Indian Journal of Science and Technology, *Vol 11(2), DOI: 10.17485/ijst/2018/v11i2/113956, January 2018*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

An Approach for Enhancing Data Access Security in Heterogeneous Database Systems

Ahmed Elbatal, Ahmed M. Gadallah and Hesham Hefny

Department of Computer Science, Institute of Statistical Studies and Research, Cairo University, Giza, Egypt;
ahmedelbatal@hotmail.com, ahmgad10@yahoo.com, hehefny@ieee.org

Abstract

Objectives: This paper proposes an enhanced data access security approach to allow virtual private database security mechanism in heterogeneous multi-tier applications regardless of the data access security features provided by each database management system. **Methods/Statistical Analysis:** An implementation of Data Access Layer has been done respecting the proposed approach. This implementation enhances Microsoft's Entity Framework that is widely used in commercial multi-tier database applications as a Data Access Layer. Accordingly, it's overloaded by the required functionality including query modification and data validation. The output assembly then is tested in a typical HR database application that targets three different DBMS's (SQL Server, MySQL, Oracle) with exactly same database state. A time measurement takes place to evaluate the processing cost of issuing CRUD operations compared with the same application architecture without using the proposed approach (e.g. relying on the row-level security provided by Oracle on the DBMS level). **Findings:** An illustrated case study respecting the proposed approach shows its scalability, reliability and efficiency. It allows data access security in both homogenous and heterogeneous database applications. On the other hand, the results show that the cost of processing both of data retrieval and data manipulation operations respecting predefined data access security policies of the proposed approach compared with Oracle VPD are reduced by around 59% and 57% respectively. **Application/Improvements:** As presented in the illustrative case study, the proposed approach can be easily applied and reused in any modern heterogeneous multi-tier database application. It allows defining data access security policies regardless of the target database management system type. Also, the results show an improvement in the processing cost of the proposed approach compared with the Oracle virtual private database with both data retrieval and data manipulation operations.

Keywords: Data Privacy; Data Access; Database; Heterogeneous Database Applications;

1. Introduction

Data access control is one of the main security issues that should be tackled in most database applications. One of the most useful data access control mechanisms is the virtual private database provided by some database management systems (e.g. Oracle[1] and PostgreSQL[2]) to facilitate row-level data access control. However, this mechanism is not yet presented in other relational database management systems (e.g. SQL Server, MySQL, SQLite, etc.). On the other hand, most of modern database applications are built respecting the multi-tier architecture. Yet, they still depend on the mechanisms provided by database management systems to implement data security in the lowest tier of their architecture, the database tier. This dependency makes it impossible to re-use the same security policies in heterogeneous database applications where different types of database management systems are used. The following subsections introduce the multi-tier application arechitecture and data access control. Also, they give an overview about the virtual private database mechanism provided by Oracle and the traditional way of implementing the virtual private database mechanism in a multi-tier application architecture

**Author for correspondence*

1.1 Multi-tier Application Architecture

Generally, most modern database applications are built respecting the multi-tier architecture. The most widely used multi-tier architecture is the three-tier architecture that incorporates three tiers: 1. Data tier that includes both of data access layer (DAL) and data persistence mechanisms as database servers, file sharing, etc. Traditionally, DAL includes a set of entity types by which each of them corresponds to one of the persistent data objects (i.e. tables or views) 2. Business logic tier that is responsible for coordinating the application, satisfying the business rules and makes the required computations and logical decisions. 3. The presentation tier represents the user interface that facilitates the interaction between the user and the application. It presents tasks and results in an easy-to-use and understandable form to the application users. Accordingly, the presentation tier represents the top-most level of an application. Other forms of the multi-tier architecture may expand one or more layer of the traditional three tiers to represent different levels of abstraction. Figure 1 shows the three-tier architecture in a single database application. Commonly, it becomes natural for most multi-tier application developers to use some sort of code generation process to help in generating the needed for of the application's DAL. Many Object-Relational Mapping (ORM) tools are used to accommodate this purpose such as NHibernate, Dapper, Massive, etc. Some of these ORM tools allow data access security at the object-level. Accordingly, a database record is firstly loaded into the DAL before deciding if it is accessible by the user or does not. In heterogeneous database applications, where different types of database management systems are used, the multi-tier architecture considers a separate DAL for each database involved in the application. Figure 2 shows the three-tier architecture in a heterogeneous database application.

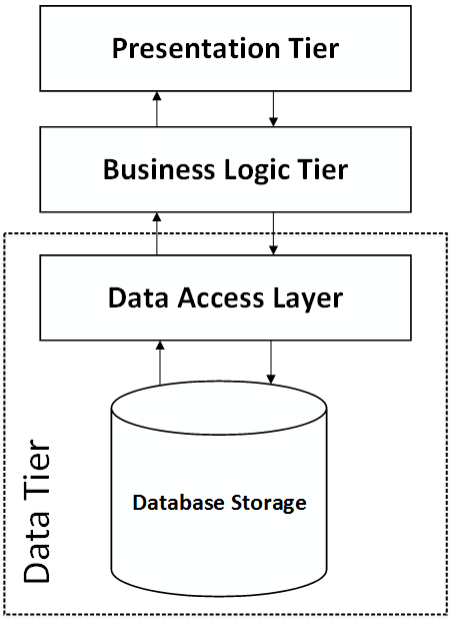

Figure 1. The three-tier architecture in a single database application.

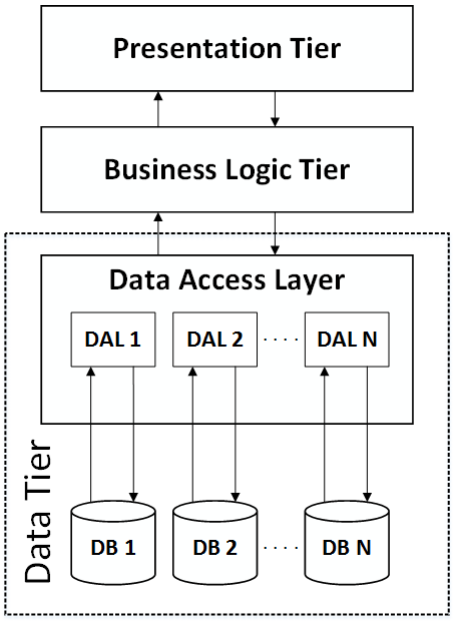

Figure 2. The three-tier architecture in heterogeneous database applications.

