Multi-tenant Database Access Control

Haitham Yaish, Madhu Goyal
Centre for Quantum Computation & Intelligent Systems
Faculty of Engineering and Information Technology
University of Technology, Sydney
P.O. Box 123, Broadway NSW 2007, Australia
haitham.yaish@student.uts.edu.au, madhu@it.uts.edu.au

Abstract— **Storing data in the cloud is a new multi-tenant database solution that has recently emerged to deliver database for multiple users, who can store and access their data over the internet. This multi-tenant database designed to be used by multiple tenants and each tenant may have multiple users. Therefore, this database type demands a special multi-tenant access control model, which provides an access control not only for multiple tenants, but also for multiple users per tenant. In this paper, we are proposing a multi-tenant access control model based on a multi-tenant database schema called Elastic Extension Tables (EET). In this model, we define access control data architecture, and the EET access grants which can be granted to tenants' users. Moreover, we propose an access control algorithm, which allows users to access the data granted to them based on a number of groups or roles assigned to these users.**

Keywords- Cloud Computing, Access Control, Multi-tenancy, Multi-tenant Database, Elastic Extension Tables.

I. Introduction

The growth of multi-tenant Cloud Computing services draws attention to security challenges, which are emerging due to the cloud vendor's resource sharing [13]. It is unlikely that the cloud users would risk their data and their computing applications over the cloud in favour of reducing the Total Cost of Ownership (TCO), or using a flexible cloud service, unless the cloud service providers provide reliable and secure services [18]. Outsourcing data to the cloud is one of the critical security challenges because this data is accessed among a large number of users from different organisations [18]. There are three data isolation approaches applied to the cloud. The first approach is called Separate Database, which is the simplest data isolation approach that stores each tenant data in a separate database. The second approach is called Shared Database - Separate Schema, which hosts all the tenants in the same database instance, but each tenant has his own database schema. The last approach is called Shared Database - Shared Schema, which allows tenants to store their data in the same database and same schema. In other words, a given table can store different table rows for different tenants, and a tenant ID column will differentiate and isolate the tenant's data [8],[9],[12],[19]. In this paper, we are focusing on the Shared Database - Shared Schema isolation approach, which requires a high degree of data isolation and configuration to ensure the security and privacy of tenants' shared data. This multi-tenant data approach consists of two data types: shared tenants' data and tenants' isolated data, by combining these data together, tenants can have the complete data they need [9],[12]. These multi-tenant data isolation approaches have challenges in supporting highly manageable database schema, and in providing configurable database fields [5],[11],[22],[26]. These challenges are (1) isolating tenants data by ensuring that each tenant can access only his own data, (2) ensuring that the tenants' data is robust and secure, (3) optimizing database performance [10],[5],[8],[27], (4) designing a database structure which works with different business domain applications [1], and (5) fulfilling different tenants' business requirements by using a tenant-aware data management based on Shared Database - Shared Schema approach [19].

There are various models of multi-tenant database designs and techniques, which have studied and implemented to overcome multi-tenant database challenges like Private Tables, Extension Tables, Universal Table, Pivot Tables, Chunk Table, Chunk Folding, and XML [11],[14],[16],[24],[25]. Nevertheless, these techniques are still not overcoming multi-tenant database challenges [24]. Based on this analysis, we have proposed a novel multi-tenant database schema design to create and configure multi-tenant applications, by introducing an Elastic Extension Tables (EET), which consists of Common Tenant Tables (CTT), Extension Tables (ET), and Virtual Extension Tables (VET) [14]. This design enables tenants creating and configuring their own virtual database schema including a required number of tables and columns, virtual database relationships, and assigning suitable data types and constraints for columns during multi-tenant application run-time execution [14]. Furthermore, EET allows tenants to choose from three database models. The first model is multi-tenant relational database. The second model combines multi-tenant relational database and virtual relational database. The third model is a virtual relational database.

In this paper, we are proposing an access control method called Elastic Extension Tables Access Control (EETAC). This method permits each tenant in the multi-tenant database to have several users with different types of grants to access the tenant's data. Further, we propose an access control algorithm which allows users to access their data that stored in columns and rows, and granted to them. Furthermore, we ran two experiments to verify the practicability of granting a tenant's user accessibility on a tenant's table columns and rows, by using EETAC method and the proposed Elastic Extension Tables Proxy Service (EETPS) [15]. In these experiments, we found that the cost of executing a query for a user who is granted access to fewer numbers of the table columns or rows is less than the

978-0-7695-5096-1/13 $31.00 © 2013 IEEE
DOI 10.1109/CSE.2013.131

cost of a user who is granted access to more numbers of the table columns or rows.

The rest of the paper is organized as follows. Section II reviews the related work. Section III describes the elastic extension tables. Section IV describes the elastic extension tables proxy service. Section V describes the elastic extension tables access control method. Section VI describes the columns and rows access grant algorithm. Section VII gives our experimental results. Section VIII concludes this paper.

II. Related Work

Access control is a security topic which was started back in the 1960s [18], and various access control models have proposed since then such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC) [2],[16]. David Ferraiolo and Richard Kuhn are the first who proposed the RBAC model in 1992, which introduces the role as a new concept to associate users to one or more roles, which are associated with one or more permissions [7][16].

