SECURITY FEATURES IN ORACLE DATABASE MANAGEMENT SYSTEM

Hennayake M Bandara
B.S., California State University, Long Beach, 2006

PROJECT

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2009
SECURITY FEATURES IN ORACLE DATABASE MANAGEMENT SYSTEM

A Project

by

Hennayake M Bandara

Approved by:

__________________________________, Committee Chair
Dr. Ying Jin

__________________________________, Second Reader
Dr. Jinsong Ouyang

__________________________________
Date
Student: Hennayake M Bandara

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator
Dr. Cui Zhang

__________________________
Date

Department of Computer Science
Abstract

of

SECURITY FEATURES IN ORACLE DATABASE MANAGEMENT SYSTEM

by

Hennayake M Bandara

Nowadays database management systems are the most crucial factors in managing and storing the data. Security enforcement in a database is very important to assure that the data stored is properly secured. Most of the database management systems are occupied with strong defense mechanisms. In addition to this, many academic researchers have been proposing solutions and mechanisms to improve security enforcement in database management systems. Understanding and Identifying of these new mechanisms and security features provided by database systems are very important. In this project, I surveyed the solutions proposed and implemented by academic researchers on Relational and Object-Oriented Databases. Also, I identified and illustrated a set of security features that can be utilized to enforce security using Oracle database management system. In conclusion, this project identified the solutions, mechanisms, and features that can be used to improve security enforcement in both Relational and Object-Oriented databases.

______________________, Committee Chair
Dr. Ying Jin

______________________
Date
ACKNOWLEDGMENTS

I would like to extend my gratitude to my project advisor Dr. Ying Jin for guiding me throughout this project and helping me in completing this project successfully. In addition, I would like to thank The Department of Computer Science at California State University for extending this opportunity for me to pursue this program and guiding me all the way to become a successful student.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>RESEARCH WORK AND SURVEY ON RELATIONAL DATABASE MANAGEMENT SYSTEMS</td>
<td>3</td>
</tr>
<tr>
<td>3.</td>
<td>RESEARCH WORK AND SURVEY ON OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS</td>
<td>19</td>
</tr>
<tr>
<td>4.</td>
<td>CASE STUDY ON SECURITY FEATURES IN ORACLE DATABASE MANAGEMENT SYSTEM</td>
<td>34</td>
</tr>
<tr>
<td>5.</td>
<td>CONCLUSION</td>
<td>71</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>73</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>No.</th>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Figure 1 Rules for a Simple Employee Database</td>
<td>6</td>
</tr>
<tr>
<td>2.</td>
<td>Figure 2 View Predicates</td>
<td>7</td>
</tr>
<tr>
<td>3.</td>
<td>Figure 3 Architecture of Dubiety-Determining Model</td>
<td>16</td>
</tr>
<tr>
<td>4.</td>
<td>Figure 4 Granting Algorithm</td>
<td>29</td>
</tr>
<tr>
<td>5.</td>
<td>Figure 5 Revoking Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>6.</td>
<td>Figure 6 Bank Database</td>
<td>36</td>
</tr>
<tr>
<td>7.</td>
<td>Figure 7 User Hierarchy in the Bank Database</td>
<td>38</td>
</tr>
<tr>
<td>8.</td>
<td>Figure 8 Result for USER1’s Query 1</td>
<td>50</td>
</tr>
<tr>
<td>9.</td>
<td>Figure 9: Result for USER1’s Query 2</td>
<td>54</td>
</tr>
<tr>
<td>10.</td>
<td>Figure 10 Oracle Output for USER2’s SQL Query</td>
<td>55</td>
</tr>
<tr>
<td>11.</td>
<td>Figure 11 Oracle Audit Log 1</td>
<td>66</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

Nowadays the technology has become the most crucial part in our day-to-day life. Facilitating a stable technology include management of information very securely and efficiently. Every large-scale application that we use today is supported by a database management system that manages and stores data. Ensuring the security of information we store in these databases is very important. Security plays a huge role in carrying out this responsibility in both user level and data level. Today various database management systems have their own ways of enforcing security. In addition to this, many academic researchers have come up with proposals to improve security in databases.

As a result of the technology, there are different types of database management systems available in the world, for example, Relational Databases, Object-Oriented Databases, Active Databases, Spatial Database, Object-Relational Databases, and Distributed Databases. This project has looked into security features that are important in Oracle Database Management System and identified them. During this project, I have focused my research on two main categories of Databases, which are Relational Database and Object-Oriented Database. The phase of this project was divided into two major parts. As first part of the project, I have researched the current existing solutions for security problems and mechanisms proposed by various academic researches regarding database security. During the second phase of my project, I have indentified important set of security features related to Oracle database management system and illustrated them.
using multiple PL/SQL queries. During this phase, I have analyzed each security feature and proposed mechanisms to improve security enforcement for features that had weaknesses. Oracle database management system supports the rich, powerful Object-Oriented Data Model. As a result of this, we have a capability to implement Object-Oriented features such as inheritance and polymorphism when we model a database in Oracle environment. I believe that security enforcement towards Relational and Object-Oriented features in Databases are still in early stage. Thus, studying the research contribution towards Relational and Object-Oriented databases and identifying and illustrating important security features to improve security enforcement using Oracle database management was the main purpose of this project.
Relational databases are heavily used in both academic and industrial fields. Many applications today use Relational databases to organize and store their data. As students, we always get to interact with Relational database in our day-to-day academic studies. Security of a Relational database depends on several important factors, for example, authorization, access control mechanism, and organization of data. To improve above factors, many academic researchers have been working on solutions and mechanisms towards Relational databases. During my research phase of this project, I was able to come across various interesting solutions proposed by academic researchers in the Computing Academia. I have presented my research work in this chapter.

