joshi et al. [10] propose a framework X-RBAC “an XML-based access control policy specification language” that extends NIST RBAC, and it provides a framework for specifying mediation policies in a multi-domain environment and extends RBAC with temporal constraints, role attributes, contextual conditions, a notion of role states, and preconditions of state transitions. According to the authors, X-RBAC provides a wide range of protection granularity for protected data and supports policy mapping in multi-domain environments. Figure 2.2 represents X-RBAC model.

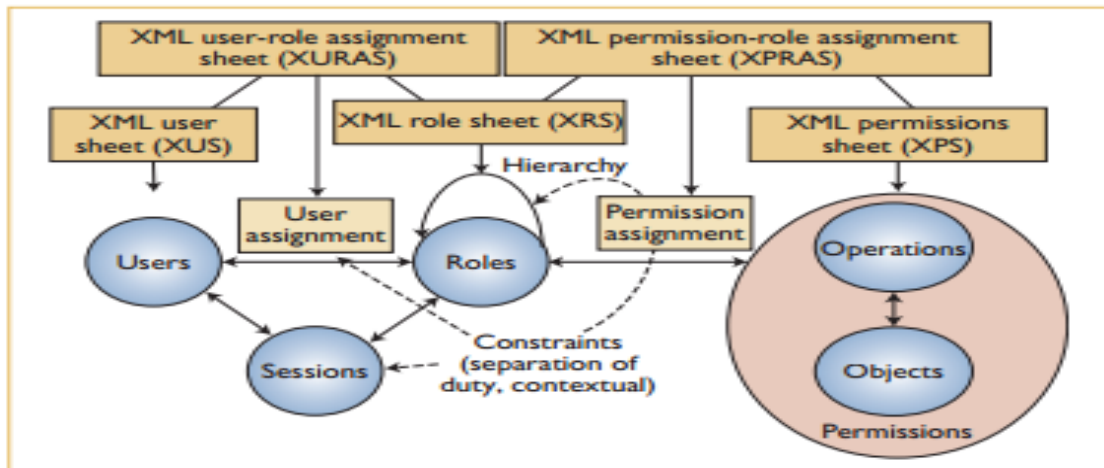


Fig 2.2: RBAC Model and X-RBAC Policy Components [Joshi et al. [10] page 41]

2.4 - Access Authorization for Relational Database

Motro [15] proposed a model for relational database. According to author, “in this model access permissions are a form of database knowledge, from which access permissions that apply to specific requests are inferred”. There are three basic principles of this model:

- 1) Database access is specified in terms of views: a set of authorization views are defined in order to control the access of data, and each user is granted permissions based on the available views.
- 2) A user queries sends to the actual database, not at any particular view.
- 3) When the request is sent to the database system, the system checks the available views of the request that could be the views of views which the user has access permissions.

According to the author, the model represents the definition of views in special “meta-relations”, and extends algebraic operators to these relations.

Example of the authorization process of the proposed model,

Assume a user sends a request to retrieve the names and sponsors of large projects:

```
retrieve (PROJECT.NUMBER, PROJECT.SPONSOR)
where PROJECT.BUDGET ≥ 250,000
```

Implementation of the above query with the following sequence of algebraic operations:

1. $A \leftarrow \sigma_{(BUDGET \geq 250,000)}(PROJECT)$
2. $A \leftarrow \Pi_{NUMBER, SPONSOR}(A)$

PROJECT' includes only tuples of views that Brown is authorized to access as defined in the following relation:

PROJECT'			
VIEW	NUMBER	SPONSOR	BUDGET
PSA	*	Acme*	*

Now the same operations that are applied to the database relations are applied to their meta-relation:

1. $A' \leftarrow \sigma_{(\text{BUDGET} \geq 250,000)}(\text{PROJECT}')$
2. $A' \leftarrow \Pi_{\text{NUMBER}, \text{SPONSOR}}(A')$

The selection retains only those view tuple which are unmodified, and the final projection is:

A'	
NUMBER	SPONSOR
*	Acme*

The above mask indicates that the user is restricted to projects sponsored by Acme, and following view definition will inform the user that permission exists only for SPONSOR = Acme.

```
permit (NUMBER, SPONSOR)
where SPONSOR = Acme
```

2.5 - Fine-grained Authorization Policies

Rizvi et al. [20] proposed a fine-grained access control model based on authorization views that allow “authorization-transparent” querying in which queries can be written against the database relation without referring to the authorization views. According to the authors, in their approach user queries can be written in terms of database relation, and the query is valid only if it can be answered using the available authorization views. The authors state that they have introduced a new notion of validity and conditional validity check in their proposed framework.

According to the authors, their model is based on six key features as follows:

1. Access control is specified using the authorization views. A view can be a traditional relational view or a parameterized view. According to the authors, a parameterized

authorization view is like the normal view, but there are some additional parameters such as *user-id*, *time* and *user-location* appearing in its definition.

```
create authorization view MyGrades as  
Select * from Grades where student-id = $user-id
```

2. Queries can be written in an authorization-transparent manner against the database relations without having referred to the authorization views.
3. A query q is unconditionally valid if there is an equivalent query q' and both the queries give the same result of all database states.
4. Certain queries can be answered using the available set of authorization views, even if they cannot be rewritten using the views.
5. Conditionally valid queries that can be answered using the information contained in a set of authorization views in a given database state.
6. Set of powerful inference rules which check the unconditional and conditional validity of queries.

2.6 - Authorization Views and Conditional Query Containment

Zhang et al. [25] proposed an algorithm that tests conditional containment of conjunctive queries respect to a set of materialized conjunctive views. According the authors, they identified the problem of \prod_2^p -complete. The authors state that based on their algorithm, they test if the query is conditionally authorized given a set of materialized authorization views.

According to the authors, they adopt the definition of Rizvi et al. [20]: a query is conditionally valid to a set of views V and a set of materialized views MV for all database states where the values of the views V agree with MV , q agrees with q' .

The authors define the conditionally contained query, conditionally empty query, and conditionally authorized query in the following definitions:

Definition 1 - For any two queries Q_1 and Q_2 ,

Q_1 is said to be conditionally contained in Q_2 with respect to V and MV ,

denoted by $Q_1 \subseteq_{V, MV} Q_2$, if for every d in D , $Q_1(d) \subseteq Q_2(d)$.

Q_1 is said to be conditionally equivalent to Q_2 with respect to V and MV ,

denoted by $Q_1 \equiv_{V, MV} Q_2$, if $Q_1 \subseteq_{V, MV} Q_2$ and $Q_2 \subseteq_{V, MV} Q_1$.

Definition 2 - A query Q is conditionally empty with respect to V and MV if $Q(d)$ is empty for every d in D .

Zhang et al. [25] states that their approach first initiates the parameterized views by extracting the parameter values associated with the user and the session, before declaring whether a query should be conditionally authorized

Definition 3 - A query Q is conditionally authorized with respect to authorization views V and materialized views MV , if there is a query Q_r , that is written using only the views in V , and that is conditionally equivalent to Q .

2.7 - Access Control to Materialized Views

In previous sections, we have discussed some of the data access control techniques which are being utilized by the organizations based on organizational requirements. Bahloul et al. [3] proposed an approach to control the access of materialized views. The authors identified the problem of automatically generating the access control rules for materialized views based on the access control rules defined over the base relations. The authors adopt the technique of authorization views in order to control the access of materialized views.

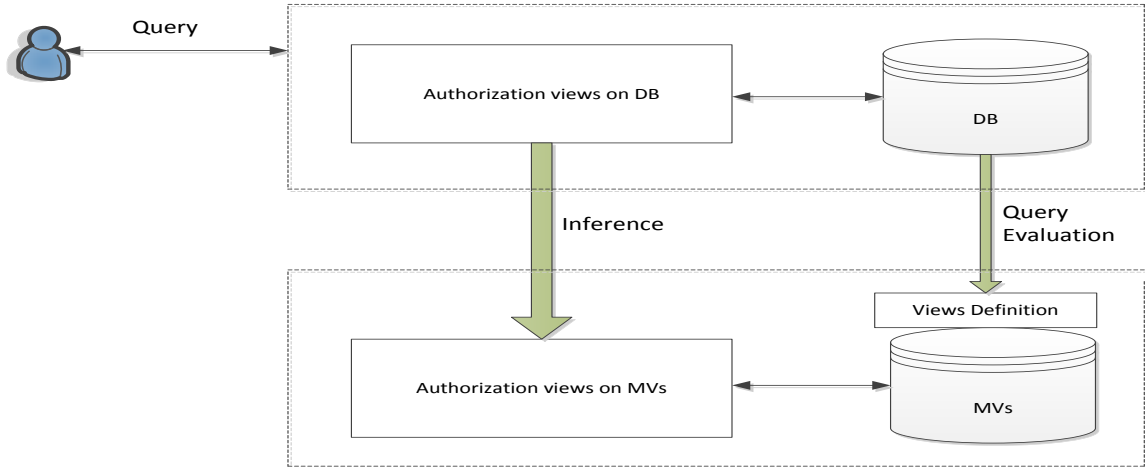


Fig 2.3: Authorization policies for materialized views (Bahloul et al. [2])

The authors propose the use of *Datalog* syntax for defining the access control rules. The authors assume the existence of three types of symbols: variables, constants and predicate names. $p(t_1, \dots, t_n)$ is a literal where p is a predicate name with arity n and each t_i for $1 \leq i \leq n$ is either a constant or a variable.

