Macrotone
Consulting Ltd.

Auditing Package for Oracle Databases

Author:	**G S Chapman**
Date:	**19th September 2007**
Version:	**1.6**
Location of Document:	

DOCUMENT HISTORY

Version	Date	Changed By:	Remarks
1.1	11/12/03	G S Chapman	Initial version.
1.2	02/02/04	G S Chapman	Updated with FGA information.
1.3	12/02/04	G S Chapman	Updated and general distribution.
1.4	06/04/04	G S Chapman	FGA explained further and some 10g features introduced.
1.5	01/06/04	G S Chapman	Minor size changes.
1.6	19/09/07	G S Chapman	Some additional information re 10G and security.

DOCUMENT DISTRIBUTION

Copy No	Name	Role	Organisation

DOCUMENT REFERENCES

Document Name	Originator	Part Number	Version	Date
Oracle9i Security Overview Release 1	Oracle Corporation	A90148-01	9.0.1	
Oracle9i Application Developer's Guide - Fundamentals Release 2	Oracle Corporation	A96590-01	9.2	
Oracle9i Security Overview Release 2	Oracle Corporation	A96582-01	9.2	

TABLE OF CONTENTS

TABLES

Appendices

PURPOSE OF DOCUMENT

This document provides the details of the generic Audit package developed and used for auditing particular schema table objects within an Oracle database. It provides a description of the methodology used and the procedures, both externally callable, but also the internal procedures within the package.

The package is supplied in an 'unwrapped' format for general implementation on Oracle databases up to and including version 10.2.

Please ensure that the version of the document matches the version of the package as installed on the Oracle database in use. Changes can and do occur between different releases although every effort is made to ensure backward compatibility with earlier releases.

1 Introduction

Traditional Oracle Database auditing options allow the tracking of the actions users perform on objects at the macro level—for example, an audit for SELECT statements on a table, enable the tracking of who selected data from the table. However, if does not provide the knowledge of the values selected.

With data-manipulating statements— such as INSERT, UPDATE, or DELETE—it is possible to capture any changes by using triggers. This type of auditing will be referred to as 'trigger auditing' in this document and this forms the basis of the AUDIT_PKG. It is available in every version of Oracle the user is likely to encounter.

In Oracle 8i and above it is worth noting that it is also possible to use the Oracle LogMiner utility to analyse the archived logs for changed records, but this is an after the fact analysis and would require the analyst to look for specific information within the logs. This particular feature will not be expanded upon further in this document and the interested user is referred to the relevant Oracle supplied documentation.

Simple SELECT statements are non-data-manipulating, so they neither fire a trigger nor go into archived logs that you can mine later, so these two techniques fall short where SELECT statements are concerned.

Oracle9i Database introduced a new feature called fine-grained auditing (FGA), which changed this. The feature allows the auditing of individual SELECT statements along with the exact statements issued by users. In addition to simply tracking statements, FGA provides a way to simulate a trigger for SELECT statements by executing a code whenever a user selects a particular set of data.

Oracle 10g has extended the functionality of FGA to also capture INSERT, UPDATE and DELETE, negating the need to use table triggers to capture the information. Triggers incur a PL/SQL process call for every row processed, and create an audit record only when a DML statement changes a relevant column.

An FGA policy, on the other hand, does not incur this cost for every row. Instead, it audits only once for every policy. Specifically, it audits when a specified relevant column occurs in a specified type of DML statement, either being changed by the statement or being in its selection criteria. This combination of criteria uncovers users who hope their information gathering will be masked because they only use the selection criteria of a DML statement. Triggers also cannot monitor the activity of another "instead-of" trigger on the same object, while fine-grained auditing supports tables and views.

The AUDIT_PKG described in this documentation incorporates both triggers and FGA for databases up to version 10.2.

NOTE: This will not work on Oracle 11 without some code changes. At the current time the specifics for Oracle 11 are not fully known. If required the changes will be made.

1.1 Trigger Auditing

Using default auditing doesn't provide the level of detail required for insert, delete or update. Audited information doesn't include for example the exact SQL statement or in case of modifications, the old and new values.

In all versions of Oracle it is possible to use triggers to capture changes but in versions of Oracle prior to 8.1 any attempt to make a change that is not committed is not registered. As of Oracle 8.1 it is possible to enhance the triggering to use autonomous transactions to simulate auditing (also all attempts are registered). The implementation that the AUDIT_PKG, makes use of the SYS_CONTEXT to get values from the environment (namespace) as osuser, ip_address and so on. This means that when a front-end application makes a change provided the application context has the details of the 'user' actually using the application, this will also be captured and saved in the audit table.

Trigger auditing as established by the AUDIT_PKG will create shadow tables with the same name as the original tables in a separate 'audit table owner' schema. The table structure will be similar to the main originating table but will have twice as many fields. These fields enable a before and after change to the table columns to be captured along with details of when, and who, made the change. For the purposes of this document a change can be defined as a delete, insert or an update on any of the rows within the original table.

When auditing is enabled for the first time, a shadow table is automatically created in the audit table owner schema.

- Nulls are always permitted in the shadow table.
- All columns in the shadow table have the same data type and sizes as their counterparts in the audited table.
- Name of the shadow table is the same name as the original table name.
- All audited columns are stored when a row is deleted.
- All audited columns are stored when data is inserted in a NULL row in the shadow table.

All Audit Trigger shadow tables contain:

- N_*column_name* where *column_name* is the name within the original source table.
- O_*column_name* where *column_name* is the name within the original source table*.*
- CHANGED_BY - name of the Oracle ID that made the change
- TIMESTAMP - the date/time when the change occurred
- CHANGE_TYPE
  - I - Insert
  - U - Update
  - D - Delete

1.2 Fine Grain Auditing

1.2.1 Audit Columns and Audit Conditions

When establishing FGA auditing the initial set-up is to look at any SELECT statement used on the table be audited. In real life, however, this is probably not necessary, and it may overwhelm the audit table that stores the trail. For example a bank may need to audit when a user selects a balance column, which contains sensitive information, but may not need to audit when a user selects the account number for a particular customer. A column named 'BALANCE', whose selection could triggers an audit, would be known as the audit column. This audit column should be added to the FGA policy.

If audit trails are recorded each time a user selects from a table, the trails will grow in size, causing space and administration problems, so you may want to conduct an audit only if certain conditions are met, not every time. Using an example, perhaps the bank require an audit only if users access the accounts of extremely wealthy account holders, for instance, only if a user selects an account with a balance of $11,000 or more. This type of condition is known as an audit condition and is passed as a parameter to the FGA package.

1.2.2 Optimizer Mode

FGA requires cost-based optimization (CBO) in order to work correctly. Under rule-based optimization, audit trails are always generated whenever a user selects from a table, regardless of whether the relevant columns are selected or not, increasing the chance of false-positive entries. For FGA to work properly, in addition to CBO being enabled at the instance level, there should be no RULE hint in the SQL statements and all the tables in the query must be analyzed at least with the estimate option.

1.2.3 FGA Data-Dictionary Views

The definition of an FGA policy resides in the data-dictionary view DBA_AUDIT_POLICIES. Table 1 - DBA_AUDIT_POLICIES columns includes a brief description of some of the important columns of this view.

Table 1 - DBA_AUDIT_POLICIES columns

OBJECT_SCHEMA	The owner of the table or view on which the FGA policy is defined
OBJECT_NAME	Name of the table or view
POLICY_NAME	Name of the policy-for example, ACCOUNTS_ACCESS
POLICY_TEXT	The audit condition specified while adding the policy-for example, BALANCE >= 11000
POLICY_COLUMN	The audit column-for example, BALANCE
ENABLED	YES if enabled; NO otherwise
PF_SCHEMA	The schema that owns the policy's handler module, if one exists
PF_PACKAGE	The package name of the handler module, if one exists
PF_FUNCTION	The procedure name of the handler module, if one exists

The audit trails are collected in the SYS-owned table FGA_LOG$. As with any SYS-owned raw table, some views on this table present the information in a user-friendly manner.