2.1 Improving Database Security through Simulation of Agents.

Integrity is a very important factor in a database management system. Integrity of a database ensures that the data we store will not be changed by unauthorized manner or by users. Chiong and Dhakal [1] have proposed a mechanism to model database security through agent based simulation. This is a simulation program that operates under a set of
pre-given permission rules and a mechanism to fix corrupted data in a database immediately. According to Chiong and Dhakal, the simulation consists of a number of agents who randomly access data objects and perform data manipulation operations. Any form of operations can corrupt data in a database. However, if we do not properly manage and fix these corrupted data, the corruption can lead the database into an unstable mode. The main logic behind this simulation program according to Chiong and Dhakal is that if privileges granted to users are based on certain rules and permissions rather than automatic or universal granting, the database security can be improved tremendously. Their simulation program keeps track of the data, the agents who manipulates the data, the privileges the agents have on the data, and the list of access privileges data owners have been granted to the agents. This program implements objects for each agent and data. Each data object keeps track of the agents who modify its data and integrity of the data. Each agent object keeps track of the data it owns and the list of privileges granted to other agents on behalf of its data. As Chiong and Dhakal describe, the simulation created a set of agents and data objects at the beginning and let agents randomly access these data objects. In addition, the researchers specify that each agent is associated with a reliability index. If an agent corrupts a certain data during an access, that corresponding agent’s unreliability index will increase. The researchers also specify that when an agent requests
access privileges from an owner of a particular data object, the owner will grant privilege after concerning the unreliability index of the requester and permission rules in the program. Chiong and Dhakal calculated the unreliability of the system based on instances of data corruption till a time ‘t’ and data pieces available at time ‘t’. They were able to compare the unreliability of data when privileges were granted based on different permission rules. The authors experienced that when privileges were granted based on much stricter rules, the data corruption in the database was less. The database security can be maintained better with stricter access to the data [1]. According to the above researches, it concludes that database security can be improved tremendously when privileges are granted based on very stricter permission rules. I believe this is a very strong mechanism to control granting privileges to users in a database and ensure the data integrity is safe.
2.2 A Reflective Database Access Control Policy Model.

The research of A Formal Framework for Reflective Database Access Control Policies (RDBAC) by Olsen, Gunter, and Madhusudan [2] aims to express a database privilege as a database query itself by incorporating access policy decisions to depend on data that are stored in the other parts of the database. According to the authors, their primary logic is based on Transaction Datalog and this was used to enforce fine-grained RDBAC. As the steps of defining policies, first the researchers had created three view predicate names for each database predicate with a fixed integer n+1. These view predicates are view.p, view.ins.p, and view.del.p. Rules for these predicate name have the interpretation that view .p(U, T₁,…, Tₙ) can be derived from the current database state if and only if user U should be granted read access to the values of p(T₁, …, Tₙ) [2]. Olsen, Gunter, and Madhusudan explained their Transaction Datalog based policies using a number of examples.

```prolog
employee('alice', 90000, 'hr', 'manager').
employee('bob', 70000, 'sales', 'clerk').
employee('carol', 90000, 'sales', 'manager').
employee('david', 80000, 'hr', 'cpa').
manager(Person, Dept) :- employee(Person, Salary, Dept, 'manager').
```

Figure 1: Rules for a Simple Employee Database [2]
Researchers explain that based on the Figure 1, the view predicates they had created could be used to accomplish access methods they needed. For an example, using the first predicate rule in the Figure 2, they showed that they can allow all employees to access their own records. They also showed that using second rule in the Figure 2, manager can view their employees’ names and positions, but not salary values.

According to the researchers, this is accomplished by replacing the Salary field by a null constant. As part of the security analysis, the above researches have proposed theorems that are related to Security Analysis and Decidability, Side-Effect-Free Policies and Append–Only Policies. The implementation of this RDBAC prototype model was targeted to evaluate reflective database access policies and perform security analysis. As
the result of the above model, the researchers were able show how access control policies can be evaluated based on privileges of user and how access privileges can be extended to users who do not have direct access to the database.
2.3 A Database Intrusion Detection and Response System.

As it is described in the previous section, a framework can be used to evaluate reflective database access control policies. However, users who are affecting the database management system as intruders can harm the structure of the security. Detecting and fighting intrusion in a database management system is very important. In the research work of Mechanism for Database Intrusion Detection and Response, researchers Kamra, Bertino, and Lebanon [3] have proposed a way to detect these intrusions by implementing an intrusion detection mechanism using database user profiles and requests that are coming from these profiles to the database management system. According to the above researchers, the architecture of this implementation breaks down as follow. Main components that infrastructure this architecture are conventional database management system mechanism, database audit log files, and database ID mechanism. When a user issues a regular query to the database management system, the query goes through a feature selector that extracts features and converts them into forms accepted by this system’s ID mechanism. The next component, the detection engine, then takes this input and checks with the current existing profiles in the database and forwards the features to the Response Engine. The Response Engine will consult the Response Policy Base and
determines whether to raise alarm, drop query or to take no action regarding a particular query that came from Detection Engine.

As Kamra, Bertino, and Lebanon explain, there are several factors they considered when detecting anomalous database requests. They primarily used database audit file system to track and identify the normal behavior of users. The ID mechanism was used to determine role intruders that behave unexpected manner. Every user request is associated with a role. The researchers mention that for every new user request, the role is predicted by the trained classifier. If the predicted role (or class) is different from the original role associated with the query, an anomaly is detected [3]. From those researches point of view, this methodology is not successful when the number of users who query the database is large. To address this issue, they proposed a way to build user-group profiles based on user’s database transactions. According to these profiles, Kamra, Bertino, and Lebanon have defined the anomaly as an access pattern that reflects a different pattern from the profiles. They also maintained a mapping for each user to its corresponding user-group. For every new query that comes into the database, its corresponding user-group for a user is determined based on mapping of the user and user-group. The researches have followed two approaches to determine anomaly in these queries. As the first approach, they used the naïve bayes classifier to check whether a particular user associated with an incoming query in fact belongs to its corresponding user-group. The
goal of the second approach was to identify if a particular query matches with its corresponding user-group by using a statistical test. Another stage in this database intrusion detection mechanism is how they detected the SQL inject attacks. Kamra, Bertino, and Lebanon have also proposed a framework to detect malicious behaviors of database application programs. Their approach was to create profiles of these database application programs using the SQL queries submitted to the database. Then their own anomaly detection program, based on data mining technique, was used to track behaviors of these application programs. As the researchers describe, the SQL Injection is primarily taking place due to application program vulnerably trying to change its behaviors. I believe the above architecture is a great solution to detect intrusions towards a database and identify vulnerabilities such as SQL injections attacks. However, identifying the behavior that deviates from normal pattern is a complex task. I believe that security in Database Management Systems can be improved dramatically by detecting anomaly of an application program that interacts externally.
2.4 Fighting Malicious Data Reads using Database Audit

One of the security features available in every database management systems is Audit capability. Another interesting security enforcement I came across during my research work was Online Detection of Malicious Data Access Using DBMS Auditing by Fonseca, Vieira, and Madeira [4]. The primary goal of the above researches was to detect illegal access to the database data in SQL commands. For this solution, the researchers have proposed a generic algorithm that studies the profiles of transactions that users use. As Fonseca, Vieira, and Madeira explain, the database system keeps a record of all the commands a user executes as a chain of commands. This sequence of commands can be used to identify the malicious data reads. The above researches implemented this solution in two stages. The first stage, the learning phase, uses database audit trail offline to generate a graph that represents each valid transaction. The second phase, the detection phase, uses database audit trail online to generate sequence of commands executed by users. According to the researchers, if a malicious transaction is found in the detection phase, then an appropriate defense action will be taken to prevent the transaction.
2.5 A Security Protocol to Protect Database Network Objects

Today database systems are connected via networks. Most enterprise level database systems are running as servers that allow clients to connect via a network. In the research of An Enterprise Policy-Based Security Protocol for Protecting Relational Database Network Objects, Itani, Kayssi, and Chehab [5] have presented an efficient end-to-end security architecture that enforces confidentiality and integrity to relational database objects that are flowing over network links. This architecture, which researchers call as ESCORT, is designed to enforce security into database objects without downgrading the performance and efficiency of the database application. According to the above researchers, ESCORT assures the security of the data based on the contents and sensitivity by exceeding the performance of SSL and TLS protocols. This is primarily carried out by using a customizable policy-based security architecture. As Itani, Kayssi, and Chehab describe, this ESCORT architecture operates both on the server and clients sides. As the main functionality, ESCORT makes sure the outgoing and incoming data are secured using its own encryption/decryption mechanism.

