DATABASE SECURITY:

AUTOMATED INTRUSION DETECTION AND DATA INTEGRITY

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Thomas Le

SPRING
2014
DATABASE SECURITY:

AUTOMATED INTRUSION DETECTION AND DATA INTEGRITY

A Project

by

Thomas Le

Approved by:

__________________________________, Committee Chair
William Mitchell, Ph.D.

__________________________________, Second Reader
Behnam Arad, Ph.D.

____________________________
Date
Student: Thomas Le

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

Nikrouz Faroughi, Ph.D.  

__________________  Date

Department of Computer Science
Abstract

of

DATABASE SECURITY:

AUTOMATED INTRUSION DETECTION AND DATA INTEGRITY

by

Thomas Le

The Internet enables people to communicate with others from different parts of the world, and businesses to expand their markets beyond national borders. The number of internet users has increased 8 folds since 1998, from 170 million in 1998 to 2760 million in 2013 [1]. According to Forbes, “for the full year it estimates that US ecommerce will generate $262.3 billion in sales, an increase of 16.4% year over year and slightly higher than the 16.2% increase last year. By 2017, eMarketer estimates that there will be $440 billion in sales for a compound annual growth rate (CAGR) of 13.8% [2].” The growths of online companies such as Amazon.com and EBAY.com have outpaced the traditional brick-and-mortar stores. To deal with the massive data generated, companies maintain commercial database systems such as Oracle® Database, MySQL, Microsoft® SQL, and IBM® DB. These databases are increasingly under attacks from hackers for their valuable customers’ information and proprietary technologies. Protecting the database is therefore, the utmost concern of these database administrators (DBAs). Commercial databases have some forms of built-in security and auditing features that allow DBAs to audit their database systems. However, the auditing process can be complicated and is very time consuming,
as DBAs have to set polices and manually perform the audit. Therefore, suspicious activities may not be discovered until much later time. Audit data are stored in the extensive built-in system logs, and they can only be queried through the predefined log views. In a similar way to encrypted data, which has no further useful purpose in the encrypted form, these audit data cannot be used to create custom security reports and alerts, other than to read-only.

To be able to use these audit data for other beneficial purposes, such as creating custom security reports and alerts, manipulation of these data is required. This project introduces an application (app) that utilizes Oracle DBMS_scheduler package to automatically create programs and jobs to collect data on pre-determined schedules. The app also creates predefined tables, which are similar to Oracle audit views, to manipulate these data, without affecting the data integrity. The application is coded in Oracle’s 3GL language, PL/SQL, and tested on Oracle® SQL database version 11g Express Edition.

This project demonstrates that a user friendly, yet effective, database-monitoring application can be quickly designed and built. The app automatically notifies the database administrators about violations of unauthorized operations and accesses to particular database system resources.

______________________, Committee Chair
William Mitchell, Ph.D.

______________________

Date
ACKNOWLEDGEMENTS

Dr. Mitchell has been instrumental in helping me define and refine this project. The knowledge I gained from his CSC204 and CSC176 has certainly helped me with finishing this project in a timely manner. I am deeply appreciative of Dr. Mitchell’s dedication, his prompt responds to my many emails, and accommodations of odd hour meetings.

I would also like to thank Dr. Arad for being my second reader. I am forever indebted to Dr. Arad for his guidance throughout the years.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Acknowledgements</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>x</td>
</tr>
<tr>
<td>List of Source Code Files</td>
<td>xii</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 About this project</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Problem</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Motivation for Project</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Objective</td>
<td>4</td>
</tr>
<tr>
<td>2. BACKGROUND OF THE STUDY</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Oracle® Database</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Oracle® 11g Express Edition (Oracle Database XE)</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Oracle® 3GL (PL/SQL)</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Oracle® Auditing</td>
<td>8</td>
</tr>
<tr>
<td>3. THE PROPOSED APPLICATION (THE APP)</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Design</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Code</td>
<td>12</td>
</tr>
<tr>
<td>3.3 Test</td>
<td>16</td>
</tr>
<tr>
<td>4. RESULTS AND SUMMARY</td>
<td>23</td>
</tr>
</tbody>
</table>
4.1 Results...........................................................................................................23

4.2 Summary ....................................................................................................... 23

Appendix A. Project source codes ................................................................. 25

Appendix B. Predefined custom log tables ................................................. 61

References ....................................................................................................... 65
<table>
<thead>
<tr>
<th>Figures</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>ORA-04089: cannot create triggers on object owned by SYS</td>
<td>3</td>
</tr>
<tr>
<td>2.</td>
<td>Oracle 11g Audit Trail View</td>
<td>9</td>
</tr>
<tr>
<td>3.</td>
<td>System Diagram</td>
<td>12</td>
</tr>
<tr>
<td>4.</td>
<td>Schema GRADE_DEPOT (CREATE, INSERT, and UPDATE)</td>
<td>16</td>
</tr>
<tr>
<td>5.</td>
<td>Oracle’s dba_audit_trail view showing audited actions</td>
<td>17</td>
</tr>
<tr>
<td>6.</td>
<td>Custom log table, gen_report, showing the same audited actions</td>
<td>17</td>
</tr>
<tr>
<td>7.</td>
<td>Table, physics101, with UPDATE, DELETE, and DROP TABLE</td>
<td>18</td>
</tr>
<tr>
<td>8.</td>
<td>Oracle’s dba_audit_trail with UPDATE, SELECT, DELETE, and DROP</td>
<td>19</td>
</tr>
<tr>
<td>9.</td>
<td>Custom log table, gen_report, with DML and DDL statements</td>
<td>20</td>
</tr>
<tr>
<td>10.</td>
<td>Failed login attempts by username SUPERMAN</td>
<td>20</td>
</tr>
<tr>
<td>11.</td>
<td>Failed login attempts alert</td>
<td>21</td>
</tr>
<tr>
<td>12.</td>
<td>Failed login attempts report of all failed logins</td>
<td>21</td>
</tr>
<tr>
<td>13.</td>
<td>UPDATEs to tables PHYSICS101 and CHEM301</td>
<td>22</td>
</tr>
<tr>
<td>14.</td>
<td>UPDATE alert showing the two UPDATEs on the tables</td>
<td>22</td>
</tr>
<tr>
<td>15.</td>
<td>UPDATE report showing the two above UPDATEs</td>
<td>22</td>
</tr>
<tr>
<td>16.</td>
<td>Predefined custom log tables: gen_report and access_report</td>
<td>61</td>
</tr>
<tr>
<td>17.</td>
<td>Predefined custom log tables: dba_grade_log, dba_grade_log2, and</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>dba_access_alert</td>
<td></td>
</tr>
</tbody>
</table>
18. Predefined custom log tables: dba_grade_depot, dba_gen_report, and
dba_access_report

