ORACLE
INSURANCE

Oracle Health Insurance Back Office

Modification Logging within

Oracle Health Insurance Back Office

version 1.4

Part number: E51467_01

December 2013

Copyright © 2011, 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are “commercial computer software” or “commercial technical data” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Where an Oracle offering includes third party content or software, we may be required to include related notices. For information on third party notices and the software and related documentation in connection with which they need to be included, please contact the attorney from the Development and Strategic Initiatives Legal Group that supports the development team for the Oracle offering. Contact information can be found on the Attorney Contact Chart.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

CHANGE HISTORY

Release	Version	Changes
10.12.2.0.0.0	1.3	Added more details regarding how modification logging works, because of received questions, and included list of ALG_LOGGING_PCK predefined configuration procedures.
10.13.3.0.0	1.4	Added paragraph about cached settings.

Introduction

This document describes the operation of the modification logging functionality (logging of changes in data) that is available within the OHI BO application. First of all, an outline is provided of the reasons for modification logging.

This is followed by a summary of modification logging within OHI BO.

Finally, a more detailed description of modification logging is provided.

Why modification logging?

There are various reasons and requirements for equipping an application with modification logging. These can be summarized as follows:

1. **Traceability**
   Providing a means to trace who performed a specific modification and when.
2. **Modification reports**
   Providing means to enable provision of simple overviews of the data modified within a specific period. This can serve a number of purposes.
3. **Interface support**
   Retaining modifications in order to pass them on to other applications in the correct sequence and manner.

Because the application is used by different organizations with varying requirements, modification logging within OHI BO is flexible, enabling fulfillment of the various requirements by means of configuration.

Modification logging operation summary

A generic type of modification logging activation has been implemented in the OHI BO application, which enables management of each individual table. Consequently, each OHI BO table has a 'shadow' log table. Furthermore, there is a central log table in which modifications to specific tables can be logged.

Specific modification logging levels can be activated by means of specific indicators that can be configured per table in the ALG_TABELLEN table.

Modification logging is not activated by default. There is no screen for this activation. Consequently, the relevant columns will have to be updated using (PL/)SQL (with the aid of the ALG_LOGGING_PCK) if required.

There are three types of configuration settings:

1. **SOORT_LOGGING**
   If modification logging (insert, update and delete operations) must be performed in the shadow table for the table:
   N(ot), B(asic), enabling retrieval of all modifications with the minimum of additional data storage, or U (for Extensive), such that complete records (both new and old values in the event of updates) are placed in the shadow table.

   The latter is used to facilitate easier tracing of who performed which transactions. It is very useful and faster/easier to retrieve what kind of change has been executed but requires more storage space.
2. **NIVEAU_LOGGING**
   If modifications must be logged in the central log table

ALG_MUTATIE_LOG:
N(ot), S(tatement) or for every R(ecord).

As a result, the central table can be used to immediately establish the modification types performed, as well as the sequence, and which table(s), without all of the tables having to be examined separately.

3. **NUMMERING_LOGGING**
   If sequence numbers must be distributed:
   N(ot), database T(ransaction) number per set of changes which are contained in one transaction, with a sequence number within transaction per record which is changed, or G(lobal). Global means that an overall modification ID is distributed per statement (insert, update or delete) in addition to the transaction ID and sequence numbers.

   A transaction number is used to track a single Oracle transaction (commit) in detail, while the global (statement) numbers are used to determine the overall sequence of statements executed in different transactions.

Detailed technical modification logging design

Modification logging is designed in such a manner that an organization need not necessarily use it.

Furthermore, if modification logging is required, the different configuration options can be used to fulfill the various requirements.

Default auditing columns in regular tables

The 'standard' tables contain seven default columns that are also important for the modification logging.

Every 'regular functional' table will be allocated a system-generated sequence number generator that serves as a unique ID generator for each record in the table. This enables standardized identification of records in every table by means of a reference to the table and the record number (so each table has default an ID column). This is useful for modification logging, but also offers advantages such as later generic functionality implementations, or for the implementation of interfaces, for example.

The other six default columns are for *auditing*. They are used to record creation and last modification data.

Three column types are distinguished for creation and last modification logging:

1. An ID for the user who performed the modification (a numerical reference to the ALG_FUNCTIONARISSEN table in order to save space).
2. The time at which the modification was performed (up to the second).
3. A source acknowledgement (e.g. the ID of another system or interface in the event that that system or interface submits a notification regarding being the source of the modification).
   The latter is optional and is only registered in the event that the invoking system or code activates/specifies this.