Siebel Systems [17] states that the present single-organisation access control model is not suitable for multi-tenant database. Accordingly, it has proposed a multi-tenant role based access control method, which allows having a plurality of tenants, where each tenant is the owner of a separate virtual database. This method supports an access control subsystem for multiple users who are seeking a data access, where each of the users has at least one organizational access attribute, and the data are stored in an underlying database. The database is divided into files; the files are divided into records within the file, and the individual records are divided into fields. This method is based on partitionability of the individual database files in the database, which are based upon an attribute of ownership and/or a granted access control.

IBM DB2 has provided several approaches of data access in Database Management System (DBMS) level including views, label-based access (LBAC), and row and column access control (RCAC). The views approach adds more management overhead because this approach uses views instead of tables. The LBAC approach is to create labels on tables and columns, and these labels are granted to users or groups. IBM introduced in DB2 V10 the RCAC approach, which represents a second layer of security, that works with the current table security model. This approach permits groups and users to access particular rows in a table and specifies the data accessed from some or all the table's columns. Additionally, some columns' data are masked with nulls, a user defined mask, or a column mask which restricts a user from accessing data within a column [6].

Salesforce has designed and developed a storage model to manage its virtual database structure by using a set of metadata, universal data table, and pivot tables that are converted to objects, objects' fields and relationships, and other object definition characteristics which are tracked by Universal Data Dictionary (UDD) [3]. Saleforce is using an access control method wherein each tenant may have one or more users. Each user or group of users can have different types of access grants, which permit them to access different rows including (1) the user rows, (2) rows for users below the user in a role hierarchy, (3) rows that are shared by a group which the user belongs to, and (4) rows that are manually shared by another user or group of users [4] [23].

In this section, we have discussed different multi-tenant role based access control methods, and different approaches to access data from a table columns and rows. However, these access control methods and approaches have similar assumptions, but for multi-tenant database designs other than EET multi-tenant database schema. Therefore, we introduce in this paper an access control method which is suitable for the proposed EET multi-tenant database schema [14].

III. Elastic Extension Tables

The proposed multi-tenant database schema is a new way of designing and creating a multi-tenant database, which consists of three types of tables. The first type is Common Tenant Tables (CTT) which are shared between tenants who are using a single instance of the multi-tenant database. These are physical relational tables, which can be applied to any business domain database such as customer relationship management (CRM), accounting, human resource (HR), or any other business domain. The second type is Virtual Extension Tables (VET), which allow tenants to extend on the existing business domain database, or having their own configurable database through creating their virtual database structures from scratch by creating (1) virtual database tables, (2) virtual database relationships between the virtual tables, and (3) other database constraints. The third type is Extension Tables (ET), which consists of eight tables that are used to construct VETs [14]. The data architecture details of these eight tables are listed below and shown in Fig. 1.

- The db_table Extension Table: This table allows a tenant to create virtual tables and give them unique names.
- The table_column Extension Table: This table allows a tenant to create virtual columns for a virtual table stored in the "db_table" extension table.
- The table_row Extension Tables: The row extension tables store records of virtual extension columns in three separate tables. These tables are separated in order to store small data values in the "table_row" extension table such as NUMBER, DATE-and-TIME, BOOLEAN, VARCHAR and other data types. Whereas the large data values stored in two other tables. The first one is the "table_row_blob" extension table, which stores a uniform resource locator (URL) for the Binary Large Objects (BLOB). The second one is the "table_row_clob" extension table, which stores Character Large Objects (CLOB) values for virtual columns with TEXT data type.
- The table_relationship Extension Table: This table allows tenants to create a virtual relationship for their virtual tables with any of CTTs or VETs.
- The table_index Extension Table: This table is used to add indexes to virtual columns, which reduce the

query execution time when a tenant retrieves data from database tables.

- The table_primary_key_column Extension Table: This table allows tenants to create single or composite virtual primary key for virtual extension columns stored in the "table_column" extension table.

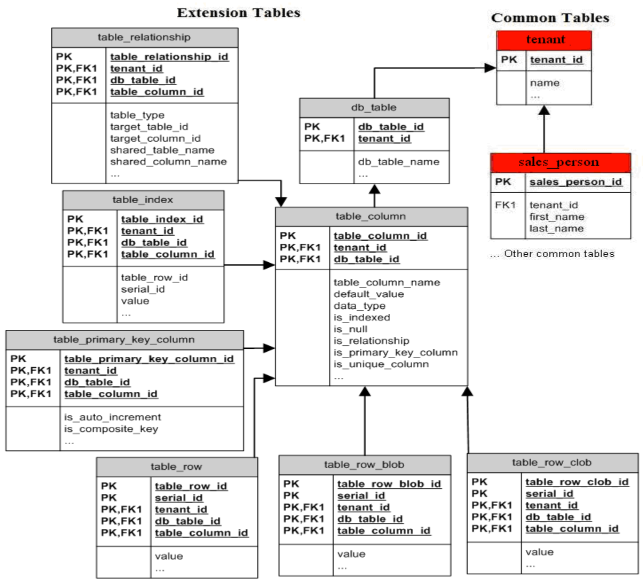


FIG. 1 ELASTIC EXTENSION TABLES (EET) [15]

