AccKW: An Efficient Access Control Scheme for Keyword-Based Search over RDBMS

Vikram Goyal, Ashish Sureka, and Sangeeta Lal

Indraprastha Institute of Information Technology Delhi
New Delhi, India
{vikram,ashish,sangeeta}@iiitd.ac.in

Abstract. Access control for relational databases is a well researched area. An SQL query is allowed or denied access to database according to the specified access control policy. On the other side, there has been a surge in research activities to provide keywords-based search interface over RDBMS. This has posed new challenges for access control enforcement as traditional solutions to access control will not be efficient for keyword-based search. This paper proposes a framework AccKW, which enforces access control policies on keyword-based search over RDBMS in the early phases of keywords based search process. The main contributions of this paper are twofold: (i) we have investigated the problem of access control in the domain of keyword-based search over relational databases, and (ii) we have implemented the framework AccKW, and found out that AccKW outperforms in terms of execution time as compared to the naive approach (brute force approach) in case of strict access control policy.

1 Introduction

Keyword-based search has become popular with the advent of Internet search engines. Although it may at times be challenging to end-users to specify a good set of keywords for their respective queries, due to ease in specification of query keyword-based search has become one of the mostly used search paradigm in many domains.

Recently, there has been a surge in research activity in the area of keyword-based search over relational databases [1,2,3,4,5,6,7]. The ability to search relational databases using keywords allows end-users to search relevant information without any knowledge of the database schema and SQL. BANKS [2,4,5], DbXPlorer [1] and DISCOVER [3] are few systems that have been proposed in the recent past. Based on our literature survey, we observed that the research on keyword-based search on structured or semi-structured databases has focused on aspects such as strategy for keyword based search, search effectiveness and efficiency.

This paper focuses on access control for keyword-based search in enterprise domain. Search in enterprise domain would be different from search in the Internet domain as the set of users and their information need may be known in enterprise domain. Further, access control is a critical aspect in enterprise domain which needs to be addressed to enable wide scale adoption and deployment of keyword-based search solutions on relational databases. Some examples of access control in enterprise domain are:

– In a health organization, it may be desired to allow patients to see only their medical information in the medical database. On the other hand, a doctor should get access to all the medical information of the patients she is treating.
– In an academic institution information system, it may be desired that a student should be able to see only her grade. On the other hand, an instructor should be able to able to access all the grades for a course she has taught [8].
– In a bank database, a customer should be able to see only her bank account information. At the same time, a manager in the bank should be able to read transactions history of each bank account but not the personal information of customers.

Several techniques and models have been proposed as well as implemented in some of the commercial database management systems [9,8,10,11]. However, access control for relational databases is a well researched area, but, access control within the context of keyword-based search over relational databases poses certain unique challenges. According to our knowledge, access control for keyword-based search over relational databases in an enterprise domain is a relatively unexplored area. The work presented in this paper lies at the intersection of access control and keyword-based search on relational databases. Existing access control mechanisms on databases assume only SQL query interface. They analyze a user provided SQL statement's FROM clause and WHERE clause to decide the authorization. There are also some solutions which transform or rewrite an SQL query into a valid authorized SQL query if the input query is unauthorized [9,8].

In this paper, we have proposed a framework, AccKW, which integrates access control routines within the keyword-based search strategy. We identified that in some cases the naive approach would generate very large number of SQL queries which should then be processed through access control routines. Incorporation of access control in the early phases of search methodology drastically reduces the time to generate authorized queries. We define two performance metrics: (i) execution time to generate the queries and, (ii) number of authorized queries generated, to evaluate the performance of our framework. Experiments have been performed by varying number of input keywords, keywords selectivity, database size and access control policy. The results show that AccKW outperforms the naive approach when the access control policy is strict.

The key contributions of this paper are:

1. we have investigated the problem of access control in the domain of keyword-based search over relational databases, and have proposed a framework AccKW which enforces access control policies cost effectively in this domain.
2. we implemented the framework AccKW, and found out that AccKW outperforms in terms of execution time as compared to the naive approach in case of strict access control policy.

The rest of the paper is organized as follows. Section 2 describes related work. In Section 3, we describe the architecture of the AccKW framework. Section 4 presents our formal model of access control. Section 5 describes the algorithm used to enforce the access control policy in keyword-based search system. In Section 6 we describe the experimentation results. Section 7 concludes the paper.

2 Related Work

The work related to this paper can be discussed from two perspectives: access control and keyword-based search in RDBMS. We describe the work done in these two areas in this Section.

