J. Sc. & Tech. Univ. Peshawar, 2010, 34 (1) 73-80.

JOINING TABLES ON SECRET PRIMARY KEYS

AZHAR RAUF[1], SHAH KHUSRO[1], SHARAFAT ALI[2] & SAREER BADSHAH[3]

[1]Department of Computer Science, University of Peshawar, Peshawar, Pakistan
[2]CECOS University, Peshawar, Pakistan
[3]Department of Statistics, Islamia College (Chartered University) Peshawar, Pakistan

Abstract: There is a need to protect data in relational databases from unauthorized access at finer levels of granularity and thereby make as much data available to the user as possible. Current state-of-the-art security techniques offer elegant ways to protect data at the column, row and cell levels. The entity integrity constraint does not allow duplicate primary keys in a table. If row level security is implemented, a user may know the secret primary key inserted by a different user in the table. This secret primary key can be misused by inserting a dummy row in the child table as a foreign key. This results in the generation of misleading reports while joining the parent and child tables on the same key. This study suggests a technique to control this issue in row level security technique.

Keywords: Fine-Grained Security, Inference, Row Level Security in Relational Databases.

Introduction

Fine grained security is becoming more and more popular in relational databases by the increasing demand for sharing the data in multiple departments of an organization and reducing the maintenance cost by managing security at the applications or views level. Organizations are stepping in to maintain the data security policies at the instance level for better management and cost effectiveness. Relational database vendors offer techniques such as Row, Column and Cell Level Security to mask data at finer levels of granularity. These techniques are cost efficient and helpful in managing the complex security policies of an organization. Protecting data in a database is more challenging than other fields of computer security for the reasons of maintaining security and making data available at finer levels of granularity. For example, in operating system security, subjects and objects are well-defined in the security matrix and precautions are made to control access to files managed by the operating system (James Cannady, 2000). Databases carry information, including sensitive information, for the purpose of querying the data. Sensitive information must be protected from unclassified users while non-sensitive information must be made available to users. The problem arises from the finer granularity of the database when handling files, attributes, and values (James Cannady, 2000). For example, an intruder might be interested in different pieces of unclassified information to leak sensitive information by making an inference or aggregation. On the other hand, too restrictive policies to protect data can hinder the useful purpose of querying the database.

There are two types of threats regarding data disclosure in databases: 1) Direct Disclosure and 2) Indirect Disclosure (Ashby, Jajodia, 1996). In the former case, a user at a low security level, directly accesses data from higher security level. This results by the flaws in the security control mechanism of the database. In the later case, a user at the low security level, indirectly accesses data from higher security level. Inference and Aggregation are two examples of indirect data disclosure. Inference refers to the derivation of sensitive information from non-sensitive data while aggregation refers to the observation that the sensitivity level of an aggregate computed over a group of values in a database may differ from the sensitivity levels of the individual elements (Dieter Gollmann, 1999). There is a need to maintain security in databases at finer levels of granularity to protect sensitive data from direct disclosure and non-sensitive data from indirect disclosure and at the same time reveal as much of the non-sensitive information as possible for our business operations.

Access Controls in Relational Databases

Relational databases make use of the two general types of access control policies – Discretionary Access Control (DAC) and Mandatory Access Control (MAC), to protect information in multilevel systems (James Cannady, 2000). In the DAC policy, access is restricted based on the authorizations granted to the user. This is the widely used access control mechanism in relational databases. In the MAC policy, information is secured by assigning sensitivity levels, or labels, to data entities. MAC policies are generally more secure than DAC policies but there is a performance overhead because of the tightened security (James Cannady, 2000). Most MAC policies also incorporate DAC measures as well. Readers are advised to refer to the Related Work section for the detailed descriptions of different security models.