1.2 Data Access Control

In relational databases, Data Access Control (DAC) is considered to be one of the main control techniques that are used to provide security of data. It is responsible for filtering data so that only the user accessible data that he/she has right to access are available. Commonly, a database management system incorporates a database security and authorization subsystem for preventing unauthorized access. Yet, some of such subsystems in traditional RDBMs do not differentiate between individual rows in a table. Accordingly, a user can access all or nothing of a table rows[3]. For example, "GRANT SELECT" on a table allows a user to access populated rows in that table. In contrary, row-level security richens data security with an additional layer of access controls on a per-row basis. Accordingly, a security requirement such as "an employee

can only view certain orders that he/she is responsible for" becomes allowable. So, it can be defined and enforced by the DBMS. The typical form of row-level security is known as label-based security in which the user is given a session label by the database administrator[4]. Consequently, such label is used by DBMS to get the data accessible by the user.

1.3 Virtual Private Database

In order to satisfy the customers' needs, some DBMSs provide different forms of security mechanisms that aim to allow row-level data access security[5]. In February 1999, Oracle introduced the concept of Virtual Private Database (VPD) as a technology that enables users to add security policies to control database access at the row-level[1]. Commonly, VPD allows dynamic WHERE clauses to SQL that attempt to access a table, synonym, or view to which a VPD security policy is attached. Accordingly, there is no way to escape this type of security. Such security policies are directly attached to the specified data objects. Consequently, the policies are automatically applied whenever a user attempts to access data. Generally, when a user attempts to access a table, view, or synonym that is protected with a VPD policy, the DBMS dynamically modifies the issued SQL statement to reflect the existed security policy. Such modification creates a WHERE condition, called a predicate, returned by a function representing the predefined security policy. In other words, the DBMS modifies the SQL statement by a condition that can be expressed by the function. Consequently, a VPD policy can be set to any of the data manipulation operations namely SELECT, INSERT, UPDATE and DELETE. For example, suppose a user has access to the orders of Sales Representative identified by ID equal 123. Assuming that, the user generates the following query statement:

```
SELECT*FROM OE.ORDERS;
```

Dynamically, the VPD policy is fired and adds a 'WHERE' clause as follows:

```
SELECT*FROM OE.ORDERS
WHERE SALES_REP_ID =123;
```

In consequence, the user can only retrieve the data of orders made by sales representative with ID equals 123. Moreover, to retrieve the orders data accessible by a user based on the session information, such as his/her user ID, the following "WHERE" clause can be used:

```
SELECT*FROM OE.ORDERS
WHERE SALES_REP_ID = SYS_CONTEXT
('USERENV','SESSION_USER');
```

Where SYS_CONTEXT is a function that retrieves the stored session-based variables in an Oracle database.

Accordingly, a function defining the data privacy restrictions is used to generate a dynamic "WHERE" clause. Usually, the security administrator creates such function in the specified database schema. The function takes as arguments a schema name and an object (table, view, or synonym) name as inputs and returns a predicate or "WHERE" clause representing the specified security policy. Generally, to set a VPD policy, the table, view, or synonym to which the security policy is applied must be specified rather than the types of statements the policy controls. The following block of code shows an example of how to add a VPD policy in an Oracle database.

```
BEGIN
DBMS_RLS.ADD_POLICY(
    object_schema =>'HR',
    object_name=>'EMPLOYEES',
    policy_name =>'SECURE_UPDATES',
    policy_function =>'CHECK_UPDATES',
    statement_types =>'SELECT,UPDATE'
);
END;
```

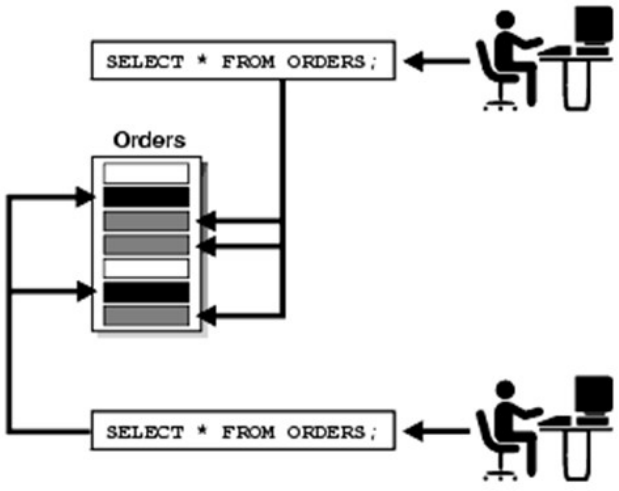

Figure 3: Virtual private. Database

This example shows how to attach a VPD policy called "SECURE_UPDATES" to the "EMPLOYEES" table in the database schema "HR". The function attached to the policy is CHECK_UPDATES. On the other hand, this policy is specified for SELECT and UPDATE SQL statements. Accordingly, the policy is applied only when issuing such statements over the "employees" table. Commonly, the combination of creating the security policy function and then applying it to a table or view is referred to as creating a VPD policy[1].

1.4 Virtual Private Database in Multi-tier Architecture

Normally, to work with VPD in a multi-tier architecture application the security policy should not leave its place in the database storage. It still defined as a predicate function in the database dictionary and executed within the DBMS runtime environment. Figure 4 describes the architecture of the typical form of VPD in a single three-tier database application. Typically, data access via VPD in a three-tier architecture, shown in Figure 5, acts as follows:

- The business logic layer issues some data manipulation operation.
- In turn, the DAL transforms the operation into the suitable SQL statement and sends it to the DBMS.
- The DBMS hits the data access security package.
- The DBMS executes the policy definition (defined as a predicate function) attached to the manipulated table(s).
- The policy definition returns the filter expression according to the user context.
- The data access security package returns the accumulative filter expression gathered from each policy attached to the manipulated table(s).
- In consequence, the DBMS queries the database using the resulted filter expression combined with the original query statement.
- The DBMS holds the returned data from the manipulated table.
- The DBMS transfers the resulted data to the DAL
- In turn, the DAL sends the resulted data to the business logic layer.

Moreover, in heterogeneous database applications, each database has its own row-level security mechanism at the optimistic case. However, some contributing DBMS's in heterogeneous applications may not have any row-level security mechanism at all like MySQL and Sqlite. Figure 6 shows a typical implementation of VPD in heterogeneous three-tier database applications.