There has been extensive work in access control enforcement in relational databases [9,8,12,13,14]. The most closely related work is the Virtual Private Database (VPD) model of Oracle [9]. It uses the functions defined on relations by the administrator to enforce an access control policy. These functions return a predicate string for a given SQL query from a user, which is appended to the query. VPD is an excellent model to enforce access control policies but works on a single SQL query at a time. In case of keyword-based search system, this model will not be efficient as it would require all the SQL queries either authorized or unauthorized to be generated, whicshould then be analyzed by VPD model. Our proposed technique enforces access control policies during formation of SQL queries from input keywords and generates only valid SQL queries. We then use Oracle's VPD model to append predicates on the generated valid SQL queries to get a set of authorized SQL queries.

Similarly, the work proposed in [8,14] also considers a single SQL query and rewrites that SQL query into an authorized SQL query. These works focus on theoretical aspects of an SQL query after authorization rewriting. The work in [14] also discusses on optimal generation of safe plan of query execution to prevent leakage due to user defined functions having side effects. All these work are complementary to our work and can be used after generation of valid SQL queries. The work in [15] and the work in [13], both have described Cell-level authorization and its implementation techniques. However, there techniques are not general and are restricted to privacy policy enforcement. They also consider a single SQL query at a time and rewrites it or filter the query's result for enforcement of privacy. We have worked for general purpose access control and prune unauthorized SQL queries in the earlier phases.

Keyword-based search has been a focus of researchers in database community in the recent years [1,2,3,4,5,6,7]. They allow an application user to make a query using a set of keywords. It has made the task of application users easier as neither they should know and remember the SQL query semantics to search the information nor should they have knowledge about the database schema. Users input a set of keywords, and they get a set of ranked results. The search using keywords over databases [1,5] is different from search over Internet, as keywords are spread among a set of relations due to normalization of database. All the keyword-based search techniques over structured databases focus on aspects such as strategy for keyword-based search, search effectiveness and efficiency. None of this work addresses access control enforcement issues.

However, there have been proposed many different strategies for keyword-based search over relational databases. Our proposed access control enforcement strategy AcKW can be applied in conjunction with all of these techniques [3,5] in a cost effective way. For this paper, we choose the technique proposed in DbXplorer [1] as an example. DbXplorer uses two phases for reporting the result. These two phases are called publish phase and search phase respectively. In the publish phase, an inverted index is created using cell values in the database tables. Each entry of inverted index also called master index or symbol table has three values, i.e. keyword, relation-id and an attribute. The pair <*relation-id, attribute*> can be seen as a pointer to the table's attribute having the keyword as its cell value. The database schema graph is created using entities as nodes of the graph and primary key and foreign key relationship as an edge. This schema graph is then used in the search phase, in which input keywords of a user are annotated on the graph using the inverted index. Then different join trees containing all the input keywords are generated. Each join tree is finally translated to an SQL query and is executed on the database to get a partial result. In this paper, we adapt this technique to our needs. We call the adapted version of this technique as VarDbXplorer.

3 System Architecture

Figure 1 presents the system architecture of the proposed AccKW framework. The system architecture consists of modules already present in the DBXplorer system and additional modules introduced by us. The modules added by us are differentiated from existing modules by applying shading in the system architecture block diagram presented in Figure 1. As illustrated in Figure 1, we extend the existing DBXplorer system by introducing three functions: Keyword Tuples Filter, Access Control Policy and Schema Graph Modifier. The functionality of each of the module is described in the following paragraphs.

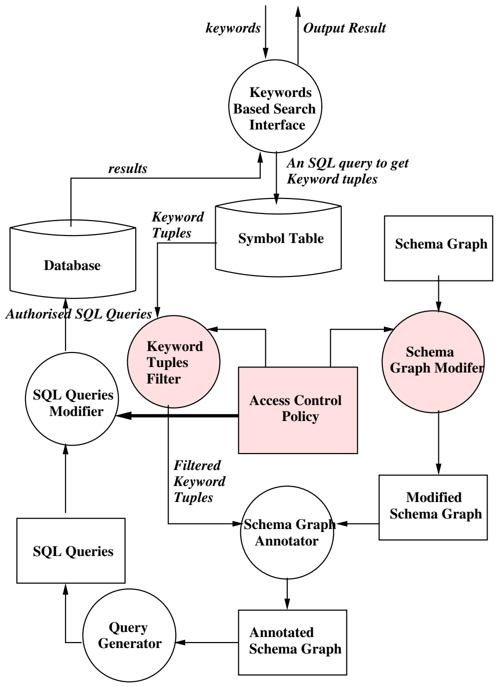


Fig. 1. Architecture of AccKW Framework

The Keywords Based Search Interface (KBSI) is the front-end module through which the application user interacts with the system. The end-user provides query keywords based on his or her information needs through the KBSI module. The search results obtained from the underlying Database is presented to the user through the same KBSI module. Typically, this module can be either a web-based interface or a form-based interface. The KBSI module is responsible for constructing an SQL query based on the supplied input keywords. The generated SQL query is then executed on the Symbol Table (a master index table implemented in the database) to get the keyword tuples. The resultant keyword tuples are passed on to the Keyword Tuples Filter (KTF) module.