According to the authors, the logical sentence associated with the Datalog rule

$P(u) \leftarrow q_1(u_1), \dots, q_n(u_n)$ is:

$\forall x_1 \dots x_n (p(u) \leftarrow q_1(u_1) \wedge \dots \wedge q_n(u_n)).$

Bahloul et al. [3] states that the relations defined by deductive rules are called intentional relations. For example,

```
Info-Doc (Id-D, Dname, Dfname, Dspeciality) ←
doctor (Id-D, Dname, Dfname, Dadr, Dphone, Dspeciality,
Dsalary)
```

is a rule that defines the (intentional) Info-Doc relation in terms of the (extensional) doctor relation.

In the following example, the authorization views are defined to control the access of hospital database.

Hospital database:

doctor (*IdD, Dname, Dfname, Dadr, Dphone, Dspecialty, Dsalary*).

nurse (*IdI, Snum, Nname, Nfname, Nadr, Nphone, Nsalary*).

Authorization views:

av1 (*IdD, Dname, Dfname, Dspecialty*) \leftarrow *doctor* (*IdD, Dname, Dfname, Dadr, Dphone, Dspecialty, Dsalary*)

av2 (*IdI, Snum, Nname, Nfname*) \leftarrow *nurse* (*IdI, Snum, Nname, Nfname, Nadr, Nphone, Nsalary*)

In order to determine what data are accessible for each intentional relation *mv* in MV, the first step is authorization view selection. According to the authors, they utilize a query rewriting technique build on the Bucket algorithm that rewrites user query based on the available authorization views.

The authors state that they modified the original Bucket algorithm [18] as if they utilize the original algorithm then the authorization view will be considered irrelevant. For example, (1) shown below is a materialized view definition. The authorization view (2) defines the tuples *x, y* which a user has right to access.

$mv(x,y,z) \leftarrow r(x,y,z)$ (1)

$av(x,y) \leftarrow r(x,y,z)$ (2)

In the original Bucket algorithm, the attribute ‘*z*’ appears in the sub-goal of the query and also in the head of the query, then it must also be in the head of the view. According to the authors, it is too strict as one can project out ‘*z*’ by generating the appropriate authorization view on *mv*.

In 2012, they authors propose the use of MiniCon algorithm [18] for query rewriting with their previous approach. According to the authors, this algorithm is more efficient in terms of matching tuples with the set of available authorization views.

2.8 - Comparison of Various Research Works

Year	Author	Proposed work	Access control approach	Rule specification language	Domain of accessed data	Contribution	Implementation
2004	Shariq Rizvi	Extending Query Rewriting Techniques for Fine -Grained Access Control	Rule-based	SQL	Generic	Gives a powerful set of inference rules to check for query validity	Not addressed
2010	Alfredo Cuzzocrea	Effectively and Efficiently Selecting Access Control Rules on Materialized Views over Relational Databases	Rule-based	Datalog	Generic	Introduces Datalog based syntax for expressing rules, and VSP-Bucket algorithm for query rewriting	Not addressed
2011	Sarah Nait-Bahloul	Access Control to Materialized Views: an Inference-Based Approach	Rule-based	Datalog	Generic	Ensures confidentiality of materialized views based on basic access control rules	Not addressed
2012	Sarah Nait-Bahloul	Authorization Policies for Materialized Views	Rule-based	Datalog	Generic	Presents S-MiniCon algorithm, an adaptation of a query rewriting algorithm to the security context	Not addressed
2013	Hassaan Yousafi	A Role-Based Access Control Schema for Materialized Views	Role + Rule-based	Datalog	Generic	Presents a fine-grained access control model based on roles for materialized views	Open source technologies

Table 2.1: Comparison of various researches works

CHAPTER III

PROPOSED FRAMEWORK

In this chapter, we present the details of our proposed framework “A Role-Based Access Control Schema for Materialized Views”. We present an architecture that enables organizations to define and manage data access control authorizations for materialized views based on roles. This thesis introduces a role-based access control schema for materialized views in which rules are associated with roles, a column level restriction is imposed on a materialized view based on a user assigned role, and a role conflicting strategy is defined if the user is gaining authorization for permissions associated with conflicting roles. In this proposed framework, we focus on open source Database Management Systems such as MySQL and PostgreSQL as the open source databases don't provide any automated process to enforce authorizations on Materialized Views. The proposed framework can also be applied to other relation databases as authorization views are defined by keeping in view the structure of materialized view tables defined in relational databases.

3.1 - Data Access Control to Materialized Views

A materialized view records query results (or simply a view) into a physical table that can be stored, and the user can then query a materialized view in the same manner as querying a database. Materialized views can be used for performance reasons in very large systems such as data warehouses or distributed systems, or for providing a filtered selection of data from a more general database. Existing proposed techniques provide rule-based access control for materialized view, but to the best of our knowledge, the administration of such systems is time consuming and cumbersome in a large environment as administrators define rules for each user to control the access to materialized views.

In this thesis, we extend the earlier work proposed by Bahloul et al. [3] that defines authorization policies in *Datalog* as a formal framework for expressing the access control rules (Abiteboul et al. [1]). In this thesis research, we define the authorization policies for roles instead of defining these policies for individuals. In second chapter, we discuss the details of related works which defines data authorization policies at the database level such as parameterized authorization views (Rizvi et al. [20]), and we have also discussed the recent work of Bahloul et al. [3] that defines authorization views in *Datalog* syntax.

3.2 - Proposed Role-Based Framework

The proposed architecture presents a new approach to define data access control authorizations for Materialized Views. We have identified the administrative problems of defining and modifying the data access control rules for materialized views in large organizations where thousands of employees perform their duties. Existing proposed techniques provide rule-based access control for materialized views; however, the administration of such systems is time consuming and cumbersome in a large environment where administrators define rules for each user to control the access to materialized views.

In our approach, we utilize the existing Role-Based Access Control (RBAC) [24] model that is proposed by National Institute of Standards and Technology (NIST) in 1992. The RBAC model is currently the widely used model among all other existing models due to its dynamic nature and ease of administration.

In RBAC, the access to organizational resources is granted on the bases of the user assigned role. When a role is assigned to a user, all the privileges associated with that role are also granted to the user. In the first chapter, we have discussed RBAC in detail.

In our proposed framework, we define authorization views for each role. The authorizations are defined in *Datalog* syntax. The authorization views that are defined for materialized views provide column-level restrictions.