DBA_FGA_AUDIT_TRAIL is a view on the table. Table 2 - Important columns in the DBA_FGA_AUDIT_TRAIL view contains a brief description of the important columns of this view.

Table 2 - Important columns in the DBA_FGA_AUDIT_TRAIL view

SESSION_ID	The Audit Session Identifier; not the same as the Session Identifier in the V$SESSION view
TIMESTAMP	The time stamp for when the audit record was generated
DB_USER	The database user who issued the query
OS_USER	The operating system user
USERHOST	The host name of the machine from which the user connected
CLIENT_ID	The client identifier, if set by a call to the packaged procedure `dbms_session.set_identifier`
EXT_NAME	The name of an externally authenticated client, such as an LDAP user
OBJECT_SCHEMA	The owner of the table on which access triggered the audit
OBJECT_NAME	The name of the table on which a `SELECT` operation triggered the audit
POLICY_NAME	The name of the policy that triggered the audit (If a table has multiple policies defined on it, each one will insert a record. In that case, this column shows which rows were inserted by which policy.)
SCN	The Oracle System Change Number at which the audit was recorded
SQL_TEXT	The SQL statement issued by the user
SQL_BIND	Bind variables used by the SQL statement, if any

One important column is SQL_BIND, which specifies the values of the bind variables used in a query, knowledge that greatly enhances the tool's power.

Another important column is SCN, which records the System Change Number when a particular query occurs. This information is useful for identifying what a user saw at a specific time, not what the value is now, using Flashback Queries which can show the data at a specified SCN value.

1.2.4 Views and FGA

FGA can also display the information upon database views. Assume a view VW_ACCOUNTS is defined on the ACCOUNTS table as follows:

create view vw_accounts as select * from accounts;

Now, if a user selects from the view instead of from the table:

```
select * from vw_accounts;
```

the following will be seen in the audit trail:

```
select object_name, sql_text from dba_fga_audit_trail;

OBJECT_NAME SQL_TEXT

----------- -----------------------------------------------
```

ACCOUNTS   select * from vw_accounts

Note that the name of the base table, not the view, appears in the OBJECT_NAME column, because the selection from the view selects from the base table. However, the SQL_TEXT column records the actual statement the user issued, and that is exactly what is required.

If there is a need to audit queries only on views, not on tables, it is possible to set the policy up on a view itself. This is done by passing the name of the view instead of that of the table to the parameter object_name in the packaged procedure dbms_fga.add_policy. The OBJECT_NAME column in DBA_FGA_AUDIT_TRAIL will then display the name of the view, and there will be no additional record for a table access.

It is also possible (optional) to define a handler_module defining additional actions; the audit function can potentially capture all user environment and application context values. Therefore, policy administration should be executable by privileged users only.

Note: This handler_module feature is not currently implemented in the AUDIT_PKG.

1.2.5 The Handler Module

The power of FGA doesn't just record events in audit trails; FGA can also optionally execute procedures. A procedure could perform an action such as sending an e-mail alert to an auditor when a user selects a certain row from a table, or it could write to a different audit trail. This stored code segment, which could be a stand-alone procedure or a procedure within a package, is known as the handler module for a policy. It does not have to be in the same schema as the base table itself; in fact, for security reasons, you may want to deliberately place it in a separate schema. Because the procedure executes whenever a SELECT occurs, much like a trigger firing on a DML statement, you can also think of it as a SELECT statement trigger. The following parameters specify a handler module is specified for the policy:

- handler_schema The schema that owns the data procedure
- handler_module The procedure name

The handler module can also take a package name instead of a procedure name. In that situation, the parameter handler_module is specified in package.procedure format.

This particular feature has not been implemented in the AUDIT_PKG code since there has been no specific request. This may be changed in the future should need arise.

1.2.6 Auditing connections through a Third party application.

This scenario is one that is increasing familiar with the growth of three tier applications with the user making a database connection through the application itself. The following extract is taken from Oracle documentation and explains the concept of the application context.

For such applications in which a single user (for example, One Big Application User) connects to the database on behalf of all users, per-user fine-grained access control is still possible. An application developer can create a context attribute to represent the application user (for example, realuser). While all database sessions (and thus all audit records) are initiated as One Big Application User, each session can nonetheless have attributes that vary, depending on who the real user is. This model works best for applications with a limited number of users where there is no requirement for session reuse. Of course, each session, from the database standpoint, is created as the same database user, so that the ability to use roles, database auditing, and others is greatly diminished for reasons previously enumerated.

Web-based applications typically have hundreds if not thousands of users, and the web is stateless. There may be a persistent connection to the database (to support data retrieval for a number of user requests), but these connections are not specific to each web-based user. Web-based applications typically set up and reuse connections instead of having different sessions for each user, to provide scalability. For example, web user Janet and John connect to a middle tier application, which establishes a session in the database used by the application on behalf of both users. Typically, neither Janet nor John are known to the database. The application is responsible for switching the username on the connection, so that, at any given time, it's either Janet or John using the session.

Oracle9*i* Virtual Private Database (VPD) capabilities facilitate connection pooling by allowing multiple connections to access one or more global application contexts, instead of setting up an application context for each distinct user session.

Applications use a CLIENT_IDENTIFIER (which could be an individual application username, or a group) to reference the global application context. Global application contexts provide additional flexibility for web-based applications to use Virtual Private Database, as well as enhanced performance through reuse of common application contexts among multiple sessions instead of setting up per-session application contexts. The CLIENT_IDENTIFIER is also viewable in the user session and accessible in the USERENV naming context. The use of a CLIENT_IDENTIFIER thus functions as an application user proxy, since the CLIENT_IDENTIFIER can be used to capture the 'application username.' The ability to pass a CLIENT_IDENTIFIER to the database for use with global application context is supported in OCI, thick JDBC, and thin JDBC. For OCI-based connections, a change in CLIENT_IDENTIFIER is automatically piggybacked on the next OCI call, for additional performance benefits.

Application user proxy authentication can be used with global application context for additional flexibility and high performance in building applications. For example, suppose a web-based application that provides information to business partners has three types of users: gold partner, silver partner, and bronze partner, representing different levels of information available. Instead of each user having his own session, with individual application contexts set up, the application could set up global application contexts for gold partner, silver partner, or bronze partner and use the client identifier to point the session at the correct context, in order to retrieve the appropriate type of data. The application need only initialise the three global contexts once, and use the client identifier to access the correct application context to limit data access. This provides performance improvements through session reuse, and through accessing global application contexts set up once, instead of having to initialise application contexts for each session individually.

1.2.7 FGA Audit Records

Fine-grained audit trails record the actual statement the user entered along with the values of bind variables, if any. A typical statement that could be captured in the trail might be:

```
select balance from accounts

where acct_no = 10034;
```

If this transaction occurred at 10:00AM and it is 11:00AM, the account balance may have been updated. If the auditor (or the DBA) decides to see exactly what the user really saw at that time, the present value of the column may not reveal accurate information. In industrial espionage cases, or to establish a motive or pattern, the exact data the user viewed is necessary. Even though FGA does not capture the exact data at that point, it is possible to see another piece of information captured in the trail.

Earlier the structure of the view DBA_FGA_AUDIT_TRAIL was shown, which records several key pieces of information related to the session and the user. The relevant column here is the SCN, which records the System Change Number at the time the trail was generated. Using flashback query, it is possible to reconstruct the data at that point in time. Assuming the SCN value was 123456 on the audit trails, issuing the following query:

```
select balance from accounts as of SCN 123456

where acct_no = 10034;
```

The clause as of SCN 123456 will return the balance from the table at that point in time, not now, precisely what is required.