IV. ELASTIC EXTENSION TABLES PROXY SERVICE

We have proposed a multi-tenant database proxy service to combine, generate, and execute tenants' queries by using a code base solution, which converts multi-tenant queries into traditional database queries, and execute these traditional database queries in any Relational Database Management System (RDBMS) [15]. This service has two objectives, first, to enable tenants' applications retrieve rows from CTTs, retrieve combined rows from two or more tables of CTTs and VETs, or retrieve rows from VETs. Second, to spare tenants from spending money and efforts on writing Structured Query Language (SQL) queries and backend data management codes by simply calling functions from this service, which retrieves simple and complex queries including join operations, union operations, filtering on multiple properties, and filtering of data based on subqueries results.

This service gives tenants the opportunity of satisfying their different business needs and requirements by choosing from any of the following three database models. First, Multi-tenant relational database. This database model allows tenants to use a ready relational database structure for a business domain database without any need of extending on the existing database structure, and this business domain database can be shared between multiple tenants and differentiate between them by using a Tenant ID. This model can be applied to any business domain database. Second, Combined multi-tenant relational database and virtual relational database. This database model allows tenants to use a ready relational database structure of a particular business domain with the ability of extending on this relational database. By adding more virtual database tables, and creating virtual relationships between them to combine the virtual tables with the existing relational database table structure. Third, Multi-tenant Virtual relational database. This database model allows tenants to use their own configurable database through creating their virtual database structures from scratch, by creating virtual database tables, virtual database relationships between the virtual tables, and other database constraints to satisfy their special business requirements for their business domain applications.

The EETPS provides functions which allow tenants to build their web, mobile, and desktop applications without the need of writing SQL queries and backend data management codes. Instead, retrieving their data by simply calling these functions, which return a two dimensional array (Object [*n*] [*m*]). Where *n* denotes the number of array rows that represents a number of retrieved table rows, and *m* denotes the number of array columns that represents a number of retrieved table columns for a particular virtual table. This two dimensional array stores the virtual table row in a structure, which is similar to any physical database table, and in return will facilitate accessing virtual rows from any VET as well as CTT. These functions were designed and built to retrieve tenants' data from the following tables:

- One table, either a CTT or a VET.
- Two tables, which have one-to-one, one-to-many, many-to-many, or self-referencing virtual relationships. These relationships can be between two VETs, two CTTs, or one VET and one CTT.
- Two tables, which may have or not have a relationship between them, by using different types of joins including left join, right join, inner join, outer join, left excluding join, right excluding join, and outer excluding join. The join operation can be used between two VETs, two CTTs, or one VET and one CTT.
- Two tables or more, which may have or not have a relationship between them, by using the union operator that combines the result-set of these tables whether they are CTTs or VETs.
- Two or more tables, which have a relationship between them, by using filters on multiple tables, or filtering the data based on the results of subqueries.

Moreover, the EETPS functions have the capabilities of retrieving data from CTTs or VETs by using the following query options: specifying query SELECT clause, specifying query WHERE clause, specifying query LIMIT, using single or composite primary keys, retrieving BLOB and CLOB values, logical operators, arithmetic operators, aggregate functions, and mathematical functions.

V. ELASTIC EXTENSION TABLES ACCESS CONTROL

In this section, we are defining the access control data architecture which is based on Elastic Extension Tables. In addition, we are defining Elastic Extension Tables access grants, which are granted to tenants' users to access a table columns and rows stored in the elastic multi-tenant database schema.

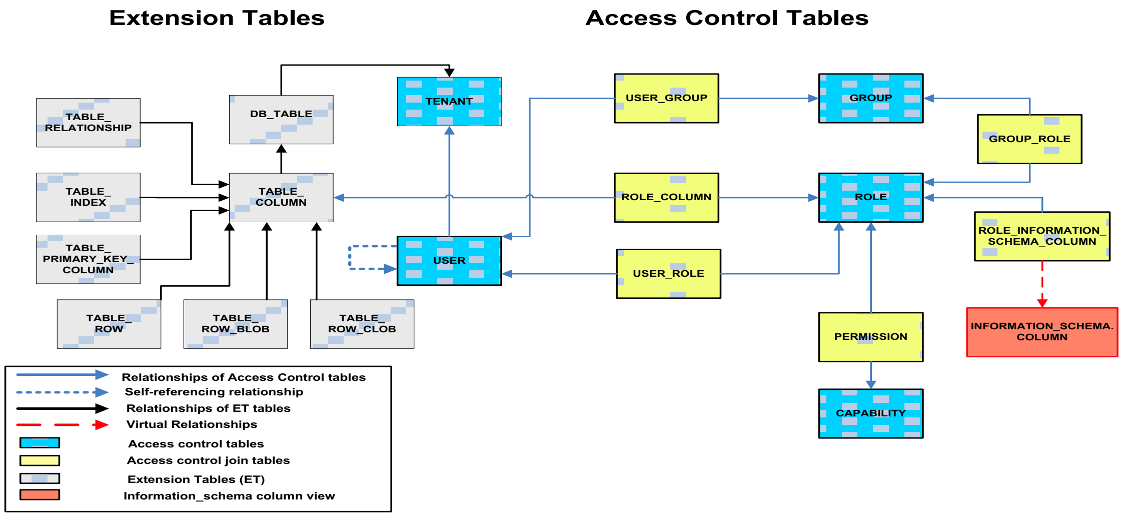


Fig. 2 EET Access Control Data Architecture

A. Access Control Tables

There are three types of EETAC tables to store tenants' access control configurations. The first type is the main entity tables of access control data architecture. The second type is the join tables, and the third type is the Information_Schema view. These tables are listed below:

1) Access Control Main Entity Tables: These tables are listed below and illustrated in a blue colour in Fig. 2.