As the researchers explain, database objects are linked with their own security policies and each policy states the level and scope of the encryption operation to be used on the database. The security policy in ESCORT architecture has two main parts. The first part contains the set of security related attributes and configuration details to be used. The
second part contains the scope and strength of the operations that are related to encryption and hashing. The main Security Engine in the ESCORT architecture is responsible for ensuring the confidentiality and integrity of the database. The above researchers have performed server side implementation on Windows and Linux servers and client side implementation on Linux, Solaris, and Windows systems. In addition, this research has shown that compared to the old traditional security policy approaches, this ESCORT system can improve performance gain on both server sides and client sides dramatically.
2.6 Enforcing Database Security through Statistical and Fuzzy Approach.

In the research work of Statistical and Fuzzy Approach for Database Security, Lu and Yi [6] has proposed a Dubiety-Determining Model, based on statistical and fuzzy logic, to monitor database transactions.

The architecture of this detection system has several main components. Figure 3 shows the hierarchical order of these components. As Lu and Yi explain, UI component assists users setup monitoring policies and manages displaying of Dubiety-Determining results. The Audit Base component organizes information about each database transactions that are coming from Sensor component. Once user set rules and monitoring policies, the Mapping to Rules component converts this information into a format of detection rules that can be stored in the Detection Rules Base. As the above researchers describe, the main component in this architecture is the Event Analyzing Module. This component processes and matches each record in the Audit Base against the rules in Rule Base. The monitoring algorithm is implemented in the Event Analyzing Module.
Above researchers have implemented this architecture on various database management systems and tests have proved that this architecture is a very effective way to indentifying suspicious user behaviors in database transactions.
2.7 Database Security Protection via Inference Detection

In the research work of Database Security Protection via Inference Detection, Chen and Chu [7] has proposed a detection system that consumes users’ current query and past query log to check if a current query can infer secure information. According to Chen and Chu, knowledge acquisition, semantic inference model, and security violation detection are the main three components in this detection system. The main responsibility of the knowledge acquisition unit is to filter out data dependency knowledge, data schema knowledge, and domain semantic knowledge. The next unit, Semantic Inference Model is constructed based on acquired knowledge. This model combines data schema, dependency, and semantic knowledge. The third unit, Security Violation Model, will combine new query request with the request log and determine whether it should answer the current query based on knowledge it acquired earlier. Since users may pose queries and acquire knowledge from different sources, we need to construct a semantic inference model for the detection system to track users’ inference intention [7]. As Chen and Chu describe, this inference detection needs to acquire knowledge from data dependency, database schema, and domain-specific semantic knowledge. This detection mechanism will check the queries submitted by users and calculate the probability of inferring the secure sensitive information by answering the user query. As the above researchers explains, if the probability calculated for a user
submitted queries exceeds the pre-specified threshold, then the query request will be
denied by this detection system.
Another important database category we use today is Object-Oriented database management systems. Nowadays Object-Oriented data model is supported widely by many database management systems. Object-Oriented data model defines states and behaviors of real world objects, their constraints, and their relationship among them. Any application that depends on data objects can use Object-Oriented databases to manage their data. In order to ensure proper access to the objects in an Object-Oriented database, strong security mechanisms must be in place. However, security enforcement over Object-Oriented databases is not very strong. This chapter surveys and investigates solutions and mechanisms various academic researchers have proposed towards Object-Oriented databases.

3.1 Object-Oriented Database Security

One of the interesting solutions proposed by Slack was Security In An Object-Oriented Database [8]. In this research, Slack talks about how object in a database can be partitioned into separate protected groups and enforce security and integrity in group level. According to Slack, the objects in the database are categorized as implementation
object and interface objects. All the objects are grouped into two-way protected groups. As Slack describes, in a two-way protected group, objects are allowed to receive messages only from interface objects and send messages only to interface objects. For an example, the database will have multiple two-way protected groups and each of these protected groups will have its own interface objects. Slack explains that instead of objects directly communicating with another object, now object will send messages to the interface object in that group and this interface object will communicate with the interface object in the receiver group to complete the communication. A combined secrecy/integrity mechanism can be constructed based on the notion of protected group [8]. As Slack explains, the interface object of a particular group has the sole responsibility to protect secrecy and integrity. In addition, each interface object has the capability to get and set access control information for all the other objects. When an interface object receives a message from an object, the interface object has the ability to get access control information from the message directly or using object identifier, which is located in the message. This way the interface object can obtain access control information of source of the message as well as the destination of the message. I believe this proposed method is a strong way to enforce security and ensure the integrity by grouping objects into protected groups instead of letting object send and receive messages directly.
3.2 A Logic Specification and Implementation Approach for Database Security

Authorization is very critical topic in database security. When it comes to Object-Oriented Databases, ensuring that each user has proper authorization access to objects is very important.

Another interesting proposal towards Object-Oriented Databases I surveyed during my research work was A Logic Specification and Implementation Approach for Object-Oriented Database Security by Xianjun [9]. The main goal of this academic research was to look into high-level authorization specification by incorporating authorization into object-oriented database and mapping specification into a logic programming language and implement it. The logic programming language Xianjun proposed for this implementation is called Lo, which has the ability to formalize various features in object-oriented database management system. As I mentioned above, to incorporate authorization into the database, Xianjun defined an access fact that holds(S, R, O), where subject S has access right R on object O. I mentioned that Xianjun used a logic specification language in this solution. This high-level database security language contains a Rule Based Parser, Logic Program Translation, and Logic Program Computation. The Rule Based Parser checks for any syntactical errors in the rule base. The Log Program Translator translates rule base to logic program, and the Logic Program Computation module computes the answer set of translated logic program. As Xianjun
described, the main goal of this proposal was to develop a formal specification and authorization rules for an object-oriented database system.
3.3 Database Security using Role Based Access Control

Authorization is an important aspect in object-oriented database security. Also, access control support is very essential when making a strong authorization structure in an object-oriented database. In the research work of RBAC Support in Object-Oriented Role Databases, Wong [10] has proposed a framework that uses a role based access control mechanism for object-oriented role databases to improve security.