Because flashback is limited to the undo retention period, transactions that occurred beyond that period will be lost. However, auditors may be looking for a trail that begins after the event occurred, so it might be prudent to perform a periodic audit collection, using flashback only to capture important columns.

1.2.8 All Types of DML

In Oracle Database 10g, FGA has been enhanced to audit all types of DML statements, not just SELECT. Under Oracle 9i Database, FGA policies could only audit SELECT statements. In Oracle Database 10g, however, this has been extended to include INSERT, UPDATE, and DELETE as well. This is achieved by specifying a parameter:

```
statement_types => 'INSERT, UPDATE, DELETE, SELECT'
```

This parameter will enable auditing on all the included statement types. One option might be to consider creating separate policies for each statement type, which would allow the ability to enable or disable policies at will—specifically, to control the generation of audit trails to manage the space occupied by them. By default, the statement_type parameter audits only SELECT statements.

Note: The AUDIT_PKG specifically includes checks for database version and will reject and use of Oracle 10g FGA features upon an Oracle 9i database.

1.2.9 Comparison with Trigger Approach

Prior to Oracle Database 10g, DML statement auditing was done via the use of a table trigger. The trigger captures the old and new values and populates the 'AUDIT' table. As implemented in the AUDIT_PKG this has been made an autonomous transaction. The biggest problem is that the trigger is fired for each row, not once per statement. For instance, a statement such as the following:

```
update accounts set balance = 1200 where balance >= 3000;
```

fires for all 10,000 records, inserting 10,000 rows into the audit table. This can seriously undermine the performance of the update statement and may even cause it to fail due to space problems in the audit table. Using a statement trigger doesn't help either, because it can't capture any new or old values of individual records. In contrast, in the FGA approach, only one record is created and the insert executes only once per statement, not once per row—affecting performance negligibly, if at all.

In FGA, it is possible to specify the relevant columns to limit generation of audit trails only when those columns are accessed. In triggers the functionality exists through the use of the WHERE of the trigger definition. However, therein lies a very important difference — in triggers the columns are checked only when they are changed, not merely accessed. In FGA, the auditing kicks in whenever the columns are accessed, whether they're changed or not. This characteristic makes FGA more versatile than triggers.

Another advantage is the applicability of the FGA facility. Sometimes, the INSTEAD OF triggers defined on a view update it on the base table; another INSTEAD OF trigger can't capture the changes made by the other trigger and hence they cannot be recorded. FGA, however, is established on views or tables and captures the changes regardless of where they come from — user statements or triggers.

1.2.10 FGA Behaviour during a Change

Data changes all the time, so it could potentially become applicable in the audit condition when it wasn't so earlier (or vice versa). This issue gives rise to some interesting questions about FGA behaviour in different scenarios. Using an example banking type scenario, where the FGA policy has been defined on UPDATE with the condition is BALANCE >= 3000 and audit column is BALANCE.

Scenario 1

Before: BALANCE = 1000

User issues:

```
update accounts set balance = 1200 where ACCOUNT_NO = ....
```

The old and new balances are less than 3,000, and the audit condition is not satisfied; hence this statement will not be audited.

Scenario 2

Before: BALANCE = 1000

User issues:

```
update accounts set balance = 3200 where ACCOUNT_NO = ....
```

The new balance is more than 3,000, and the audit condition is satisfied; hence this statement will be audited.

Scenario 3

Before: BALANCE = 3200

User issues:

```
update accounts set balance = 1200 where ACCOUNT_NO =# …
```

The new balance is less than 3,000 but the old balance was more. Hence the audit condition is satisfied and this statement will be audited.

Scenario 4

User inserts a row with BALANCE < 3000.

```
insert into accounts values (9999,1200,'X');
```

Because the balance 1,200 does not satisfy the audit condition, the statement is not audited. If the value of the balance column were greater than or equal to 3,000, it would have been audited.

Scenario 5

User inserts a row with null value in balance.

```
insert into accounts (account_no, status) values (9997, 'X');
```

Because the balance is null, and the column does not have any default value, the audit condition is not satisfied (the comparison NULL >= 3000 results in FALSE) and the statement is not audited. Important note: Should the column have had a default value of more than 3,000, the statement still would not have been audited, even though the balance column value of the inserted row is greater than 3,000.

1.2.11 All Relevant Columns?

Consider a policy defined on the table ACCOUNTS as follows.

```
dbms_fga.add_policy (
    object_schema   => 'FRED
    object_name     => 'ACCOUNTS',
    policy_name     => 'ACCOUNTS_SEL',
    audit_column    => 'ACCOUNT_NO, BALANCE',
    audit_condition => 'BALANCE >= 3000',
    statement_types => 'SELECT'
 );
```

The policy is defined on the columns ACCOUNT_NO and BALANCE. Assuming the balance for account 9995 is 3.200, if the following statement is issued:

```
select balance from accounts where account_no = 9995;
```

the statement will be audited, since the balance column is chosen and the balance is 3,200, greater than 3,000, satisfying the audit condition. The statement will trigger an audit regardless of which of the three columns are selected.

In some cases the combination of columns may be of importance, but not a specific column. For instance, if a user wants to find out the total balance in the bank, she issues:

```
select sum(balance) from accounts;
```

This query is fairly innocent; it does not specifically identify an account holder and the account balance. Security policy might not require this query to be audited, however, the query

```
select balance from accounts where account_no = 9995
```

must be audited, as it specifically identifies an account. By default all statements, regardless of the combination of columns used, are audited. This will create a large number of unneeded audit trail entries and perhaps some space constraint problems. To limit them, you can specify auditing to kick in only when the desirable combinations of columns are used in the query. While defining the policy, there is a parameter:

```
audit_column_opts => DBMS_FGA.ALL_COLUMNS
```

This parameter will make the policy create audit trail entries only when both the columns ACCOUNT_NO and BALANCE are accessed in the query. For instance, the following query will produce an audit trail entry.

select account_no, balance from accounts;

But, this one will not.

select account_no from accounts;

Use of this parameter will limit the amount of auditing to a more manageable size. If the default behaviour—that is, auditing when any of the columns are selected—is desired, then it is possible to use the different value for the same parameter.

audit_column_opts => DBMS_FGA.ANY_COLUMNS

1.2.12 Capturing Bind Variables

With Oracle Database 10g, additional pertinent information can be written to the regular audit trails such as the values of the bind variables used in the query. Using the initialization parameter performs this

audit_trail = DB_EXTENDED

In FGA audit trails, it may or may not make sense to have the values of the bind variable. To stop recording the values, there is another parameter in the add_policy() procedure as follows.

audit_trail => DB

By default, the bind variables are captured and the value of this parameter is DB_EXTENDED.

1.2.13 Combining Regular and Fine-Grained Auditing

In Oracle Database 10g, regular auditing has also been improved. Implemented by the AUDIT command, regular auditing can now capture a lot of other useful information, such as:

- Extended, granular timestamp
- Operating system process ID
- Transaction identifier (when the audit trail is generated by a data modifying transaction, such as via an update, the transaction id is recorded here, which can be joined later with the view DBA_TRANSACTION_QUERY to identify the exact statement, its undo SQL, the row id, and much more)
- SQL Statement Text
- Values of bind variables
- SCN at the time of change.