- **Tenant Table**. This table stores the tenants' information details. The Tenant ID column of this table is used to isolate the tenants' data, which is stored in a CTT or a VET. This isolation is applied by having a master-details relationship between this table and any CTT or VET. By adding the Tenant ID column as a reference column to a CTT or a VET, to refer the data in any of these tables to an existing tenant who has a unique Tenant ID in the Tenant table.

- **User Table**. Each tenant in the multi-tenant database can have multiple users accessing the tenant's data. This table can store three types of these users. The first type is an admin or a super user. The second type is a single user. The third type is a parent-child user, which allows tenants to have an admin or a super user and assign to this user one or more users by using the self-referencing relationship, which this table has. Each user type can have different levels of database access based on groups and/or roles they associated with.

- **Group Table**. This table is used to define different levels of a tenants' group. Any of these groups logically associating users with similar data access needs. Once a tenant group is defined, some roles, which have granted permissions assigned to that group. Then, any tenant user who is associated with the group inherits all of the permissions granted to that group.

- **Role Table**. Tenants' users and groups can have roles, which are granted permissions needed to perform database activities on a tenant's data among multiple tenants' data, which is stored in a multi-tenant database. In addition, this role table is granted permissions for both types of tables CTTs and VETs.

- **Capability Table**. This table allows tenants to authorise their user privileges to any operation performed upon data. These operations have different access levels including full access, read/write access, read access, and other access types.

2) Access Control Join Tables: These tables are used to establish many-to-many relationships between the access control main entities, and the join tables listed below and illustrated in a yellow colour in Fig. 2.

- **User_Group Table**. This join table is used to allocate an access classification level between groups and tenants' users. Typically this allocation is used to group users like administrator users, super users, or public users.

- **Group_Role Table**. This join table is used to authorise a group of users to access one or more database access roles, and any user who is allocated to this group will inherit all the permissions which are granted to the group.

- **User_Role Table**. This join table is used to authorise a user to access one or more database access roles.

- **Role_Column Table**. This join table is used to allocate a role to access some or all the tenant's VET columns. Once the tenant has this allocation, he/she can add business rules to access some or all rows from these VET columns. The details of these business rules are presented in section B.

- **Permission Table**. This join table is used to allocate roles to different kinds of database access capabilities such as full access, read/write access, read access, and other access types.

- **Role_Information_Schema_Column Table**. The purpose of this join table is similar to the purpose of the Role_Column Table. However, this join table allocates a role to access some or all the tenant's CTT columns, once the tenant has this allocation, he/she can add business rules to access some or all rows from these CTT columns.

3) Information_Schema.column View: This view allows getting information about columns for tables and views within the PostgreSQL database [20]. This Information_Schema view is also used by databases like Oracle, Mysql, and others. We are using this view in the EETAC data architecture to give access grants for tenants' users to access CTTs columns. This view is illustrated in a red colour in Fig. 2.

B. Elastic Extension Tables Access Grants

EETAC has two main types of grants. The first type is Group Access Grant, in which a user is assigned to a group, and this user inherits all of the roles granted to that group. The second type is Role Access Grant, in which a user is assigned to a role assigned to a user directly or inherited via a group. To allow the user to access CTTs or VETs in the EET database schema. The two types of grants are shown in Fig. 3. The group access grant is illustrated in the blue arrows, and the role access grant is illustrated in the grey arrow.

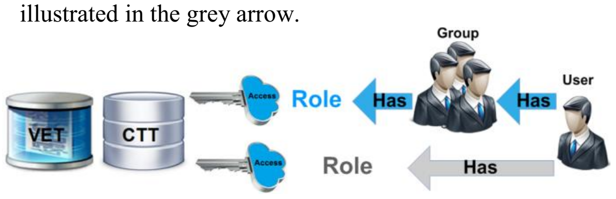


Fig. 3 EET access control grants

These two main types of grants have two subtypes of grants. Table Columns Access Grant, and Table Rows Access Grant. These access grants control the access of multi-tenant data in CTTs and VETs. Since CTTs and VETs are using the Tenant ID to isolate the tenants' data in EET multi-tenant database and divide it into partitions, then each single tenant can have his/her own partitions to store their own data. Moreover, these partitions are divided by tenants' users according to these two grants, which are discussed in details, in the following two points:

- Table Columns Access Grant. This grant allows tenants to give user permissions to access some or all columns of a CTT or a VET. These permissions can restrict tenants' users from accessing some or all columns of a table. For example, Fig 3 is showing two types of users, the first user is a super user called Adam, who has roles which can access all the table's columns of a table. The second user is Abraham who has roles which can access only three columns of the same table that Adam can access. In addition, this grant helps in deciding the optimal query execution plans, by knowing whether a user can access all, or some of a table columns. In the case when a user can access some of the table's columns, these columns can be retrieved from the table by generating a query structure different from the structure of retrieving all the columns.