Internet has changed the techniques of sharing the data between businesses and their customers. It has enabled large amounts of data to be stored in large repositories and shared among organizations and stakeholders for easy and quick access. Organizations, having branches located in different geographical areas and running different security rules across the world require sharing data with each other and their customers. These security rules may vary for different departments, divisions, regions, countries, cultures, and customers around the world. A central database repository is required to satisfy these varying information needs of such organizations. For instance, consider a large organization that stores the human resource data in a central database that can be accessed by multiple subsidiaries, departments, and divisions. Different users want to access this HR information and different security policies apply to these users. For example, Managers can view all information of employees who work with them. HR people can view and update employees' records within their area only. A specialist might only need to view the records of 'Engineering' department and an employee can view and modify his/her own personal data such as marital status, number of dependents, address, phone# etc. but cannot access his/her salary information. Such organizations which are dispersed in a wide geographical area and depend on internet-based systems to manage data centrally for administrative ease and cost reduction purposes need to control access to the data at a finer level of granularity, often to the level of individual users or customers. (Oracle Virtual Private Database, 2005). Another trend in the business community is the out-sourcing of routine tasks and focusing on the core competencies only. This further necessitates the importance of varying security policies on centralized data.

Application Level Security has always been a challenge in organizations. If access to data is controlled at the application level instead of the data instance level, users who bypass the application security e.g., by issuing ad-hoc queries or reporting tools can access secret data. Complexity is another issue to implement strong security policies in multiple applications. Managing security by a database server at the data instance level is more cost effective as compared to implementing the same security policies in each application separately. Views have been widely used for application level security and controlling access at fairly granular level, but they have limitations which make them less optimal for very fine-grained access control. The following is a short discussion of the limitations of views.

If we need large number of views to enforce security policy, views are not always practical. For example, if we want to restrict customer's access to their data and we have few customers then it is ok. But if we have hundreds and thousands of customers then managing as many views is impossible (Oracle Virtual Private Database, 2005).

Views can not be used to implement complex security policies. For example, if an organization wants to implement the Cell Level Security then it is hard for a database to evaluate such a complex condition. Sometimes users have direct access to the base tables. Such users can bypass the view security by issuing ad-hoc queries or running reports to access secret data.

Views may complicate the administration of security policy. When a security policy is changed, added, or removed, it is difficult to determine what to do with each view. An administrator cannot tell that he/she is breaking an application by altering or dropping a view. Views might reduce performance by fetching data from the base table first and then making it available to the application.

The above discussion fairly explains the need of granular security on the base tables in the database.

Labeling Techniques in Relational Databases

There are three state-of-the-art labeling techniques in relational databases:

Column Level Security

Column suppression is one of the commonly used techniques to secure secret data of a column. The Virtual Private Database (VPD) feature of Oracle 10g implements 'Column Relevance' and 'Column Filtering' types of granular securities (Oracle Virtual Private Database, 2005). In the 'Column Relevance' feature, the applied security policy is invoked only when a specific column of the table is part of the Data Manipulation Language (DML) command. For example, an employee is able to see all rows (first name, last name, location etc.) from the HR table as long as he is not querying the salary column. If the salary column is referenced in his SQL statement, the database returns only the entries he is allowed to see depending on the security policy. In the 'Column Filtering' or Column Masking approach, all rows are returned, but only the rows he is disallowed contain null values (Oracle Virtual Private Database, 2005).

Row Level Security

This technique suppresses the entire row from the user whose clearance level is less than the Label of the record. The database automatically attaches the WHERE predicate to the query made by any user to check his/her clearance from the Label attribute associated with each record. This technique was originally implemented in Oracle 8i and now is a part of Oracle's 10g VPD feature (Oracle Virtual Private Database, 2005). Microsoft's consultants claim that MS SQL Server 2005 can be used to implement Row Level Security (Art Rask, 2005).

Cell Level Security

In this technique, individual cells carrying secret data are suppressed only. This technique is implemented in OLAP and MS SQL Server 2000 in data cubes and can be implemented in SQL 2005, according to MS consultants (Art Rask, 2005). Oracle's Column Filtering approach is a closely related technique to Cell Level Security, depending on the security policy. A new fine grained security technique called cell-plus-innocent security has been identified in (Rauf A. 2009).

There are positive and negative aspects of these labeling techniques with regards to data accessibility and allowing chances of inference because of the data suppression. These researchers are investigating the issues associated with the row level security in relational databases.

Research Issues and Problem Statement

Row level security provides security at finer levels of granularity but problem arises when an intruder knows a secret primary key by inserting a row in a table and the database management system does not allow him because of the integrity constraint as another user has already stored a row using the same primary key. This intruder can insert a dummy record in a child table using the same secret primary key. It may result in the generation of misleading reports for the management while joining the parent and child tables. This study is focused on finding the solutions for the above problem.

