

## Chapter 6: Developing a Proper Audit Trail for your EBS Environment

In Chapter 2, we looked at the inherent architecture of EBS and some implications regarding the lack of a detailed audit trail. Three implications we looked at relate to application controls, change management, and privileged user access and monitoring. The lack of a sufficient audit trail can also have an impact on Sarbanes-Oxley audits, the ability to prevent fraud, and the ability to diagnose application problems.

To develop a proper audit trail, we need to look at the technologies available and their pros and cons. The four categories of audit trails we will review are:

1. Standard application audit information (aka row who, created by/last updated by information)
2. Sign-on audit
3. Snapshot-based technologies
4. Advanced application audit trail methodologies

### Standard Application Audit Information

As discussed in Chapter 2, the standard audit information depends on the level of detail created by the application logic. Most of the transactions, master data maintenance, and configurations contain only the most basic level of information. This information is sometimes referred to as the “Row Who” information because it identifies who entered and/or last maintained information related to a particular row in the database. The information stored is as follows:

- Created by – which application user created the record
- Creation date – the time date stamp the record was created
- Last updated by – which application user last updated the record
- Last update data – the time date stamp the record was last updated

As was illustrated in in Chapter 2, the above information is NOT sufficient for most auditing purposes. The data only reflects the state of the record as of the last time it was updated and provides no history of detailed changes between the time the record was created and the last time it was updated.

Having said that, in cases where the created by and last updated by values are the same, it appears that the record has NOT been updated and would likely represent the only activity related to that record. Once a record has an update to it, you would need an advanced auditing technique in order to create a detailed, sufficient audit trail of all changes made to the record, not just the last change.

## Sign-on Audit Information

Many organizations enable the profile option “Sign-On: Audit Level” to give them some additional auditing. This profile option can be set as: None (default), User, Responsibility or Form; with “Form” being the lowest level (i.e. most audit information created). I recommend, as does Oracle in its Metalink Note 189367.1, to set this profile option to “Form.” By setting it to “Form” the application tracks every login to the application, every responsibility used by the users, and every form opened by the users. However, this functionality only exists to track user activity in professional forms, not OA framework forms.

Professional forms are those that are accessible via the Navigator. An example is Enter Journals in Figure 1.

Figure 1

OA framework forms are HTML web pages and are not tracked through the SignOn Audit process. An example is Account Analysis and Drilldown page in Figure 2.

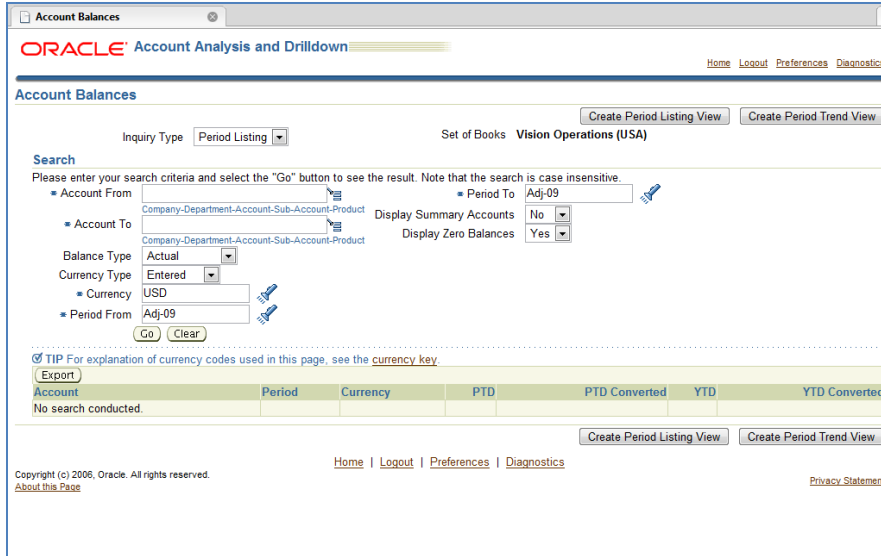


Figure 2

Since the Signon Audit report information only relates to professional forms, it is not at all complete. However, it provides some very basic information (i.e. what forms are accessed) which is of use in auditing professional forms. As more new forms are developed in OA framework pages and existing forms are re-developed in the OA framework, this information becomes less and less valuable since it will represent an increasingly smaller portion of the activity. For example, in release 12, two critical forms related to fraud (Suppliers and Bank Accounts) have been re-developed us OA framework technology.