The rest of this paper is organized as follows: Section 2 introduces the related work. The proposed approach is presented in the Section 3. In consequence, Section 4 gives an illustrative case study with results and findings. Finally, the conclusion and future works are given in Section 5.

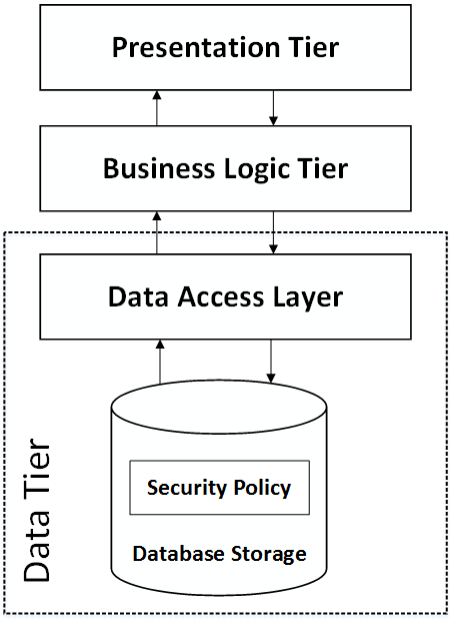

Figure 4. VPD in three-tier single database applications.

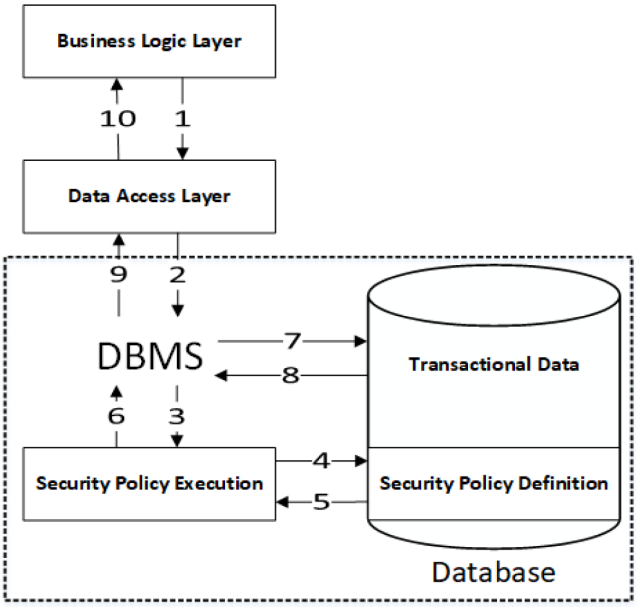

Figure 5. Security policy processing in VPD with the three-tier architecture.

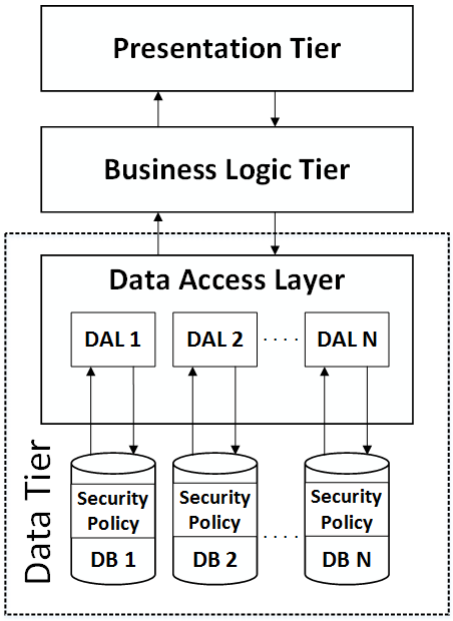

Figure 6. VPD in three-tier heterogeneous database applications.

2. Related Work

Generally, data access control mechanisms provided by a DBMS are not the only factor to put in mind when choosing the appropriate DBMS for a database application[6–9]. Some customers choose a DBMS that fits their business and system requirements such as license cost, scalability and performance regardless of data access control. Such DBMS's may lack the appropriate data access control mechanism needed to fulfill the data access security issues. In such case, it becomes very important to build a custom data access security mechanism or using some other mechanisms as a work around.

Some previous works uses VIEWS to implement row-level data access security as a work around that is suitable for all kinds of existed RDBMS. However, this technique has some disadvantages: It is hard to administer, as the data access rules are embedded in the views. Also, its implementation is limited and requires careful design and development[10–12]. In addition, another crucial problem of using such approach takes place when attempting to migrate to another DBMS. It would then costs a lot time and effort to transform the existing data access control implementation to the other DBMS. Moreover, this would costs more if the existing application uses some mechanisms that do not even exist in the other DBMS (e.g. like the VPD). On the other hand, in a Multi-tier Application, it is more appropriate to put all the system's business rules including data access rules in the business layer. Unfortunately, putting them in a lower layer (i.e. Database DDL) would violate the multi-tier architecture of the application[13].

Although, moving data access mechanism one layer up in a multi-layer application is not a new idea; Corcoran et al. proposed a new programming language called SELinks[14,15]. It provides a uniform programming model for building secure multi-tier web applications. Yet, it focused on label-based security (i.e. not as flexible as the VPD mechanism) and as a programming model, it is hard to be implemented and integrated with existing applications. This is due to the effort needed to transform the existing code to meet its equivalent syntax.

In consequence[16] survey the area of securing web applications from the server side, with the aim of systematizing the existing techniques into a big picture that promotes future research. They discussed SELinks in the perspective of application logic vulnerabilities within new web applications security construction. Other mechanisms mentioned in their survey are more about input validation and data flow tracking between layers rather than securing the data that resides on the data base layer (i.e. attached to a DBMS). Recently, researchers are working on enhancing the use of SELinks for the purpose of controlling data flow over multi-tier applications. Balliu et al. toke SELinks as an inspiration to develop a new framework called JSLINQ as an extension of the WebSharper library to track data flow[17].

3. The Proposed Approach

The proposed approach in this paper aims mainly to enhance the DAL itself to allow attaching security policies to its entity types that may existed on heterogeneous DBMSs. Accordingly, such defined data security policies can be used later to append the predefined filter expressions when executing data manipulation operations. An example of implementing data security polices in a heterogeneous database system based on the proposed approach is introduced in the case study section. The implementation uses the publically used DAL of Microsoft, the Entity Framework. However, the same approach can be implemented using any kind of code generated DAL.