19. Predefined custom log tables: dump_all_statements, dump_access_alert, and
error_message
# LIST OF SOURCE CODE FILES

<table>
<thead>
<tr>
<th>Files</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20. Scheduler_api_sys.sql</td>
<td>25</td>
</tr>
<tr>
<td>21. Scheduler_run_sys.sql</td>
<td>30</td>
</tr>
<tr>
<td>22. Scheduler_api_dba.sql</td>
<td>32</td>
</tr>
<tr>
<td>23. Scheduler_run_dba.sql</td>
<td>51</td>
</tr>
<tr>
<td>24. Trigger_chem301.sql</td>
<td>56</td>
</tr>
<tr>
<td>25. Trigger_physics101.sql</td>
<td>57</td>
</tr>
<tr>
<td>26. Access_trigger.sql</td>
<td>58</td>
</tr>
<tr>
<td>27. Update_alert_trigger.sql</td>
<td>59</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

1.1 About this project

This project presents an application that incorporates the built-in Oracle audit package, with DBMS_schedulers, and event triggers to improve the monitoring of a database. It is not another Oracle Audit tutorial. There are already a number of quality tutorials and books about the subject on and off line. Although Oracle provides an excellent built-in database security and auditing library, the actual auditing process can be complicated and tedious. Customization of security reports and alerts is limited as data from Oracle auditing logs are read-only, and built-in security restrictions prohibit the use of triggers on SYS’s objects. The application automatically reorganizes and manipulates these log data, without affecting the data integrity; these data are then used in conjunction with triggers to create custom reports and alerts.

In this report, a hypothetical school database schema, GRADE_DEPOT, contains tables representing classes that are being taught at a school. Chem301 and Physics101 are two such classes with name, Id, and grade as columns. These represent each student’s name, student’s Id, and grade, respectively.

It is the job of the school’s DBA to keep the database and its contents secured and safe. The DBA relies on a series of custom audit reports and security alerts to monitor the
database for suspicious login attempts and unauthorized modifications of each database object and its data. These reports and alerts have to be accurate and generated as soon as the database security policy is violated.

Some custom report logs have been created. Appendix B has the complete list of custom log tables used in this project. The predefined tables, SYS.access_report and dba_access_report, are used to store all login attempts to the database. All DML and DDL statements are reported to SYS.gen_report and dba_gen_report. DBA_grade_depot, dba_access_alert, dba_grade_log, and dba_grade_log2 tables keep logs of any changes made to the grade column of the audit object and failed login attempts. With these custom reports, the DBA is able to create custom alert reports according to the database security policy. In this report, failed login attempt is audited. If failed login attempts for a particular user exceed a threshold, the DBA is alerted. A printed message includes the user name, the terminal that he used, and the number of failed login attempts. The audit policy also includes the monitoring of all logins (successful or otherwise) to the database along with all DML and DDL actions (CREATE TABLE, DROP, INSERT, UPDATE, DELETE, and ALTER TABLE). If an UPDATE statement is executed on the object, old value as well as new value are reported; also recorded are the exact SQL statements used.

1.2 Problem

There are a number of potential problems that have to be considered:
1. Oracle offers a superb auditing library. The auditing features are very rich and powerful. However, they can be complicated and hard to use; DBAs have to set polices and manually perform the audit. Data collected and recorded in Oracle audit logs are extensive but they cannot be used to create custom security alerts.

2. Audit records are stored in SYS.AUD$, SYS.FGA_LOG$, and system logs. To be able to access the records, one has to query 18 predefined views. It is time consuming. Suspicious activities may not be discovered until much later time.

3. Data logs from these views are read-only; therefore, event triggers cannot be used. Oracle and other databases prohibit the use of triggers on any SYS objects (tables):

```sql
create or replace trigger test_trigger
after insert
on failed_login_report
declare
  count1 integer;
  username failed_login_report.username%type;
  terminal failed_login_report.terminal%type;
  ora_time failed_login_report.timestamp%type;
begin
  select count(1) into count1, username into user_name,
  terminal into terminal, ora_time into ora_time
  from failed_login_report
  where returncode = 2
  group by username, terminal;
  if count1 > 4 then
    dbms_output.put_line('Failed login attempts: '+count1);
  end if;
end;
```

Figure 1: ORA-04089: cannot create triggers on object owned by SYS
4. Last but not least, the system resource used by the application has to be considered. Any triggers or custom reports generated from SYS.AUD$ and SYS.FGA_LOG$ will strain the system overall.

1.3 Motivation for Project
My motivation for this project is to improve the database auditing process. By combining the automate workload creation feature of Oracle DBMS_scheduler package and the security audit package, the auditing process can be greatly improved and enhanced. Enhancing the excellent security features of Oracle security audit by automating it may be beneficial to people who maintain databases.

1.4 Objective
One important objective is to be able to efficiently collect and report the audit information from SYS.AUD$ and SYS.FGA_LOG$ on a pre-set time schedule. These audit information are used to create custom security reports and alerts.