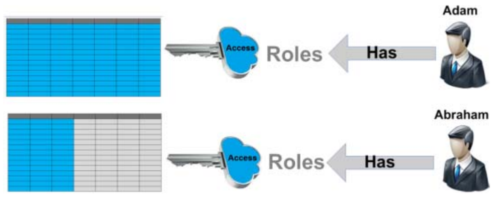


Fig. 4 Table columns access grant

- Table Rows Access Grant. This grant allows tenants to offer user permissions to access some or all rows of a CTT or a VET. These permissions can restrict tenants' users from accessing some or all rows of a table. For example, Fig. 5 is showing the same users who were shown in Fig.4, but this time Adam has roles that can access all the table's columns and rows, and Abraham has roles that can access all columns, but only some rows of the same table that Adam can access. Also, this grant optimizing the query execution by considering the number of rows which are accessed by a user, and generating a query structure different from the structure of retrieving all the table rows.

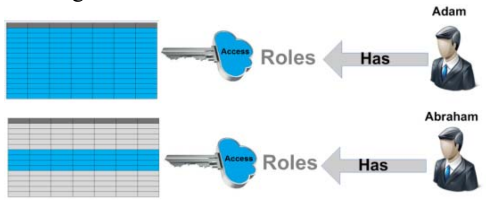


Fig. 5 Table rows access grant

VI. Columns and Rows Access Grant Algorithm

In this section, we are presenting an access control algorithm which is used to allow tenants' users to access the data granted to them when they assigned roles permitted to access columns and rows of a CTT or a VET. This algorithm invokes two other subsidiary algorithms that presented in this section.

A. Get User Query Access Main Algorithm

This access control main algorithm defines the SELECT and the WHERE clauses and returns their values, by determining which columns and rows a user can access. These SELECT and WHERE clauses are used to construct the user's query statement, which retrieves data from a CTT or a VET based on access grants assigned to the user. The details of this algorithm are shown in Algorithm 1.

Definition 1 (Get User Query Access). T denotes a tenant ID. U denotes a tenant's user. B denotes a table name. S denotes a string of the SELECT clause parameter. Q denotes the table type whether it is a CTT or a VET. B_{column} denotes a CTT or a VET columns. $\varnothing$ denotes an empty set. R_{return} denotes a set of user roles returned by calling GetUserRoles algorithm. C_{return} denotes a row matrix with 1 row and 2 columns which has two elements, the first one is $C_{return\,0,0}$ that denotes the tenant's user SELECT clause attributes, and the second one is $C_{return\,0,1}$ that denotes the access control part of the tenant's user Where clause. The values of C_{return} returned by calling GetUserColumns algorithm. UQA_{select} denotes a string of query SELECT clause. UQA_{where} denotes a string of query WHERE clause. UQA_{return} denotes a row matrix with 1 row and 2 columns which has two elements, the first one is $UQA_{return\,0,0}$ that stores into it the value of UQA_{select}, and the second one is $UQA_{return\,0,1}$ that stores into it the value of UQA_{where}.

Algorithm 1: GetUserQueryAccess (T, U, B, S, Q)

```
Input: T, U, B, S, and Q
Output: $UQA_{return}$
1.  if Q = CTT then
2.      $B_{column}$ ← retrieve the number of columns for a CTT from
        role_information_schema_column table by using T and B query
        filters
3.  else
4.      $B_{column}$ ← retrieve number of columns for a VET from
        table_column extension table by using T and B query filters
5.  end if
6.  $R_{return}$ ← getUserRoles(T, U, B, Q)
7.  $C_{return}$ ← getUserColumns (T, U, B, Q, $R_{return}$)
8.  if size of $B_{column}$ = size of $C_{return\,0,0}$ then
9.      if S = ∅ then
10.         $UQA_{select}$ ← ∅
11.     else
12.         $UQA_{select}$ ← S
13.     end if
14. else
15.     if S = ∅ then
16.         $UQA_{select}$ ← ∅
17.     else
18.         $UQA_{select}$ ← S ∩ $C_{return\,0,0}$
19.     end if
20. end if
21. if $C_{return\,0,1}$ ≠ ∅ then
22.     $UQA_{where}$ ← $C_{return\,0,1}$
23. else
24.     $UQA_{where}$ ← ∅
25. end if
26. $UQA_{return\,0,0}$ ← $UQA_{select}$
27. $UQA_{return\,0,1}$ ← $UQA_{where}$
28. Return $UQA_{return}$
```

B. Get User Roles Subsidiary Algorithm

This access control subsidiary algorithm is used to retrieve the tenant's user roles assigned to CTT or VET columns. The details of this algorithm are shown in Algorithm 2.