Figure 7 shows the architecture of the proposed approach in three-tier database applications. In such

architecture, the security policy is moved away from the database dictionary. Accordingly, the functionality of modifying a SQL statement with the filtering condition is executed in the DAL which resides in the application runtime environment rather than the DBMS. Such a way makes it easy to work with heterogeneous database applications. Also, it facilitates migration from one DBMS to another one. Figure 8 presents the architecture of the proposed approach in heterogeneous three-tier database applications. Consequently, the security policy definition becomes part of the business logic layer. Accordingly, the security policies are considered as collection of business rules that are defined in the business logic layer in the form of predicate functions. On the other hand, the security policy execution package is shared among all DALs involved in the heterogeneous application. It contains the necessary classes to do the functionality of DAC; which is abstracted away from being a part of each separated DAL. A simple implementation of the relationship between DALs and a shared security policy execution package can be done by using inheritance. Such that each separated DAL inherits from the base classes that contain the security policy execution functionality. In the proposed approach, data access security is fulfilled as shown in Figure 9 and according to the following steps:

- The business logic layer issues some data manipulation function.
- The DAL hits the data access security package resides in its runtime environment.
- The data access security package executes the policy definition (defined as a predicate function in the business logic layer) attached to the manipulated tables.
- The policy definition returns the filter expression according to the user context.
- The security policy package returns the accumulative filter expression gathered from each policy attached to the manipulated table(s).
- The DAL then hits the DBMS with the resulted filter expression combined with original statement.
- The DBMS executes the final query statement including the filter expression.
- The DBMS holds the returned data from the manipulated table.
- The DBMS transfers data to the DAL.
- In turn, the DAL sends the resulted data to the business logic layer.

Respecting the processing scenario of the proposed approach, the added value of its applicability compared with traditional ones can be addressed as follows:

- The defined data access security policy is executed in the application runtime environment rather than delegating it to the DBMS. Accordingly, the proposed approach leads to an earlier execution of the defined data security policy that is more suitable in heterogeneous data base applications. So, the generation of the filter expression becomes related to the runtime environment used in the application. It also keeps the DBMS away from managing the data access security and leaves it dedicated to the data manipulation operations.
- Also, the predicate function representing the required security policy is defined and executed within the business logic layer. Accordingly, each security policy is considered as a part of the business rules defined in that layer and may re-use other predefined rules.
- In heterogeneous applications, the same defined data access security policy can be accessed by different DALs.
- Moreover, migration to different DBMS becomes easier since no change required to be done on the predefined security policies. In addition, the same approach can be used with any type of DBMS regardless of whether it provides VPD capability or not.

According to the proposed approach, it becomes easy to define a data security policy to be applied in a heterogeneous database application. The following code segment instantiate a database model and define a security policy.

This example shows how to instantiate a new Database Context ”Northwind Context”. It adds a security policy called ”Filter By Department” to be applied on the” Employees” table respecting any type of data manipulation operations denoted by ”VPDbStatementType.All”. The security policy uses a pre-defined predicate function called ”Filter By Department Function” that returns a dynamic filter string. Consequently, four data manipulation operations including SELECT, INSERT, UPDATE and DELETE on the targeted employees table are executed. Accordingly when executing the ”Save Changes” function, if there is any attempt to violate one of the added security policies then an exception will be thrown. On the other hand, the following code block shows an example of a predicate function ”Filter By Department

```
//Instantiate a database context
var NorthwindContext =new NorthwindModel();
//Modifying its application context
NorthwindContext.applicationContext["CurrentUserDepartmentId"]=5;
//Adding a security policy
VPDbPolicies.AddPolicy(NorthwindContext, NorthwindContext.Employees,
   "FilterByDepartment", FilterByDepartmentFunction, VPDbStatementType.All);
//Selecting filtered employees
var employees = NorthwindContext.Employees;
//Add new employee with id=12
var newEmployee =new Northwind.Model.Employee();
newEmployee.EmployeeID =12;
newEmployee.FirstName ="John";
newEmployee.LastName ="Michael";
newEmployee.DepartmentId =5;
NorthwindContext.Employees.Add(newEmployee);
NorthwindContext.SaveChanges();
//Update an existing employee whose id=12
var existingEmployee=NorthwindContext.Employees.Find(12);
existingEmployee.FirstName ="Peter";
NorthwindContext.SaveChanges();
//Delete an existing employee whose id=12
var deleteEmployee = NorthwindContext.Employees.Find(12);
NorthwindContext.Employees.Remove(deleteEmployee);
NorthwindContext.SaveChanges();
```

Function" that uses the value of the "Current User Department" key stored in the application context to return the appropriate filter expression that would be then applied, through the "Filter By Department" security policy, to allow the current user to manipulate only the data of employees belong to his department.

```
string FilterByDepartmentFunction (VPDbApplicationContext applicationContext)
{
   if(applicationContext["CurrentUserDepartmentId"]!=null)
      return@"DepartmentId=" + applicationContext["CurrentUserDepartmentId"];
   else
      return@"1=0";
}
```

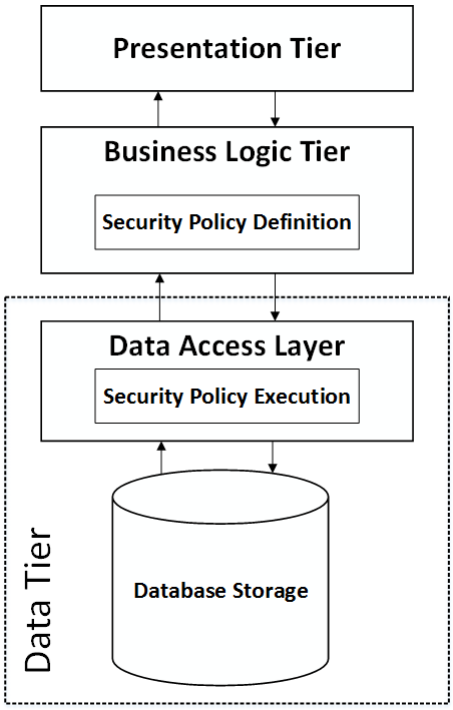

Figure 7. The proposed approach in a three-tier application architecture.