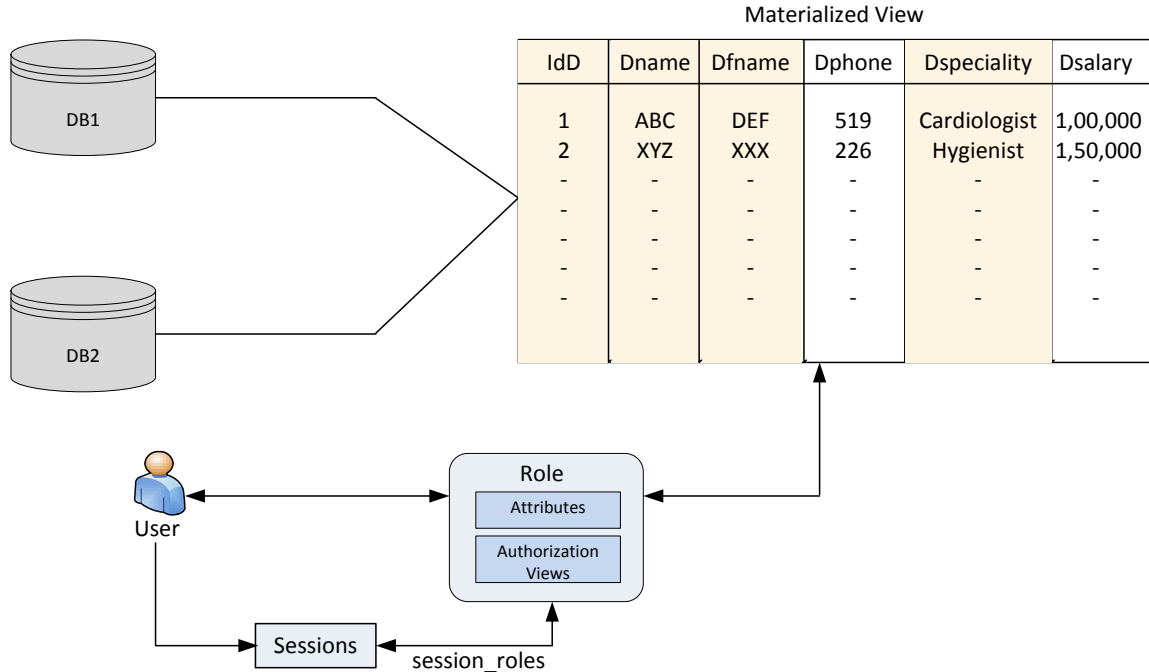


Fig 3.1: Proposed RBAC Architecture for Materialized Views

In figure 3.1, a materialized view is created by joining two tables from two different databases, and the view is deployed in a local database on a distributed site. As the proposed architecture depicts, the authorization views are associated with user assigned role, and the user profile attributes are extracted during the role assignment process. These attributes are extracted from database table where user profile attributes such as *username*, *department name*, *user role*, and other attributes are stored. The attributes are used for authentication and role assignment.

In our proposed framework, we also maintain the *session_roles*, as we have noticed during our thesis research that a user can be assigned multiple roles such as a Project Manager can also work as a Programmer same time. So keeping in view the multiple role assignment, we allow a user to open multiple *session_roles*. We provide the detail of the *session_roles* in section 3.2.3.

As we allow users to open multiple accounts in the same session, we also need to keep track of Role Conflict that occurs due to the conflict of interest between two roles. A user has to deactivate one session in order to activate another conflicting *session_role*. We discuss role conflicts in detail in section 3.2.4.

3.2.1 - Role Assignment

The Role Assignment process starts after the authentication process, if the user credentials such as *Username* and *Password* are valid then the process proceeds further. Figure 3.2 represents the work flow of role assignment process.

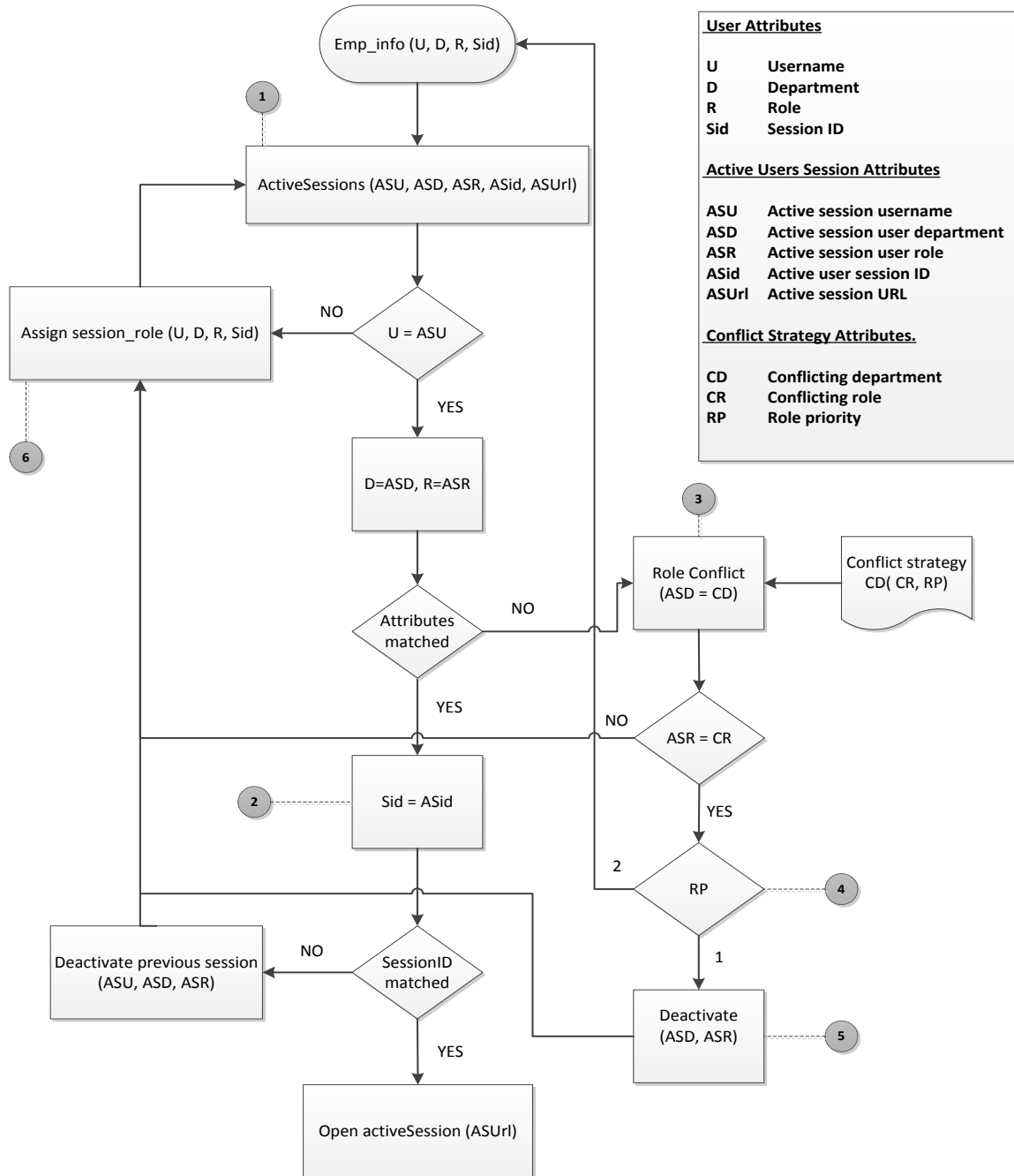


Fig 3.2: Workflow of Role Assignment process

Once the user credentials are verified then the process checks the already active sessions. Active Session is a table where all active user sessions' information is stored.

We describe the above workflow in the following six steps as represented in figure 3.2:

1. The process checks if the user information already exists in the *active_session* table. If the user information doesn't exist in the *active_session* table then the user information (i.e. username, department, and role) that is extracted from database based on user credentials (i.e. username, password) is sent to role assignment module where the role is assigned to the user. Otherwise, if the user session information exists then the user attributes are matched with the already existing user session attributes. If the attributes are matched with one of already active sessions then it is considered that the user is already an active user with the same role.
2. If the user is already an active user, then the user session ID is checked, if the session ID is matched with active role session ID then the user is redirected to the same session, otherwise, if the session ID is different, the process deactivates already active session and proceeds to the role assignment process.
3. If the user session attributes are not matched with already active user session except the *username* as described in step 1, then it is considered that the user is an active user with a different role. In this case, role conflict is checked by extracting the attributes defined in role conflict strategy with already active user sessions' attributes.
4. If the role conflict exists after analyzing the role conflict strategy then the role priority is further checked. Role priorities are defined in conflict strategies. If the priority is '1' then the already existing conflicting session is deactivated without notifying the user, and the new user proceeds to role assignment process. If the priority is '2' then a notification is sent with conflicting role information to the user that a role conflict exists, and in order to activate a new *session_role*, user must deactivate the conflicting *session_role*.

5. After deactivating the conflicting *session_role*, the user can proceed to role assignment process as represented in figure 3.2.
6. If the role conflict doesn't exist as described in step 4, then the user is assigned a role that is based on user credentials.

3.2.2 - Role-Based Authorizations

In our proposed framework, the authorization to access a materialized view is based on user role. Figure 3.3 represents Role-Based Authorization architecture.