Wong used a data model called DOOR object-oriented data model for this mechanism. This DOOR model supports objects with roles, attributes, methods, object classes and role classes, and types. According to Wong, this model also has the capability to implement inheritance and delegation. Since this is a role based access control mechanism, privileges associated with each role are different. Wong explains according to Role Based Access Control Model, whenever an object changes its role, its corresponding privileges will also change. In addition to this, he mentions that whenever a job function of an object changes, its privileges must be updated.

Wong modeled the authorization such a way when subject S is invoking a method that tries to change attribute of an instance, the authorization mechanism would check whether the subject S has proper privileges to do so. There are two main categories under this authorization mechanism. The first category, Role Related Authorization was modeled by using authorization objects Setof-Owned-Roles and Owned-Roles. The
second category, Class Hierarchy Authorization was modeled based on creation of class hierarchies and sub classes of a class. As Raymond describes, all the users in this framework are represented as objects. Also, issues such as Temporal Authorization and Separation of Duties were addressed in this research under authorization. This proposed framework has showed how Role Based Access Control model can be used along with proper authorization mechanism to improve security in object-oriented role databases.
3.4 Statically Detecting Security Flows in Databases

During my research work and survey phase, I learned that many of the security issues that are being addressed by academic researchers are related to Access Control. In the research of Static Detection of Security Flaws in Object-Oriented Databases [11], researcher Tajima has developed a static detection technique for security flaws that can allow malicious users to bypass the encapsulation and harm primitive operations inside functions. In an object-oriented database system, users have the ability to invoke primitive operations inside functions as long as the users have proper grant privilege. As part of this detection mechanism, firstly Tajima developed a framework that encompasses security requirements that need to be satisfied in a database. As the next step, he implemented an algorithm that can analyze program code of the function and check if the given security requirements are met syntactically. His basic data model is consisted of mutable classes and objects, and users are given privileges to invoke functions to access data objects in the database system.

As I described earlier, once the security requirements are in the framework, the next phase is the algorithms to detect the security flaws. This algorithm detects flaws using a reasonable amount of computation and determines whether the given security requirements are satisfied properly. Even though this algorithm is very efficient in
detecting flaws, Tajima explains that this is very pessimistic and another way of detecting security flaws would be to dynamically analyze the execution query of a given user.
3.5 Authorization and Revocation Mechanisms in Object-Oriented Databases.

In object-oriented databases, we have users trying to access objects and invoke their attributes. In this case, ensuring that users have proper privilege to invoke methods in an object is very important. Majetic and Leiss have showed how important the authorization can be in an object-oriented database environment in their research of Authorization and Revocation in Object-Oriented Databases [12]. According to the above researchers, the researches that have been conducted towards the object-oriented database systems are very less. In this research, the above authors propose a solution that states users have more direct control over their resources by associating each grant propagation numbers. Majetic and Leiss explained that bounded propagation of privileges gives users chance to manipulate control over access privileges after the privileges have been granted. The propagation numbers govern the grantability and exercisability of the privileges [12]. The researchers have targeted this solution primarily on inheritance feature. As a foundation to their solution, the authors have proposed the following authorization policies that can be applied to object-oriented database systems.

Policy 1: A user who has access to parent type will have the same level of access to the inherited attributes in sub classes.
Policy 2: Access to a class entirely defines access to all the attributes in that class and attributes inherited from parent class. In the case of multiple inheritance, then the access will be to union of the sub classes

Policy 3: Access to the super class does not give access to the attributes defined in the sub class.

The authorization mechanism in this research work was based on propagation numbers. They have used Vertically, Horizontally, Combined and Limited bounded propagations. The same way the privileges are granted, users can lose their privileges if the granter believes that users have failed to maintain the privileges. For example, any user who can grant privileges to another user can also revoke it. The Figures 4 and 5 describe the algorithm Majetic and Leiss created to implement this authorization and revocation mechanisms.
Figure 4: Granting Algorithm [12]
Revoke( )

{ find UG with object & negative status for each right in given 
user’s UG

{ if (fn & granter is OK ) 
for all attributes delete user’s attribute upon the policy

if ( UG has no attribute ) delete it

Figure 5: Revoking Algorithm [12]
3.6 Object-Oriented Database Security through Different Access Controls

In the research work of A Implementation of Object-Oriented Database Security [13], Ambhore, Meshram, and Waghmare have addressed efficiency of incorporating high-level authorization into object-oriented database. Throughout this research, these researchers discuss issues associated with current security models such as Discretionary Access Control Model, Mandatory Access Control Model, and Role Based Access Control Model.

Using Discretionary Access Control, authors in this research describe the approaches taken towards authorization on object-oriented databases. According to Ambhore, Meshram, and Waghmare, in object-oriented databases, authorization breaks into full and partial authorization. In the case of full authorization, users who have rights on an object have same operation rights for components of that object. In the case of partial authorization, having access to an object does not necessarily grant access to object’s components. In addition, this research states that in the case of inheritance, if a user has access to a parent object, then same level of access will exist for all its descendent objects. However, a user who has access to descendent objects will not have access to view the objects’ ancestors. Under the Mandatory Access Control, Ambhore, Meshram, and Waghmare discuss the proposed security models for object-oriented database management systems. These models are Sorion Security Model and Jajodia-Dogan
Security Model. Using Role Based Access Control, the above researchers describe how important the access control decisions that are based on roles and how GRANT security feature affects Role Based Access Control in an object-oriented database environment. As this research points out, today object-oriented database management systems have become more commercialized. Therefore, when an access control mechanism is designed for a database, we need to consider the issues that are being faced by current security models such as authorization, access types, authorization types, and visibility from above and below.
3.7 The Constraints of Object-Oriented Database

Zaqaiheh and Daoud have addressed security constraints facing Object-Oriented database in their research work of The Constraints of Object-Oriented Databases [14]. According to the above researchers, data accuracy, consistency, and integrity in an object-oriented database environment are very crucial.