Another objective of the project is to be able to overcome Oracle’s restriction on placing triggers on these audit records, as previously mentioned in Section 1.3. These data will have to be manipulated, without affecting the data integrity, before triggers can be used on them. Event trigger is an important component of the application. A trigger will set off to generate custom security alert, every time the security policy is violated.
CHAPTER 2
BACKGROUND OF THE STUDY

2.1.1 Oracle® SQL Database

Oracle database, also known as Oracle DBMS, is an object-relational database management system. It is developed, and marketed by the Oracle Corporation. According to Oracle:

“An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery. Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.”
The database has **logical structures** and **physical structures**. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures [3].”

**2.1.2 Oracle® 11g Express Edition (Oracle Database XE)**

Oracle Database version 11g Express Edition (Oracle Database XE), release 11.2.0.1.0-64 bit, is used for the development of this application. As of this writing, Oracle Database version 12c is available. Version 12c is the latest release from Oracle with many powerful features added. Oracle Database 12c Enterprise Edition “features more than 500 new features including a new architecture that simplifies the process of consolidating databases onto the cloud, enabling customers to manage many databases as one without changing their applications [4].” However, a free Express Edition of 12C is not yet available. Oracle Database 11g XE is “an entry-level edition of Oracle Database that is quick to download, simple to install and manage, and is free to develop, deploy, and distribute. Oracle Database XE makes it easy to upgrade to the other editions of Oracle without costly and complex migrations. Oracle Database XE can be installed on any size machine with any number of CPUs, stores up to 11 GB of user data, using up to 1 GB of memory, and using only one CPU on the host machine [5].” Therefore, Oracle Database 11G-XE is selected as it provides real application testing options, PL/SQL stored procedures, and DML triggers. It also features database audit feature that is the core of this application.
It is important to recognize that Oracle Database XE is a stripped-down version of Oracle Database Enterprise Edition. An application that works on Oracle Database XE, will work on Oracle Database Enterprise Edition, but not vice versa. The Enterprise Edition contains all the components of Oracle Database. It provides “the performance, availability, scalability, and security required for mission-critical applications such as high-volume online transaction processing (OLTP) applications, query-intensive data warehouses, and demanding Internet applications [5].”

Safe and security database practices such as security configuring the database, enabling Oracle data dictionary protection, and enforcing password management are not within the scope of this project. Oracle Database 2 Day + Security Guide is an excellent reference for Oracle Database Security [10]. Oracle Database Security Checklist White Paper “provides guidance on configuring the Oracle Database based on security best practices for operational database deployments [6].”

Oracle Database Express Edition Licensing Information [9] contains a comprehensive list of features that are available with Oracle Database XE.
2.2 Oracle® 3GL (PL/SQL)

PL/SQL is a database programming language that was developed by Oracle. PL/SQL is an imperative 3GL that was “designed specifically for the seamless processing of SQL commands. It provides specific syntax for this purpose and supports exactly the same datatypes as SQL. Server-side PL/SQL is stored and compiled in Oracle Database and runs within the Oracle executable. It automatically inherits the robustness, security, and portability of Oracle Database [7].”

2.3 Oracle Auditing

According to Oracle, “auditing is the monitoring and recording of selected user database actions. Auditing is normally used to:

- Investigate suspicious activity. For example, if an unauthorized user is deleting data from tables, the security administrator might decide to audit all connections to the database and all successful and unsuccessful deletions of rows from all tables in the database.

- Monitor and gather data about specific database activities. For example, the database administrator can gather statistics about which tables are being updated, how many logical I/Os are performed, or how many concurrent users connect at peak times [8].”
Oracle supports three types of auditing: Regular audit, Fine-grained audit, and Common audit (combination of regular and fine-grained audit). Once enabled, Oracle audit can track actions on a specified schema, its objects, and users’ login and logoff information.

Statement audit audits DDL and DML SQL statements, such as SELECT, CREATE, UPDATE, DELETE, and DROP. Privilege audit audits the use of the target privilege.

Audit records can be stored in either a data dictionary table, called the database audit trail, or an operating system audit trail. The database audit trails for fine-grained audit, and regular audit are reported to table FGA_LOG$ and table named SYS.AUD$, respectively. These tables are in the SYS schema of each Oracle database's data dictionary. Oracle provides several pre-defined views to help you access the information in this table. As shown in Figure 2, there are 18 views:

Figure 2: Oracle 11g Audit trail views
CHAPTER 3
THE PROPOSED APPLICATION (THE APP)

3.1 Design

The automation aspect of this app is achieved by utilizing Oracle DBMS_scheduler. This app comprises of two set of schedulers that run independently of each other. Each scheduler manages and runs several jobs and programs according to the security policy. While one scheduler gathers data from Oracle’s pre-defined audit views on a pre-set schedule, the other manipulates these data and generates audit report and security alerts.

Oracle® SQL has an extensive auditing facility that provides system logs for all system events, data manipulation language (DML) and data dictionary language (DDL) actions. Oracle offers three types of audits: standard audit, fine-grained audit, and common audit. Actions that trigger the audit are stored the built-in system logs. To view these data, Oracle provides eighteen (18) different pre-defined audit views with three commonly used views: dba_audit_trail, dba_fga_audit_trail, and dba_common_audit_trail.

Trigger is an important component in designing this system. For example, the grade column of a student table is audit-enabled, any update or change in that column will trigger a notification to the DBA. However, triggers are not allowed on these system log views. To be able to utilize the important log information, custom log tables have to be
created. These log tables store the collected data from AUD$ and FGA_LOG$. They are updated only when .SYS.AUD$ and SYS.FGA_LOG$ change.

Oracle prohibits the creation of triggers on any tables (objects) that are owned by SYS. In addition, it is a good database practice that one performs DBA routines using a separate account with DBA privileges granted by SYS [6]. This app uses a separate schema with full dba’s privileges, DBA_ADMIN, to monitor the database. Since DBA_ADMIN is not SYS, triggers can be placed on DBA_ADMIN’s custom audit reports.