Definition 2 (Get User Roles). T denotes a tenant ID. U denotes a tenant's user. B denotes a table name. S denotes a string of the SELECT clause parameter. Q denotes the table type whether it is a CTT or a VET. R_{table} denotes a set of role ID values assigned for a CTT or a VET. R_{group} denotes a set of role ID values assigned to the tenant's user groups. R_{user} denotes a set of role ID values assigned to the tenant's user. R_{id} denotes a role ID. R_f is a flag used in the algorithm to check whether any of R_{user} and R_{group} elements exist in R_{table}. R_{return} denotes a set of role ID values, which the tenant's user can access. ∅ denotes an empty set.

Algorithm 2: GetUserRoles (T, U, B, Q)

```
Input: T, U, B, and Q
Output: $R_{return}$
1.  if Q = CTT then
2.      $R_{table}$ ← retrieve roles assigned to a CTT from
        role_information_schema_column table by using T and B query
        filters
3.  else
4.      $R_{table}$ ← retrieve roles assigned to a VET from the role_column
        table by using T and B query filters
5.  end if
6.  $R_{group}$← retrieve roles assigned to U from the group_role table by
    using T, and U query filters
7.  $R_{user}$ ← retrieve roles assigned to U from user_role table by using T,
    and U query filters
8.  $R_{return}$ ← ∅
9.  $i$ ← 0
10. for all $R_{table}$ do
11.     $R_{id}$ ← $R_{table\,i}$
12.     if $R_{id} \in R_{group} \vee R_{id} \in R_{user}$ then
13.         $R_f$ ← true
14.         exit loop
15.     end if
16.     $i \leftarrow i + 1$
17. end for
18. if $R_f$ ≠ true then
19.     $R_{return}$ ← $R_{group}$
20.     $j$ ← 0
21.     for all $R_{user}$ do
22.         $R_{id}$ ← $R_{user\,j}$
23.         if $R_{id} \notin R_{return}$ then
24.             $R_{return} \leftarrow R_{return} \cup R_{id}$
25.         end if
26.         $j \leftarrow j + 1$
27.     end for
28. end if
29. Return $R_{return}$
```

C. Get User Columns Subsidiary Algorithm

This access control subsidiary algorithm is used to retrieve columns and columns' rules granted to a tenant's user. The details of this algorithm are shown in Algorithm 3.

Definition 3 (Get User Columns). T denotes a tenant ID. U denotes a tenant's user. B denotes a table name. S denotes a string of the SELECT clause parameter. Q denotes the table type whether it is a CTT or a VET. C_{user} denotes the tenant's user Columns and columns' rules retrieved from the role_column access control table, and stored in a matrix with n rows and 2 columns. Where $C_{user\,i,0}$ is the first column of the matrix which represents the tenant's user columns, and $C_{user\,i,1}$ is the second column of the matrix which represents the tenant's user columns' rules. C_{id} denotes a column ID. C_{select} denotes a set that is storing a table columns values which constructs the tenant's user query SELECT clause, where $C_{select} = \{C_{select\,1}, C_{select\,2}, \ldots C_{select\,n}\}$. Each element in this set represents a column name in B. C_{where} denotes a string that is storing the access control part of the query WHERE clause, which typically is used to grant rows access to tenant's users. C_{return} denotes a row matrix with 1 row and 2 columns which has two elements, the first one is $C_{return\,0,0}$ that stores the value of C_{select}, and the second one is $C_{return\,0,1}$ that stores the value of C_{where}. R_{return} denotes a set of roles the tenant's user can access.

```
Algorithm 3: GetUserColumns(T, U, B, $Q, R_{return}$)

Input: T, U, B, Q, and $R_{return}$
Output: $C_{return}$
1.  if Q = CTT then
2.      $C_{user}$ ← retrieve columns and columns' rules for U who has $R_{return}$
        from role_information_schema_column table by using T, U, B, and
        $R_{return}$ query filters
3.  else
4.      $C_{user}$ ← retrieve columns and columns rules for U who has $R_{return}$
        from role_column table by using T, U, B, and $R_{return}$ query
        filters
5.  end if
6.  $i \leftarrow 0$
7.  for all $C_{user}$ do
8.      $C_{select\,i} \leftarrow C_{user\,i,0}$
9.      $C_{where} \leftarrow C_{where} \cup C_{user\,i,1}$
10.     $i \leftarrow i + 1$
11. end for
12. $C_{return\,0,0} \leftarrow C_{select}$
13. $C_{return\,0,1} \leftarrow C_{where}$
14. Return $C_{return}$
```

VII. EXPERIMENTS

After developing the EETPS [15], we applied on this service the EETAC method that we propose in this paper, and we carried out two types of experiments to verify the practicability of applying our EETAC on the EETPS. We have evaluated the response time through invoking the EETPS functions, which converts multi-tenant queries into traditional database queries, instead of accessing the database directly.

A. Experimental Setup

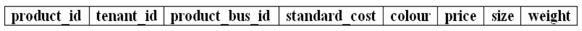
The EETAC method was implemented in Java 1.6.0, Hibernate 4.0, and Spring 3.1.0. The database is PostgreSQL 8.4 and the application server is Jboss-5.0.0.CR2. Both of database and application server is deployed on the same PC. The operating system is Windows 7 Home Premium, CPU is Intel Core i5 2.40GHz, the memory is 8 GB, and the hard disk is 500 GB.