Research Hypothesis and Experiment Design

The following hypothesis represents the central thesis of this investigation:

Hypothesis

The joining problem on secret primary keys can be prevented by allowing each user to insert a record on the foreign key only if he/she has created the primary key for it.

In this hypothesis, the security policy of the row level security is enhanced based on which a user is disallowed to insert a new record in a table that could cause insertion anomaly. Each user of the database is allowed to insert a record in the child table with his user id as it is the case of row level security. When the user relates this new record to the parent table by putting a foreign key in the new record, the security policy will check whether this user has generated the primary key for the same foreign key, in the parent table or not? If this user has not generated the primary key in the parent table he is disallowed by the security policy to insert a new record otherwise he will be able to insert one.

Experiment Design

A new technique is suggested in this study to prevent insertion anomaly in row level security. An experiment was designed using Oracle 10g to apply the row level security technique on a hypothetical database. The security policy of the row level security in Oracle 10g was enhanced by adding an extra trigger for each

table. A user tried to insert a new record in the child table with a foreign key that a different user had generated the primary key record on the same foreign key, in the parent table. The trigger did not allow the user to create a dummy record using another user's primary key, in the child table. This way the enhanced row level security stopped the possibility of misusing the secret primary key.

Assumptions and Research Scenario

In this research, it is assumed that a user knows a secret primary key. If a user inserts a record in a table on a primary key and a record is already entered by a different user using same primary key in the same table with a different security level, the integrity constraint will restrict him to insert a new record. This initiates an inference and the user knows the secret primary key generated by a different user.

In a typical organization, there are multiple security classes and users' security levels. This research considers only three users: a) User1 b) User2 and c) Manager. User1 is not allowed to insert/view a record in the area of User2 and vice versa. Manager can view/insert records of both users. Only the 'SELECT' and 'INSERT' statements of the SQL DML are considered for the research purposes.

Analysis

An integrity constraint exception is raised if a foreign key is inserted in a child table whose primary key is missing in the parent table. This way the DBMS maintains referential integrity by not allowing a user to enter a record on a foreign key whose primary key does not exist. If we take the example of Oracle 10g, the VPD feature apparently has an issue. In case of row level security, it deals referential integrity on the old fashion of without the VPD feature. If a user creates a primary key in the parent table only this particular user should be allowed to use his/her primary key as a foreign key in the child table. But this is not the case in row level security feature of Oracle. It rather allows a user to use a foreign key in the child table whose primary key is generated by a different user in the parent table. This results the possibility of an insertion anomaly in the child table as this user is not a legitimate user of the foreign key. These researchers have identified a solution to take care of this issue. The solution is implemented with the use of a trigger.

Trigger Level Control

In this solution, a trigger is generated in each child table where a foreign key is to be used. The trigger is generated by the schema builder user and fired before the INSERT statement for all users of the table except the schema builder user. The trigger checks whether the foreign key inserted by the user has been generated by this particular user in the parent table. If this user has generated the primary key in the parent table then the user is allowed to insert a record using the foreign key otherwise an exception is raised. The trigger stores the inserted foreign key by the user in a variable and searches this value in the parent table using the SELECT statement.

Following is the implementation of row level security and then implementing the trigger to avoid the problem discussed.

The following statements create a schema builder user 'SHARAFAT' and two other users 'USER1' and 'USER2' for the sake of example.

```
CONNECT SYS AS SYSDBA;

CREATE USER SHARAFAT IDENTIFIED BY IMS
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;
GRANT CONNECT, RESOURCE TO SHARAFAT;
GRANT CREATE ANY CONTEXT, CREATE PUBLIC SYNONYM TO SHARAFAT;

CREATE USER SECURITY_USER1 IDENTIFIED BY SEC1
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;
```

table. A user tried to insert a new record in the child table with a foreign key that a different user had generated the primary key record on the same foreign key, in the parent table. The trigger did not allow the user to create a dummy record using another user's primary key, in the child table. This way the enhanced row level security stopped the possibility of misusing the secret primary key.