The KTF module applies access control policy rules on these tuples and filters out keyword tuples having either unauthorized relations or unauthorized attributes. The filtered keyword tuples are eliminated and is not passed on to the next module in the pipeline for further processing. For example, consider a fine-grained access control rule which says that if the value of the age variable or field in a record is less than 30, then such a record should not be accessible to a specific user or role. In such a case, any keyword tuple consisting of age attribute and having the age value less than 30 will be filtered out from further processing. The module uses the application users context information to decide the access control rules for that user. Context information may include attributes such as role-id, user-id, time and location information.

The Schema Graph Modifier (SGM) module prunes the schema graph for a specific user using the access control rules applicable to the user. The pruning consists of operations such as relation node deletion, edge deletion, and specific node attributes deletion (i.e., node label modification). This module generates a Modified Schema Graph for the specific user.

The Schema Graph Annotator module takes filtered keyword tuples and modified schema graph as the inputs and produces an Annotated Schema Graph. This module annotates the keywords from the keyword tuples on the nodes of the schema graph. The Query Generator module generates all the valid join trees as described in [1] and forms the set of valid SQL queries. These valid SQL queries are then processed by the SQL Queries modifier (SQLM) module. The SQLM module takes each SQL query and the access control policy as inputs. It appends the predicates to the query by analyzing the relation and attribute information present in both the SQL query and the access control policy. For example, consider a rule wherein an access to a relation is authorized only if the user is accessing his or her information. In such a situation, a predicate checking for the match in user-id will be appended to the query. The queries generated by this module are called authorized SQL queries and are executed on the database. The result set of each query is forwarded to the KBSI module for final display to the application user.

4 Formal Model of Access Control

We assume that the access control policy has been defined as a set of rules. Each rule is a quad-tuple *(S, A, O, Auth)*. A rule defines an authorization decision $Auth$ (allow, deny) for a subject S for an object O (a set of tuples of a table in database) for an action A. For example, consider a rule *(manager, select, project, allow)*. The rule defines that a *manager* is allowed to access *project* table tuples. Subjects represent end-users as well as groups of users (wherein a group of users can be represented as a role). In general, an action on databases can be any common operation such as select, insert, update, and delete. In this paper, we investigate the select operation as it is the most widely used operation in keyword-based search over RDBMS.

The object component of the rule is specified as a pair: *(table_exp, predicate)*. The first tuple of the pair i.e., *table_exp* is a valid table expression which can be any valid projection of attributes of a table. The second tuple of the pair i.e., *predicate*, is either any valid WHERE clause condition of an SQL query as specified in [9] or a sequence of tables. If predicate is like a WHERE clause, then it can include exists sub-queries or conditions on more than one table attributes. For example, consider a rule *(salesman, select, (Employee.salary, salary >20000), deny)*. The rule denotes that a salesman cannot access salary of an employee if the salary of that employee is more than Rs 20,000. We specify following types of rules using above specified quad-tuple $(S, A, O, Auth)$:

1. **Entity Constraint.** An entity constraint defines authorization for a database relation for a subject. For an example, rule *(student, select, (faculty,), deny)* is an entity constraint and defines that a student is denied any type of access on faculty table. In other words, relation faculty can not occur in FROM clause of an SQL query issued by subject student.
2. **Context based Entity Constraint.** A context based entity constraint defines authorization for a relation by using predicates on attribute's value. As an example, rule *(student, select, (faculty,student.instructor_id != faculty.id), deny)* is a context based entity constraint and specifies that a student is not allowed to access faculty table tuples if faculty is not student's instructor.
3. **Entity-Attribute Constraint.** This constraint defines authorization for attributes of a database relation. As an example, rule *(student, select, (faculty.salary,), deny)* specifies that a student can not access any faculty's salary information. Further, salary attribute of faculty relation can not be used any where in the WHERE clause of an SQL query from student.
4. **Context-based Entity-Attribute Constraint.** This constraint uses predicates to define authorization for attributes of a relation. As an example, rule *(faculty, select, (student.marks, student.instructor_id != faculty.id), deny)* is a context-based entity-attribute constraint and specifies that a faculty can not access marks of a student if the faculty is not the instructor of that student.
5. **Entity Path Constraint.** An entity path constraint defines authorization on a set of relations in the database. For an example, rule *(student, select, (faculty, personal_detail), deny)* specifies that a student can not access faculty and personal_detail relation together in a single SQL query.