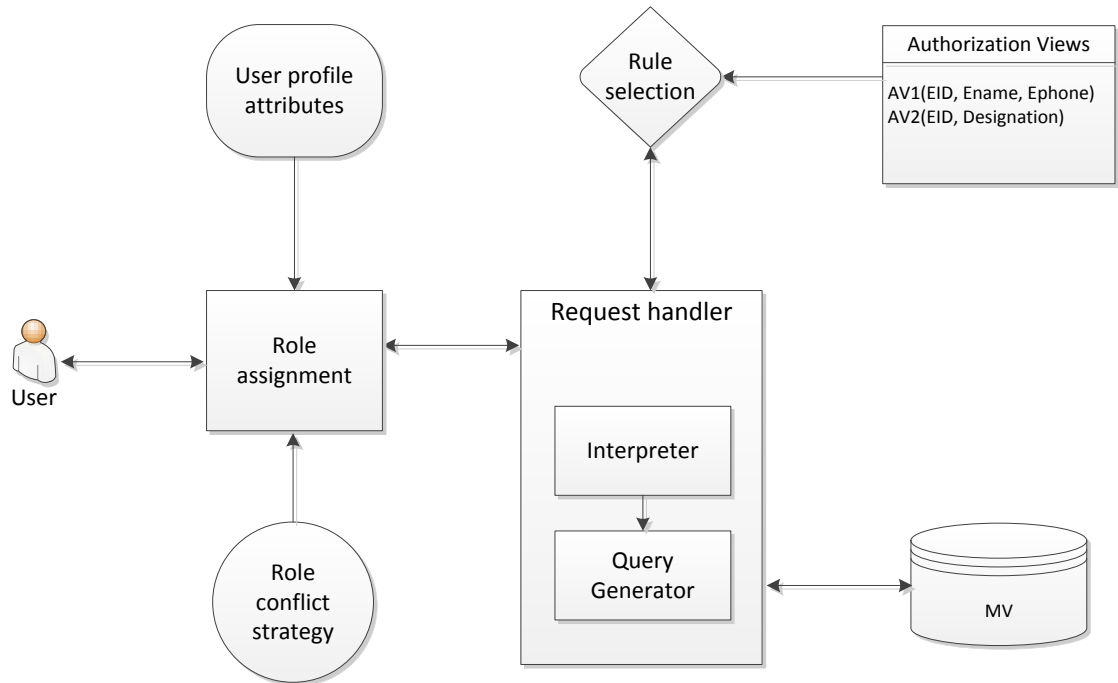


Fig 3.3: Architecture of Role-Based Authorization

The process starts with role assignment, user attributes are extracted based on user credentials which user provides during the login process. The following example describes the above architecture.

Example:

Once a user is assigned a role after going through the Role Assignment process as discussed in previous section, the user query is sent to the Request Handler.

The Request Handler sends the materialized view name as mentioned in the user query to the Rule Selection module.

The Rule Selection module searches for the given materialize view name in the associated authorization views file of user assigned role. If the materialized view name is found in the list of authorized views, the Rule Selection process extracts the authorized view information and returns it to the Request Handler.

The Request Handler sends the authorized view to the Interpreter.

The Interpreter extracts the column names from the given authorized view and sends it to the Query Generator.

The Query Generator module generates the query on the basis of the provided column names and materialized view name, and returns it to the Request Handler.

The Request Handler further sends the query to database system and returns the results to user.

We use an existing approach to define authorization views proposed by Bahloul et al. [3]. In the following example, the authors define the authorization views to control the access of hospital database.

Example:

Hospital database:

doctor (*IdD, Dname, Dfname, Dadr, Dphone, Dspecialty, Dsalary*).

nurse (*IdI, Snum, Nname, Nfname, Nadr, Nphone, Nsalary*).

Authorization views:

$av1 (IdD, Dname, Dfname, Dspecialty) \leftarrow \mathbf{doctor} (IdD, Dname, Dfname, Dadr, Dphone, Dspecialty, Dsalary)$

$av2 (IdI, Snum, Nname, Nfname) \leftarrow \mathbf{nurse} (IdI, Snum, Nname, Nfname, Nadr, Nphone, Nsalary)$

In the above example, there are two parts of authorization view. The left hand side defines the body of an authorization view, and the right hand side defines a complete definition of a materialized view from which an authorization view is derived. The body of an authorization view is used to define the authorizations for a particular user. In our proposed framework, we define the authorization views for roles as represented in figure 3.4.

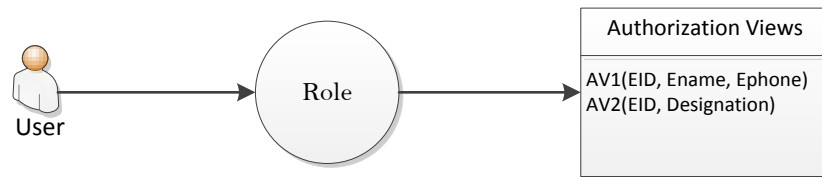


Fig 3.4: Role and Authorization Views

3.2.3 - Session Roles

In our proposed framework, we use Role-Based Access Control (RBAC) model proposed by NIST. RBAC is used by majority of organizations due to its dynamic nature and ease of administration. Three primary rules are defined for RBAC:

1. Role assignment: A user can access the resources only if he/she has selected or been assigned a role.
2. Role authorization: User active role must be authorized to the user.
3. Permission authorization: A user can access the resources only if the permission is authorized to the user's active role.

RBAC specifications:

- *User Assignment* = $UA \subseteq USERS \times ROLES$, a many-to-many mapping user to role assignment relation.

- *Permission Assignment* = $PA \subseteq PREMS \times ROLES$, a many-to-many mapping permission to role assignment relation.
- *session_users* ($s: SESSIONS$) $\rightarrow USERS$, the mapping of session s onto the corresponding user.
- *session_roles* ($s: SESSIONS$) $\rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles.

A user may open multiple simultaneous sessions with different roles and permissions. In our proposed framework, we follow the above rules and also allow user to open multiple simultaneous sessions. A user can open multiple sessions with different role, and each concurrent session authorizes a user to access the views based on an assigned role. Each session extracts user attributes (i.e. username, department, and role) that are passed through a generic URL to the Request Handler after Role Assignment. When a user sends a query, the Request Handler extracts the attributes and authorizes the user to view the records of a requested materialized view based on the available authorization view that is defined in authorized views file associated with user role within a particular department.

3.2.4 - Role Conflicts

In previous section, we discuss multiple simultaneous sessions which allow users to open multiple sessions with different roles and permissions. But we also need to restrict users to avoid role conflicts. Role conflict defines that no individual can assume the power of two or more conflicting roles at the same time. Role conflicts occur when individuals have various conflicting responsibilities

RBAC introduces Separation of Duties (SoD) that restricts users to perform duties in confliction roles at the same time by enforcing the constraints.

We take an example of finance system in which user has access to the billing system through a “Finance Billing” role, and he also has access to the payment system through the “Finance Paying” role. In this case, user cannot activate both the roles at the same time. User has to deactivate one session in order to activate the other session.

In our proposed framework, we introduce Conflict Strategies which are associated with user role same as the authorization views to avoid any role conflict. In our proposed conflict strategy, we introduce role priority.

If a role conflict occurs during the role assignment process, the system checks the priority of the role that is defined in the conflict strategy by the administrator. We describe the conflict strategy as follows.

After identifying the role conflict between two roles, the conflict strategy is defined in the following manner.

Department (Role, Priority)

The conflict strategy is defined in a text file associated with each role. During the Role Assignment process as we discuss in previous section. The Request Handler checks already activated *session_roles* of the same user with different roles. It extracts the attributes of the active *session_roles* one by one, and matches with the attributes defined in conflict strategy (i.e. department, role). If the attributes are matched then it is considered a role conflict. In this case, the Request Handler checks for the role priority defined in the conflict strategy. If the priority is '1' then the system activates the new *session_role* and deactivates the previous session. If the priority is '2' then the system sends a notification with the conflicting role attributes, and the user can activate the new *session_role* after deactivating the already active *session_role*.

In the following example, we define conflict strategies for Finance Billing and Finance Paying roles which belong to Finance department.

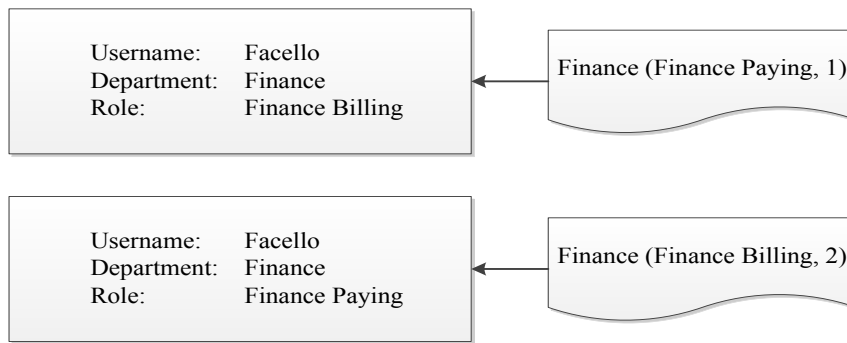


Fig 3.5: Role Conflict Strategies

In figure 3.5, user *Facello* is assigned two roles, but he cannot activate both roles at the same time. The role priorities are defined in the conflict strategy associated with both the roles. In the above scenario, the Finance Billing role is assigned first priority and Finance Paying role is assigned second priority.