If you set this profile option at the “Form” level, as recommended, this setting will track the users logins, which responsibilities are used, and which forms they access. However, the reporting gives you no indication of WHAT was done when the user accessed the forms.

Some organizations require their privileged users to log which responsibilities and forms are used when addressing help desk tickets in order to provide some level of accountability for their activities. Those organizations may use the SignOn Audit information to verify which forms were accessed by a privileged user and compare the information to what was logged by the user. While this provides some accountability, it does not provide absolute assurance. For example, if a user noted in a help desk ticket that they were required to look up some information in the Suppliers form, the person reviewing the activity of the user would only see the privileged user accessed the Suppliers form. The reviewer would have no way of knowing whether the privileged user merely viewed a record or set up a new fictitious supplier.

As noted, sign-on audit information may provide some basic information to hold users accountable, but it doesn’t answer the question of WHAT the user did while accessing the forms. Therefore, the information cannot be considered a complete audit trail and does not provide you with any information as to whether or

not the user(s) accessing the form(s) are following procedures or give you any assurance that the user(s) is not committing fraud while accessing the form(s).

## Snapshot-based Technologies

Another common ‘auditing’ technology is the use of snapshots. Snapshot technologies take an image or snapshot of the data as of a point in time. Snapshots can be taken at whatever intervals an organization requests – every 10 minutes, hourly, daily, weekly, etc.

Organizations can take a snapshot of an instance to create an initial version generally referred to as a baseline version. Another snapshot is then taken at a later point in time and compared to the baseline version to identify differences. Organizations may use this to review for changes to application controls or object changes. They may also use the technology to compare two different instances to identify where configuration differences may exist.

The primary benefit of this type of technology is the ability to view certain changes made to tables with very little performance impact, especially when compared to the impact of triggers. The primary drawback is that the data cannot be relied upon to create a detailed audit trail due to the fact that all changes are not captured.

Snapshot-based technologies are best used for operational purposes to identify differences between two instances or points in time to address configuration differences. This may help identify differences between instances to help an analyst determine why some functionality may not be working in a production environment. However, snapshot technologies are not effective to build a sufficient audit trail from a compliance and fraud perspective.

Snapshot technologies provide you with information about what happens when comparing values between two different points in time, but not the details of all changes made. Let’s look at an example and examine the usefulness of snapshot technologies when applied to the baselining of automated or application controls. Look back at the illustration made in Chapter 2 regarding the GL\_JE\_SOURCES table. Refer to page 18.

Remember, we were looking at how a change to the Freeze Journals configuration is stored at the database level. Table 1 shows the change history we noted:

<u>Source Name</u>	<u>Freeze Journals</u>	<u>Require Journal Approval</u>	<u>Last Updated By</u>	<u>Last Updated Date</u>
Receivables	Yes	Yes	DOEJ	01-Jan-06
Receivables	No	Yes	HAREJ	01-Dec-07
Receivables	Yes	Yes	HAREJ	02-Dec-07

Table 1

If a snapshot was taken before the first change (Freeze Journals value was “Yes”) and another snapshot was taken after the second change (where Freeze Journals value was set back to “Yes”) and the two versions were compared, the value in the Freeze Journals column would have been the same. However, the value for the Last Updated By and Last Updated Date columns would indicate that there was a change made to this record (stored as a row) in the database. A reasonable auditor would like to know WHAT the change was and based on the snapshots stored, there would be no record that the Freeze Journals value was changed from “Yes” to “No” and then back to “Yes.”