As described earlier in this Section, access control policy is a set of rules and we do not define any priority on the basis of ordering of rules in the policy specification. However, due to presence of both allow and deny decision there may occur a conflict for a query, in that case we give higher precedence to deny semantics.

5 Algorithm

We present an algorithm in this Section which enforces all the constraint types described in Section 4 in keyword-based search domain.

Algorithm 1. Algorithm to determine authorized SQL queries for a given set of keywords

```
Input:  1. Access control Policy, $A$
        2. Symbol Table, $ST$
        3. A schema graph, $G$
        4. User's Context Information, $U$
        5. Input keywords from a user $U$, $input\_keywords$

Output: Authorized SQL Queries set $output\_queries$

Method:
 1. $output\_queries$ = []
 2. $key\_tuples$ = []
 3. for all $keyword$ in $input\_keywords$ do
 4.     $k\_tuples$ = get $key\_tuples$ from $ST$
 5.     $key\_tuples$.extend($k\_tuples$)
 6. end for
 7. $filtered\_key\_tuples$ = prune-key-tuples($key\_tuples$, $A$, $U$)
 8. if $\forall\ k \in keywords,\ k \notin key\_tuples$ then
 9.     return $output\_queries$
10. end if
11. $pruned\_graph$ = prune-graph($G$, $A$, $U$)
12. $annotated\_gr$ = annotate-graph($pruned\_graph$,$filtered\_key\_tuples$
13. $valid\_trees$ = get-valid-join-trees($annotated\_gr$)
14. $pruned\_trees$ = prune-trees($valid\_trees$)
15. $valid\_queries$ = get-valid-queries($pruned\_trees$)
16. $output\_queries$ = rewrite-queries($valid\_queries$, $A$, $U$)
17. return $output\_queries$
```

We now describe functions prune-key-tuples(key_tuples, A, U), prune-graph(G, A, U), prune-trees($valid\_trees$) and rewrite-queries($valid\_queries$, A, U) invoked in Algorithm 1.

– **prune-key-tuples(key_tuples, A, U)**: In this function, all the keyword tuples which are not accessible to the user are removed. This function uses first four types of constraints to determine removable keyword tuples *(keyid,relid,attid)* in the following way:
  - A keyword tuple will be removed if it contains a relid value also present in object tuple of any entity constraint rule with deny authorization.
  - A keyword tuple will be removed if it contains a relid value also present in object tuple of context based entity constraint rule with deny authorization and the rule's predicate is true for keyid value.

  - A rule of type entity attribute constraint would remove a keyword tuple if the rule has deny authorization and rule's relation and attribute ids match with the tuple's relid and attid.
  - Context based attribute constraint rule with deny authorization removes a keyword tuple if the tuple's keyid value makes rule's predicate true and also matches in relid and attid value of the rule's object tuple.
- **prune-graph(**G**,** A**,** U**):** This function prunes the schema graph and uses first and third type of constraints in the following way:
  - An entity constraint rule with deny authorization removes a node with relation label from the schema graph.
  - A rule of type entity attribute constraint removes attribute from the label of the relation node specified in the rule. It also removes an edge from the schema graph if the edge label has the attribute specified in rule.
- **prune-trees(**$valid\_trees$**):** This function uses path constraint type rules to delete inaccessible join trees, i.e., if a join tree has relation nodes present in any of the path constraint rule then the join tree is deleted.
- **rewrite-queries(**$valid\_queries$**,** A**,** U**):** In this function, valid queries are rewritten to get the authorized set of queries. This step is similar to Oracle's VPD approach and uses context based entity constraint, and context based entity attribute constraint types rules. These constraints are very similar to dynamic ACL association tuple of instance set in Oracle's VPD.

We illustrate Algorithm 1 through an example.

Illustrative Example

We present below an example to demonstrate our algorithm. The schema for example is given as below:

Employee(eid, name, age, gender, pan, salary, phone, address)
Project(pid, name, budget, start_date, duration, details)
Emp-Proj(eid, pid, join_date, role)
Sponsors(sid, name, phone, type, investment, did)
Complains(cid, pid, date, details)

The schema graph for this schema would be as given in Figure 2.
Let us further assume the following authorizations for a user on this database.