B. Experimental Data Set

The EETPS has designed and developed to serve multiple tenants on one instance application [15]. However, in this paper the aim of the experiments is evaluating the performance after applying the EETAC method on the EETPS for one tenant. We executed the experiments for one tenant, because, in the multi-tenant database the data of each tenant's user is isolated in a table partition. Thus, these experiments can evaluate the effectiveness of retrieving data for each single tenant's user from the multi-tenant database. In our experiment, we used one machine and invoked the function which retrieves a 100 of rows from the 'product' VET, which is shown in Fig. 6. There are 200,000 rows stored in this table that belongs to a tenant whose "tenant_id" equals 1000, and the "db_table_id" of this table equals 16. All the queries implemented in these experiments, are filtered by tenant_id, db_table_id, and other filters specified in the below experiments. These experiments are divided into two types sharing the details of this data set. The experiments are listed below, and the queries of these experiments are shown in Appendix 1.

1) Accessing Data from Table Columns Experiment (Exp.1): In this experiment, we executed Query 1 (Q1) and Query 2 (Q2) to benchmark the query execution time difference between a tenant's user who can access data from all columns of a table by executing Q1, and another tenant's user who can access data from only three out of eight columns of the same table by executing Q2.

2) Accessing Data from Table Rows Experiment (Exp.2): In this experiment, we executed Query 1 (Q1) and Query 3 (Q3) to benchmark the query execution time difference between a tenant's user who can access data from all the table rows by executing Q1, and another tenant's user who can access 10% of the table data which equals approximately 20,000 rows by executing Q3.



product_id	tenant_id	product_bus_id	standard_cost	colour	price	size	weight

FIG. 6 THE 'PRODUCT' TABLE COLUMNS.

C. Experimental Results

1) Accessing Data from Table Columns Experimental Results: Typically, users are granted access to table columns from the application level, because, in a single-tenant database, users are not granted database access on the column level. Whereas, the EETAC method, is granting users a database access on the column level. This capability reduces the query execution time in the multi-tenant database. The experimental study of Exp.1 is showing that the execution time of Q1 for a user who can access fewer numbers of columns of a table is less than the execution time of Q2 for a user who can access all of the table columns. The details results of this experiment are shown in Table 1.

2) Accessing Data from Table Rows Experiment Results: Typically, users cannot be granted a database access to table rows from the database. Whereas, the EETAC method, is granting users a database access on the row level. This capability reduces the query execution time in the multi-tenant database. The experimental study of Exp. 2 is showing that the execution time of Q3 for a user who can access a percentage of a table rows is less than the execution time of Q1 for a user who can access all the table rows. The details results of this experiment are shown in Table 1.

TABLE I. THE QUERY EXECUTION TIME OF EXP. 1 AND EXP.2

Experiment	Query executed	Time in seconds
Exp. 1	Q1	1.35
Exp. 1	Q2	1.10
Exp. 2	Q1	1.45
Exp. 2	Q3	0.48

VIII. Conclusion

In this paper, we have proposed a multi-tenant access control method, called Elastic Extension Table Access Control (EETAC) that allows each tenant in a multi-tenant database to have several users with different types of access grants to access the tenant's data. The concept of retrieving data from the multi-tenant database is slightly different from the single-tenant database. Single-tenant database does not differentiate between the data of different tenants' users. Whereas, the data of the multi-tenant database is partitioned by differentiating between the data owned by a particular tenant, and by accessing columns and rows granted to a tenants' users based on a number of groups or roles assigned to them.

Furthermore, we carried out two types of experiments and we verified the practicability of applying the proposed access table columns and rows grants on the EETPS. The first experiment verified that the cost of executing a query for a user who can access some numbers of columns of a VET is less than the cost of executing the same query for a user, who can access all the VET columns. The second experiment verified that the cost of executing a query for a user who can access a percentage of a VET rows is less than the cost of executing the same query for a user, who can access all of the VET rows.

Appendix 1

Q 1: SELECT tr.table_column_id, tr.value, tr.table_row_id, tr.serial_id FROM table_row tr WHERE tr.tenant_id =1000 AND tr.db_table_id = 16 AND tr.table_row_id IN (SELECT distinct tr.table_row_id FROM **table_row** tr WHERE tr.tenant_id = 1000 AND tr.db_table_id = 16 AND tr.table_column_id = 50 AND (cast(value as numeric) > '9000'))
ORDER BY 3,4 LIMIT 800 OFFSET 0 [15]

Q2: SELECT tr.table_column_id, tr.value, tr.table_row_id, tr.serial_id FROM table_row tr WHERE tr.tenant_id =1000 AND tr.db_table_id = 16 AND t**r.table_column_id in (47,48,49)** AND tr.table_row_id IN (SELECT distinct tr.table_row_id FROM **table_row** tr WHERE tr.tenant_id = 1000 AND tr.db_table_id = 16 AND tr.table_column_id = 50 AND (cast(value as numeric) > '9000')) ORDER BY 3,4 LIMIT 800 OFFSET 0 [15]

Q3: SELECT tr.table_column_id, tr.value, tr.table_row_id, tr.serial_id FROM table_row tr WHERE tr.tenant_id =1000 AND tr.db_table_id = 16 AND tr.table_row_id IN (SELECT distinct tr.table_row_id FROM **table_index** tr WHERE tr.tenant_id = 1000 AND tr.db_table_id = 16 AND tr.table_column_id = 50 AND (cast(row_value as numeric) > '9000')) ORDER BY 3,4 LIMIT 800 OFFSET 0 [15]

References

[1] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi, "Towards an elastic and autonomic multitenant database," in Networking Meets Databases, Athens, Greece, 2011.