In terms of content and capability, the regular auditing resembles the fine-grained version. As a DBA, however, there is interest in knowing all audit entries, not just one. A new view, DBA_COMMON_AUDIT_TRAIL, combines regular and FGA trails. The following query could be used to check them both: select * from dba_common_audit_trail;

This view is a union of DBA_AUDIT_TRAIL and DBA_FGA_AUDIT_TRAIL, with relevant information from each. From the data dictionary, the view is created as shown below.

```
select 'Standard Audit', SESSIONID,
    PROXY_SESSIONID, STATEMENTID, ENTRYID, EXTENDED_TIMESTAMP, GLOBAL_UID,
    USERNAME, CLIENT_ID, Null, OS_USERNAME, USERHOST, OS_PROCESS, TERMINAL,
    INSTANCE_NUMBER, OWNER, OBJ_NAME, Null, NEW_OWNER,
    NEW_NAME, ACTION, ACTION_NAME, AUDIT_OPTION, TRANSACTIONID, RETURNCODE,
    SCN, COMMENT_TEXT, SQL_BIND, SQL_TEXT,
    OBJ_PRIVILEGE, SYS_PRIVILEGE, ADMIN_OPTION, GRANTEE, PRIV_USED,
    SES_ACTIONS, LOGOFF_TIME, LOGOFF_LREAD, LOGOFF_PREAD, LOGOFF_LWRITE,
    LOGOFF_DLOCK, SESSION_CPU
  from DBA_AUDIT_TRAIL
```

```
    UNION ALL
    select 'Fine Grained Audit', SESSION_ID,
        PROXY_SESSIONID, STATEMENTID, ENTRYID, EXTENDED_TIMESTAMP, GLOBAL_UID,
        DB_USER, CLIENT_ID, EXT_NAME, OS_USER, USERHOST, OS_PROCESS, Null,
        INSTANCE_NUMBER, OBJECT_SCHEMA, OBJECT_NAME, POLICY_NAME, Null,
        Null, Null, STATEMENT_TYPE, Null, TRANSACTIONID, Null,
        SCN, COMMENT$TEXT, SQL_BIND, SQL_TEXT,
        Null, Null, Null, Null, Null,
        Null, Null, Null, Null, Null,
        Null, Null
      from DBA_FGA_AUDIT_TRAIL
```

1.2.14 FGA and Regular Auditing: Differences

There are differences between standard and fine-grained auditing in Oracle Database 10g. These are as follows:

- Standard auditing must be enabled at the database level using the parameter AUDIT_TRAIL. This parameter is not dynamic; the database has to be restarted to make it take effect. In contrast, FGA does not require any parameter change.
- Once in place on an object, standard auditing stays there. To deactivate it, you must remove the audit option using the NOAUDIT command. That can be inconvenient because dropping the audit option on a table also drops metadata information. FGA, however, can be temporarily disabled and enabled, without losing any metadata information.
- FGA can handle only four types of statements: SELECT, INSERT, UPDATE, and DELETE. Regular audit, in contrast, can handle many other statements and privileges, even session connections and disconnections.
- Standard audit creates only one record per session (by session) or one per each access to the object (by access). This modest footprint is important for controlling space inside the audit trail tables. FGA isn't as low-profile; it operates on a per-access basis — making the trail bigger.
- Standard auditing can be used to detect any attempts to break in, by recording the trail, and if the attempt was unsuccessful, the error code. FGA can't.
- Standard auditing can write to either database tables or an OS file. The latter is useful when an auditor, not the DBA, has access to the trails. In Windows, the non-DB audit trails are recorded in the Event Log and accessible differently. This option protects the integrity of the audit trails. FGA logs, however, are written only to the database table FGA_LOG$. It is possible to create user-defined audit handlers in FGA to write to OS files, but their integrity is not assured.
- Standard auditing can be set up for default objects. This facility becomes extremely useful in cases where tables are created at runtime: The default auditing option enables auditing without the DBA's intervention. This is not possible in FGA; one must create a policy on an existing table, and that can happen only after the table has been created.
- In FGA, auditing is much more flexible--only when certain columns are accessed, when a certain condition is evaluated to true, and so on. That versatility comes in handy when there is a need to control the growth of the audit trail.
- SQL bind variables are captured by default in FGA. In regular auditing, the initialization parameter audit_trail must be set to db_extended to enable that.
- Privileges differ: regular auditing requires audit system or statement privilege; FGA needs only an execute privilege on dbms_fga package.

From the above comparison, it is apparent that FGA may prove useful in certain cases. With the enhanced regular auditing features in Oracle Database 10g, some tasks previously considered impossible such as capturing values of bind variables, become quite easy.

1.2.15 FGA Summary

FGA enables the support of privacy and accountability policies in an Oracle database. In addition because auditing occurs inside the database, not in an application, actions are audited regardless of the access methods employed by users (through tools such as SQL*Plus or applications), allowing foolproof setup.

1.3 Triggers or FGA?

There are two situations where triggers may be a better choice than FGA:

- FGA inserts the audit trails using an autonomous transaction, which is committed within its own context. If the DML statement fails or is rolled back, the inserted trail record is not rolled back. If a user updates something but does not commit, the change is not made but the audit trail is created anyway. This may lead to several false positive entries in the audit trail, a potentially undesirable situation. Subsequent analysis of the table using SCN numbers captured through flashback queries can probably reveal this problem, but the process may be complicated. But if this risk is not acceptable, then a trigger-based approach is preferred over FGA.
- FGA records the SQL statements issued by the user and the SCN number, but not the values before and after the change. A separate facility must be used to extract those values using flashback queries from the tables. Because the flashback queries depend on the information contained in UNDO segments, which is limited, the facility may not extract the old values from points that far into the past. A trigger-based approach captures the change at the source; hence the recording of old and new values is guaranteed.

1.4 Features and Database versions

The package is built using dynamic SQL. This enables the same code to be compiled upon different database versions, and in this way provide the appropriate functionality upon that particular database. This implication is that not all of the features of the AUDIT_PKG will be available upon every database. The table below describes the features available on the different database versions.

Database Version	Features Available
Oracle 8.1.7 and below	Trigger Auditing only
Oracle 9.x (9i)	Trigger Auditing and FGA for SELECT statements only
Oracle 10.1 and 10.2 (10g)	Trigger Auditing and FGA for SELECT, INSERT, UPDATE and DELETE operations

Table 3 - Features and Database versions

2 Installation Instructions

The AUDIT_PKG the package was designed to be owned by a single individual schema. This schema would then be used to establish the auditing requirements within the database. This schema owner may or may not be the same 'audit table' schema owner. The reason for the distinction is that the package owner will have a number of additional database privileges that it might not be desirable to provide to the Audit table owner. An example of this is if the audit table owner is used for generating reports, or carrying out other tasks within the database and it is necessary to restrict what they can or cannot do.

The first step is to create a user who will 'own' the audit_pkg package. The script provided in Appendix A-3 can be used. This script called 'create_aud_pkg_owner.sql' in the distribution will prompt for the 'audit_pkg owner', a default tablespace and a suitable password for the owner. If the supplied owner already exists within the database, the user will be dropped and re-created. All the required database privileges are provided to the user, enabling the user to perform all the required actions when the procedures are run.

Once installed and compiled cleanly the package is available to use by any user given execute privilege on the package and with the appropriate database privileges.

3 Using the package

There are a number of ways in which the package can be used. Some of these will be outlined below.

3.1 Quick Setup

3.1.1 Trigger Auditing

The package is initially called using the 'setup_audit' procedure, providing the current name of the database schema that is to be audited and the name of a schema to hold the triggering audit tables.

```
exec audit_pkg.setup_audit('Schema Owner','Audit Schema Owner');
```

Where 'schema owner' is the owner of the tables to be audited.

'audit schema owner' is the owner who is to own the audit tables.

The following steps will be performed by the above procedure to generate the required procedures, triggers and privileges to enable the auditing to work. These steps are procedurised and are done automatically.