CHAPTER IV

IMPLEMENTATION AND VERIFICATION

4.1 - Background

The main objective of this thesis research is to provide architecture to control the access of Materialized Views based on user roles. The case study is built on top of research done at the University of Windsor (Kent et al. [11] and Kobi et al. [12]) towards the creation of automated tools to conduct healthcare surveys, decision support system, and real-time data management system. In this research work, we identified the need of securing sensitive healthcare data. In our proposed framework, we present a Role-Based Access Control architecture in which authorizations are associated with user roles. Moreover, we introduce a mechanism to resolve role conflicts by defining role priorities in role conflict strategies which are also associated with user role same as authorization views.

4.2 - Implementation

In previous chapter, we discuss the steps involved in authentication, role assignment, and user authorizations in our proposed framework. In this section, we provide the details of implementation of our proposed framework.

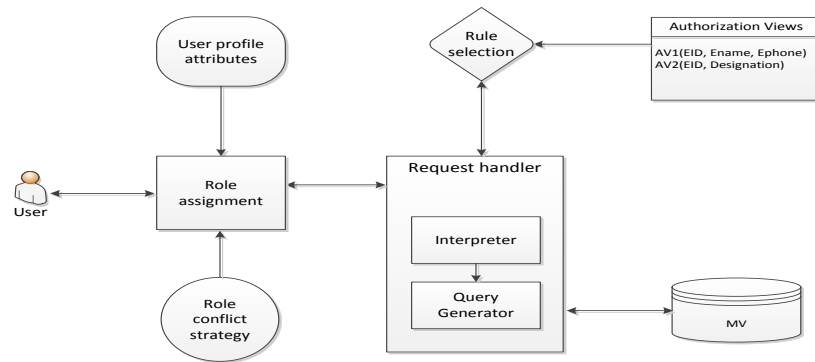


Fig 4.1: Proposed Architecture

In the proposed framework, we focus on open source Database Management Systems (DBMS) as the open source DBMS doesn't provide any automated process to regulate the access of Materialized Views. In the implementation of our proposed framework, we use MySQL database, the internal structure of our application is developed in PHP, and the User Interface (UI) is developed in HTML and CSS.

4.2.1 - Role Assignment Module

Role Assignment is a process to assign a role to the user based on the credentials that a user provides during the authentication process. Before a role is assigned, a user has to pass few checks in order to avoid any role conflicts and duplication of *session_roles*. In previous chapter, we explain a complete workflow of Role Assignment module. In this section we provide pseudo code of this module.

```

Role Assignment Module
CHECK if the user is already an active user
IF user exists in Actives_Sessions

    EXTRACT user Active_Sessions attributes

    IF user attributes match with any Active_Sessions
        CHECK for SessionID
        IF SessionID match with current user SessionID
            REDIRECT user to same Session_Role
        ELSE
            Deactivate ActiveSession role
            Activate new Session_Role

    BREAK

CHECK Role Conflict with existing Session_Roles in ActiveSessions

EXTRACT attributes of ActiveSessions
EXTRACT Conflict_Strategy of current role

IF any of ActiveSessions role match with role defined in Conflict_Strategy for current user
    CHECK for the role priority defined in Conflict_Strategy
    IF Priority is 1
        Deactivate ActiveSession role
        Activate current role
    IF Priority is 2
        Send an alert with conflicting role information

BREAK

```

Fig 4.2: Role Assignment module pseudo code

4.2.1.1 - Active Sessions

In Active Sessions table, we store user information (i.e. username, role, department, sessionID, and session URL) whenever a user login to the system. The information available in this table is extracted during the role assignment process in order to match user active roles and current role attributes to avoid any role conflict and duplication of *session_roles*. The session ID is stored in order to prevent a user to activate multiple sessions for same *session_role* from different system. A user cannot open a *session_role* from two different session IDs at the same time. If a user tries to activate a *session_role* which is already in active session list then the system deactivates the existing *session_role*, and creates a new entry in active sessions with new Session ID. In this way, we can track user's activities.

We extract user's active session's information by calling activeSession function.

Active Sessions

```
function activeSession (username) {  
    EXTRACT username, role, department, session_id, url from active_sessions  
    WHERE username == username  
    IF found  
        Store user ActiveSessions attributes in an ARRAY  
        Return ARRAY  
    ELSE  
        Return NULL  
}
```

Fig 4.3: Active Sessions pseudo code

4.2.2 - Request Handler Module

The Request Handler processes all users' request once a user is assigned a role. It receives users' request and sends it to the View Selection module. Before processing a user request, the Request Handler verifies that the user still exists in Active Sessions. If the user is still an active user, the Request Handler confirms that the user Session ID matches with the existing Session ID in Active Sessions. If it so then it processes user's

request, otherwise, it deactivates user *session_role* and redirects the user to the login page. The Request Handler operates by the following steps:

1. The Request Handler receives a user query, extracts the requested Materialized View name from the query, and sends it to the View Selection module.
2. The View Selection module search for the requested Materialized View name in Authorization Views file that is associated with the user's role.
3. The View Selection module returns the Authorization View information to the Request Handler.
4. The Request Handler sends the authorized view information to the Interpreter.
5. The Interpreter extracts authorized column names from the authorization view and sends it to the Request Handler.
6. The Request Handler sends the view name and the authorized columns name to the Query Generator.
7. The Query Generator generates the query in SQL based on the provided information from the Request Handler, and sends the query to a database.
8. The Request Handler receives the records from the Query Generator, and returns it to the user.

4.2.2.1 - View Selection

View Selection is a process to search and select an Authorization View based on a user selected Materialized View. View Selection process starts when a request is sent to the View Selection module from the Request Handler that contains a requested Materialized View name. In order to link the Authorization Views file that is associated to each role, the View Selection modules receives user's attributes (i.e. department name and role) along with a Materialized View name from the Request Handler, and generates a path in order to link to Authorization Views file of user assigned role.

View Selection
<pre> function ViewSelection (MaterializedView Name, Department Name, Role) { Path to user Auhtorization Views file = "AuthorizationViews/department_name/role" LOOP Search for MaterializedView in defined Authorization Views IF found Store AuthorizationView body information in an ARRAY Return AuthorizationView ELSE Return NULL } </pre>

Fig 4.4: View Selection pseudo code

4.2.2.2 - Interpreter

The Interpreter translates an Authorization View in SQL syntax. The Request Handler sends authorization view information that is returned by the View Selection module to the Interpreter. The authorization view is received in the following format.

av1(emp_no, first_name, last_name)

The Interpreter removes the view name and the brackets that contain authorized column names from the given Authorization View, and returns the column names to the Request Handler in order to send it to the Query Generation module.

emp_no, first_name, last_name

4.2.2.3 - Query Generation

Query Generation is a process to generate a query in Sequential Query Language (SQL) syntax, and send it to the Request Handler in order to send the query to a database. The following architecture describes query construction process.