1. *(clerk, select, (Complains,), deny)* : A clerk can not execute any operation on Complain relation.
2. *(clerk, select, (Sponsors.investment,), deny)* :A clerk can not execute select query on sponsors investment attribute.
3. *(clerk, select, (Employee,Project.budget$>$ 30 and Employee.eid=Emp-Proj.eid and Emp-Proj.pid=Project.pid), deny)* : A clerk can not access other employee information if employee is involved in at least one project with budget greater than 30 lakhs.

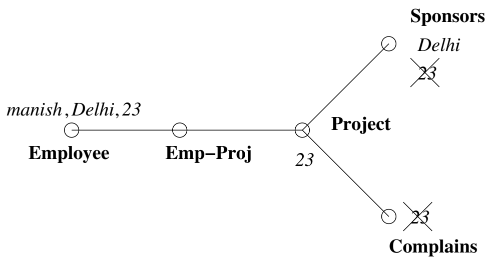


Fig. 2. Example Schema Graph

4. *(clerk,select, (Employee, Employee.age > 35), deny)*: A clerk can not access employee information if employee age is more than 35.

Let us now assume that the user with clerk role inputs Manish, Delhi and 23 keywords for information search. Keywords annotation in the schema graph for the above given authorization rules is shown in Figure 2. Figure shows crosses on relation Complains and keyword '23'. This is due to the reason that Complains relation and investment attribute of relation Sponsors are not accessible to the user. We have shown these crosses for the clarity purpose but in actual, these keywords will not be annotated to the schema graph due to filtering of keyword tuples and pruning of schema graph.

The following set of join trees as shown in Figure 3 will be generated in this case.

The authorized SQL queries would be as given below:

– Select name, address, age from Employee where name = 'manish' and address = 'Delhi' and age = 23 and not (age > 35)
– Select name, age, address from Employee, Emp-Proj, Project, Sponsors where Employee.eid = Emp-Proj.eid and Emp-Proj.pid = Project.pid and

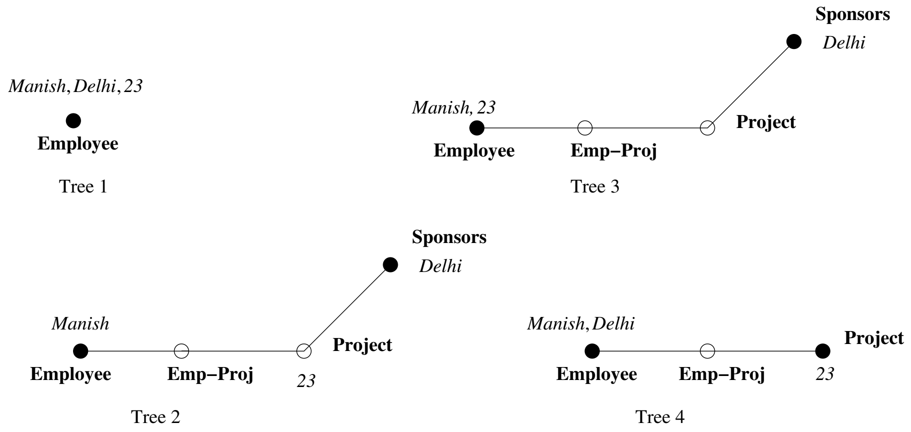


Fig. 3. Join Trees

Project.pid = Sponsors.pid and Employee.name = 'manish' and Employee.age = 23 and Sponsors.address = 'Delhi' and not (Project.budget $>$ 20) and not (age $>$35)

- Select name, budget, address from Employee, Emp-Proj, Project, Sponsors where Employee.eid = Emp-Proj.eid and Emp-Proj.pid = Project.pid and Project.pid = Sponsors.pid and Employee.name = 'manish' and Project.budget = 23 and Sponsors.address = 'Delhi' and not (Project.budget $>$ 30) and not (age $>$ 35)
- Select name, budget, address from Employee, Emp-Proj, Project where Employee.eid = Emp-Proj.eid and Emp-Proj.pid = Project.pid and Employee.name = 'manish' and Project.budget = 23 and Employee.address = 'Delhi' and not (Project.budget $>$ 30) and not (age $>$ 35)

6 Experimental Study

Our performance evaluation consists of experiments on our implementation of the VarDbXplorer (which is an adapted version of DBXplorer) and Access Control on TPC-H [16] database on a HP desktop machine. The machine is a Intel(R) Core Due CPU 2.66GHz with 3GB of main memory. The experiment is implemented in Python version 2.5.2 and uses MySQL database system [17] at the backend.