1. The audit table owner if they exist in the database will be dropped.
2. The specified Audit table owner will be created in the database. This is the second parameter supplied to the setup_audit procedure. The first parameter is the owner of the tables to be audited. The audit table owner cannot be the same as the original table owner. This restriction may be lifted in later versions.
3. The Audit table owner is given the connect role, and 'create table', and create procedure privilege.
4. The originating table owner is given the 'create trigger' privilege.
5. An internal procedure is then run to provide the 'audit table owner' with select privileges on the 'originating tables'.
6. The audit tables will then be created from the 'originating tables'.
7. The originating table owner is then provided with 'select and insert privileges' on the generated audit tables.
8. The next step is to generate the originating table triggers. These triggers will populate the audit tables when the table entries are inserted, updated or deleted.
9. The last step is to test that insert, deletes and updates on the main tables generate entries in the relevant audit tables.

Once the triggering is set up, the user can decide to update or change the triggers to restrict what columns to audit using the package generated triggers as a starting point.

The triggers generated will be suitable for capturing changes on every single column within the original table. It will be necessary to refine the trigger if this generates too much information. For example it may be sufficient to only capture changes on certain columns of the originating table. In these situations the trigger can be used as a model for making the required changes.

Note that LOB column changes will not be captured. The reason for this is that a change to a LOB does not in itself cause a table change and hence the trigger will not be fired and changes captured. In these situations it is necessary for the application to capture the before and after

LOB data in the audit table. The audit table itself will contain the columns ready to accept the LOB data.

3.1.2 Fine Grain Auditing of DML statements.

The Fine grain auditing is included as part of the AUDIT_PKG and has specific procedures available to enable it. FGA works by establishing a policy on the desired table. There is a broad *setup_fga* routine which will establish a FGA policy for every table in a schema using a default condition such that every column and hence every SELECT statement is audited. Similarly there are routines to remove FGA from all the tables in a schema. These routines are used to quickly set up the FGA auditing and over time the conditions and auditing criteria can be refined prior to the application being officially released.

Heavily used tables will generate a lot of entries in the FGA audit table and hence it is a common requirement to change the FGA condition and the columns to which it applies. Procedures within the package enable these to be assembled with a internal table and all applied at once or for changes to be made one at a time.

With Oracle 10g databases the Fine Grain Auditing allows for auditing of INSERT, UPDATE and DELETE statements. With Oracle 9i this is restricted to SELECT statements.

The three main FGA procedures are as follows:

3.1.2.1 Setup_fga

This is the procedure that sets up the Fine Grain Accounting on the tables of the entered user. The schema owner to be audited is the only mandatory parameter to the 'setup_fga' routine and it will establish database policies on all of the schema tables using the default condition '1=1' which is that all columns are audited. Policies may be modified or refined later when more experience in the auditing requirements are known.

To use it make a call to the procedure as follows:

SQL> exec audit_pkg.setup_fga(*'Schema Owner','Test Only flag'*);

The test only flag accepts one value, either 'TRUE' or 'FALSE'. Setting it to false will tell the user what it will do but not actually perform the action.

3.1.2.2 Remove_fga

This is similar to the setup described above. To use it make a call to the procedure as follows:

SQL> exec audit_pkg.remove_fga('*Schema Owner','Test Flag'*);

The parameters are as described for the setup procedure.

3.1.2.3 List_fga

This procedure lists all the existing Fine Grain tables being audited to the screen. It requires no parameters and is invoked as follows:

SQL> exec audit_pkg.list_fga;

3.2 Controlled setup

If the user knows the requirements for auditing then the auditing can be added on a table-by-table basis. This provides a much finer establishment of auditing criteria. In this scenario the first stage would be to create the audit table owner using the 'create_audit_owner' procedure. The default password will be the name of the audit schema. Remember to change the default user password using the 'alter_user_password' routine before final implementation. Then there is the add_auditing_to_table procedure to add trigger auditing to a specified table, and the 'add_fga_on_table' procedure to add FGA. These routines individually perform all the grants required when the audit table is generated and the appropriate grants given. The advantage is that these routines provide finer control especially as the FGA routine allows any particular auditing condition to be specified.

An update FGA routine 'update_fga_condition' allows refinement of the auditing condition at any time. In this release this effectively removes any existing FGA policy and creates a new policy.

Similar routines exist to remove trigger auditing 'remove_auditing_from_table' or FGA 'remove_fga_from_table' from individual tables.

3.3 List Setup.

Similar to the controlled setup described above but it allows the user to build up an internal list within the package and then to run the whole setup from a single command. The initial stage is to ensure that the audit schema owner is created in the database (see above). The procedure 'add_table_to_list' will add triggering auditing or fga auditing criteria to the internal list. Routine 'remove_table_from_list' will similarly remove the specified auditing criteria from the internal list. Routine 'display_list_to_process' along with secondary routines 'display_trig_list_to_process' and 'display_fga_list_to_process' enable the user to see the contents of the internal list to allow further changes before executing the list with the 'process_list' procedure.

3.4 Removing Auditing

Procedure 'remove_fga' will remove all the FGA policies within the database for the specified schema. The procedure 'remove_fga_from_table' would be used to remove a specific policy. Procedure 'drop_audit_user' will perform a similar action for trigger auditing by removing any table triggers updating the audit table. The procedure will then finally remove the audit tables and the audit table owner from the database.

3.5 Other options

There are three routines that were used in development of the audit_pkg, which can be used to display additional user information when the package is used. These are the 'set_dbg' and 'set_nodbg' routines, which turn on output information or off depending on which routine is called.

Some of the procedures will accept an option parameter 'p_list_only' which can be used to show the actions that would be performed were the routine to be called without the parameter set, but will not perform the actions. This is a Boolean variable and would be either TRUE or FALSE.

There is also a small routine to trip the FGA logs. By Oracle design the use of FGA in Oracle is such that if two separate audit policies exist on two individual tables, and a select statement is issued referring to the two tables, then the FGA log will contain two entries. One entry for each policy is generated. This may generate a lot of apparently duplicate table entries in the audit log. It is possible to search through the audit log and remove any of these 'duplicate' entries, based upon the table entries other than the policy name and the object (table) name. Whether this is suitable in the users specific implementation is left to the user to decide.

Notes

1. Input of user names is case insensitive. It will convert them to upper case in the procedure itself.
2. FGA table will get very big. It should be moved to a separate tablespace. See Appendix A.3 on the method used to perform this. Note that with Oracle 10g the default tablespace for this table is SYSAUX, where as with Oracle 9i it is within the SYSTEM tablespace.
3. The default setting for the FGA on the tables is set to audit every column in the table. This will generate copious amounts of output. The FGA condition can be changed on individual tables by using the update_fga_condition procedure.
4. If the package owner, who runs the set up is the same as the owner of the Audit tables, there is a specific check in the package such that privileges are not reassigned.
5. The user name in the FGA Audit tables will be the value established by the application if it is using the sys_context settings. If this value is not set by the application then the application owner will be the logged as the user name.
6. There is still a need to control the FGA audit tables on a periodic basis to keep the table size within a reasonable range.
7. There is still a small issue with duplicate FGA audit entries if two (or more) tables, which are being audited, are contained within the same select statement. This means that some disk space is being used that shouldn't be. A small procedure to periodically remove the duplicates has been written but whether this is desirable is a decision for the user.
8. FGA policy names are generated from the first 30 characters of the table name appended with '_P'. Trigger names are the first 30 characters of the table name appended with '_T'.

4 Viewing Audit History

4.1 Viewing the Trigger Audit History

Each individual table has its own audit trail table so a select statement such as the following would be applicable:

```
select  O_APPEALTYPEID,  N_APPEALTYPEID ,
        O_ACTIONINSTANCEID,  N_ACTIONINSTANCEID,
        O_USERLOGINNAME , N_USERLOGINNAME,
        CHANGED_BY,
        CHANGE_TYPE,
        To_char(TIMESTAMP,'YYMMDDHH24MI')
from AUDITOR.APPEAL
where rownum < 11;
```

The above statement will display the first 10 rows from the APPEAL audit table owned by the schema 'AUDITOR', with the old and new values for the fields APPEALTYPEID, ACTIONINSTANCEID and USERLOGINNAME, along with details of when they were changed and by whom, and whether the change was an insert 'I', delete 'D' or update 'U'.