Assumptions and Research Scenario

In this research, it is assumed that a user knows a secret primary key. If a user inserts a record in a table on a primary key and a record is already entered by a different user using same primary key in the same table with a different security level, the integrity constraint will restrict him to insert a new record. This initiates an inference and the user knows the secret primary key generated by a different user.

In a typical organization, there are multiple security classes and users' security levels. This research considers only three users: a) User1 b) User2 and c) Manager. User1 is not allowed to insert/view a record in the area of User2 and vice versa. Manager can view/insert records of both users. Only the 'SELECT' and 'INSERT' statements of the SQL DML are considered for the research purposes.

Analysis

An integrity constraint exception is raised if a foreign key is inserted in a child table whose primary key is missing in the parent table. This way the DBMS maintains referential integrity by not allowing a user to enter a record on a foreign key whose primary key does not exist. If we take the example of Oracle 10g, the VPD feature apparently has an issue. In case of row level security, it deals referential integrity on the old fashion of without the VPD feature. If a user creates a primary key in the parent table only this particular user should be allowed to use his/her primary key as a foreign key in the child table. But this is not the case in row level security feature of Oracle. It rather allows a user to use a foreign key in the child table whose primary key is generated by a different user in the parent table. This results the possibility of an insertion anomaly in the child table as this user is not a legitimate user of the foreign key. These researchers have identified a solution to take care of this issue. The solution is implemented with the use of a trigger.

Trigger Level Control

In this solution, a trigger is generated in each child table where a foreign key is to be used. The trigger is generated by the schema builder user and fired before the INSERT statement for all users of the table except the schema builder user. The trigger checks whether the foreign key inserted by the user has been generated by this particular user in the parent table. If this user has generated the primary key in the parent table then the user is allowed to insert a record using the foreign key otherwise an exception is raised. The trigger stores the inserted foreign key by the user in a variable and searches this value in the parent table using the SELECT statement.

Following is the implementation of row level security and then implementing the trigger to avoid the problem discussed.

The following statements create a schema builder user 'SHARAFAT' and two other users 'USER1' and 'USER2' for the sake of example.

```
CONNECT SYS AS SYSDBA;

CREATE USER SHARAFAT IDENTIFIED BY IMS
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;
GRANT CONNECT, RESOURCE TO SHARAFAT;
GRANT CREATE ANY CONTEXT, CREATE PUBLIC SYNONYM TO SHARAFAT;

CREATE USER SECURITY_USER1 IDENTIFIED BY SEC1
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;
```

```
GRANT CONNECT, RESOURCE TO SECURITY_USER1;

CREATE USER SECURITY_USER2 IDENTIFIED BY SEC2
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;
GRANT CONNECT, RESOURCE TO SECURITY_USER2;
```

Following statement grants the execute right on package DBMS_RLS to all users.

```
GRANT EXECUTE ON DBMS_RLS TO PUBLIC;
```

Now the schema owner creates two tables DEPT and EMP as parent and child tables respectively.

```
CONNECT SHARAFAT/IMS@SERVER
CREATE TABLE DEPT
(
DEPTNO  NUMBER(2) CONSTRAINT PK_DEPTNO_IN_DEPT PRIMARY KEY,
DNAME  VARCHAR2(30),
LOC      VARCHAR2(30),
USER_ID VARCHAR2(30)
);

CREATE TABLE EMP
(
EMPNO      NUMBER(4) CONSTRAINT PK_EMPNO_IN_EMP PRIMARY KEY,
ENAME      VARCHAR2(30),
JOB          VARCHAR2(30),
MGR          NUMBER(4) CONSTRAINT SK_MGR_IN_EMP REFERENCES EMP(EMPNO),
HIREDATE DATE,
SAL          NUMBER(7,2),
COMM       NUMBER(4),
DEPTNO    NUMBER(2) CONSTRAINT FK_DNO_IN_EMP REFERENCES DEPT(DEPTNO),
USER_ID VARCHAR2(30)
);
```

Following statements grant SELECT and INSERT rights on the two tables to USER1 and USER2.

```
GRANT SELECT, INSERT ON DEPT TO SECURITY_USER1, SECURITY_USER2;
GRANT SELECT, INSERT ON EMP   TO SECURITY_USER1, SECURITY_USER2;
```