The design has to take into consideration the resources used by the triggers and the copy and update of the audit records, especially for large database systems with potential million rows of records. Traditional row insert, update, cursor, fetch are not be as efficient.

Figure 3 shows a diagram of the system:
3.2 Code

Audit features need to be enabled as Oracle audit is disabled by default. To enable auditing and save audit records to the database audit trail:

SQL> ALTER SYSTEM SET audit_trail=db SCOPE=SPFILE;

The database needs to be re-started for the change to take affect:

SQL> SHUTDOWN immediate

To restart:
Once audit is enabled, audit policies (audited actions) need to be specified. In this case, the audit policy calls for audit CREATE SESSION, which monitors failed/successful login attempts, DML and DDL statements by schema GRADE_DEPOT:

```
CONNECT sys/password AS SYSDBA

SQL> AUDIT ALL BY GRADE_DEPOT BY ACCESS;
SQL> AUDIT CREATE SESSION WHENEVER NOT SUCCESSFUL;
SQL> AUDIT CREATE SESSION WHENEVER SUCCESSFUL;
SQL> AUDIT SELECT TABLE, UPDATE TABLE, INSERT TABLE, DELETE TABLE BY GRADE_DEPOT BY ACCESS;
```

For fine-grained audit, to audit the grade column, the policy can be set as follow:

```
CONN sys/password AS SYSDBA

BEGIN

   DBMS_FGA.add_policy(
      object_schema => 'GRADE_DEPOT',
```


The application consists of two sets of schedulers. The first scheduler (scheduler_api_sys.sql and scheduler_run_sys.sql) operates in the SYS schema. The sole purpose of this scheduler is to automatically collect and update the two master custom log tables, gen_report and access_report. In the DBA_ADMIN schema, the second scheduler (scheduler_api_dba.sql and scheduler_run_dba.sql) generates specific jobs and programs according to the audit policy. Some of these jobs and programs generate and update the predefined custom log tables, while others manipulate the data and print custom security reports. In addition, in the DBA_ADMIN schema, a series of triggers are set on these custom log tables.

It is worth mentioning that these custom logs are similar in design to the 18 pre-defined views in Oracle audit package. Each contains data for specific purpose. DBA_ACCESS_ALERT holds all failed login information; DBA_GEN_REPORT contains all DDL and DML statements performed on the audit schema objects, and so on. Partitioning a large master table into smaller and manageable tables that hold similar type
of data can improve the processing and query time, especially when the database is large and the master audit log tables, AUD$ and FGA_AUD$, are very large with million rows of records.

As stated in Section 3.1, this app was designed with efficiency in mind. Therefore, instead of traditional row by row update to the custom master log tables, the app dynamically updates the audit information by creating new log table (same name), only when there is a change in the SYS.AUD$ or SYS.FGA.AUD$.

Oracle defines a trigger as “a named PL/SQL unit that is stored in the database and executed (fired) in response to a specified event that occurs in the database [11].” Therefore, the manipulation of data is required in order for triggers to be set, or be defined on these tables. To create or mimic the “event” to a specific table that a trigger is to be defined on, an explicit cursor is used to fetch each row into a temporary table, namely dump_all_statements, using INSERT INTO statement. These temporary dump tables are emptied shortly thereafter.

Please refer to Appendix A for the complete PL/SQL source codes of this project.
3.3 Test

The application is tested on Oracle Database version 11g Express Edition. I am using an incremental life cycle, and unit testing is practiced.

Figure 4 shows the hypothetical schema, GRADE_DEPOT. There are two objects, CHEM301 and PHYSICS101, each with NAME, Id, and GRADE as columns. SQL statements to create, insert, select, and update the table are shown:

Figure 4: Schema GRADE_DEPOT (Create table, insert, update)

Figure 5 below shows the table name, action name, and the sql text commands that trigger the audit as queried from Oracle internal dba_audit_trail view. The view correctly shows all audited actions.
Figure 5: Oracle’s dba_audit_trail view showing audited actions

Figure 6 shows the exact information from the custom log table, gen_report.

Figure 6: Custom log table, gen_report, showing the same audited actions

Figure 7 shows UPDATE, DELETE, and DROP TABLE commands on the table physics101.
Figure 7: Table, physics101, with UPDATE, DELETE, and DROP TABLE.

Again, Oracle dba_audit_trail view shows the above audited actions:
Figure 8: Oracle’s dba_audit_trail view with UPDATE, SELECT, DELETE, and DROP

The same audited actions are also in gen_report log:
Figure 9: Custom log table, gen_report, with DML and DDL statements

Figure 10 shows failed login attempts for user name SUPERMAN. The total attempts exceed the threshold of two (2) attempts. SUPERMAN failed to login to the database eight (8) times.

Figure 10: Failed login attempts by user SUPERMAN.
Repeated failed login attempts could be a sign of suspicious activity that deserves the extra attention of the DBA. Shown below is the failed_attempts job with an alert message. The alert message includes the number of failed attempts, username, and the computer terminal used.

```
$SQL>
$SQL> exec failed_attempts;
Failed login attempts:8 by username : SUPERMAN from terminal : LE-PC
Failed login attempts:3 by username : KETAN_PATEL from terminal : LE-PC
PL/SQL procedure successfully completed.
$SQL>
```

Figure 11: Failed_login_attempts alert

A complete list of all failed login attempts can also be printed:

```
$SQL> exec print_failed_log;
*******************************************************************************
<table>
<thead>
<tr>
<th>username</th>
<th>action_name</th>
<th>terminal</th>
<th>timestamp</th>
<th>returncode</th>
</tr>
</thead>
<tbody>
<tr>
<td>KETAN_PATEL</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>KETAN_PATEL</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>SUPERMAN</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>SUPERMAN</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>SUPERMAN</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>SUPERMAN</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>SUPERMAN</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>SUPERMAN</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>SUPERMAN</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>16-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>Bob</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>17-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>Bob</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>17-APR-14</td>
<td>1017</td>
</tr>
<tr>
<td>Bob</td>
<td>LOGON</td>
<td>LE-PC</td>
<td>17-APR-14</td>
<td>1017</td>
</tr>
</tbody>
</table>

PL/SQL procedure successfully completed.
$SQL>
```

Figure 12: Failed_login_attempts report of all failed logins.