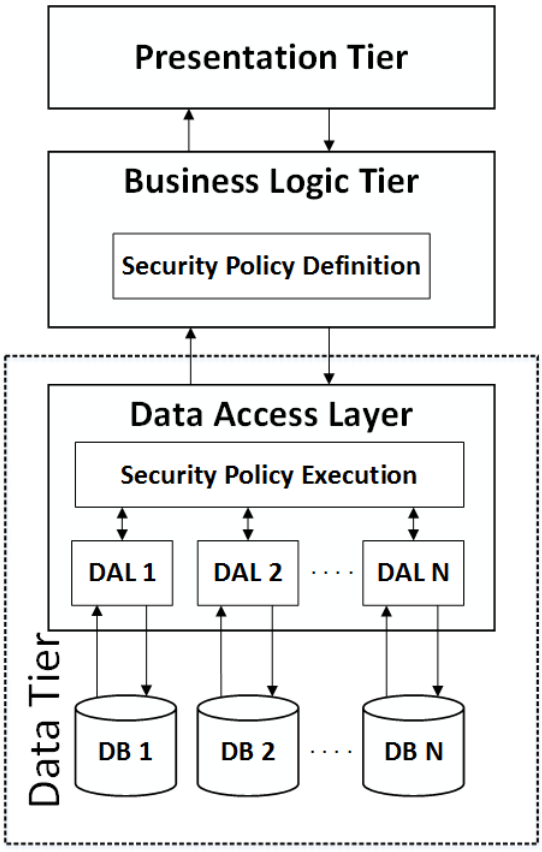

Figure 8. The proposed approach in a heterogeneous three-tier application architecture.

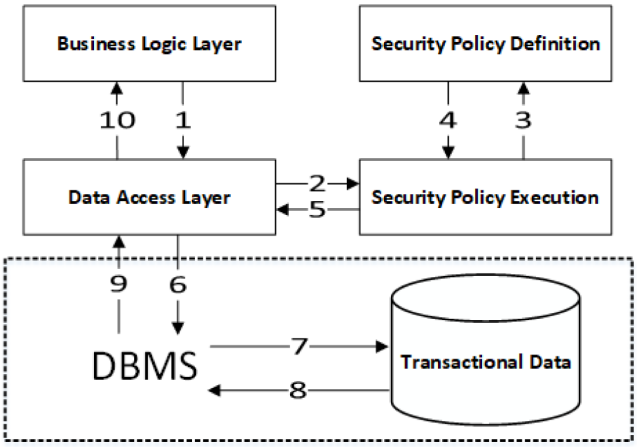

Figure 9. The flow of executing a security policy in the proposed approach.

4. An Illustrative Case Study

This case study represents an implementation of the proposed approach. It incorporates a set of most popular DBMS's (i.e. MSSQL, MySQL and Oracle) using a simple HR database that existed on each DBMS with typical large amount of data.

4.1 Objectives

The main objective of this case study is to show the flexible applicability of the proposed approach with any type of DBMS that is supported by the Entity Framework. Another objective is to compare the performance of the proposed approach against Oracle VPD in the same environment (i.e. using Entity Framework as the DAL). Also, the case study shows the cost of processing different manipulation operations on a set of heterogeneous DBMSs that obey or violate a predefined data access security policy.

4.2 Schema and Data Sample

A simple HR schema is considered for the case study that includes two tables as follows:

- Employees (Employee Id, First Name, Last Name, Email, Phone, Address, City, State, Zip Code, Country, SSN, Salary, Birth Date, Department Id) with 10 million records (about 2.5 GB).
- Departments (Department Id, Department Name), with 10 records.

4.3 Application Structure

The .NET application of the case study has been made up of a set of layers and sub-projects. The database layer includes three databases with the same database state on MSSQL, MySQL and Oracle DBMSs. On the other hand, the DAL incorporates a library named EFRLS that contains the required classes that encapsulate the implementation of the data access security policies. Also, The DAL includes four entity framework sub-projects. Three of them are Hr.Model.MSSQL, Hr.Model.MySQL and Hr.Model.Oracle that represent the entity framework DAL for each corresponding DBMS (MSSQL, MySQL and Oracle DBMSs respectively). The fourth sub-project called Hr.Model.OracleVPD represents the normal entity framework DAL of the Oracle database without applying the proposed approach. The Hr.Model.OracleVPD is responsible for creating the required security policies respecting the Oracle supported VPD feature.

4.4 An Implementation of the Proposed Approach

Accordingly, the areas of DAL code that have been modified to accommodate the implementation the proposed approach are as follows:

- The Data base Context that represents a combination of the Unit of Work and Repository patterns (Datasets) such that it can be used to query from a database and group together changes that will then be written back to the store as a unit. It has been modified to override the base Entity Framework's Database Context on two major functionalities: 1. Allow passing an Application Context (user defined collection of key/value pairs that may contain some shared or session-based variables) to any new instance of the Database Context. 2. Validate the changes made on its datasets before saving them to the database store.
- The dataset representing the collection of all entities in the context that can be queried from the database, of a given type. It has been modified to override the base Entity Framework's Dataset on two major functionalities: 1. Validate added, updated or deleted entities on the server before affecting the original dataset and throw an exception if any of the working entities violates the added security policies. 2. Exposing the filtered dataset, instead of the original one, to any SELECT operation made on a dataset instance.
- The database model that inherits from the Database Context and represents an implementation of a specific database. It has been modified to add two major functionalities: 1.Allow passing an application context to any new instance of the database model. 2. Using the modified dataset instead of the base Entity Framework's Dataset.

4.5 Testing the Proposed Approach

The implemented application starts with instantiating each of the four DALs. In consequence, it assigns their application context that will be used later in the security policy's predicate function. It then adds a simple security policy to all of those instances. Finally, it separately executes the operations stated in Table 1, from operation Op01 to operation Op13, on HR databases existed on different database servers via their corresponding entity framework's DAL.

4.6 Results

The illustrative case study shows that the proposed approach can be easily applied with the entity framework's DAL. It also shows good performance compared with the Oracle VPD especially on data manipulation operations (i.e. insert, update and delete). This because the process of checking the assigned security policies on adding a new entity or updating or deleting an existing one is performed on the DAL runtime environment rather than passing it to the DBMS. Also, it can be noted that the currently used approaches in data manipulation operations in a DBMS that does not support row level security is to put the validation process (i.e. an IF condition) in line in a higher level than the DAL. So, it does not take more than one tick to proceed.