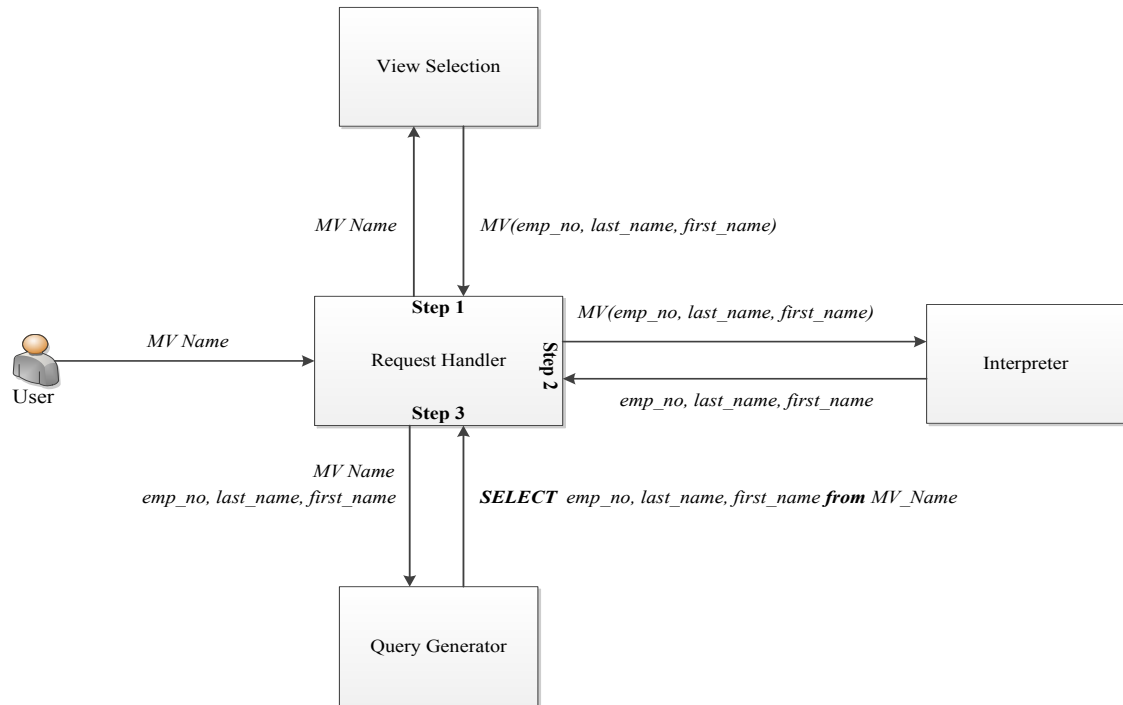


Fig 4.5: Query Construction

In figure 4.5, the Query Generation module receives the materialized view name and column names from the Request Handler. The materialized view name and column names are assigned to the dynamic variables which are declared in Request Handler. The Request Handler passes the variables to Query Generator in order to construct a complete query. SQL SELECT and FROM clause is statically defined in query syntax and the dynamic variables are placed for column names and a view name. There are some other clauses that are used in SQL SELECT statement such as WHERE, GROUP BY, ORDER BY, and LIMIT which can be defined by the user through the options available in the user interface. The values of these additional SELECT clauses are also assigned to dynamic variables which are declared in Request Handler, and the Request Handler passes these values to the Query Generator along with MV name and column names.

4.3 - Verification

In order to test our proposed framework, we have implemented an application based on our proposed architecture. The application is developed in accordance with the specification described in Chapter 3. We discuss the implementation part in previous

section where we define steps that each module performs. In this section, we examine our application that we have developed to test and verify our proposed approach.

4.3.1 - Basic Requirements

The verification of basic requirements is to test the core elements of the application. In our thesis research, we are focused on providing architecture to authorize users to access the data based on their role. The Role Assignment process starts after the authentication process. A user needs to provide a valid username and password for authentication. Figure 4.6 represents login page.

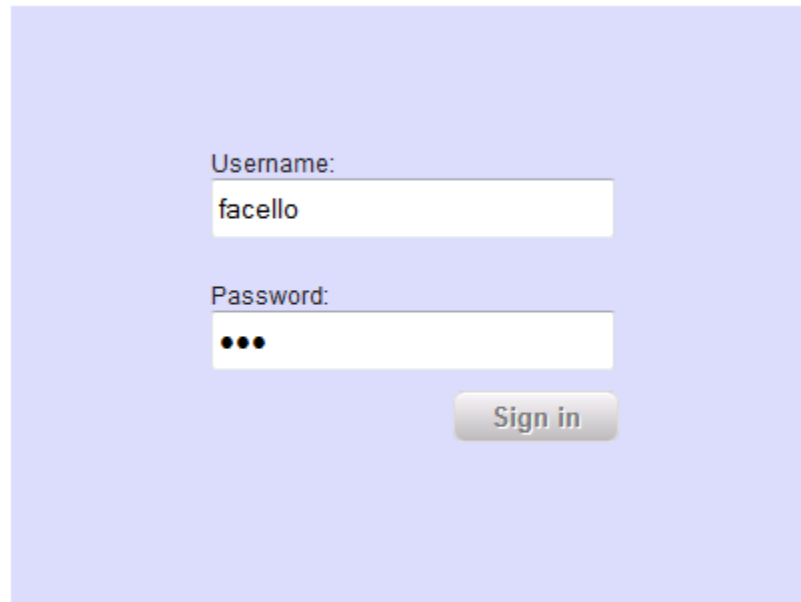


Fig 4.6: Login page

After the authentication process, the user is redirected to the homepage. The following image represents the homepage.



Fig 4.7: Homepage

Username, department, and the role of the user appear on the top right of the header section as in figure 4.7. There is a logout button available that deactivates user session and redirects user to the login page. The left panel down to the header is used to place Query Generator. A user can generate a query using the features available in Query Generator. There is a dropdown list available to select a materialized view. This dropdown list contains only those authorized views which are defined in authorization view file that is associated with user assigned role. There are some check boxes which are used to enable and disable additional SQL statement clauses, and 'Display None' feature is used for analysis purposes.

The basic requirement of this framework is to restrict the user to view only those columns of materialized view which the user has authorization. The following image represents a simple query execution without mentioning any additional clauses.

Facello | Finance | Finance Paying
[Logout](#)

Query Generator

Views: emp_info

WHERE
 GROUP BY
 ORDER BY
 Display None

Rows per page: 10

EMP_INFO			
EMP_NO	FIRST_NAME	LAST_NAME	GENDER
10001	Georgi	Facello	M
10002	Bezael	Simmel	F
10003	Parto	Bamford	M
10004	Chirstian	Koblick	M
10005	Kyoichi	Maliniak	M
10006	Anneke	Preusig	F
10007	Tzvetan	Zielinski	F
10008	Saniya	Kalloufi	M
10009	Sumant	Peac	F
10010	Duangkaew	Piveteau	F

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | > [30003]

Fig 4.8: Simple query execution and results

In figure 4.8, user *Facello* selects a view *emp_info* from the dropdown list of views and executes the query without adding any additional clauses. The system returns only those columns information which the user authorized to view as defined for users’ assigned role in its associated authorization views file as below.

```

1 departments(dept_no, dept_name) <- department(dept_no, dept_name)
2 dept_emp(emp_no,dept no) <- dept_emp(emp no, dept no, from date, to date)
3 emp_info(emp_no,first_name,last_name,gender) <- emp_info(emp_no,birth_date,first_name,last_name,gender,hire_date)

```

Fig 4.9: Authorization Views

In figure 4.9, the complete view definition is defined at the right side that contains six columns, but the user is not authorized to view the information of two columns (i.e. *birth_date* and *hire_date*) as defined at the left side. In Fig 4.8, the system returns the results based on the defined authorization view.

In our proposed framework, we introduce role conflict strategies. A user can activate multiple *session_roles* simultaneously as a user can be working in multiple roles in same organization. We define conflict strategy to avoid any conflict of interest between two

roles as the conflicting roles cannot be activated at the same time by same user. In our proposed framework, we assign priorities to conflicting roles. The conflict strategy is associated with roles same as the authorization views.

In chapter 3, we discuss role conflicts in detail with an example. In order to validate this requirement, we define role conflict strategy for ‘Finance Paying’ and ‘Finance Billing’ role in Finance department, and we assign both the roles to same user. The strategies are defined in following manner.

```
.ConflictStrategies\Finance\Finance Billing.txt 1 Finance (Finance Paying,1)  
.ConflictStrategies\Finance\Finance Paying.txt 1 Finance (Finance Billing,2)
```

Fig 4.10: Conflict Strategy

In figure 4.10, a conflict strategy is defined for ‘Finance Billing’ and ‘Finance Paying’ roles in their associated role files as both the roles cannot be activated at the same time. The ‘Finance Billing’ role is given first priority and ‘Finance Paying’ role is at second priority. The user activates ‘Finance Billing’ role and at the same time the user wants to activate ‘Finance Paying’ role. As the ‘Finance Billing’ role is given first priority, therefore, the user gets the following message during the role assignment process and redirected to the Login page as represented in figure 4.11.