The predicament you’d find yourself in under this scenario is trying to convince an auditor of the nature of the change. There is no independent evidence available in order to prove the nature of the change. If your organization is using the Journal Approval Workflow as an application control, this scenario could prove to be problematic. From reviewing this data, the auditor has no basis for understanding whether the change was related to the Require Journal Approval flag or another column for this record.

In these circumstances, the auditor could question the integrity of the application control and ‘downgrade’ the control. This would likely mean the process related to this application control (i.e. controls related to approval of journals) would be tested manually rather than relying on the defined application control. Given the significance of this process, this could lead to a material deficiency in your SOX 404 audit. It would likely also result in increased audit fees.

Some have suggested that increasing the frequency of the snapshots could overcome this issue. However, when you think about some changes that could be made intraday (say within an hour), snapshots would have to be made very frequently in order to overcome these issues. What would happen if changes were made within the hour or within the minute? If the schedule of snapshots weren’t frequent enough, the changes wouldn’t be captured.

## Impact of Snapshot Technologies on Application Controls

Automation controls or application controls are controls that are embedded in an application such as EBS. Some of these application controls are dependent on application setups. One example is the journal approval workflow in the General Ledger which is dependent on Profile Option and Journal Source configurations. Some application controls are embedded in the application and cannot be overridden such as the requirement of a journal entry to be balanced (i.e. debits must equal credits) before it can be posted (i.e. acknowledging that a suspense account can be defined to plug the difference, in some cases, where a journal entry's debits and credits aren't equal). From an audit perspective, it is important to distinguish between those that can and cannot be controlled by application configurations.

Examples of application controls include:

- All journal entries must have the sum of debits and credits equal before they can be posted.
- All external requisitions must be approved by someone with appropriate \$ limit authority before they can be converted into a purchase order.
- All orders are subject to a credit review before they are released to be shipped.
- Currency exchange rate must be set up in order for a foreign currency invoice to be entered in Payables.

Some application controls are dependent upon the configuration of the application. In the above examples, both the requisition approval requirement and credit review requirement are processes that can be automated only when properly configured to do so.

Application controls are more reliable because they are automated and cannot be overridden by an end user. Those that rely on certain configurations in order for them to be automated need to have a baseline of their configuration settings. The initial values of these configurations are documented in their initial year of reliance. To the extent that these configurations haven't been changed in subsequent years, an auditor would not have to re-test the process (PCAOB guidance suggests the control should be re-tested every three years).

To the extent that an organization wants to rely on the application controls dependent on certain configurations, they must prove that the configurations related to that application control have not changed. If the configurations have changed, they would need to provide detailed documentation of such changes and show where the changes had been requested and approved by someone authorized by management to make such changes. An auditor would need to determine whether or not they want to re-test the process by developing new baseline values or rely on the testing and approval of changes documented by the organization.

An organization needs to be able to capture any and all changes to the configurations for each application control and provide an effective audit trail for each change.

Organizations looking to baseline their configurations as of a financial statement date, such as December 31, 2007, would take a snapshot as of that date. Then, when an auditor came in the subsequent year, they would compare the current values to the baseline values as of 12/31/07. If the values were the same, the auditor may assume that the values haven't changed. However, the comparison of the two values doesn't prove they didn't change, only that they were the same values at those two points in time. They could have been changed and then changed back to the same value. You may want to revisit the example scenario related to application controls in the earlier chapter.

Snapshot technologies have some value but are NOT fully reliable to build a complete detailed audit trail.

## Advanced Application Audit Trail Methodologies

To have a detailed audit trail, you need to employ an advanced audit trail methodology. Advanced audit trail technologies can be employed using logs or triggers. We will discuss several different types of logs as well as different types of trigger technologies.

### Log-based Technologies

Logs can be either network logs or database logs. Network logs are details of activity that flow across the network. As a user interacts with the application, the activity before the presentation layer (i.e. their desktop) and the application and database layers (which are likely to be in a data center) flows across the network. Network logs track such activity and would provide the new values entered by the user to the extent that it can be understood. Some applications encrypt data between the presentation layer and the database so that no one can steal the data as it passes across the network.