In schema GRADE_DEPOT, two UPDATE statements were executed on table PHYSICS101 and CHEM301. The user has changed the grades for a student with ID=9999999999:
Figure 13: UPDATEs to tables PHYSICS101 and CHEM301

The application promptly generates an UPDATE alert:

```sql
SQL> show user;
USER is "GRADE_DEPOT"
SQL> update physics101 set grade=92 where id=999999999;
1 row updated.
SQL> update chem301 set grade=89 where id=999999999;
1 row updated.
```

Figure 14: UPDATE alert showing the two UPDATEs on the tables.

The app also prints out a detailed report showing the value that was changed as highlighted below:

```sql
SQL> exec update_attempts;
USER:GRADE_DEPOT performed : UPDATE on table : PHYSICS101
USER:GRADE_DEPOT performed : UPDATE on table : CHEM301
PL/SQL procedure successfully completed.
SQL>
```

Figure 15: UPDATE report showing the two above UPDATEs

```sql
SQL> exec print_chem301;
*************************GRADE CHANGE REPORT FOR CHEM301*************************
<table>
<thead>
<tr>
<th>STUDENT_ID</th>
<th>OLD_GRADE</th>
<th>NEW_GRADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>59</td>
<td>89</td>
</tr>
<tr>
<td>8999999998</td>
<td>99</td>
<td>69</td>
</tr>
</tbody>
</table>
PL/SQL procedure successfully completed.
SQL>
```

```sql
SQL> exec print_physics101;
*************************GRADE CHANGE REPORT FOR PHYSICS101*************************
<table>
<thead>
<tr>
<th>STUDENT_ID</th>
<th>OLD_GRADE</th>
<th>NEW_GRADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>42</td>
<td>92</td>
</tr>
<tr>
<td>8999999998</td>
<td>99</td>
<td>39</td>
</tr>
<tr>
<td>5559999998</td>
<td>95</td>
<td>56</td>
</tr>
</tbody>
</table>
PL/SQL procedure successfully completed.
SQL>
```
4.1 Results
The test results above show that the application does an effective monitoring job of the database. Working independently of one another, both schedulers in the application work as expected. The scheduler in SYS schema correctly collects the information from the corresponding Oracle audit, AUD$. Figures 5, 6, 8, and Figure 9 show that the correct data are collected from Oracle audit views. DBA_ADMIN schema’s scheduler also executes all the jobs and programs as scheduled. All the custom log tables are updated with their respective date-types. Event triggers to alert the database administrator of suspicious activity are working properly. They fire when conditions in the specific tables met the preset security policy. Figure 11 and Figure 12 show the correct security alert and audit report are generated once the failed login attempts exceed the threshold of two failed attempts. DML statement audit alert and report also work. Shown in Figure 14 and Figure 15 are the report and alert that UPDATE statements were committed on table PHYSICS101 and table CHEM301 of GRADE_DEPOT schema.

4.2 Summary
Applications such as this can improve the task of monitoring database. The application, though runs silently in the background, provides full monitoring of a database. It is important to note that this application can easily be adapted to suite the database
resources. For instance, in a constrained environment, where trigger operations can seriously affect the normal database operating conditions, some of the results showed above can be achieved by different means, such as dynamic querying the underlying log tables. Because of time constraints, only few, but common audit features are implemented in this project. It is relatively easy to customize this app to suit your needs. For instance, to create an alert message for a DDL statement such as CREATE TABLE, a trigger that is similar to update_alert_trigger.sql for dump_all_statements table. A new create_table job and create_table program will execute a stored procedure that is similar to update_attempts. Using this framework, more features can easily be added, and implemented.
-- Source code: scheduler_api_sys.sql

-- Author: Thomas Le

create or replace procedure g_report as

drop_cmnd1 varchar2(150);
c_cmnd1 varchar2(400);
old_count number;
new_count number;

begin

select count(*) into old_count from gen_report;
select count(*) into new_count from dba_audit_trail;
if new_count != old_count then

drop_cmnd1 := 'drop table gen_report';
EXECUTE IMMEDIATE drop_cmnd1;
c_cmnd1 := 'create table gen_report as
select username,owner,obj_name,action_name,sql_text
from dba_audit_trail
order by timestamp';
EXECUTE IMMEDIATE c_cmnd1;
execute immediate 'grant all on gen_report to DBA_ADMIN';

end if;
exception

    when others then

        NULL;

end g_report;
/

show errors

create or replace procedure session_report as

drop_cmnd2 varchar2(150);
old_count number;
new_count number;

begin

    select count(*) into old_count from access_report;
    select count(*) into new_count from dba_audit_session;

    if new_count != old_count then

        drop_cmnd2 := 'drop table access_report';

        EXECUTE IMMEDIATE drop_cmnd2;

        EXECUTE IMMEDIATE 'create table access_report as

            select *

            from dba_audit_session';

        execute immediate 'grant all on access_report to DBA_ADMIN';

    end if;

end;
exception
    when others then
        NULL;
end session_report;
/
show errors
begin
    dbms_scheduler.drop_job(job_name => 'gen_job');
end;
/
show errors
begin
    dbms_scheduler.drop_job(job_name => 'session_job');
end;
/
show errors
begin
    dbms_scheduler.drop_program(program_name => 'gen');
end;
/
show errors
begin
dbms_scheduler.drop_program(program_name => 'session_program');

end;
/

show errors

begin

dbms_scheduler.drop_schedule(schedule_name => 'schedule1');

end;
/

show errors

begin

begin

    dbms_scheduler.create_program(
        program_name => 'gen',
        program_type => 'STORED_PROCEDURE',
        program_action => 'g_report',
        number_of_arguments => 0,
        enabled => TRUE,
        comments => 'stored procedure: gen_report table');