The select statement would be modified as appropriate to specify which columns were to be displayed and the appropriate name of the audit table owner and table name.

4.2 Viewing the FGA Audit history

The following is a simple query to view the results of the FGA audit trail.

```
SELECT to_char(timestamp, 'YYMMDDHH24MI')
          AS timestamp, db_user, policy_name, sql_bind, sql_text
 FROM dba_fga_audit_trail;
```

Result:

```
TIMESTAMP  DB_USER POLICY_NAME      SQL_BIND SQL_TEXT
---------- ------- ---------------- -------- --------------------

0201301822 MILLER  AUDIT_EMP_SALARY          SELECT  sal
                                              FROM  scott.emp

0201311143 SCOTT   AUDIT_EMP_SALARY          SELECT ename
                                             FROM dept pd, emp pe
                                             WHERE pd.deptno=pe.deptno
                                             AND pd.deptno=10
                                             AND sal>1000
```

5 Procedures and Function Calls

This section describes the calling parameters for every procedure and function used within the package. The external procedures are the ones that will be used in most usual day to day activities. The internal procedures are provided for completeness but are not callable except from within the package itself.

5.1 External procedures and functions

Procedure Name	Parameters	Param Type	Default value	Description
		Debugging procedures		
PROCEDURE set_dbg;	NONE	N/A	-	Used to allow the generation of informational messages intended to assist debugging during development.
PROCEDURE set_nodbg;	NONE	N/A	-	As above but turns of the message generation.
FUNCTION debugging	NONE	N/A		RETURN BOOLEAN Generally only of use internally, determines the state of the internal debug flag.
		Trigger Audited Table Procedures		
PROCEDURE create_audit_table	p_table_name	IN VARCHAR2	-	Generic routine that will create an audit table based on the supplied table. The format of the table created is described elsewhere in the document.
	p_table_owner	IN VARCHAR2	-	
	p_list_only	IN BOOLEAN	FALSE	
	p_audit_owner	IN VARCHAR2	NULL	
PROCEDURE create_audit_trigger	p_table_name	IN VARCHAR2	-	Generic routine to create an audit trigger based on the criteria used for the audit table created in the 'create_audit_table' procedure. This trigger may be used as a model if additional checks or code is required.
	p_table_owner	IN VARCHAR2	-	
	p_list_only	IN BOOLEAN	FALSE	
	p_audit_owner	IN VARCHAR2	-	
PROCEDURE set_audit_table_owner	p_audit_owner	IN VARCHAR2	-	Establishes the default audit table owner, which saves providing the details multiple times if a number of routines are called from a script.
PROCEDURE setup_audit	p_table_owner_in	IN VARCHAR2	-	Routine to set up trigger auditing

Procedure Name	**Parameters**	**Param Type**	**Default value**	**Description**
	p_audit_owner_in	IN VARCHAR2	-	upon **ALL** of the tables within a specified schema, under the specified 'audit table schema.
PROCEDURE create_audit_owner	p_audit_owner	IN VARCHAR2		Creates the specified audit table owner with all appropriate grants etc.
	p_def_tspace	IN VARCHAR2	USERS	
	p_temp_tspace	IN VARCHAR2	TEMP	
PROCEDURE alter_user_password	p_user	IN VARCHAR2		Changes the specified users database password.
	p_password	IN VARCHAR2		
PROCEDURE drop_audit_owner	p_audit_owner	IN VARCHAR2	-	Drops the specified audit table owner. Also removes any triggers referencing the audit table owners tables.
Fine Grain Auditing Procedures				
PROCEDURE setup_fga	p_table_owner_in	IN VARCHAR2	-	Establishes FGA on all the tables within the specified schema using a default 'SELECT' condition, which is upon all of the table columns being audited. The listing only action will inform the user of the tables so affected.
	p_list_only_in	IN BOOLEAN	FALSE	
PROCEDURE remove_fga	p_table_owner_in	IN VARCHAR2	-	Removes FGA from all the tables within the specified schema. The listing only action will inform the user of the tables so affected. Note this will remove ALL FGA policies for the specified schema.
	p_list_only_in	IN BOOLEAN	FALSE	
PROCEDURE list_fga;	NONE	N/A	-	Lists to the screen the tables upon which FGA is active.
List processing options				
PROCEDURE add_table_to_list	table_owner	IN VARCHAR2	-	The default audit_table_owner (v_audit_owner) is obtained from the value contained within the package or from the owner previously specified in the call to procedure 'set_audit_owner' previously described above.
	table_name	IN VARCHAR2	-	
	audit_type	IN VARCHAR2	'FGA'	
	audit_table_owner	IN VARCHAR2	v_audit_owner	

Procedure Name	Parameters	Param Type	Default value	Description
	fga_condition	IN VARCHAR2	NULL	
	action_type	IN VARCHAR2	'ADD'	
	policy_name	IN VARCHAR2	NULL	
	column_names	IN VARCHAR2	NULL	
	statement_types	IN VARCHAR2	NULL	
PROCEDURE display_list_to_process	audit_type	IN VARCHAR2	'ANY'	Lists all the contents of the tables held within the current list prior to the actions on the list items being performed. Used to check that items are correct within a SQL session before proceeding.
PROCEDURE display_fga_list_to_process	audit_type	IN VARCHAR2	'FGA'	As with the 'list_aud_tables' procedure described above but only lists FGA tables.
PROCEDURE remove_table_from_list	table_owner	IN VARCHAR2	-	Removes the indicated table auditing option from the internal list prior to processing. If policy name is specified it overrides the audit type specification. If neither are specified all audit types are removed on the specified table.
	table_name	IN VARCHAR2	-	
	audit_type	IN VARCHAR2	NULL	
	policy_name	IN VARCHAR2	NULL	
PROCEDURE display_trig_list_to_process	audit_type	IN VARCHAR2	'TRIG'	As with the 'list_aud_tables' procedure described above but only lists Trigger audited tables.
PROCEDURE clear_tables_from_list	audit_type	IN VARCHAR2	'ANY'	Removes all tables of the specified type from the internal lists prior to processing.
PROCEDURE process_list	NONE	N/A	-	Processes all the entries in the build up internal table. Internal table is cleared after this procedure has been called.
Individual table changing procedures				
PROCEDURE add_auditing_to_table	table_name	IN VARCHAR2		Adds trigger auditing to specified table, including all grants etc.
	table_owner	IN VARCHAR2		

Procedure Name	Parameters	Param Type	Default value	Description
	audit_table_owner	IN VARCHAR2		
PROCEDURE add_fga_on_table	table_name	IN VARCHAR2		Adds an FGA policy with the specified condition to the system. If the policy name is specified it will be used in preference to the internally generated policy name. If statement_types is not specified the default of 'SELECT' will be used.
	table_owner	IN VARCHAR2		
	fga_condition	IN VARCHAR2		
	column_names	IN VARCHAR2	NULL	
	policy_name	IN VARCHAR2	NULL	
	statement_types	IN VARCHAR2	NULL	
PROCEDURE remove_auditing_from_table	table_name	IN VARCHAR2		Removes trigger auditing from specified table. Removes trigger and audit table.
	table_owner	IN VARCHAR2		
	audit_table_owner	IN VARCHAR2		
	audit_table_name	IN VARCHAR2	NULL	
PROCEDURE remove_fga_from_table	table_owner	IN VARCHAR2		Removes the FGA policy from the specified table. If policy_name is NULL (or not specified) the internal 'SELECT' policy will be removed. If policy_name is the name of a specific policy it will be removed. If policy name is ALL then all policies on the specified table will be removed.
	table_name	IN VARCHAR2		
	policy_name	IN VARCHAR2	NULL	
PROCEDURE update_fga_condition	table_owner	IN VARCHAR2		Updates (changes) the FGA condition on the specified table.
	table_name	IN VARCHAR2		
	fga_condition	IN VARCHAR2		
	column_names	IN VARCHAR2	NULL	
	policy_name	IN VARCHAR2	NULL	
PROCEDURE trim_fga_log	NONE	N/A		Intended to enable trimming of the audit trail logs to remove duplicate entries caused when the same select statement on two or more tables is caught by two or more policies.