Database logs are intrinsic to the database and are built into the core of the database activities. They are integral to the proper functioning of the database. The database logs are useful for functions such as rollback of transactions and mirroring of instances.

Network logs and database logs have similar pros and cons. The most significant pro is that they are less likely to cause performance impact, when compared to triggers. The most significant con is that it can be difficult to determine what was changed because of the way the data is stored. Let me illustrate with an example. Figure 3 shows a table that stores certain information related to responsibilities in EBS.

Columns				
Name	Datatype	Length	Mandatory	Comments
APPLICATION_ID	NUMBER	(15)	Yes	Application identifier
RESPONSIBILITY_ID	NUMBER	(15)	Yes	Responsibility identifier
LAST_UPDATE_DATE	DATE		Yes	Standard Who column
LAST_UPDATED_BY	NUMBER	(15)	Yes	Standard Who column
CREATION_DATE	DATE		Yes	Standard Who column
CREATED_BY	NUMBER	(15)	Yes	Standard Who column
LAST_UPDATE_LOGIN	NUMBER	(15)	Yes	Standard Who column
DATA_GROUP_APPLICATION_ID	NUMBER	(15)	Yes	Data group application identifier
DATA_GROUP_ID	NUMBER	(15)	Yes	Data group identifier
MENU_ID	NUMBER	(15)	Yes	Menu identifier
TERM_SECURITY_ENABLED_FLAG	VARCHAR2(1)			Flag to indicate if Security by Terminal is enabled for the responsibility
START_DATE	DATE		Yes	The date the responsibility becomes active
END_DATE	DATE			The date the responsibility expires
GROUP_APPLICATION_ID	NUMBER	(15)		Application identifier from report security group definition
REQUEST_GROUP_ID	NUMBER	(15)		Identifier of report security group assigned to the responsibility
VERSION	VARCHAR2(1)			Version of responsibility. For example, web (W) or AOL (4)
WEB_HOST_NAME	VARCHAR2(80)			IP address or alias of machine where the Webserver is running. Defaults to the last agent
WEB_AGENT_NAME	VARCHAR2(80)			Name of Oracle Web Agent. Defaults to the last agent
RESPONSIBILITY_KEY	VARCHAR2(30)		Yes	Internal developer name for responsibility

Figure 3

In Figure 3 we see the columns in the FND\_RESPONSIBILITY table. This table stores the data related to the definition of a responsibility definition. In my illustration, let's consider that a menu is being changed. Perhaps the organization has agreed that all menus associated with a responsibility should be custom menus rather than seeded menus. So, they have built a new custom menu and, in the Responsibilities form, have updated the responsibility definition with the new menu. In the table above, the data stored that is related to the menu is the MENU\_ID. A change log visible through either network logs or database logs would show that the new value entered was the new MENU\_ID. To an auditor or someone reviewing the logs, the information they would receive in an audit report is that a new value was entered in the table FND\_RESPONSIBILITY for the column MENU\_ID. For an auditor to know whether this change was the approved change or was another menu that was not approved, they would have to know more about the MENU\_ID. Details on the MENU\_ID are stored in another table, FND\_MENUS. The columns in the FND\_MENUS table can be seen in Figure 4.

Columns				
Name	Datatype	Length	Mandatory	Comments
MENU_ID	NUMBER		Yes	Menu identifier
MENU_NAME	VARCHAR2(30)		Yes	Menu name
LAST_UPDATE_DATE	DATE		Yes	Standard Who column
LAST_UPDATED_BY	NUMBER		Yes	Standard Who column
LAST_UPDATE_LOGIN	NUMBER	(15)	Yes	Standard Who column
CREATION_DATE	DATE		Yes	Standard Who column
CREATED_BY	NUMBER	(15)	Yes	Standard Who column
TYPE	VARCHAR2(30)			TYPE

Figure 4

You can see the amount of information related to a menu is limited in this table. Perhaps other tables have additional information about the menus as well. Let's take a look at FND\_MENUS\_TL in Figure 5.