end;
/

show errors

begin

    dbms_scheduler.enable(
begin
  dbms_scheduler.create_program(
    program_name =>'session_program',
    program_type =>'STORED_PROCEDURE',
    program_action =>'session_report',
    number_of_arguments => 0,
    enabled =>'TRUE',
    comments =>'stored procedure: access_report table');
end;
/

show errors
begin
  dbms_scheduler.enable(
    name =>'session_program');
end;
/

show errors
-- Source code: scheduler_run_sys.sql

-- Author: Thomas Le

define

set echo on
set serveroutput on

begin
    dbms_scheduler.create_schedule(
        schedule_name => 'schedule1',
        start_date => systimestamp,
        repeat_interval => 'freq=minutely; interval=1',
        end_date => null,
        comments => 'test_run, every min.');
end;
/

show errors

begin
    dbms_scheduler.create_job(
        job_name => 'gen_job',
        program_name => 'gen',
        schedule_name => 'schedule1',
        comments => 'every min.');
end;
/
begin

    dbms_scheduler.create_job(
        job_name => 'session_job',
        program_name => 'session_program',
        schedule_name => 'schedule1',
        enabled => FALSE,
        comments => 'audit failed login attempts');

    dbms_scheduler.enable('session_job');

end;
/

show errors
create or replace procedure duplicate_login as
    cmnd  varchar2(150);
    old_count  number;
    new_count  number;
begin
    select count(*) into old_count from DBA_ADMIN.dba_access_report;
    select count(*) into new_count from SYS.access_report;
    if new_count != old_count then
        cmnd := 'drop table DBA_ADMIN.dba_access_report';
        execute immediate cmnd;
        execute immediate 'create table DBA_ADMIN.dba_access_report as
            select * from SYS.access_report';
        -- update failed Attempts_log
        execute immediate 'drop table dba_access_alert';
        execute immediate 'create table dba_access_alert as
            select count(*) as attempts,username,terminal,returncode
            from dba_access_report
            where returncode != 0
            group by username,terminal,';returncode';
end if;

exception

when others then
   insert into error_message values('duplicate_login FAILED !');

end duplicate_login;

/

show errors

create or replace procedure duplicate_gen as
   cmnd2 varchar2(150);
   old_count number;
   new_count number;

begin
   select count(*) into old_count from DBA_ADMIN.dba_gen_report;
   select count(*) into new_count from SYS.gen_report;
   if new_count != old_count then
      cmnd2 := 'drop table DBA_ADMIN.dba_gen_report';
      execute immediate cmnd2;
      execute immediate 'create table DBA_ADMIN.dba_gen_report as select * from SYS.gen_report';
   end if;
end;
-- monitor chema : GRADE_DEPOT

execute immediate ' drop table dba_grade_depot ';' 

execute immediate 'create table dba_grade_depot as

    select username,obj_name,action_name,sql_text
    from dba_gen_report
    where owner="GRADE_DEPOT"';

end if;

exception

when others then

    insert into error_message values('duplicate_gen FAILED !');

end duplicate_gen;

/

show errors

create or replace procedure failed_attempts as

cursor a1_cur is

    select * from dba_access_alert;

    a1 a1_cur%ROWTYPE;
begin
    open a1_cur;
    loop
        FETCH a1_cur INTO a1;
        EXIT WHEN a1_cur%NOTFOUND;
        insert into dump_access_alert
        values(a1.attempts,a1.username,a1.terminal,a1.returncode);
        truncate table dump_access_alert;
    end loop;
    close a1_cur;
end failed_attempts;
/
show errors
create or replace procedure update_attempts as
cursor update_cur is
    select * from dba_grade_depot;
u  update_cur%ROWTYPE;

begin

open update_cur;

loop

    FETCH update_cur INTO u;

    EXIT WHEN update_cur%NOTFOUND;

    insert into dump_all_statements
        values(u.username,u.obj_name,u.action_name,u.sql_text);

    truncate table dump_all_statements;

end loop;

close update_cur;

end update_attempts;

/

show errors

create or replace procedure print_chem301 as

cursor c3_cur is
select * from dba_grade_log2;

c3 c3_cur%ROWTYPE;
begin

dbms_output.put_line('************GRADE CHANGE REPORT FOR CHEM301*************');
dbms_output.put_line('-----------------------------------------------');
dbms_output.put_line('STUDENT ID          OLD_GRADE  NEW_GRADE  ');
dbms_output.put_line('---------------------------------------------------');

open c3_cur;
loop
  FETCH c3_cur INTO c3;
  EXIT WHEN c3_cur%NOTFOUND;
  dbms_output.put_line(c3.id||'         '||c3.old_grade||'           '||c3.new_grade);
end loop;
close c3_cur;
end print_chem301;
show errors

create or replace procedure print_physics101 as

    cursor c4_cur is
        select * from dba_grade_log;

    c4    c4_cur%ROWTYPE;

begin
    dbms_output.put_line('**************GRADE CHANGE REPORT FOR PHYSICS101**************

STUDENT ID              OLD_GRADE  NEW_GRADE  
---------------------------------------------------------------

     OPEN c4_cur;

     LOOP
         FETCH c4_cur INTO c4;
     END LOOP;
EXIT WHEN c4_cur%NOTFOUND;

dbms_output.put_line(c4.id||'         '||c4.old_grade||'           '||c4.new_grade);

end loop;
close c4_cur;

end print_physics101;
/

show errors

create or replace procedure print_failed_log as

cursor c1_cur is

    select * from dba_access_report
    where returncode>0 ;

    c1 c1_cur%ROWTYPE;

begin

    dbms_output.put_line('**************************FAILED LOGIN ATTEMPTS**************************');

    dbms_output.put_line('------------------------------------------------------------------');

');
dbms_output.put_line('username    action_name    terminal   timestamp
returncode');

dbms_output.put_line('--------------------------------------------------------------');