In this case study a set of different data manipulation operations are considered. Table 1 shows a set of data manipulation operations that are applied in the case study. On the other hand, both of Table 2 and Table 3 show the result of the case study. The first column of each table shows the SQL operation that is being executed by the DAL. On the other hand, the second column shows the number of records affected/selected per each executed SQL operation. For each DBMS involved in the case study, "Normal" refers to the currently used approach in three tiers architecture (i.e. putting the required filter

Table 1. A set of SQL operations that are considered in the experiment

Operation ID	SQL Statement	SQL Operation
Op01	SELECT	Retrieving the first 1 record that matches a given filter expression.
Op02	SELECT	Retrieving the first 10 records that match a given filter expression.
Op03	SELECT	Retrieving the first 100 records that match a given filter expression.
Op04	SELECT	Retrieving the first 1000 records that match a given filter expression.
Op05	UPDATE	Updating an existing entity with values that obey an assigned security policy.
Op06	UPDATE	Updating an existing entity with values that violate an assigned security policy.
Op07	UPDATE	Updating an existing entity with no security policy assigned.
Op08	INSERT	Adding a new entity with values that obey an assigned security policy.
Op09	INSERT	Adding a new entity with values that violate an assigned security policy.
Op10	INSERT	Adding a new entity with no security policy assigned.
Op11	DELETE	Removing an existing entity with values that obey an assigned security policy.
Op12	DELETE	Removing an existing entity with values that violate an assigned security policy.
Op13	DELETE	Removing an existing entity with no security policy assigned.

expression inline in a higher layer than the DAL), "VPD" refers to the using of virtual private database approach (i.e. in Oracle Database only) and "Proposed" refers to the implemented one respecting the proposed approach. Also in this case study, the time cost of executing each SQL statement is measured by a number of CPU ticks (one tick = 10^{-4} millisecond) elapsed until processing that SQL statement. Accordingly, the numbers in the table show the CPU ticks of performing each operation. It has been calculated as an average of the elapsed ticks of 100 iterations per process. For better visualization, Figures 10, 11, 12 and 13 also show the same results in a bar chart format.

Table 2. The processing cost of executing different data retrieval operations (SELECT statements) respecting a predefined data access security policy

Operation ID	Count of Records	Oracle			MSSQL		MySQL	
		Normal	VPD	Proposed	Normal	Proposed	Normal	Proposed
Op01	1	5800	15872	6858	3001	3074	3398	3653
Op02	10	6342	16361	7096	8083	8468	8876	8934
Op03	100	7014	22818	7958	31050	31654	10425	11402
Op04	1000	7015	29048	12620	32038	34207	25206	15943

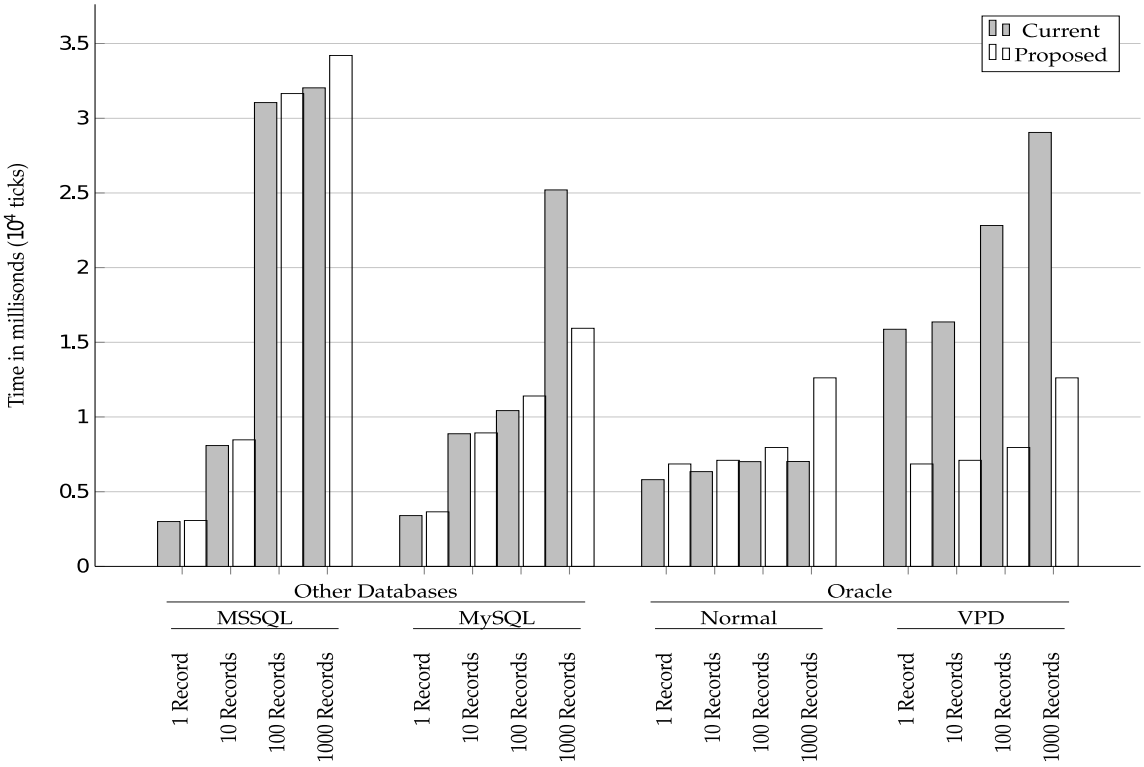

Figure 10. The case study results of the select statement.