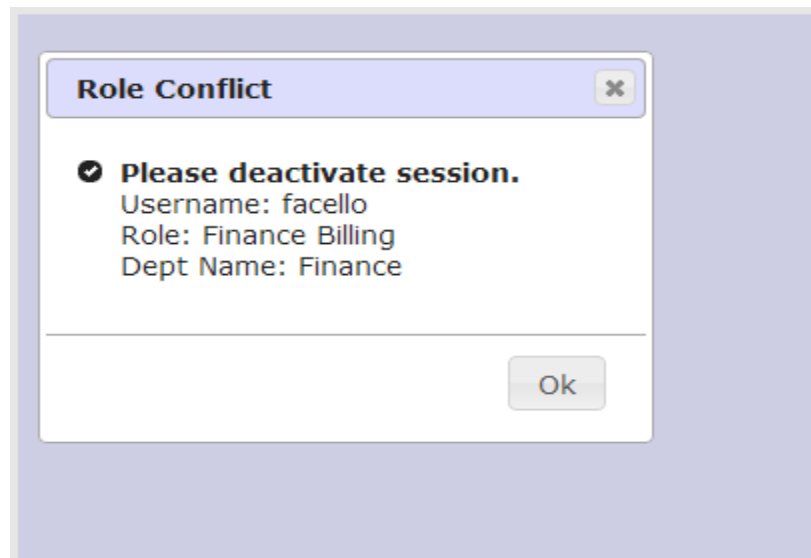
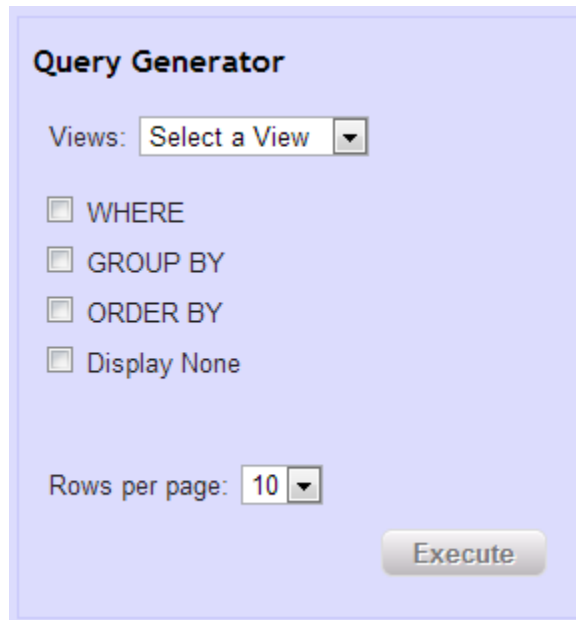


Fig 4.11: Role conflict detected

If the user activates 'Finance Paying' role first and at same time the user activates 'Finance Billing' role. In this case, the system deactivates 'Finance Paying' role without notifying to the user as the 'Finance Billing' role is given first priority.

4.3.2 - Additional Functions

There are some additional functions in our application. These functions enable users to filter the results coming from the database. The additional functions include the additional SQL statement clauses such as WHERE, GROUP BY, and ORDER BY. These features are available in Query Generator. The following figure represents the additional clauses available in the User Interface.



The image shows a user interface for a 'Query Generator'. It features a title 'Query Generator' at the top. Below the title, there is a 'Views:' label followed by a dropdown menu with the text 'Select a View'. Underneath, there are four checkboxes, each followed by a label: 'WHERE', 'GROUP BY', 'ORDER BY', and 'Display None'. At the bottom left, there is a 'Rows per page:' label followed by a dropdown menu showing the number '10'. At the bottom right, there is a button labeled 'Execute'.

Fig 4.12: Additional SQL statement clauses

In figure 4.12, there are additional SQL statement clauses which a user can enable to filter the results. A user can enable them by clicking on the checkboxes available in each clause. The following figure represents query generation using the additional clauses.

Query Generator

Views:

WHERE

GROUP BY

ORDER BY

Display None

Rows per page:

Fig 4.13: Query Generation using SQL statement clauses

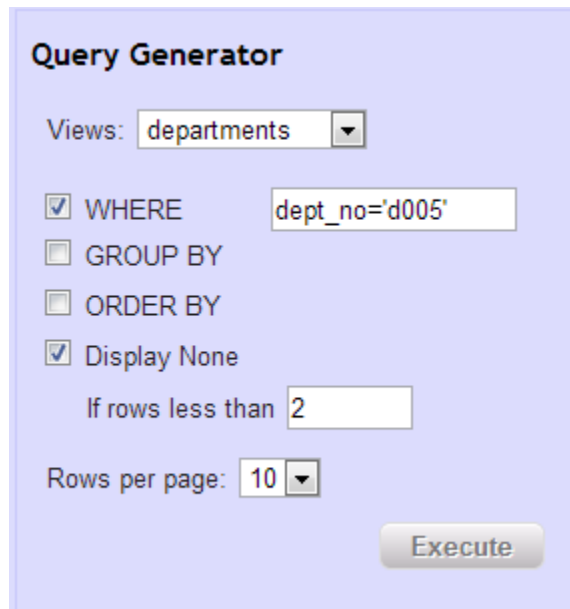
Figure 4.13 represents Query Generation using additional SQL statement clauses, the user executes the query and the Request Handler returns the results based on authorization view defined for user assigned role as represented in figure 4.14.

EMP_INFO

EMP_NO	FIRST_NAME	LAST_NAME	GENDER
10001	Georgi	Facello	M
10003	Parto	Bamford	M
10004	Chirstian	Koblick	M
10005	Kyoichi	Maliniak	M
10008	Saniya	Kalloufi	M
10012	Patricio	Bridgland	M
10013	Eberhardt	Terkki	M
10014	Berni	Genin	M
10015	Guoxiang	Nooteboom	M
10016	Kazuhito	Cappelletti	M

Fig 4.14: Results of query with additional SQL clauses

Apart from the additional SQL clauses, there is another additional features ‘Display None’ that is added in the User Interface for data analysis purpose. The Request Handler extracts the number records for a requested query, and if the number of records is less than the value defined in “Display None” field then the request handler returns a total number of rows.



The screenshot shows a 'Query Generator' window with a light blue background. At the top, it says 'Query Generator'. Below that, there is a 'Views:' dropdown menu with 'departments' selected. There are four checkboxes: 'WHERE' (checked), 'GROUP BY' (unchecked), 'ORDER BY' (unchecked), and 'Display None' (checked). The 'WHERE' clause is set to 'dept_no='d005''. Below the 'Display None' checkbox, there is a text input field with '2' and the label 'If rows less than'. At the bottom, there is a 'Rows per page:' dropdown menu with '10' selected and an 'Execute' button.

Fig 4.15: Query Generation using Display None function

In figure 4.15, the ‘Display None’ function is enabled and in “rows less than” field, a value ‘2’ is given. In this case, the Request Handler doesn’t return any results if the number of records is less than two. The following figure represents the results after executing the query as mentioned in Fig 4.15.



The screenshot shows the 'Query Generator' window from Figure 4.15, but with the 'Execute' button highlighted. To the right of the window, the text '1 record(s) available.' is displayed. The 'Query Generator' window itself is identical to Figure 4.15, showing the 'WHERE' clause 'dept_no='d005'', the 'Display None' checkbox checked, and the 'If rows less than' field set to '2'.

Fig 4.16: Results of query using Display None function

4.4 - Scalability Test Results

NU = Number of users

SARO = Number of Session Roles

SARU = Number of rules associated with roles

Load			RoleAssg time (secs)	RuleSelection time (secs)	QueryExecution time (secs)	QueryResponse time (secs)
NU	SARO	SARU				
15	21	234	0.95854	0.00126	1.59740	0.00050
30	40	468	0.98607	0.00163	1.59740	0.00036
45	57	520	1.07722	0.00140	1.50204	0.00035

Table 4.1: Scalability test results

4.5 - Summary Comments

In this chapter, we have discussed the implementation of our proposed framework in detail. We have also validated our approach and presented the results. The proposed framework is not domain specific. It can be adopted by any organization which has huge number of employees working at different remote locations in a distributed environment. The organizations create Materialized Views database and deploy such databases at remote sites in order to ease the network load, and also to reduce load of the main database server. There is no such automated mechanism available in open source databases which can be utilized to control the access of Materialized Views locally at the remote sites. The proposed framework can be utilized for local Materialized View sites in a distributed environment.

CHAPTER V

CONCLUSION AND FUTURE WORK

In this chapter, we conclude our proposed framework and discuss some areas for future work.

5.1 - Conclusion

This thesis work presents a framework to control the access of Materialized Views based on user role. The authorizations are associated with user role, and these authorizations are defined in Datalog syntax. Our work extends an earlier work proposed by Bahloul et al. [3]. The authors define authorization views for individual users to control the access of materialized views, but in our approach we define authorization views for roles.

In our proposed framework, we utilize a Role-based access control approach specification that is proposed by National Institute of Standard and Technology (NIST), and we authorize user to view the data based on an assigned role. We provide column based authorization on a requested Materialized View.