We have used TPC-H database [16] for performance study as TPC-H database use is common in researchers working on keywords based search on RDBMS [1,3]. The sizes of the database are 10 MB, 100 MB, and 1GB generated using scaling factor of .01, .1 and 1 with dbgen utility of TPC-H benchmark.

We adapt DbXplorer keyword-based search strategy [1] for our scenarios and term that as VarDBXplorer. The master index or symbol table is implemented as an inverted index table in MySQL which is a column based approach [1]. We have written a python program to make a master index for every value present in the database. Each value in the database is stored with its occurrence information in the symbol table. A value is a single word and is an alphanumeric expression which is obtained from an attribute value after lexical analysis process. For an example, an address value is decomposed into words using space as a delimiter and removing control characters. A value may occur more than once in an attribute of a relation or in more than one tuple of a relation. An entry of symbol table can be represented as a 3-tuple $< value, T_i, A_{ij} >$, where $value$ is a keyword, T_i is a relation name and A_{ij} is j_{th} attribute in relation T_i. To select keywords with a particular frequency, we created another table *key_stats*, which is derived from master index table by executing a *group by* SQL query on master index table. Each tuple of this table is a (value, frequency) pair. Here, frequency specifies the number of attributes in which the value occurs.

We measure the efficiency of applying access control policy in the early phases of search in terms of performance parameters: time of execution (T_{exec}) in seconds and number of generated queries (Num_{ge}). For that, we specified five types of access control policies which vary in terms of access restrictions for a user. Policy-1 has the least restriction and Policy-5 has the maximum restriction among these 5 policies. Each policy rule is of the type defined in 4, i.e., entity constraint, context based entity constraint etc. We considered different database sizes for evaluation, for that we have generated data using different scaling factor of dbgen utility, i.e., 10 MB, 100 MB and 1000 MB using scaling factor of .01, .1 and 1. We have also studied the effect on time and number of generated queries for variation in number of input keywords. For that we have used 2 to 6 keywords in a query. We have not used more than 6 keywords as it is quite uncommon that a user search has more than 6 keywords. We have also studied the effect of variation in keyword selectivity. Keyword selectivity of a keyword is defined as number of attributes in the database in which the keyword is present. The parameters table is given as Table 1

Table 1. Parameters Table

PARAMETER	DEFAULT VALUE	VARIATIONS
Access Control Policy Quantifier (λ)	3	1,2,4,5
Number of Keywords (n)	5	2,3,4,6
Selectivity of Keywords (F)	5	1 to 4, 6 to 10
Database Size (S)	1 GB	100 MB, 10 MB

Effect of Variation in Access Control Policy

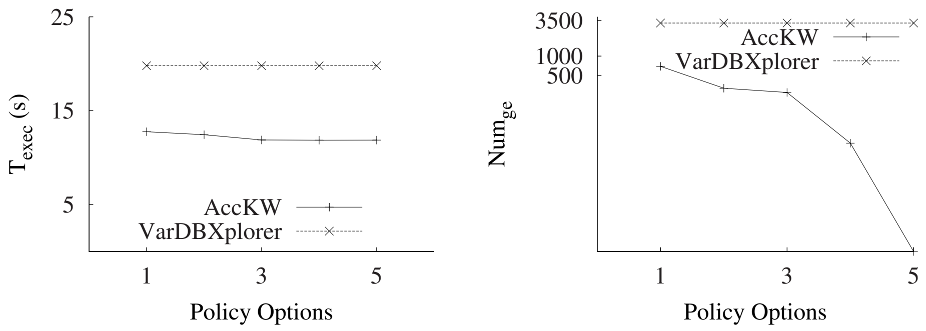
The performance graphs of effect in variation of access control policy is shown in Figure 4a and Figure 4b. As explained earlier, the policies from policy-1 to policy-5 varies in terms of strictness. Policy-1 is the most relaxed one and policy-5 is the strictest one among the five. We see that as the access control policy becomes more restrictive for a user, it reduces the execution time T_{exec} to generate authorized queries as well as the number of authorized queries. Figure 4a shows that VarDBXplorer approach always takes more time as compared to AccKW. Figure 4b shows the effect of different access control policies on Num_{ge} and has log scale for Y-axis. It shows that the value of Num_{ge} goes on decreasing as the access control policy becomes more restrictive.

We find that this is due to pruning of schema graph size and authorized tuple set in the early phases. This reduction in the size of the schema graph results into less processing and hence reduction in both time T_{exec} and number of queries Num_{ge}.