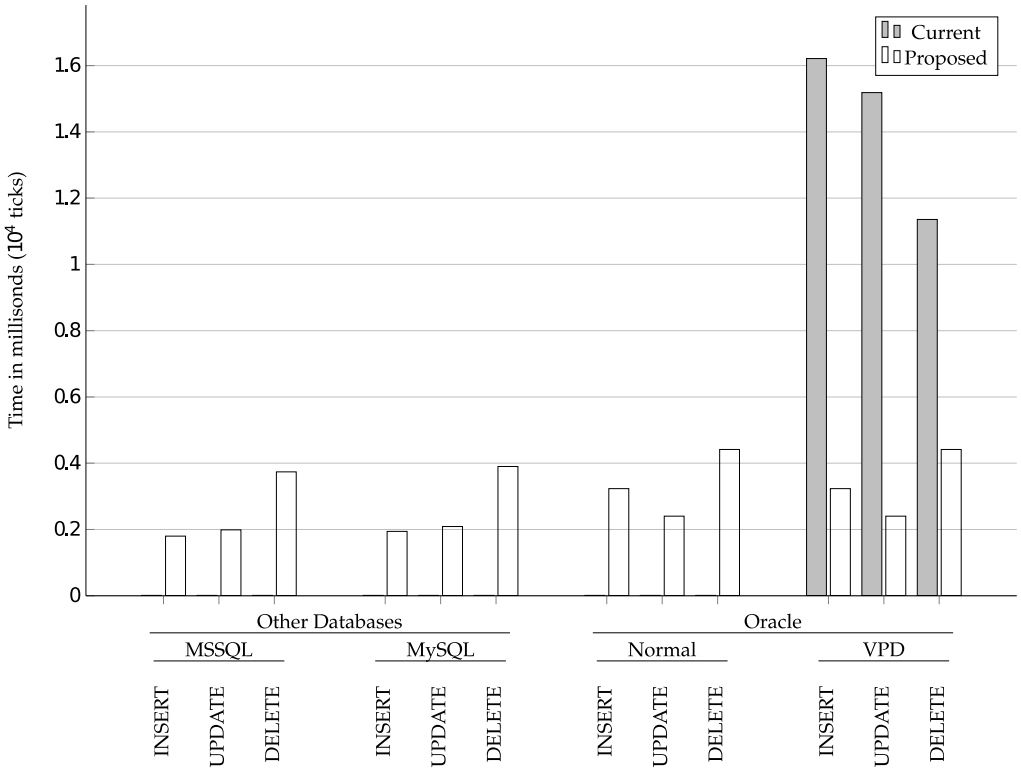

Figure 11. The case study results of the CRUD statements with invalid rules.

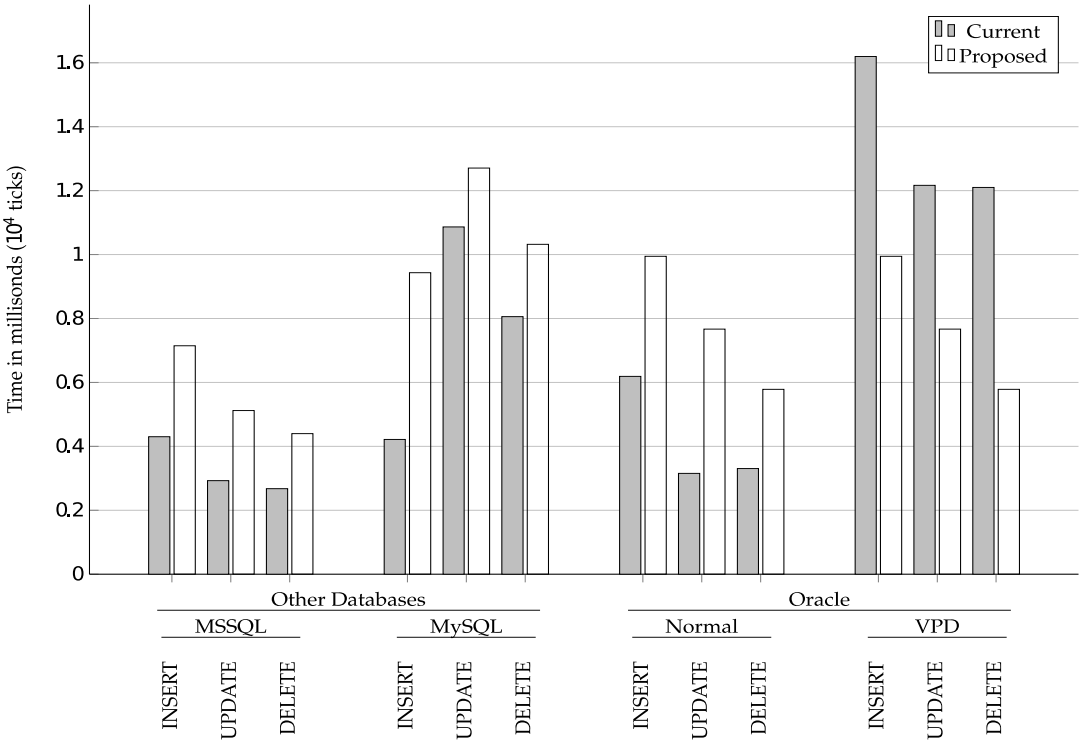

Figure 12. The case study results of the CRUD statement with valid rules.

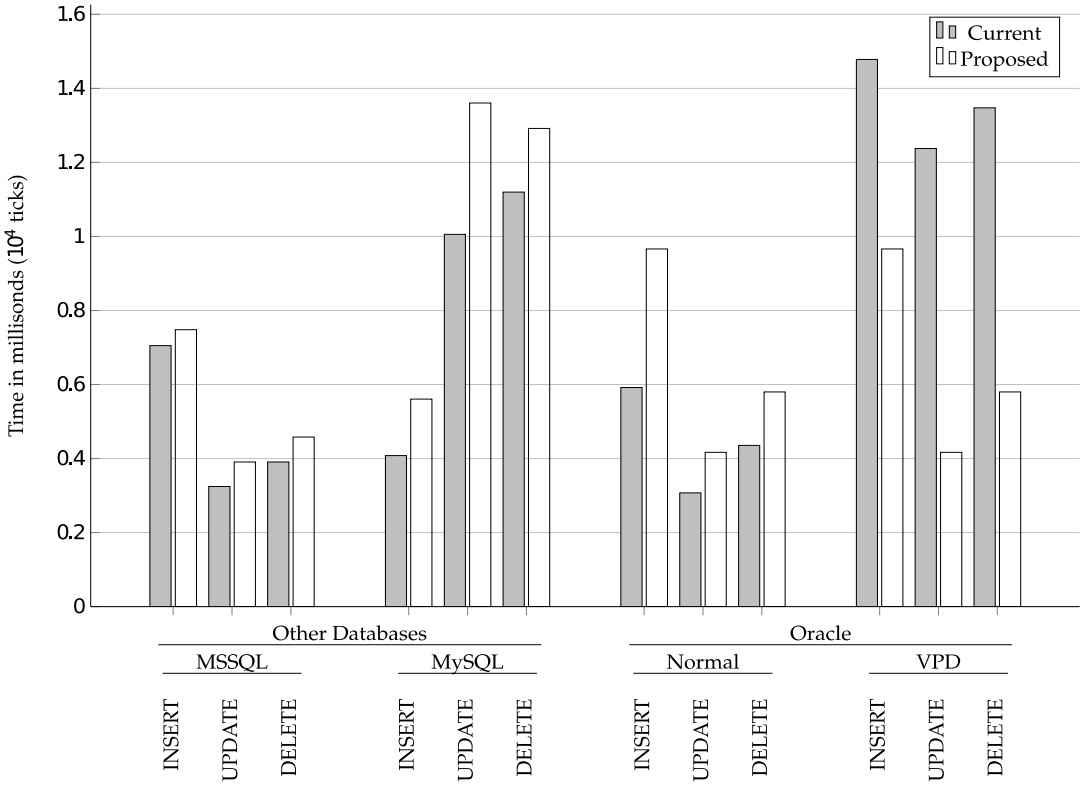

Figure 13. The case study results of the CRUD statements with no assigned rules.