Some of the open problems above researchers brought out in this research are in a scenario where inheritance is implemented and it is related to integrity constraints. The first open problem is the efficiency of enforcing and maintaining integrity constraints when copying object to another object in run-time, especially the down casting of objects. The other open problem they were interested in was when a class inherits attributes from multiple parent classes. In his case, if the same attribute name exists in the parent class, then we need to declare a pure virtual functions or virtual class. Another problem the above researchers brought out was that object-oriented data model and relational data model can be combined and the result will be an object relational model. However, the data organization will be similar to an object-oriented structure. The research in [14] states that integrity model can be built on the Object-Oriented data mode. However, these researchers state that in the integrity model, there are more constraint domains. They believe as object-oriented database applications move into different areas, more constraints such as image, video, and audio will be developed.
This chapter will cover the case study of Oracle Database Management System and its security features. I have identified a set of important security features in Oracle and illustrated them in detail. I have used Oracle Database Management System released version 11g for this project and all the PL/SQL queries were generated in a local system for experimentation and illustration.

Because of the technology today, we have many database management systems. Among them, Oracle provides a large number of security features that database administrators can use to enforce security. Oracle Database Management System supports object-oriented data model as well as Relational data model. We have the capability to implement inheritance and polymorphism by creating types in Oracle database. I believe identifying how security affects object-oriented features, as well as relational features is very crucial. Thus, my illustrations on these security features have spread on both relational and object-oriented features in Oracle Database Management System. The set of Oracle security features I studied and illustrated in this project are Grant, Virtual Private Database, Revoke, Table Space Security, Profiles, and Audit. To illustrate each of
the above features, I created a database in a local Oracle database environment that involves secure data and generated my own PL/SQL queries.

In Oracle, the main database administrator has the first privileges of using each security feature. If a certain administrator uses the above security features in a very efficient way, then the security enforcement in Oracle can be improved and maintained for a long time. During this illustration, I have identified how strong and weak these security features are and what are the options we have as a database administrator when using these security features.
4.1 The Bank Database

The database in Figure 6 was used as the reference database for my illustration. This database contains secure data similar to a bank environment.

![Bank Database Diagram]

Figure 6: Bank Database
4.2 Database administrator and users

In Oracle Database Management System, the main administrator by default is SYS. Any other types of database administrators who need to perform administrative tasks that are defined in SYSDBA or SYSOPER, need to log into the database system as SYS and using SYSDBA role. Using the SYSDBA role, administrators have the capability to create more administrators with limited privileges. Using SYSDBA role, I created a main administrator, BANKADMIN for my database and granted DBA privileges to BANKADMIN. Oracle has defined a large number of administrative privileges under DBA role. After login into my bank database as BANKADMIN, I created five regular users and used them accordingly for illustration purposes in my each security feature. The following figure, Figure 7, describes the user hierarchy in my Bank database.
4.3 GRANT Security Feature

In Oracle Database Management System, Grant feature is being used to grant any access privileges on objects and user roles to users. Any owner of an object has the ability to grant privileges to other users to access its own object. However, the most important fact is that this Grant feature must be used very carefully and appropriately. Granting access privileges on an object to an unknown user can harm the security enforcement of the database. To illustrate this security feature, I created the following example SQL queries. Newly created users do not have CREATE SESSION privilege in Oracle. Therefore, the
administrator first has to grant the CREATE SESSION privilege to new users. The following query describes this scenario.

BANKADMIN submits the query:

grant create session to user3;
grant select on customer to user3;

The above grant privilege will allow USER3 to log into the BANK database at any time and perform a read operation on CUSTOMER table, which is under BANKADMIN schema. The above grant statement gave USER3 the entire access to object CUSTOMER.

For example, now USER3 does the following query.

select * from bankadmin.customer;

As the result, now the entire table will be visible to USER3.

Another benefit of Grant security feature in Oracle is ability to grant roles to users. Administrators can create roles that encompass multiple privileges and grant that role to users. For example, as the BANKADMIN, I created a role named DEVELOPER and granted couple of privileges to the DEVELOPER. The following query explains this scenario.

create role developer;
grant insert on bank to developer;
grant update, delete on bank to developer; grant developer to user3;
The above last SQL query will grant insert, update, and delete privileges on BANK table to user3. For example, submitting the following query will insert data into BANK table from USER3.

```
insert into bankadmin.bank values(3, 'Citi', '444 L Street Sacramento CA 95816');
```

Grant security privilege indeed helps object owners to grant privileges to users on behalf of their objects. However, could allowing a user to access the entire table of data be harmful? Yes. For example, CUSTOMER table in BANK database contains secure information such as Social Security numbers and user ID of other users. Granting the select privilege to user can expose the entire table to that user. Currently the Grant security feature does not have a control over information of a table. The other questions is to how to enforce row level security to CUSTOMER table, thus USER3 can see only the information pertaining to USER3. The solution is Virtual Private Database. I have illustrated this security feature in the next section.

Also, the Grant feature can be used in an Object-Oriented database environment to enforce the security in object level, especially when a user needs to access a table of objects belongs to another user.
The following example illustrates Grant security feature in an environment where an Object-Oriented feature, inheritance, is implemented. Currently ACCOUNT table is part of the BANK database. For example, USER5 creates an object type similar to ACCOUNT table.

USER5 submits the query:

```sql
create type account_type as object(
  accno number(7),
  balance number(10),
  branchid number(2),
  userid varchar(10)) not final;
/
```

The above type, ACCOUNT_TYPE, is not a final type. Thus, a subtype can be created by inheriting this parent type.

Now, USER5 creates a sub type, ACCOUNT_TYPE_CHILD, by deriving the parent type ACCOUNT_TYPE.

USER5 submits the query:

```sql
create type account_type_child under account_type(
  user5points number(4));
```
In Oracle database, users have the capability to create objects and tables of objects based on predefined user type. The following query submitted by USER5 creates a table of objects of type ACCOUNT_TYPE and ACCOUNT_TYPE_CHILD.

create table user5_account_type_obj of account_type;
create table user5_account_type_child_obj of account_type_child;

Now USER5 has the ability to create objects of type ACCOUNT_TYPE and store them in ACCOUNT_TYPE_OBJ table. The following query illustrates the creation of these objects by USER5.

insert into user5_account_type_obj values(account_type(5555555,400,11,'user5'));
insert into user5_account_type_child_obj
values(account_type_child(5555556,800,10,'user3', 66));
insert into user5_account_type_obj values(account_type(5555557,600,13,'user3'));

In Oracle database, an object owner can grant execution privileges to another user to reference owner’s object type and create objects. The following query illustrates USER5
granting execution privileges on USER5’s inherited type ACCOUNT_TYPE_CHILD to user USER4.

grant execute on account_type_child to user4;

Now USER4 has the ability to create objects referencing USER5’s subtype ACCOUNT_TYPE_CHILD.

USER4 submits the queries:

create table user4_account_child_obj of user5.account_type_child;

insert into user4_account_child_obj values(user5.account_type_child(4444444, 300, 12, 'user4',33));

I have experimented several scenarios in this Object-Oriented feature using Grant security feature. My first scenario was to find out as the owner of the type ACCOUNT_TYPE_CHILD, whether USER5 has access to USER4’s USER4_ACCOUNT_CHILD_OBJ table.