Effect of Variations in the Number of Keywords

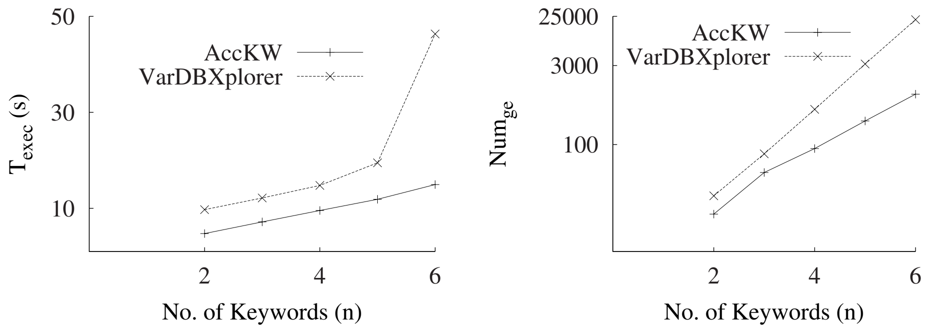
We did this experiment by selecting 2 to 6 keys at random from the master index which has selectivity (F) equal to 5.

The performance graphs of this experiment are shown in Figure 5a and Figure 5b. Figure 5b has log scale for Y-axis. The experiment shows that T_{exec} and Num_{ge} increases as the number of keywords (n) increases for a given access control policy option (default access control policy 3). This is due to generation of more number of tuples with the increase in number of keywords, that need to be annotated with the pruned schema graph. More keywords annotation to schema graph leads to more number of queries generation as well as more execution time.



(a) Effect on T_{exec} **(b)** Effect on Num_{ge}

Fig. 4. Effect of variation in Access Control Policy



(a) Effect on T_{exec} **(b)** Effect on Num_{ge}

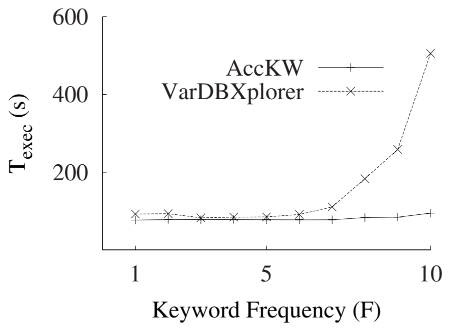
Fig. 5. Effect of Variation in Number of Keywords

Effect of Variations in Keyword Selectivity

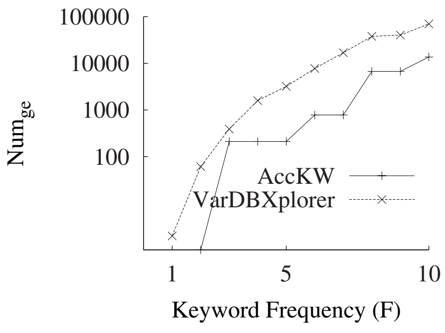
The performance graphs of this experiment are shown in Figure 6a and Figure 6b (with log scale on Y-axis). The graphs show that the increase in the selectivity of keywords increases the T_{exec} as well as Num_{ge} for a given access control policy. The reason for this is similar to the previous experiment that the increase in selectivity of a keyword results into more tuples generation. This increase in tuples in the tuple set results in more number of keys to be annotated with the schema graph and hence increase in execution time Figure 6a and generated authorized queries Figure 6b.

Effect of Variations in the Database Size

Figure 7a and Figure 7b shows the performance results of variation in database size on T_{exec} and Num_{ge} respectively. Graph in Figure 7b uses log scale for Y-axis. As

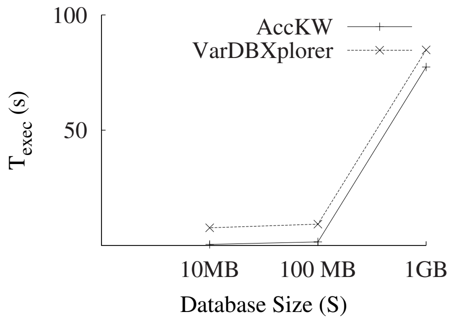


(a) Effect on T_{exec}

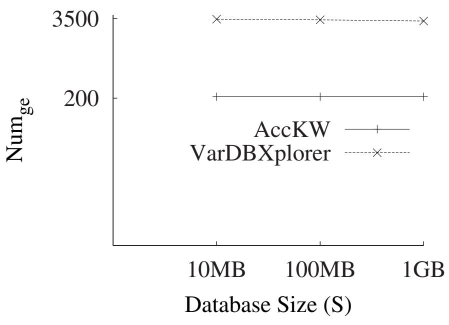


(b) Effect on Num_{ge}

Fig. 6. Effect of variations in keyword selectivity



(a) Effect on T_{exec}



(b) Effect on Num_{ge}

Fig. 7. Effect of variations in the database size

discussed earlier, we generated data of different sizes using scaling factor parameter of dbgen utility of TPC-H benchmark for this experiment.