Name	Datatype	Length	Mandatory	Comments
LANGUAGE	VARCHAR2 (4)		Yes	Language
MENU_ID	NUMBER		Yes	Menu identifier
USER_MENU_NAME	VARCHAR2 (80)		Yes	Displayed name of menu
LAST_UPDATE_DATE	DATE		Yes	Standard Who column
LAST_UPDATED_BY	NUMBER (15)		Yes	Standard Who column
LAST_UPDATE_LOGIN	NUMBER (15)		Yes	Standard Who column
CREATION_DATE	DATE		Yes	Standard Who column
CREATED_BY	NUMBER (15)		Yes	Standard Who column
DESCRIPTION	VARCHAR2 (240)			Description
SOURCE_LANG	VARCHAR2 (4)		Yes	The Language the text will mirror. If text is not yet translated into LANGUAGE then any changes to the text in the source language row will be reflected here as well.

Figure 5

For an auditor to make an assessment of whether or not the change to the responsibility’s menu is reasonable, they would need to know more information about the new menu and ideally this information would be pulled into the audit records at the time of the change in the main table rather than cross-referencing this information at the time the audit information is being reviewed. The most valuable information would likely be the Menu Name (stored in the FND\_MENUS table) and the User Menu Name and Description columns (which are stored in the FND\_MENUS\_TL table).

Although the information stored in logs is accurate, it is less than perfect when trying to understand whether or not the change was appropriate. Audit trails built via Logs are useful but are perhaps not ideal.

### Trigger-based Technologies

Triggers are another type of technology we’ll look at and, admittedly, are my favorite technology to use. Triggers have received a bad rap over the years because of their perceived performance impact on the system. The performance impact *does need to be taken into consideration*, but the major use of triggers to create an audit trail would be on low-volume tables. Triggers only create overhead (i.e. use system resources) when they ‘fire’. Triggers only fire when transactions are written to tables that contain the triggers. Therefore, as long as triggers are used on low volume tables, the performance impact should be negligible. With any use of triggers, an evaluation of the code (i.e. a peer review) should be done by a qualified expert, and a performance evaluation should be part of the testing process.

What are the benefits of triggers and why should they be preferred over other technologies? The benefits are:

- Before and after values are written to the audit record
- Triggers can have conditions placed on them (i.e. a where clause added to minimize the scope and, therefore, performance impact and the amount of data being stored)
- Additional data (i.e. metadata) can be captured at the time the audit trail record is written

*Before and after values*

The ability to capture the before and after values of a change to the record greatly enhances the ability to understand and evaluate the appropriateness of the change. This allows a reviewer of the change to understand the nature of the data before the change as well as the new value. It should also allow them easier comparison to the documentation in your change management process.

*Conditions for triggers*

Triggers can be used to monitor activity of certain users or activity with certain conditions. This is done by narrowing the scope of the trigger by adding a 'WHEN' clause to the trigger. For example, if you are monitoring activity of certain users (such as privileged users) or transactions above certain levels (for adherence to policy), a "WHEN" clause would be important. A trigger has two components to it – a header and a body. The header is where the WHEN condition is written and determines whether or not the body should 'fire.' The body contains that 'what to do' portion of the trigger which in this case is to write an audit record to a certain table. The 'expensive' portion of this transaction from a system perspective is the 'firing' of the trigger body. Therefore, reducing the conditions that cause a trigger body to 'fire' by placing limiting clauses in the header (via the 'WHEN' clause) will minimize the performance impact (unless, of course, the 'WHEN' clause is poorly written).

Reducing the scope of the trigger via the 'WHEN' clause also could significantly reduce the amount of data stored, a key concern for the DBAs.

*Additional data can be captured when writing the audit trail record*

When an audit trail record is written, additional data related to the change can be captured. In the example above, we looked at a change to the menu in the definition of a responsibility. A trigger can "SELECT" additional data related to the before and after values in order to provide the auditor of the data more information. This means that an audit record can store the Menu Name, User Menu Name and Description fields from other tables in addition to the MENU\_ID changed in the FND\_RESPONSIBILITY table.