To illustrate this, USER5 submits the query:

select * from user4.user_account_type_obj;
But according to Oracle, even though USER5 is the owner of type ACCOUNT_TYPE_CHILD, USER5 does not have privilege to view USER4’s objects in USER4_ACCOUNT_CHILD_OBJ table. The Oracle gave following error.
ORA-00942: table or view does not exist

However, USER5 can view USER4’s above table, if USER5 is granted with proper privileges by USER4. In order to achieve this, first USER5 has to grant execute privilege on ACCOUNT_TYPE_CHILD to user4 with GRANT OPTION. The following query illustrates this scenario.
USER5 submits the query:
grant execute on account_type_child to user4 with grant option;

Now, USER4 can grant SELECT privilege behalf of his or her objects to USER5.
grant select on user4_account_child_obj to user5;

As a result of the above query, now USER5, the owner of type ACCOUNT_TYPE_CHILD, is able to view objects of USER4. The above scenario illustrates the powerful functionality of Grant feature in object level.
Another scenario I experimented was to find out whether USER4 has the ability to reference USER5’s parent type when USER4 has reference privileges only on USER5’s child type.

I have illustrated this scenario by using the following query.

USER4 submits the query:

```sql
create table user4_account_type_obj of user5.account_type;
```

However, Oracle outputs the error:

ORA-00902: invalid datatype

This illustrates that even though a USER4 is able to reference USER5’s inherited child type, USER4 does not have privilege to reference USER5’s parent type.

This can only be achieved by granting reference, EXECUTE, privilege explicitly to USER4 to reference USER5’s parent type.

To illustrate this, USER5 submits the query:

```sql
grant execute on account_type to user4;
```

Now, USER4 is capable of creating objects reference USER5’s parent type ACCOUNT_TYPE. The above scenarios illustrate how Oracle’s object-oriented data
model environment behaves and how Grant security feature affects when inheritance is implemented.
4.4 Virtual Private Database Security Feature

4.4.1 Row level security with Oracle Virtual Private Database

Virtual Private Database provides us the capability to create table level security policies and attach them to tables. By enabling Virtual Private Database, users can enforce row level as well as column level security to tables and control the information other users can access. Virtual Private Database feature in Oracle enforces security, to a fine level of granularity, directly on the database table, views, or synonyms [15]. In the previous section, I illustrated that after granting SELECT privilege on object CUSTOMER to a user, the entire table is visible to USER3. This visibility can be controlled by using Oracle Virtual Private Database feature. For example, when a user queries a particular table using the following query, the Oracle system enforce the Virtual Private Database mechanism by embedding a predicate into the user submitted query.

Assume a user does the following query:

```sql
select * from john.orders;
```

As a result of Virtual Private Database, the above user SQL query will be then changed into:

```sql
select * from john.orders
where order_id='2332';
```
To enable Virtual Private Database feature in Oracle, database administrators need to implement the security policy and a function to inject predicate to the user’s SQL query. I have implemented Virtual Private Database feature to CUSTOMER table under BANKADMIN schema. I have used column USERID in CUSTOMER table as part of the predicate to implement this security feature. The following code describes the function that was used to get user information from current user session.

create or replace package body pkg_cust_vpd

as

function predicate (obj_schema varchar2, obj_name varchar2) return varchar2 is

begin

return 'userid =lower(sys_context(''userenv'',''session_user''))';

end predicate;

end pkg_cust_vpd;
/

The following code creates a custom security policy CUST_POLICY and attaches to CUSTOMER table.
begin

sys.dbms_rls.add_policy(
  'bankadmin',
  'customer',
  'cust_policy',
  'bankadmin',
  'pkg_cust_vpd.predicate',
  'select,update');

end;
/

The key point of the above policy is that when a user tries to query the CUSTOMER table using any SELECT or UPDATE statements, the above policy will invoke the function PREDICATE to enforce Virtual Private Database feature.

The following SQL query by USER1 illustrates that this Oracle Virtual Private Database can be used to control row level information.

USER1 issues the query:

select * from bankadmin.customer;
Figure 8 shows the result for the above SQL query. In another scenario, USER4, who has SELECT privilege on CUSTOMER table, executes the following query.

Select * from bankadmin.customer;

Even though USER4 has read access to the CUSTOMER table, CUST_POLICY will not allow USER4 to see any information since USER4 does not exist in the CUSTOMER table. The Oracle will simply output no result for the above query. This confirms that once the Virtual Private Database feature is enforced, users can only see data that are pertaining to them.

However, I believe there is a one drawback in this security feature in terms of implementation. To implement this feature along with the security feature, a good predicate must be used. In this example, I have used USERID column, which is a sensitive information in CUSTOMER table. However, implementing this Virtual Private Database feature into a table that does not contain a column that can qualify as a strong
predicate can be a problem. In this case, adding predicate column to a table that has a large number of data would be a complex task.

Also, currently the Oracle Virtual Private Database feature supports the capability of adding one predicate. Thus, administrators can use only one column field to implement virtual private database feature. Currently it is not possible for administrator to include multiple column fields when implementing Oracle Virtual Private Database feature.

The above scenario illustrates applying row level security to a table using Oracle Virtual Private Database feature

4.4.2 Column level security with Oracle Virtual Private Database

Oracle provides capability to add column level security the same way by using PL/SQL functions and Oracle policy. Tables contain column level data that are very secure. The same way we block users seeing row level data that are pertaining to them, we should allow users to see only the column level data that are pertaining to them. I have illustrated this scenario using the CUSTOMER table. For example, CUSTOMER table contains both USERID and PASSWORD data for each user. BANK can have more than one manager. For example, a manager who is login into the BANK database as USER1 should be able to see all of the customer information except their passwords. This can be
achieved by implementing column level security via Oracle Virtual Private Database security feature.

I implemented column level security using the similar steps I took to implement row level security. Firstly, I created a function, which is part of the policy HIDE_PWD_POLICY, I implemented as BANKADMIN. The following code implements the function.

```
create or replace function hide_pwd (  
    v_schema in varchar2,  
    v_objname in carchar2)  
return varchar2 as  
con carchar2 (200);  
begin  
    return 'userid =lower(sys_context(''userenv'',''session_user''))';  
end hide_pwd; /
```

The following code implements the security policy for this column level security.

```
begin  
    dbms_rls.add_policy (  
        object_schema => 'bankadmin',  
        object_name   => 'customer',
```
The above HIDE_PWD_POLICY will be executed when a user queries the CUSTOMER table and the function HIDE_PWD will be invoked. According of this policy, any user, for example, the bank manager, who queries CUSTOMER table, will be able to see all the user information except their passwords. The user can see only his or her password and the password field (PWD) for other users will be blocked.