');

open c1_cur;

loop

    FETCH c1_cur INTO c1;

    EXIT WHEN c1_cur%NOTFOUND;

    dbms_output.put_line(c1.username||'     '||c1.action_name||'

    '||c1.terminal||'     '||c1.timestamp||'      '||c1.returncode);

end loop;

close c1_cur;

end print_failed_log;

/

show errors

begin

    dbms_scheduler.drop_job(job_name => 'access_log_job');

end;
end;
/

show errors

begin
    dbms_scheduler.drop_job(job_name => 'gen_log_job');

end;
/

show errors

begin
    dbms_scheduler.drop_job(job_name => 'failed_log_print_job');

end;
/

show errors

begin
    dbms_scheduler.drop_job(job_name => 'failed_attempts_job');

end;
/

show errors

begin
    dbms_scheduler.drop_job(job_name => 'print_grade_job');
end;
/

show errors

begin
    dbms_scheduler.drop_job(job_name => 'print_grade_job2');
end;
/

show errors

begin
    dbms_scheduler.drop_job(job_name => 'update_alert_job');
end;
/
show errors

begin
    dbms_scheduler.drop_program(program_name => 'access_log_program');
end;
/
show errors

begin
    dbms_scheduler.drop_program(program_name => 'gen_log_program');
end;
/
show errors

begin
    dbms_scheduler.drop_program(program_name => 'failed_attempts_program');
end;
/

show errors

begin
    dbms_scheduler.drop_program(program_name => 'failed_log_print_program');
end;
/

show errors

begin
    dbms_scheduler.drop_program(program_name => 'print_grade_program');
end;
/

show errors

begin
    dbms_scheduler.drop_program(program_name => 'print_grade_program2');
end;
/
show errors

begin
    dbms_scheduler.drop_program(program_name => 'update_alert_program');
end;
/
show errors

begin
    dbms_scheduler.drop_schedule(schedule_name => 'schedule2');
end;
/
show errors

begin
    dbms_scheduler.create_program(
        program_name => 'access_log_program',
        program_type => 'STORED_PROCEDURE',
        program_action => 'duplicate_login',
    )
end;
number_of_arguments => 0,
enabled                => TRUE,
comments               => 'making copy of sys.access_report');
end;

/show errors

begin
  dbms_scheduler.enable(
    name                   => 'access_log_program');
end;

/show

begin
  dbms_scheduler.create_program(
    program_name           => 'gen_log_program',
    program_type           => 'STORED_PROCEDURE',
    program_action         => 'duplicate_gen',
    number_of_arguments    => 0,
    enabled                => TRUE,
    comments               => 'making copy of sys.gen_report');
end;
/*

show errors
begin
    dbms_scheduler.enable(
        name => 'gen_log_program');
end;
/

show errors
begin
    dbms_scheduler.create_program(
        program_name => 'failed_attempts_program',
        program_type => 'STORED_PROCEDURE',
        program_action => 'failed_attempts',
        number_of_arguments => 0,
        enabled => TRUE,
        comments => 'creating failed_attempts log');
end;
/

show errors
begin
dbms_scheduler.enable(
    name => 'failed_attempts_program');
end;
/
show errors

begin
    dbms_scheduler.create_program(
        program_name => 'failed_log_print_program',
        program_type => 'STORED_PROCEDURE',
        program_action => 'print_failed_log',
        number_of_arguments => 0,
        enabled => TRUE,
        comments => 'print failed_log');
end;
/
show errors

begin
    dbms_scheduler.enable(
        name => 'failed_log_print_program');
end;
/
show errors

begin

dbms_scheduler.create_program(
  program_name => 'print_grade_program',
  program_type => 'STORED_PROCEDURE',
  program_action => 'print_chem301',
  number_of_arguments => 0,
  enabled => TRUE,
  comments => 'print grade change for chem301');
end;
/

show errors

begin

dbms_scheduler.enable(
  name => 'print_grade_program');
end;
/

show errors

begin
dbms_scheduler.create_program(
    program_name => 'print_grade_program2',
    program_type => 'STORED_PROCEDURE',
    program_action => 'print_physics101',
    number_of_arguments => 0,
    enabled => TRUE,
    comments => 'print grade change for physisc101');
end;
/
show errors
begin
    dbms_scheduler.enable(
        name => 'print_grade_program2');
end;
/
show errors

begin
    dbms_scheduler.create_program(
        program_name => 'update_alert_program',
        program_type => 'STORED_PROCEDURE',
        program_action => 'update_alert',
        number_of_arguments => 0,
        enabled => TRUE,
        comments => 'update alert change for physisc101');
end;
number_of_arguments => 0,
enabled => TRUE,
comments => 'monitor update statements ');

end;
/

show errors

begin

    dbms_scheduler.enable(
        name => 'update_alert_program');

end;
/

show errors

-- Source code: scheduler_run_dba.sql

-- Author: Thomas Le

set echo on
set serveroutput on

begin

    dbms_scheduler.create_schedule(
        schedule_name => 'schedule2',
        start_date => systimestamp,
        repeat_interval => 'freq=minutely; interval=1',
        end_date => null,
        => null,
begin
    dbms_scheduler.create_job(
        job_name => 'access_log_job',
        program_name => 'access_log_program',
        schedule_name => 'schedule2',
        enabled => FALSE,
        comments => 'making copy of access_report');
    dbms_scheduler.enable('access_log_job');
end;
/
show errors

begin
    dbms_scheduler.create_job(
        job_name => 'gen_log_job',
        program_name => 'gen_log_program',
        schedule_name => 'schedule2',
        enabled => FALSE,
        comments => 'making copy of access_report');
end;
/
show errors
begin
  dbms_scheduler.create_job(
    job_name => 'failed_attempts_job',
    program_name => 'failed_attempts_program',
    schedule_name => 'schedule2',
    enabled => FALSE,
    comments => 'create failed_attempts log');
  dbms_scheduler.enable('failed_attempts_job');
end;
/
show errors