[2] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A flexible attribute based access control method for grid computing," Journal of Grid Computing, vol. 7, no. 2, 2009, pp. 169-180.

[3] C. D. Weissman, and S. Bobrowski, "The design of the force.com multitenant internet application development platform," in international conference on Management of Data, Rhode Island, USA, 2009, pp. 889-896.

[4] C. Weissman, D. Moellenhoff, S.Wong, and P. Nakada, "Query optimization in a multi-tenant database system," U.S. Patent 8 229 922, July 24, 2012.

[5] C. P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. Hart, "Enabling multi-tenancy: An industrial experience report," in Software Maintenance,Timisoara, Romania, 2010, pp. 1-8.

[6] D. Arnold, S. Diniro, V. Lee, S. Musker, and J. A. Woods, "Row and column access control," in Unleashing DB2 10 for Linux, UNIX, and Windows, vol. 10, no. 1, 2012, pp.65-86.

[7] D. F. Ferraiolo and D. R. Kuhn, "Role-Based Access Controls," in 15th NIST-NCSC National Computer Security Conference Gaithersburg, USA, 1992, pp. 554-563.

[8] D. Li, C. Liu, Q. Wei, Z. Liu, and B. Liu, "RBAC-based access control for SaaS systems," in Information Engineering and Computer Science, Wuhan, China, 2010, pp. 1-4.

[9] E. J. Domingo, J. T. Nino, A. L. Lemos, M. L. Lemos, R. C. Palacios, and J. M. G. Berbís, " CLOUDIO: A cloud computing oriented multi-tenant architecture for business information systems," in Cloud Computing, Miami, USA, 2010, pp.532-533.

[10] F. Chong, G. Carraro, and R. Wolter. (2013, July 10). Multi-tenant data architecture [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479086.aspx.

[11] F. S. Foping, I. M. Dokas, J. Feehan, and S. Imran, "A new hybrid schema-sharing technique for multitenant applications," in Digital Information Management, Michigan, USA, 2009, pp. 1-6.

[12] G. Liu, "Research on independent saas platform," in Information Management and Engineering, Chengdu, China, 2010, pp. 110-113.

[13] H. Takabi, J. B. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments," IEEE Security and Privacy, vol. 8, no. 6, 2010, pp. 24-31.

[14] H. Yaish, M. Goyal, and G. Feuerlicht, "An elastic multi-tenant database schema for software as a service," in Ninth IEEE International Conference on Dependable, Autonomic and Secure Computing, Sydney, Australia, 2011, pp. 737-743.

[15] H. Yaish, M. Goyal, and G. Feuerlicht, "Proxy service for multi-tenant database access," in The International Cross Domain Conference and Workshop, Regensburg, Germany, 2013, pp.100-117.

[16] J. Du, H. Y. Wen, and Z. J. Yang, "Research on data layer structure of multi-tenant e-commerce system," in IEEE 17th International Conference on Industrial Engineering and Engineering Management, Xiamen, China, 2010, pp. 362-365.

[17] K. Brodersen, T. M. Rothwein, M. S. Malden, M. J. Chen, and A. Annadata, "Database access method and system for user role defined access," U.S. Patent 6 732 100, May 4, 2004.

[18] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," IEEE Internet Computing, vol. 16, no. 1, 2012, pp. 69-73.

[19] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang, "Native support of multi-tenancy in RDBMS for software as a service," in Proceedings of the 14th International Conference on Extending Database Technology, Uppsala, Sweden, 2011, pp.117-128.

[20] PostgreSQL. (2013, July 10) Columns [Online]. Available: http://www.postgresql.org/docs/8.3/static/infoschema-columns.html.

[21] R. Anderson, "Technical perspective: A chilly sense of security," Communications of the ACM, vol. 52, no. 5, 2009, pp. 90-90.

[22] R. Mietzner, T. Unger, R. Titze, and F. Leymann, "Combining different multi-tenancy patterns in service-oriented applications," in Enterprise Distributed Object Computing Conference, Auckland, New Zealand, 2009, pp. 131-140.

[23] Salesforce, "Record-level access: Under the hood," white paper, salesforce.com, inc., July 5, 2013.

[24] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold, "A comparison of flexible schemas for software as a service," in Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, Rhode Island, USA, 2009, pp. 881-888.

[25] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-tenant databases for software as a service: schema-mapping techniques," in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, Vancouver, Canada, 2008, pp. 1195-1206.

[26] T. Kwok, T. Nguyen, and L. Lam, "A software as a service with multi-tenancy support for an electronic contract management application," in Services Computing, Honolulu, USA, 2008, pp.179-186.

[27] V. Lazarov, "Comparison of different implementations of multi-tenant databases," B.A. thesis, Technische niversität München, München, Germany, 2007.