**EBS System Administrator Advanced Auditing; Trigger-Based**

Oracle EBS has a 'trigger-based' mechanism available to build an advanced audit trail. It is enabled through the System Administrator menu by defining Audit Groups where you choose the tables and columns that you'd like to audit. This process enables triggers on these tables and writes audit records to shadow tables (suffixed with \_s) for each table you are auditing. This feature is a standard function built into the EBS system.

However, the limitations and drawbacks of this process are significant. First, the triggers cannot be customized to incorporate necessary meta-data. This means that the cross reference data stored in other

tables such as Menu Name, User Menu Name, and Description will not be written to the audit trail so the information provided in the audit trail will be just the basic data changed (i.e. the MENU\_ID). You will also not be able to add a 'WHEN' clause to the data to reduce the scope and potential performance and data storage impact.

Finally, and perhaps most significantly, the audit trail data is stored decentralized. That is, each table's audit trail data is stored in its own table (a shadow table). This means that for every table you audit, a separate query would need to be written to extract the data for reporting purposes. In the case in which you would be auditing a significant number of tables (well over a hundred and perhaps significantly more), the development of reporting of such data would be a substantial project in and of itself. Data growth and management of data in these decentralized tables is another concern.

Some of Oracle's other solutions as well as other available third-party software (from companies such as Absolute Technologies and CaoSys) have a standard reporting schema such that audit trail is stored in a central location to allow for ease of reporting and data management. Some of these tools also have built-in processes to manage the data retention, based on a pre-defined number of days, to help manage the data growth.

While the mechanism that is available as part of the standard EBS features is free, the drawbacks, in my opinion, drive me towards looking at other solutions available from Oracle and from third-party vendors.

### **Evaluating Advanced Application Auditing Technologies**

The challenge for organizations in evaluating various technologies such as trigger and log-based solutions is to get beyond the marketing hype and to gain a thorough understanding of what features each organization has to offer. This is an area I do my best to cover as an analyst and would be happy to discuss pros/cons of various company's solutions and technologies. Because of the evolving nature of each company's offerings, I have decided not to provide detailed analysis of offerings in this space in this book. However, feel free to contact me at [jhare@erpseminars.com](mailto:jhare@erpseminars.com) if you'd like to set up a call to discuss your options further.

### **What to Audit**

Once you have resolved which types of technology to evaluate, the natural next question to answer is; "What types of information do you need to audit?"

I have mentioned the types of information that should go through your organization's change management process in the chapter on change management and included it again in Table 2

<u>Category</u>	<u>Form / Function</u>
Application Controls	Journal Sources (GL), Journal Authorization Limits (GL), Approval Groups (PO), Adjustment Approval Limits (AR), Receivables Activities (AR), OM Holds (OM), Line Types (PO), Document Types (PO), Approval Groups (PO), Approval Group Assignments (PO), Approval Group Hierarchies (PO)
Affect Business Process	Profile Options, DFFs, KFFs, Value Set Changes
Development	Concurrent Programs, Executables, Functions, SQL forms
Security	Menus, Roles, Responsibilities, Request Groups, Security Profiles, SQL forms such as Dynamic Trigger Maintenance, Define Profile Options, Alerts, Collection Plans, etc (see Metalink Note 189367.1 for more information on SQL forms)

Table 2

In addition to these types of records that should be audited to support an audit of your change management process, other records need to be audited as well. Examples include Suppliers, Bank Accounts, Remit To Addresses, and Locations. Detailed audit records and related monitoring on these types of records are needed to support a review of fraud and operational considerations as well as to monitor the activity of privileged users.

## Audit Trail Conclusions

Many organizations still have significant risk due to a lack of detailed audit trail records to support internal and external audit requirements of application changes. In many cases, organizations have chosen to put off addressing these risks because external auditors haven't pointed out such risks. However, time may be running out for organizations to address these significant issues. Not addressing the risks may result in audit findings, errors or actual fraud taking place within the organization.