begin
  dbms_scheduler.create_job(
    job_name => 'failed_log_print_job',
    enabled => FALSE,
    comments => 'make copy of gen_report');
  dbms_scheduler.enable('gen_log_job');
end;
/
show errors
program_name => 'failed_log_print_program',
schedule_name => 'schedule2',
enabled => FALSE,
comments => 'making copy of failed_log');

dbms_scheduler.enable('failed_log_print_job');
end;
/
show errors

begin

dbms_scheduler.create_job(
    job_name => 'print_grade_job',
    program_name => 'print_grade_program',
    schedule_name => 'schedule2',
    enabled => FALSE,
    comments => 'print grade change for chem301');

dbms_scheduler.enable('print_grade_job');
end;
/
show errors

begin
dbms_scheduler.create_job(
    job_name => 'print_grade_job2',
    program_name => 'print_grade_program2',
    schedule_name => 'schedule2',
    enabled => FALSE,
    comments => 'print grade change for physics101');

dbms_scheduler.enable('print_grade_job2');

end;
/

show errors

begin

dbms_scheduler.create_job(
    job_name => 'update_alert_job',
    program_name => 'update_alert_program',
    schedule_name => 'schedule2',
    enabled => FALSE,
    comments => 'audit update statements');

dbms_scheduler.enable('update_alert_job');

end;
/

show errors

-- Source code: Trigger_chem301.sql

-- Author: Thomas Le

set echo on

set serveroutput on

create or replace trigger grade_trigger3

after insert or update

on grade_depot.chem301

for each row

declare

s_name    grade_depot.chem301.name%type;

s_id      grade_depot.chem301.id%type;

o_grade   grade_depot.chem301.grade%type;

n_grade   grade_depot.chem301.grade%type;

begin

if :new.grade != :old.grade then

    s_id := :old.id;

    o_grade := :old.grade;

    n_grade := :new.grade;

    insert into dba_grade_log2 values(s_id,o_grade,n_grade);

dbms_output.put_line('dba-new grade changed: '||n_grade || ', old grade: '||o_grade || ', student Id :'||s_id);
end if;
end;
/

show errors

-- Source code: Trigger_physics101.sql

-- Author: Thomas Le

set echo on
set serveroutput on
create or replace trigger grade_trigger1
after update or insert
on grade_depot.physics101 referencing new as new old as old
for each row
declare
id_no grade_depot.physics101.id%type;
old_g grade_depot.physics101.grade%type;
new_g grade_depot.physics101.grade%type;
begin
if :new.grade != :old.grade then
id_no := :old.id;
old_g := :old.grade;
new_g := :new.grade;
insert into table1 values('physics101');
insert into dba_grade_log values(id_no,old_g,new_g);
dbms_output.put_line('sys-new grade changed: '||new_g || ', old grade: '|| old_g || ',student Id :'||id_no);
end if;
end;
/
show errors

-- Source code: access_trigger.sql

-- Author: Thomas Le

set echo on
set serveroutput on
create or replace trigger a_trigger
before insert or update
on dump_access_alert referencing new as new old as old
for each row
declare
begin
    if :new.attempts > 2 then
        dbms_output.put_line('Failed login attempts: ' || :new.attempts || ' by username: ' || :new.username || ' From terminal: ' || :new.terminal);
    end if;
end;
/
show errors

-- Source code: update_alert_trigger.sql

-- Author: Thomas Le

set echo on
set serveroutput on
create or replace trigger up_trigger
before insert or update
on dump_all_statements referencing new as new old as old
for each row
declare

begin
if :new.action_name = 'UPDATE' then
    dbms_output.put_line('USER:'||:new.username ||' performed : '||
    :new.action_name ||' on table : '||:new.obj_name);
end if;
end;
/
show errors
APPENDIX B

PRE-DEFINED CUSTOM LOG TABLES

The followings are predefined custom log tables in SYS schema:

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL> desc gen_report;
Name               Null?  Type
----------------------------------------        -------------
USERNAME           VARCHAR2(30)
OWNER              VARCHAR2(30)
OBJ_NAME           VARCHAR2(28)
ACTION_NAME        VARCHAR2(28)
SQL_TEXT           VARCHAR2(2000)

SQL> desc access_report;
Name               Null?  Type
----------------------------------------        -------------
OS_USERNAME        VARCHAR2(255)
USERNAME           VARCHAR2(30)
USERHOST           VARCHAR2(30)
TERMINAL           VARCHAR2(255)
TIMESTAMP          DATE
ACTION_NAME        VARCHAR2(28)
LOGOFF_TIME        DATE
LOGOFF_LREAD       NUMBER
LOGOFF_LREDA       NUMBER
LOGOFF_LWRITE      NUMBER
LOGOFF_LWRITEA     VARCHAR2(48)
SESSIONID          NOT NULL NUMBER
SESSIONCODE        NOT NULL NUMBER
CLIENT_ID          NOT NULL VARCHAR2(64)
SESSION_CPU        NUMBER
EXTENDED_TIMESTAMP NUMBER
REMOTE_SESSIONID   NUMBER
LOGON_HOST         VARCHAR2(32)
GLOBAL_UID         NUMBER
INSTANCE_NUMBER    NUMBER
OS_PROCESS         VARCHAR2(16)
```

Figure 16: Predefined custom log tables: gen_report and access_report.
There are nine predefined custom tables in the DBA_ADMIN:

**Figure 17:** Predefined custom log tables: dba_grade_log, dba_grade_log2, and dba_access_alert.
Figure 18: Predefined custom log tables: `dba_grade_depot`, `dba_gen_report`, and `dba_access_report`. 
Figure 19: Predefined custom log tables: dump_all_statements, dump_access_alert, and error_message.
REFERENCES


http://docs.oracle.com/cd/E17781_01/license.112/e18068.pdf

http://docs.oracle.com/cd/B28359_01/server.111/b28337.pdf

http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#LNPLS020