Table 4 - External procedures/functions

5.2 Internal procedures and functions

The following table describes the internal procedures and functions available within the package. Internal procedures and functions are not available for external usage unless specifically exposed. Some of these procedures may be usable in future developments or for other packages and are therefore described below for completeness.

If interested in using any of these procedures and or functions please contact the author, or alternatively if you feel confident about exposing those procedures yourself feel free to alter the code at your own risk. No responsibility is taken for any changes made, or the consequences of those changes.

Procedure Name	Parameters	Parameter Type	Default value	Description
PROCEDURE pl	p_str	IN VARCHAR2		
	p_len	IN INTEGER	80	
FUNCTION db_9i	None	N/A		RETURN BOOLEAN. TRUE if Database version is 9i, otherwise FALSE
FUNCTION db_10g	None	N/A		RETURN BOOLEAN. TRUE if Database version is 10g, otherwise FALSE
PROCEDURE copy_blob	p_src_table	IN VARCHAR2		
	p_src_col	IN VARCHAR2		
	p_dest_table	IN VARCHAR2		
	p_dest_col	IN VARCHAR2		
	p_src_off	IN NUMBER	1	
	p_dest_off	IN NUMBER	1	
PROCEDURE issue_grant	p_table_owner	IN VARCHAR2		
	p_table_name	IN VARCHAR2		
	p_grantee	IN VARCHAR2		
	p_grants	IN VARCHAR2		
PROCEDURE	p_table_name	IN VARCHAR2		

Procedure Name	Parameters	Parameter Type	Default value	Description
remove_audit_trigger	p_table_owner	IN VARCHAR2		
PROCEDURE remove_audit_table	p_table_name	IN VARCHAR2		
	p_table_owner	IN VARCHAR2		
PROCEDURE create_fga_policy	p_table_name	IN VARCHAR2		Overloaded version of routine with ability to specify an audit column. In Oracle 9i only single column variable is acceptable. In 10g multiple columns can be specified. If no columns are specified all columns are assumed.
	p_table_owner	IN VARCHAR2		
	p_column	IN VARCHAR2	NULL	
	p_condition	IN VARCHAR2	NULL	
	p_policy_name	IN VARCHAR2	NULL	
	p_statement_types	IN VARCHAR2	NULL	
	p_list_only	IN BOOLEAN	FALSE	
PROCEDURE remove_fga_policy	p_table_name	IN VARCHAR2		
	p_table_owner	IN VARCHAR2		
	p_policy_name	IN VARCHAR2	NULL	
	p_list_only	IN BOOLEAN	FALSE	
PROCEDURE generate_all_audit_tables	p_table_owner	IN VARCHAR2		
PROCEDURE generate_all_audit_triggers	p_table_owner	IN VARCHAR2		
	p_audit_owner	IN VARCHAR2		
PROCEDURE remove_all_audit_triggers	p_owner	IN VARCHAR2		
PROCEDURE generate_all_fga_policies	p_table_owner	IN VARCHAR2		Generate fga policies on all tables for the specified schema using default condition.
	p_list_only	IN BOOLEAN	FALSE	
PROCEDURE remove_all_fga_policies	p_table_owner	IN VARCHAR2		Removes all fga policies on all tables in specified schema.
	p_list_only	IN BOOLEAN	FALSE	
PROCEDURE	p_table_owner	IN VARCHAR2		Generates all tables

Procedure Name	Parameters	Parameter Type	Default value	Description
generate_all_table_grants	p_grantee	IN VARCHAR2		grants on the tables in the specified schema.
	p_grants	IN VARCHAR2		

Table 5 - Internal procedures/functions

6 Outstanding Issues

1. Audit table owner cannot be the same as the table owner.
2. Audit table names are the same as the main table name.
3. LOB triggering cannot be implemented in the package but code to allow the LOB's to be placed in the audit tables is within the package. This code need further testing before production use.
4. The code for handling LOB columns within the trigger generation needs a review before being generally available for production usage. In the mean time the user is advised to check carefully the triggers generated for tables containing LOB columns and modify if required. Warning messages are generated and will be displayed on the screen provided that dbms_output is enabled within the session.
5. No support for FGA handling procedures are currently allowed for.
6. Tables containing LONG variables can not be used with the trigger auditing. The reason for this is that an Oracle table can only contain one long variable. The audit trigger would try to create two columns, both of type long. Since the use of LONG variables is deprecated, it is not considered appropriate to develop a solution to this problem. Note that the Oracle generated 'PLAN_TABLE' contains a LONG variable.
7. The internal string buffer used to hold the code that is then executed is of a finite size. (8192 characters on version 1.4 and below, 16000 characters on 1.5 and above). The consequence of this is that very large tables may cause the internal string buffers to be exceeded.
8. Oracle 11 is not yet supported.

A. Appendix

A.1 Moving the AUD$ table

You should be aware that moving AUD$ out of SYSTEM tablespace is *not* a supported procedure. Oracle does not support changing ownership of AUD$, or any triggers on it.

Oracle stores audit trail records in the SYS.AUD$ base data dictionary table. The problem is that this table grows inside the SYSTEM tablespace and must have records deleted from it or be truncated, otherwise it will take up all the room in the SYSTEM tablespace. This deleting and truncating of the SYS.AUD$ table can cause fragmentation of the SYSTEM tablespace.

The following script allows a DBA to move SYS.AUD$ out of the SYSTEM tablespace. By moving it out of SYSTEM tablespace, the table's size can be controlled without filling or fragmenting the SYSTEM tablespace.

```
create tablespace "AUDIT"
   datafile '$HOME/data/aud01.dbf' size 500k
      default storage (initial 100k next 100k pctincrease 0)
/
create table audx tablespace "AUDIT"
   storage (initial 50k next 50k pctincrease 0)
      as select * from aud$ where 1 = 2
/
rename AUD$ to AUD$$
/
rename audx to aud$
/
create index i_aud2
  on aud$(sessionid, ses$tid)
    tablespace "AUDIT" storage(initial 50k next 50k pctincrease 0)
/
```

Change the script to have the correct values for the new ‘aud$’ tablespace data file, and if desired different storage parameters.

A.2 Moving the FGA$ table

Note this is not an Oracle supported process. You use it at your own risk.

Prior to Oracle 10g being released the Fine Grain tables were located within the SYSTEM tablespace along with the standard AUDIT tables. With Oracle 10g a new tablespace called SYSAUX is defined as the default location for these tables. Part of the 10g changes were that instructions are provided for moving some of these tables out of the SYSAUX tablespace, however at the current time this is not true for the FGA tables. There may be circumstances where it is desirable to relocate the tables to another tablespace and thus the following instructions are retained.

Steps to move sys.fga_log$ from the system tablespace to a user defined tablespace.