In our proposed framework, we enable users to activate multiple *session_roles* simultaneously. In our thesis research, we identified that if two roles have conflict of interest then they cannot be activated at the same time. In order to avoid role conflicts, we introduce role conflict strategies, these strategies are associated with user role same as authorization views. In a role conflict strategy, the attributes are defined to enforce Separation of Duties (SoD), and conflicting roles are given priority to resolve role conflicts.

In order to test our proposed architecture, we developed an application using the open source technologies (i.e. MySQL and PHP). The implementation is done in accordance with the basic requirement of our proposed architecture.

The results show that the application meets the basic requirements of our proposed framework which include defining authorization views for roles, provide column based authorizations, activation of multiple *session_roles*, detection of conflicting roles, and setting up priorities to resolve role conflicts.

Our proposed framework is not domain specific; it is a generic framework and can be utilized in any domain.

5.2 - Future Work

We address some other potential areas which can be addressed in future work based on the experience gained in this thesis research.

5.2.1 - Role Automation

In this thesis research, we identified that the role automation is another important area of research. The role automation is required in big organizations where thousands of employees perform their duties. In such organizations to create, assign, and change the role of each employee is a time consuming process. Currently, a dedicated department such as Human Resources department assigns a role to each employee based on his/her job descriptions. The entire process is manually controlled; once a role is assigned to an employee by the concerned department, the system administrator creates a new entry in a database table or Access Control List (ACL) in which the employee profile attributes are stored.

Future work includes a role automation process that assign roles to employees based on their job descriptions.

5.2.2 - Workflow Management in RBAC

In Role-based Access Control system, the chain of command (role hierarchies) should be defined clearly where top-most role is first in the command and the next in command is down one level and so on. Each role in the role hierarchy is assigned responsibilities such

as a Supervisor is responsible to submit the working hours of his subordinates. In this case, if the Supervisor is not available then it can delay the process.

In order to avoid such delays, we need a mechanism that authorizes each role to transfer its responsibilities to next in the role hierarchy. The redirection of responsibilities to another role in the role hierarchy is supposed to be temporarily, and a time slot must be assigned during the transfer process. After the assigned time, the authorizations associated with the given responsibilities must be revoked by the system.

Future work includes implementing and integrating role automation and workflow management system with our existing framework.

5.2.3 - Global vs. Local Authorizations

In our thesis research, we have presented a framework that enforces authorizations based on user assigned role on a local materialized view database. Future work includes extending our Role Authorization Framework that will allow user to access data from remote databases in a distributed environment based on authorization defined on local database (Local authorization views) and remote databases (Global authorization views). The local and global authorization must be analysed by the system before granting access to users. In order to resolve conflicts between both local and global authorization views, we need to design a strategy to resolve such conflicts.

BIBLIOGRAPHY

- [1] Abiteboul, S., Hull, R., AND Vianu, V. 1995. Foundation of Databases. Addison-Wesley.

- [2] Bahloul, S. N., Coquery, E., AND Hacid, M. 2012. Authorization Policies for Materialized Views. Proceedings of 27th IFIP TC 11 Information Security and Privacy Conference. Heraklion, Crete, Greece. 525-530.

- [3] Bahloul, S. N., Coquery, E., AND Hacid, M. 2011. Access Control to Materialized Views: an Inference Based Approach. Proceedings of the 2011 Joint EDBT/ICDT Ph.D. Workshop. New York, USA. 19-24.

- [4] Cuzzocrea, A., Hacid, M., AND Grillo, N. 2010. Effectively and Efficiently Selecting Access Control Rules on Materialized Views over Relational Database. IDEAS' 10 Proceedings of the Fourteenth International Database Engineering & Application Symposium. 225-235.

- [5] Cruz, I. F., Gjomemo, R., Lin, B., AND Orsini, M. 2008. A location aware role and attribute based access system. Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems. New York, NY, USA.

- [6] Ferrini, R., AND Bertino, E. 2009. Supporting RB AC with XACML + OWL. Proceeding of the 14th ACM symposium on access control models and technologies. ACM New York, NY, USA.

- [7] Finin, T., Joshi, AND A., Kagal,. 2008. Using OWL to Model Role Based Access Control. UMBC Ebiquty Laboratory Technical Reort, University of Maryland, Baltimore County, Baltimore, USA.

- [8] Feng, F., Lin, C., Peng, D., AND Li, Junshan. 2008. A Trust and Context Based Access Control Model for Distributed System. In Proceeding of 10th International Conference on High Performance Computing and Communications. 629-634.
- [9] Haddad, A., Hacid, M., AND Laurini, R. 2012. Data integration in presence of authorization policies. Proceedings of IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. Lyon, France. 92-99.
- [10] Joshi, J. B. D., Bhatti, R., Bertino, E., AND Ghafoor, Arif. 2004. Access-Control Language for Multi-domain Environments. IEEE Internet Computing. 40-50.
- [11] Kent, R.D., Koltiz., Snowdon, A.W., AND Aggarwal, A. 2010. Towards a Unified Data Management and Decision Support System for Health Care. Intelligent Interactive Multimedia Systems and Services, Volume 6, 205-220.
- [12] Koltiz., Snowdon, A.W., Kent, R.D., Bhandari, G., Rahman, S.F., Preney, P.D., Kolga C.A., Tiessen, B., AND ZHU, L. 2011. Towards a “Just-in-Time” Distributed Decision Support System in Health Care Research. Annals of Information Systems: Supporting Real Time Decision-Making, Volume 13 (3), 253-285.
- [13] Kulkarni, D. AND Tripathi, A. 2008. Context-Aware Role-based Access Control in Pervasive Computing Systems. In *Proceeding of the 13th ACM Symposium on Access Control Models and Technologies*. Estes Park, Colorado, USA, 113-121.
- [14] Liang, C. A Faster Way to Temporarily Redirect the Role Based Access Control Workflow Processes. Proceeding of 21st Computer Science Seminar.

- [15] Motro, A. 1989. An Access Authorization Model for Relational Databases Based on Algebraic Manipulation of View Definitions. Proceedings of IEEE 5th International Conference on Data Engineering. 339-347.
- [17] Ni, Q., Trombetta, A., Bertino, E., AND Lobo, J., 2007. Privacy-Aware Role-Based Access Control. Proceedings of the 12th ACM symposium on Access control models and technologies. ACM New York, NY, USA.
- [18] Pottinger, R. AND Halevy , A. 2001. MiniCon: A scalable algorithm for answering queries using views. The VLDB Journal. 10. 182-198.
- [19] Priebe, T., Dobmeier, W., Schlager, C., AND Kamprath, N., 2007. Supporting Attribute-based Access Control in Authorization and Authentication Infrastructures with Ontologies. Journal of Software, 2, 27-38.
- [20] Rizvi, S., Mendelzon, A., Sudarshan, S., AND Roy, P. 2004. Extending Query Rewriting Techniques for Fine-Grained Access Control. Proceedings of the 2004 ACM SIGMOD international conference on Management of data. 551-562.
- [21] Ryutov, T., AND Kichkaylo, T., Neches, R., 2009. Access Control Policies for Semantic Networks, IEEE International Symposium on Policies for Distributed Systems and Networks. 150-157, 20-22.
- [22] Toninelli, A., Montanari, R., Kagal, L., AND Lassila, O. 2006. A Semantic Context-Aware Access Control Framework for Secure Collaboration in Pervasive Computing Environment. In *Proceeding of International Semantic Web Conference*. Verlag, Berlin, Heidelberg, 473-486.
- [23] Wang, J., Maher, M., AND Topor, R. 2002. Rewriting Unions of General Conjunctive Queries Using Views. Proceedings of 8th International Conference on Extending Database Technology. Prague, Czech. 52-69.

- [24] Weber, A., Hazen. 2003. Role Based Access Control: The NIST Solution.
- [25] Zheng, Z., AND Mendelzon, A. 2005. Authorization Views and Conditional Query Containment. Proceeding of 10th International Conference. Edinburgh, UK, 259-273.
- [26] http://en.wikipedia.org/wiki/Role-based_access_control
- [27] http://docs.oracle.com/cd/B10501_01/server.920/a96567/repview.htm
- [28] <https://code.google.com/p/flexviews/>
- [29] <http://www.mysql.com/>

VITA AUCTORIS

Hassaan Yousafi was born in 1978 in Lahore, Pakistan. He received his B.Sc. (honours) in Computer Science from the University of Management and Technology in 2005. After completion of his undergraduate degree, He worked as IT Administrator, Software Engineer, and Web & Systems Engineer in Schlumberger, TricastMedia, and IBM Canada Ltd respectively. Currently, he is a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in Winter, 2013.