Following code creates a context and defines its body.

```
CREATE CONTEXT SHARAFAT USING SHARAFAT.CONTEXT_PACKAGE;
CREATE OR REPLACE PACKAGE CONTEXT_PACKAGE AS
      PROCEDURE SET_CONTEXT;
END;

CREATE OR REPLACE PACKAGE BODY CONTEXT_PACKAGE IS
      PROCEDURE SET_CONTEXT
      IS
        V_OUSER VARCHAR2(30);
```

```
    BEGIN
            DBMS_SESSION.SET_CONTEXT('SHARAFAT','SETUP','TRUE');
            V_OUSER:=SYS_CONTEXT('USERENV','SESSION_USER');
            DBMS_SESSION.SET_CONTEXT('SHARAFAT','USER_ID',V_OUSER);
            DBMS_SESSION.SET_CONTEXT('SHARAFAT','SETUP','FALSE');
    END set_context;
    END context_package;

GRANT EXECUTE ON SHARAFAT.CONTEXT_PACKAGE TO PUBLIC;
CREATE PUBLIC SYNONYM CONTEXT_PACKAGE FOR
SHARAFAT.CONTEXT_PACKAGE;

CONNECT SYS AS SYSDBA;
CREATE OR REPLACE TRIGGER SHARAFAT.SET_SECURITY_CONTEXT
AFTER LOGON ON DATABASE
BEGIN
    SHARAFAT.CONTEXT_PACKAGE.SET_CONTEXT;
END;

CONNECT SHARAFAT/IMS@SERVER
CREATE OR REPLACE PACKAGE SECURITY_PACKAGE AS
    FUNCTION USER_DATA_INSERT_SECURITY(OWNER VARCHAR2,OBJNAME VARCHAR2)
    RETURN VARCHAR2;
    FUNCTION USER_DATA_SELECT_SECURITY(OWNER VARCHAR2,OBJNAME VARCHAR2)
    RETURN VARCHAR2;
END Security_package;
```

Following code creates a package SECURITY_PACKAGE that actually implements the row level security on SELECT and INSERT statements.

```
CREATE OR REPLACE PACKAGE BODY Security_Package IS
    FUNCTION User_Data_Select_Security(Owner VARCHAR2, Objname VARCHAR2)
    RETURN VARCHAR2 IS
      Predicate VARCHAR2(2000);
    BEGIN
      Predicate := '1=2';
      IF (SYS_CONTEXT('USERENV','SESSION_USER') = 'SHARAFAT') THEN
       Predicate := NULL;
      ELSE
       Predicate := 'USER_ID = SYS_CONTEXT("SHARAFAT","USER_ID")';
      END IF;
      RETURN Predicate;
    END User_Data_Select_Security;

    FUNCTION User_Data_Insert_Security(Owner VARCHAR2, Objname VARCHAR2)
    RETURN VARCHAR2 IS
      Predicate VARCHAR2(2000);
    BEGIN
      Predicate := '1=2';
      IF (SYS_CONTEXT('USERENV','SESSION_USER') = 'SHARAFAT') THEN
       Predicate := NULL;
```

```
    ELSE
     Predicate := 'USER_ID = SYS_CONTEXT("SHARAFAT","USER_ID")';
    END IF;
    RETURN Predicate;
   END User_Data_Insert_Security;
END Security_Package;
```

Then a synonym is created and the add_policy method of the DBMS_RLS package is set.

```
GRANT EXECUTE ON SHARAFAT.Security_Package TO PUBLIC;
CREATE PUBLIC SYNONYM Security_Package FOR SHARAFAT.Security_Package;

BEGIN
 DBMS_Rls.Add_Policy('SHARAFAT', 'DEPT', 'USER_DATA_INSERT_POLICY',
                     'SHARAFAT',
                     'SECURITY_PACKAGE.USER_DATA_INSERT_SECURITY',
                     'INSERT', TRUE);

 DBMS_Rls.Add_Policy('SHARAFAT', 'DEPT',
                     'USER_DATA_SELECT_POLICY','SHARAFAT',
                     'SECURITY_PACKAGE.USER_DATA_SELECT_SECURITY','SELECT');

 DBMS_Rls.Add_Policy('SHARAFAT', 'EMP', 'USER_DATA_INSERT_POLICY',
                     'SHARAFAT',
                     'SECURITY_PACKAGE.USER_DATA_INSERT_SECURITY',
                     'INSERT', TRUE);

 DBMS_Rls.Add_Policy('SHARAFAT', 'EMP',
                     'USER_DATA_SELECT_POLICY','SHARAFAT',
                     'SECURITY_PACKAGE.USER_DATA_SELECT_SECURITY','SELECT');
END;
```