For example, USER1 is the bank manager in my BANK database. Now, USER1 logs into the BANK and queries the CUSTOMER table to see all the customers.

USER1 submits the query:

Select * from bankadmin.customer;

Figure 9 shows the Oracle output for the above query. As the Figure 9 shows, USER1 has the ability to see only his or her password field. The password fields of all the other users are blocked and showed empty values.
Figure 9: Result for USER1’s Query 2

Oracle Virtual Private Database security feature gives database administrators a great opportunity to protect data in their databases. Securing data row level and column level improve the security enforcement in database management systems tremendously.
4.5 Revoke Security Feature

Same way the object owners and administrators have ability to grant privileges, they can also revoke privileges from users. If a trust level and reliability of a user decreases over time, the granter can revoke the previously granted set of privileges and roles. To illustrate this security feature, I have used the following scenario. Currently USER2 has SELECT privilege on BANK_BRANCHES table.

For example, USER2 submits the following query to the Oracle system:

```sql
select * from bankadmin.bank_branches;
```

The Figure 10 shows the output for the above query. Now, the BANKADMIN revokes the SELECT privilege on BANK_BRANCHES table from USER2.

```sql
Revoke select on bank_branches from user2;
```

Now, USER2 issues the same query to read BANK_BRANCHES table, but since SELECT privilege is revoked, Oracle will give the following error message.

ORA-00942: table or view does not exist

<table>
<thead>
<tr>
<th>MAINBANK</th>
<th>BRANCH_NO</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>123 P Street Sacramento CA 95817</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>200 Bicentennial Cir Sacramento CA 95826</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>56 Main Street Los Angeles CA 90015</td>
</tr>
</tbody>
</table>

Figure 10: Oracle Output for USER2’s SQL Query
The same way, using the following query, owners can revoke the entire set of previously
granted privileges.

BANKADMIN submits the query:

Revoke all on customer from user2;

For example, USER2 submits the following query:

update bankadmin.customer set(name)=('John') where ssn=111111112;

As the result, the Oracle will now output the following error message:

ORA-00942: table or view does not exist
4.6 Tablespace Security Feature

Another important security feature provided by Oracle database management system is Tablespace feature. According to Oracle Concepts Guide, in terms of storage, every database is divided into different storage units named tablespaces. By default, Oracle creates database in a tablespace named SYSTEM. When an administrator first creates a database, Oracle creates this new database in the SYSTEM tablespace. In addition to this, Oracle has another tablespace called SYSAUX tablespace, which is an auxiliary tablespace to the SYSTEM tablespace.

The SYSAUX tablespace provides a centralized location for database metadata that does not reside in the SYSTEM tablespace [16]. As I mentioned, these tablespaces are provided by Oracle by default. However, Oracle provides administrators the capability to create custom tablespaces and create new users in these tablespaces. With this security feature, database administrators can control the storage requirements pertaining to database and its users.

The following code describes the creation of a tablespace named TS_ONE to illustrate tablespaces feature in my BANK database.

For example, BANKADMIN executes the query:

```
create tablespace ts_one
logging
```
Once the tablespace TS_ONE is created, administrators have the ability to create users in this tablespace.

To illustrate this scenario, I created a new user called USER4 in the tablespace TS_ONE. The following code illustrates the user creation.

```sql
create user user4 identified by user4
default tablespace ts_one
quota 1m on ts_one;
```

Once the USER4 has been granted with proper privileges, USER4 can create tables. For example, USER4 executes the following query to create a new table:

```sql
create table user4_personal(    id number,
    fname varchar(10),
);```
INAME VARCHAR(10),
TAXAMOUNT NUMBER);

Since the USER4 was created in the tablespace TS_ONE, USER4_PERSONAL table will be created in tablespace TS_ONE.

One of the important functionality embedded in the tablespace security features is the ability to bring the tablespace offline. Administrators can enforce security by bringing any tablespaces into an offline mode. This enforcement blocks any users in offline tablespaces accessing their objects or performing any operations. To illustrate this functionally, I logged into the BANK database as BANKADMIN and brought the TS_ONE tablespace into an offline mode.

As BANKADMIN, I executed the query:
ALTER TABLESPACE TS_ONE OFFLINE NORMAL;

Now USER4 tries to execute the query:
SELECT * FROM TMP_PERSONAL;

As the result, Oracle will now output the error message:
ERROR at line 1:
ORA-00376: file 6 cannot be read at this time
ORA-01110: data file 6: 'C:\ORACLEDATA\TS_ONE.DBF'
Above functionality is a strong way to enforce security by taking the tablespace offline. If an administrator needs to prevent any user performing any operations temporary, that user can be blocked by bringing the user’s tablespace offline. Once the tablespace is offline, users in that tablespace are unable to perform any DML or DDL operations.

For example, USER4 tries to create a table by submitting the SQL query:

```sql
create table points(
    userid varchar(10),
    points number,

    CONSTRAINT points_pk PRIMARY KEY (userid));
```

However, tablespace TS_ONE is currently offline. Thus, Oracle will output the following error message.

ORA-01542: tablespace 'TS_ONE' is offline, cannot allocate space in it

Another way database administrators can enforce the security in a database is by limiting the user quota in the tablespace. Database administrators explicitly specify the quota when users are created in a specific tablespace. This is a one way of controlling the amount of storage users are allowed to use. When it comes to security, user quota is important. I have illustrated this functionality in tablespace security using the following scenario.
I created a new user, USER5, in the TS_ONE tablespace with a user quota of 0 megabytes using the query:

```sql
create user user5 identified by user5
default tablespace ts_one
quota 0 on ts_one;
```

For example, now USER5 tries to execute the query:

```sql
create user user5 identified by user5
default tablespace ts_one
quota 0 on ts_one;
```

As the result, Oracle will output the following error message.

ORA-01536: space quota exceeded for tablespace 'TS_ONE'

This is very useful when an administrator needs to maintain a large pool of users in a secure database. When users are created and granted with object related privileges, they have ability to create and invoke any methods. If a user is given with a very large or unlimited quota amount, then this could create other drawbacks. Many intruders who are trying to harm a database would try to use large disk storage amount to accomplish their
tasks. However, this can be controlled by creating users in a controlled tablespace. If the quota for a particular tablespace is fixed, then all the users in this tablespace will have a control over the disk amount they are allowed to use. The following queries illustrate this scenario.

BANKADMIN create a tablespace TS_TWO by submitting the query:

```
create tablespace ts_two
  logging
  datafile 'C:\OracleData\ts_two.dbf'
  size 1m
  autoextend off;
```

Now, BANKADMIN creates two users in this tablespace.