The result shows a minimal effect of data sizes on execution time and number of queries for a given access control policy, keyword selectivity and number of keys. This constant difference is due to the difference in time to get the keyword-tuples from the master index. After keyword tuple selection, the process of authorized query generation will be same for each data size version.

7 Conclusions

This paper proposes a novel framework called AccKW which enforces access control in keyword-based search over RDBMS. Solutions for access control enforcement in the domain of keyword-based search over RDBMS is a relatively unexplored area and the work presented in this paper addresses the stated research gap. The paper proposes a solution framework and also discusses issues and challenges regarding access control in this domain. The proposed framework is implemented and emperically evaluated. The paper presents performance results from different perspectives such as efficiency and performance impact as a result of variation in number of keywords, access control policy, database size, and keyword selectivity. Based on the empirical evaluation and simulation results, we conclude that the proposed framework outperforms the naive approach in most of the cases.

References

1. Agrawal, S., Chaudhuri, S., Das, G.: Dbxplorer: A system for keyword-based search over relational databases. In: ICDE 2002: Proceedings of the 18th International Conference on Data Engineering, Washington, DC, USA, p. 5. IEEE Computer Society, Los Alamitos (2002)
2. Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, P., Sudarshan, S.: Banks: browsing and keyword searching in relational databases. In: VLDB 2002: Proceedings of the 28th international conference on Very Large Data Bases, VLDB Endowment, pp. 1083–1086 (2002)
3. Hristidis, V., Papakonstantinou, Y.: Discover: keyword search in relational databases. In: VLDB 2002: Proceedings of the 28th international conference on Very Large Data Bases, VLDB Endowment, pp. 670–681 (2002)
4. Hulgeri, A., Nakhe, C.: Keyword searching and browsing in databases using banks. In: ICDE 2002: Proceedings of the 18th International Conference on Data Engineering, Washington, DC, USA, p. 431. IEEE Computer Society, Los Alamitos (2002)
5. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB 2005: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, pp. 505–516 (2005)
6. Koutrika, G., Simitsis, A., Ioannidis, Y.: Précis: The essence of a query answer. In: International Conference on Data Engineering, vol. 0, p. 69 (2006)
7. Simitsis, A., Koutrika, G., Ioannidis, Y.: Précis: from unstructured keywords as queries to structured databases as answers. The VLDB Journal 17(1), 117–149 (2008)
8. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. In: SIGMOD 2004: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 551–562. ACM, New York (2004)
9. Murthy, R., Sedlar, E.: Flexible and efficient access control in oracle. In: SIGMOD 2007: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 973–980. ACM, New York (2007)
10. Olson, L.E., Gunter, C.A., Madhusudan, P.: A formal framework for reflective database access control policies. In: CCS 2008: Proceedings of the 15th ACMconference on Computer and communications security, pp. 289–298. ACM, New York (2008)
11. Shin, H., Atluri, V.: Spatiotemporal access control enforcement under uncertain location estimates. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security XXIII. LNCS, vol. 5645, pp. 159–174. Springer, Heidelberg (2009)
12. Chaudhuri, S., Dutta, T., Sudarshan, S.: Fine grained authorization through predicated grants. In: ICDE 2007: Proceedings of the 23rd International Conference on Data Engineering, Istanbul, Turkey, pp. 1174–1183. IEEE, Los Alamitos (2007)
13. Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan, S., Rjaibi, W.: Extending relational database systems to automatically enforce privacy policies. In: ICDE 2005: Proceedings of the 21st International Conference on Data Engineering, Washington, DC, USA, pp. 1013–1022. IEEE Computer Society, Los Alamitos (2005)

14. Kabra, G., Ramamurthy, R., Sudarshan, S.: Redundancy and information leakage in fine-grained access control. In: SIGMOD 2006: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pp. 133–144. ACM, New York (2006)
15. Lefevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., DeWitt, D.: Limiting disclosure in hippocratic databases. In: VLDB, pp. 108–119 (2004)
16. TPC-H decision support benchmark (Transaction Processing Council), http://www.tpc.org/
17. MySQL Database, http://www.mysql.com/