Row level security is tested using SELECT and INSERT statements of the DML. The strike-out statements represent the rejected statements by the row level security.

```
CONNECT SECURITY_USER1;
INSERT INTO SHARAFAT.DEPT VALUES (10,'Urdu','Peshawar','SECURITY_USER1');
INSERT INTO SHARAFAT.DEPT VALUES (20,'Urdu','peshawar','SECURITY_USER2');

CONNECT SECURITY_USER2;
INSERT INTO SHARAFAT.DEPT values (10,'English','Peshawar','SECURITY_USER2');
SELECT * FROM SHARAFAT.DEPT WHERE DEPTNO=10;
INSERT INTO SHARAFAT.EMP VALUES
        (1,'Asad','Analyst',null,sysdate,2000,120,10,'SECURITY_USER1');
INSERT INTO SHARAFAT.EMP VALUES
        (1,'Asad','Analyst',null,sysdate,2000,120,10,'SECURITY_USER2');

SELECT DEPT.DNAME,DEPT.LOC,EMP.ENAME,EMP.JOB
FROM SHARAFAT.DEPT,SHARAFAT.EMP WHERE DEPT.DEPTNO=EMP.DEPTNO;
```

Following is the code of the trigger:

```
CREATE OR REPLACE TRIGGER RLSP
BEFORE INSERT ON EMP
FOR EACH ROW
DECLARE
        TEMP_VAR NUMBER (2);
        CURSOR RS (V_DEPTNO IN NUMBER)  IS SELECT DEPTNO FROM DEPT
                                                WHERE DEPTNO = V_DEPTNO;

BEGIN
        OPEN  RS(:NEW.DEPTNO);
        FETCH RS INTO TEMP_VAR;
        IF RS%NOTFOUND THEN
            RAISE_APPLICATION_ERROR(-20725, 'Integrity Constraint violated  parent record no
            found');
        END IF;
END;
```

The trigger does not allow a user to enter a foreign key that is not created by him/her as a primary key in the parent table. This technique stops insertion anomaly but there is a performance overhead as the trigger is fired each time when there is an INSERT activity because the trigger validates whether the user has the SELECT right on the table's data. Furthermore, the parent table name is hard coded in the SELECT query inside the trigger for each child table.

Conclusion

The entity integrity constraint does not allow duplicate primary keys in a table. If row level security is implemented, a user may know the secret primary key inserted by a different user in the table. This secret primary key can be misused by inserting a dummy row in the child table as a foreign key. This results in the generation of misleading reports while joining the parent and child tables on the same key. This study suggests a technique to control this problem.

A trigger is created in each child table that checks the validity of a foreign key entered by a user. The INSERT statement is rejected if the foreign key entered by the user, is created by a different user as a primary key in the parent table. The hypothesis was proved by designing an experiment on a hypothetical database and implementing the trigger level control mechanism.

References

Art Rask, 2005. Don Rubin, Bill Neumann. Implementing Row and Cell Level Security in Classified Databases Using SQL Server 2005, MS SQL Server Technical Center, April 2005.

Cannady, J., 2000. Security Models for Object-Oriented Databases, CRC Press LLC, 2000.

Gollmann, D., 1999. Computer Security, Copyright 1999 by John Wiley and Sons Ltd., ISBN 0 471 97844 2.

Jajodia, A., 1996. NCSC (National Computer Security Center) Technical Report, Volume 1/5, Library No. S-243,039, May 1996.

Oracle Virtual Private Database, 2005. An Oracle Database 10g Release 2 White Paper, June 2005.

Rauf, A., Badshah, S., Khusro, S., 2009. A Comparative Study of Fine Grained Security Techniques Based on Data Accessibility and Inference World Academy of Science Engineering and Technology Vol. 55 July 2009 ISSN 2070-3724, Page 302.