Run the following create table script (suitably altered for the correct tablespace of course.)

```
CREATE TABLE SYSTEM.FGA_LOG$
(
 SESSIONID     NUMBER NOT NULL,
 TIMESTAMP#    DATE NOT NULL,
 DBUID         VARCHAR2(30) NULL,
 OSUID         VARCHAR2(255) NULL,
 OSHST         VARCHAR2(128) NULL,
 CLIENTID      VARCHAR2(64) NULL,
 EXTID         VARCHAR2(4000) NULL,
 OBJ$SCHEMA    VARCHAR2(30) NULL,
 OBJ$NAME      VARCHAR2(128) NULL,
 POLICYNAME    VARCHAR2(30) NULL,
 SCN           NUMBER NULL,
 SQLTEXT       VARCHAR2(4000) NULL,
 LSQLTEXT      CLOB NULL,
 SQLBIND       VARCHAR2(4000) NULL,
 COMMENT$TEXT  VARCHAR2(4000) NULL,
 PLHOL         LONG NULL
)
INITRANS 1
MAXTRANS 255
PCTFREE 10
PCTUSED 40
STORAGE (INITIAL 65536
    NEXT 1048576
    PCTINCREASE 0
    MINEXTENTS 1
    MAXEXTENTS 2147483645
    FREELISTS 1
    FREELIST GROUPS 1)
TABLESPACE USERS
LOGGING
/
```

There might be a desire to turn off logging on the table/tablespace to avoid filling up the redo logs. This has some implication on recovery scenarios and tracking of audit changes, which may be unacceptable for the application concerned.

No index has been identified upon the FGA$ table so none has been created upon the new table above. Experience may indicate that an index is required and if so it would be added at this stage.

As the SYS user run the following:

```
rename fga_log$ to fga_log_temp$;
create view fga_log$ as
select * from system.fga_log$;
```

As the SYSTEM user issue the following commands:

```
grant all on fga_log$ to sys with grant option;
grant delete on fga_log$ to delete_catalog_role;
```

Then as the SYS user run the $ORACLE_HOME\rdbms\admin\catfga.sql script.

A.3 Maintaining FGA audit records

The following may provide assistance to the maintenance of the audit tables. To remove entries one must connect as the SYS or internal user to remove entries if the original table is used, or the system owner if the steps outlines in A1 above have been followed.

If using the original, unmoved table the following is expected:

```
 SQL> connect system/manager
 SQL> delete from DBA_FGA_AUDIT_TRAIL;
delete from DBA_FGA_AUDIT_TRAIL
            *
 ERROR at line 1:
 ORA-01031: insufficient privileges

SQL> connect / as sysdba
SQL> delete from DBA_FGA_AUDIT_TRAIL;
2 rows deleted.
 select count(*) from DBA_FGA_AUDIT_TRAIL;
```

```
  COUNT(*)
---------
        0
```

A.4 Installation script

This script is run from a SQL*Plus prompt. It is advisable to run this as the sys user connected as 'sysdba'. If you are unsure how to do this, please consult your Oracle database documentation.

Note that if the security recommended by Oracle is implemented that there will be no public execute grants upon the DBMS_LOB, DBMS_JOB, UTL_TCP, UTL_HTTP, UTL_SMTP and UTL_FILE executables and hence any usage of these will cause a compilation error. This affects the DBMS_LOB package in this particular situation. The script below has been modified to allow for this change.

```
rem
rem Script to create the audit pkg owner.
Rem
rem Version 1.3A   26-Feb-04  GSC
rem Version 1.4    26-Apr-04  GSC
rem Version 1.6    17-Sep-07  GSC
rem
prompt 'This script needs to run as the sys user to ensure'
prompt' that the correct permissions are given.'
prompt
prompt 'Please enter audit package owner:'
accept audown
prompt 'Please enter audit package owner default tablespace'
accept deftblspce

prompt 'Dropping audit package owner if they exist.
drop user &audown cascade;
create user &audown identified by t3 default tablespace &deftblspce temporary tablespace temp;
alter user &audown quota unlimited on &deftblspce;
GRANT CREATE USER TO &audown;
GRANT DROP USER TO &audown;
grant create session to &audown;
GRANT GRANT ANY ROLE TO &audown;
GRANT GRANT ANY OBJECT privilege TO &audown;
GRANT CREATE ANY TABLE TO &audown;
GRANT SELECT ANY TABLE TO &audown;
GRANT DROP ANY TABLE to &audown;
GRANT CREATE ANY TRIGGER TO &audown;
GRANT DROP ANY TRIGGER TO &audown;
grant create procedure to &audown;
grant select on dba_tab_columns to &audown;
grant select on dba_tables to &audown;
grant select on dba_users to &audown;
grant grant any privilege to &audown;
grant execute on dbms_fga to &audown with grant option;
grant execute on dbms_lob to &audown with grant option;
grant select on fga$ to &audown with grant option;
grant select on dba_triggers to &audown;
grant select on dba_objects to &audown;
grant select on dba_fga_audit_trail to &audown with grant option;
prompt 'Might also need to grant delete privilege on fga_log$ IF it has been moved.'
prompt 'The assumption here is that it has not.'
grant delete on fga_log$ to &audown with grant option;
prompt 'Changing to Audit Table owner'
connect &audown/t3
```

```
prompt 'Generating the package.'
prompt
@audit_pkg.sps
@audit_pkg.spb
prompt
prompt 'Now we must change the audit owners password.'
prompt 'Please enter the required password.'
accept passwd
alter user &audown identified by &passwd;
prompt
prompt ' Now check that everything is fine and you can run '
prompt ' audit_pkg.setup_audit('Schema','Audit_schema');'
prompt
```

A.4.1 Alternative PL/SQL script

This following is enclosed for those users that which to implement the install process into an alternative mechanism, such as an ANT deployment.

```
--
-- Create Audit package owner.
--
-- Version 1.4    26-Apr-04   G S Chapman
-- Version 1.5A 17-Sep-07   G S Chapman
--
declare
   audown    varchar2(32):= 'CMTSAUDITOR';
   audpasswd varchar2(32) :='T3';
   deftbl    varchar2(32) := 'USERS';
   temptbl   varchar2(32) := 'TEMP';
begin
  begin
    execute immediate
    'drop user '||audown||' cascade';
  exception
    when others then
      NULL;
  end;
  execute immediate
  'create user '||audown||' identified by '||audpasswd||
  ' default tablespace '||deftbl||' temporary tablespace '||temptbl;

  execute immediate 'GRANT CREATE USER TO '||audown;
  execute immediate 'GRANT DROP USER TO '||audown;
  execute immediate 'grant create session to '||audown;
  execute immediate 'GRANT GRANT ANY ROLE TO '||audown;
  execute immediate 'GRANT GRANT ANY OBJECT privilege TO '||audown;
  execute immediate 'GRANT CREATE ANY TABLE TO '||audown;
  execute immediate 'GRANT SELECT ANY TABLE TO '||audown;
  execute immediate 'GRANT DROP ANY TABLE to '||audown;
  execute immediate 'GRANT CREATE ANY TRIGGER TO '||audown;
  execute immediate 'GRANT DROP ANY TRIGGER TO '||audown;
  execute immediate 'grant create procedure to '||audown;
  execute immediate 'grant select on dba_tab_columns to '||audown;
  execute immediate 'grant select on dba_tables to '||audown;
  execute immediate 'grant select on dba_users to '||audown;
  execute immediate 'grant grant any privilege to '||audown;
  execute immediate 'grant execute on dbms_fga to '||audown;
  execute immediate 'grant execute on dbms_lob to '||audown;
  execute immediate 'grant select on fga$ to '||audown;
  execute immediate 'grant select on dba_objects to '||audown;
  execute immediate 'grant select on dba_triggers to '||audown;
  execute immediate 'grant select on dba_fga_audit_trail to '||audown;
```

```
   dbms_output.put_line('Need to grant delete on fga_log$ if the table has moved.');
   execute immediate 'grant delete on fga_log$ to '||audown;
end;
```