Table 3. The processing cost of executing different data manipulation operations respecting a predefined data access security policy

Operation ID	SQL Statement	Rules	Oracle			MSSQL		MySQL	
			Normal	VPD	Proposed	Normal	Proposed	Normal	Proposed
Op05	INSERT	Valid Rules	6190	16198	9945	4301	7144	4216	9432
Op06	INSERT	Invalid Rules	1	16206	3223	1	1790	1	1934
Op07	INSERT	No Rules	5919	14781	9661	7051	7482	4080	5607
Op08	UPDATE	Valid Rules	3153	12167	7669	2927	5119	10864	12706
Op09	UPDATE	Invalid Rules	1	15179	2393	1	1980	1	2082
Op10	UPDATE	No Rules	3073	12374	4169	3244	3908	10057	13604
Op11	DELETE	Valid Rules	3303	12100	5782	2673	4398	8057	10319
Op12	DELETE	Invalid Rules	1	11350	4406	1	3731	1	3894
Op13	DELETE	No Rules	4355	13475	5798	3908	4582	11196	12917

As shown in the results of the case study presented in Table 2 and Table 3, the processing cost of one DAP is almost the same when applied on different DBMS as it takes place in a shared layer, the DAL runtime environment. Yet, there are different processing costs of the same data manipulation operation due to different processing scenario of each DBMS to manipulate such operation. Accordingly, the overall cost of performing a SQL operation respecting a defined data access security policy differs from one DBMS to another one.

Also, it can be noted that the time cost of executing a data security policy respecting the proposed approach is less than executing the same policy in Oracle DBMs using its supported virtual private data security VPD. As shown in Table 2, the proposed approach compared with Oracle VPD reduces the processing cost of data retrieval operations respecting a predefined data access security policy by around 59%. On the other hand, it reduces the processing cost of data manipulation operations compared with Oracle VPD respecting a predefined data access security policy by around 57% in average. This is reasonable as the process of generating a query statement in the proposed approach takes place in the application's runtime environment rather than the DBMS. Such generated query statement incorporates a where clause satisfying the assigned policies. Accordingly, the DBMS becomes responsible for processing a query statement without suffering from the cost of checking a security policy.

5. Conclusion

The proposed approach attempts to move the data access security one level up in a multi-tier application to the DAL instead of being a part from the DBMS itself. Accordingly, it can be used with all types of DBMS regardless of the security features supported by the selected DBMS for the database application. Also, the proposed approach is more flexible when migrating from one DBMS to another. The effort needed in case of such migration is just rebuilding the DAL (e.g. using the Entity Framework) to cope with the new DBMS with its appropriate database provider. On the other hand, the proposed approach fits better with the multi-tier architecture in that the business rules related to data privacy can be now represented in its appropriate position in a multi-tier application, the business logic layer. It modularizes the data access security needed in heterogeneous database applications.

Nevertheless, we must notice that it also comes with some shortages. It doesn't prevent data access to the DBMS directly. That is, if someone issues some SQL statement to the database directly using a database tool like SQL Plus, he would easily retrieve all data. Also, since the implementation of the proposed approach depends on the framework used to build the DAL. So that, if needed to use another framework (e.g. other than the Entity Framework), a modification of the code generated from the new framework is required to perform the same mechanism. It also takes the task of maintaining data access security away from the database administrator.

6. References

1. Huey P. Oracle Database Security Guide 11g. Oracle Corp; 2008.
2. Row-security - PostgreSQL Wiki. N.p., n.d. Web. 2016 01 Feb.
3. Elmasri R. Fundamentals of database systems. 7th ed. University of Texas at Arlington; 2015.
4. Afyouni H. Database security and auditing: Protecting data integrity and accessibility. Cengage Learning; 2005.
5. Knox D. Effective Oracle database 10g security by design. McGraw-Hill, Inc; 2004.
6. Thomsen C, Torben BP. A survey of open source tools for business intelligence. Data Warehousing and Knowledge Discovery. Springer Berlin Heidelberg; 2005. p. 74–84. Crossref.
7. Angles R. et al. Bench marking database systems for social network applications. First International Workshop on Graph Data Management Experiences and Systems. ACM; 2013.
8. Kuhlenkamp J, Markus K, Oliver R. Benchmarking scalability and elasticity of distributed database systems. Proceedings of the VLDB Endowment 7.12; 2014. p. 1219–30. Crossref.
9. Difallah DE, et al. Oltp-bench: An extensible test bed for benchmarking relational databases. Proceedings of the VLDB Endowment 7.4; 2013. p. 277–88. Crossref.
10. Licis ND. Desktop database data administration tool with row level security. U.S. Patent No. 7,155,612; 2006 Dec 26.
11. Cotner C, Miller RL. Row-level security in a relational database management system. U.S. Patent No. 7,240,046; 2007 Jul 3.
12. Thomson RD, Geiwitz R. Data security system and method. U.S. Patent No. 5,751,949; 1995 May 23.
13. Fowler M. Patterns of enterprise application architecture. Addison Wesley; 2002.
14. Corcoran BJ, Swamy N, Hicks M. Cross-tier, Label-based Security Enforcement for Web Applications; 2009.
15. Swamy N, Corcoran BJ, Hicks M. Fable: A language for enforcing user-defined security policies. Proceedings of the 29th IEEE Symposium on Security and Privacy. 369383. Oakland '08. 2008. Crossref.
16. Li X, Xue Y. A survey on server-side approaches to securing web applications. ACM Computing Surveys (CSUR). 2014; 46(4):54.
17. Balliu M, et al. JSLINQ: Building secure applications across Tiers. Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy; ACM. 2016.