```
create user user3 identified by user3
  default tablespace ts_two
  quota 1m on ts_two;
```

```
create user user6 identified by user6
  default tablespace ts_two
  quota 1m on ts_two;
```
According to the above query, both users user3 and user6 will be limited to the tablespace TS_TWO quota 1m.
4.7 Audit Security Feature.

Looking at the users’ operations and their activities is very important to get a good picture of the security of the entire database system. Oracle Audit security feature is a strong logging functionality that helps administrators and users to follow the activities and operations other users perform on objects in their schema. Using the Oracle standard auditing capability, any user can audit SQL statements, privileges given to users, schema objects, and activity of the network in their own schema. When auditing is enabled to track certain user operations, Oracle tracks each and every operation individually and then administrators and users can take look at them at a later point in time to check or investigate the operations. To illustrate this security feature, I enabled auditing for certain operations as BANKADMIN in BANK database. The following SQL code illustrates Oracle audit feature that audits all the accesses to the CUSTOMER table by any user.

For example, BANKADMIN submit the SQL query:

```
audit select on customer by access;
```

Now all the SQL statements that interact with the CUSTOMER table are audited. In Oracle database system, all the audit logs are saved in DBA_AUDIT_TRAIL by default. As BANKADMIN, I could query the DBA_AUDIT_TRAIL investigate the entire audit logs.
For example, to look at the audit logs enabled by the above query, I submit the SQL query:

```
select username, obj_name, action_name, extended_timestamp from dba_audit_trail
where owner='BANKADMIN' and username='USER1';
```

The above SQL query will read all the audit logs stored in the DBA_AUDIT_TRAIL that are pertaining to USER1. Figure 11 shows the Oracle output that displays audit information for this scenario.
<table>
<thead>
<tr>
<th>USERNAME</th>
<th>OBJ_NAME</th>
<th>ACTION_NAME</th>
<th>EXTENDED_TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>19-OCT-09 11.22.10.986000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>19-OCT-09 11.22.10.987000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>20-OCT-09 12.02.23.636000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>20-OCT-09 12.02.23.637000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>20-OCT-09 01.22.54.028000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>20-OCT-09 01.22.54.029000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 10.39.31.995000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 10.39.31.995000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 10.40.06.567000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 10.40.51.883000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 10.40.51.883000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 10.41.30.412000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 10.41.30.412000 AM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.31.00.689000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.31.00.690000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.44.09.536000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.44.09.537000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.51.07.533000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.51.07.533000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.54.36.152000 PM -07:00</td>
</tr>
<tr>
<td>USER1</td>
<td>CUSTOMER</td>
<td>SELECT</td>
<td>24-OCT-09 07.54.36.153000 PM -07:00</td>
</tr>
</tbody>
</table>

Figure 11: Oracle Audit Log 1
Oracle also provides a way to enable audit for individual users. The following SQL query illustrates this scenario.

For example, to audit USER5’s activities as BANKADMIN, I submit the SQL query:

```sql
audit select table, update table by user5;
```

To monitor USER5’s operations, I used the following SQL query.

```sql
select username, timestamp, obj_name, action_name from dba_audit_object where username='USER5' and (obj_name='BANK' or obj_name='CUSTOMER');
```

Figure 11 shows the audit log generated for USER5. I have illustrated the basic functionality in Oracle standard audit security feature. Using this security feature, administrators and users can monitor activities that are related to their schema.

Only way database administrators can keep track of their users activities is using Oracle Audit feature. By configuring and scheduling the audit feature efficiently, security in a database can be improved dramatically. Bad users always take time and build up malicious activities to harm a system. But using Oracle Audit security feature, database administrators have the ability to track users activities in detail such as number of logins, time of login, statements executed, and objects accessed. One way we can improve security is by preventing harmful actions by users. Oracle Audit is a very strong security
feature that can be used to study users’ behaviors and enforce take proper security actions.
4.8 Profile Security Feature

When users are created in the Oracle database, administrators have capability to create users under a profile. Oracle Profile security feature enables administrators control users behavior in the database and limit the resources users can use. Maintaining a control over users’ loggings and password usage is very important to database security. One way database administrators can control user authorization mechanism is through Oracle Profiles. I have illustrated this security feature by creating a profile in the BANK database and new users under the new profile. As BANKADMIN, when I apply a profile to a set of users, all the rules in that profile affect the representing set of users.

The following SQL query creates a basic Oracle Profile named USER_PROFILE.

create profile user_profile
limit
  sessions_per_user 1
  failed_login_attempts 2
  connect_time 5
  idle_time 3;
Under the above profile, users can create only one logging sessions at a time. Also, this profile limits the number of login attempts to two. Thus, if a user fails login two consecutive times, his or her account will get locked. Also, if a user under this profile stays idle in the database more than three minutes, Oracle will rollback all of her or his last operations. I included USER2 into the USER_PROFILE using the following SLQ query.

```
alter user user2 profile user_profile;
```

I manually failed USER2’s login attempts two times. During the third login attempt, Oracle gives the following error message to USER3 denying the login access.

```
ORA-28000: the account is locked
```

I believe this is a strong functionality under Oracle Profile feature to enforce security. Today many intruders attempt to break into database systems by carrying out multiple login attempts. For example, by limiting unsuccessful login attempts for a user, we can prevent malicious users attempting to login to the database using false credentials. Also, this is a strong mechanism to control actions of individual users. We have ability to create one profile and associate it with all the users or create profiles for different groups of users and associate them.
Today many database management systems are equipped with latest security defense mechanisms and features. How strong these security features can enforce security depends on how accurate we implement, enable, and use them with our database applications. As I presented, improving the authorization and access control mechanism in Relational and Object-Oriented databases are the most critical security mechanism academic researchers are working on. If any database management system fails its authorization mechanism, then its first line of defense is in danger. Research work in [2][10][12] are very strong solutions that can improve security enforcement. Once the authorization and access control security are in place, the next stage in database security is to secure the data. The research work in [4][5][11] provide very strong mechanisms to achieve data security. The research conducted academic researches towards database security has offered us many solutions. Developers and database administrators can use these solutions to improve security enforcement in databases during their design phases. In addition to the research work, I was able to get a good understanding of how security features work in Oracle database management system. By identifying these security features, I realized that if I can enable and use these features, the security enforcement
could be improved dramatically. For example, Oracle Virtual Private Database feature offers a unique capability to implement security in row level and column level. As academic students, we always interact with a database management system as regular database users. During this project, I was able to identify and illustrate security features in Oracle database management system as a database administrator as well as a regular user. The security features that Oracle provides are very useful to design and implement a secure database. In conclusion, security features in Oracle database management system can be utilized to improve security enforcement in both Relational and Object-Oriented database management systems.
BIBLIOGRAPHY


ACM SIGMOD Record, pp. 341-352, 1996.