An example of a table (ALG_BRONNEN, see later) is displayed below (the first column and final six columns are the above-mentioned seven columns):

```
ID                                                NOT NULL NUMBER(14)
CODE                                              NOT NULL VARCHAR2(10)
OMS                                               NOT NULL VARCHAR2(200)
CREATIE_BRON_ID                                            NUMBER(14)
```

```
CREATIE_MOMENT                                  DATE
CREATIE_DOOR                                    NUMBER(14)
LAATSTE_MUTATIE_BRON_ID                         NUMBER(14)
LAATSTE_MUTATIE_MOMENT                          DATE
LAATSTE_MUTATIE_DOOR                            NUMBER(14)
```

In the event of initial data entry, the 'last modification' columns are filled with the same data as the 'creation' columns.

The sources and user 'descriptions' are maintained in separate tables (in ALG_BRONNEN and ALG_FUNCTIONARISSEN respectively; ALG_BRONNEN is the table that contains source identifiers used to describe the ID as stored in the 'source columns', CREATIE_BRON_ID and LAATSTE_MUTATIE_BRON_ID, that are also present in this table). The auditing columns contain a reference in order to minimize the storage space used: columns %BRON_ID reference a record in ALG_BRONNEN and columns %DOOR reference a record in ALG_FUNCTIONARISSEN.
A source or user 'definition' can be deleted from the specification tables even if there are still records that refer to them. This is the responsibility of the organization using the tables. The implementation of a check, by means of a foreign key for example, for this purpose would be much too 'costly' from a performance point of view.

Configurable modification logging

Because of the various configuration possibilities, the data model features an entity that contains the tables comprising the application. This entity also enables entry of various settings per table. Consequently, modification logging can be configured for each individual table.

However, this only applies to the regular 'functional' data tables (please see the paragraph about availability of modification logging later on). There are no modification logging possibilities for technical tables containing system maintained or temporary data, etc.

In general, it can be assumed that the greater the accessibility of the logging data, the higher the required overhead and the greater the storage space required. Logging must also enable relatively rapid retrieval of specific modifications for disclosure to other parties, for example.

In the light of the above, an implementation method has been chosen that can meet the following configuration requirements − which have served as base requirements − by means of three settings:

1. **Logging?**
   Is it necessary to log modifications?
2. **Log including complete record with new(est)values?**
   In the event of logging, is it necessary to log only old and modifiable values during an update (after all, the current table contains the new values and fixed values), or is it necessary to log all of the new values in addition to the old values for the purpose of simplicity?

   In such cases the records are immediately saved in the logging table with all of the new values in the event of additions/inserts. Furthermore, in the event of modifications/updates, all new values are saved in the same record with the modifiable old values.

   In the event of modifications/updates and removals/deletions, the auditing values from the last update or deletion are placed in the modification record, in addition to the time stamp of the previous modification, regardless of the setting for logging new values.

This can be used to immediately determine the validity of the data (old modification moment up to and including the new modification moment).

In the event of deletions the record is always saved in its entirety if logging is activated to avoid the values being lost.

The unique record ID is always recorded for every logging activity, regardless of the type of modification (insert/update/delete).

This approach enables recording of logging information with minimum overhead if required. This only applies to the previously modifiable values in old columns for which all modifications are traceable.

3. **Log unique sequential modification ID in log record?**
   Is it necessary to record a detailed modification ID in the log record that determines a unique sequence number for all modifications in the application? This number is required additional to a timestamp, as a large number of modifications can be performed within the same second. Consequently, an ID of this type is necessary to determine the sequence of modifications over various records and/or tables.

   Because this number must be generated for each SQL statement (DML) and each record across all tables, the ID generator is a potential 'hot spot' (although is optimized and extremely fast), which means it should be employed only when really required.

4. **Log statement ID in central overall modification table?**
   Because the modifications performed within a specific period should easily be retrieved without the need to scan a lot or all of the tables, it is possible to indicate that DML statements that have actually resulted in a modification must be recorded in a *central transaction log table* (in which an entry is made stating that a record has been inserted in a specific table, for example).

   The type of statement (insert, update, delete), executing user account and (optional, see next bullet) involved record are then recorded for the table concerned.

5. **Should the modification ID also be logged centrally for all records?**
   If the modification flow must be produced in greater detail for passing on modifications to another system in exactly the correct sequence, it is possible to indicate that the detailed modification ID must also be recorded centrally for each record, i.e. in the central table. This may result in a greater number of entries in the central table including a record ID.

   Consequently, at this stage only the central table can be used to determine what record types are required from the modification tables in order to reconstruct an outgoing transaction without the various separate log tables having to be scanned (that would be very inefficient to reproduce transactions that span a set of unknown tables; all log tables would need to be scanned for each transaction).

6. **Log commit ID in log table?**
   For certain traceability activities (as well as software problems, for example) it can be desirable to establish which modifications were performed within a specific database transaction and in what sequence.

   This can also be desirable in order to make it relatively easy to see by means of which transactions a user produced a specific modification.
   This functionality can be activated by means of the generation of a commit ID (an ID for each transaction being committed) in the log table with a sequence number within the commit ID for each modified record (across the various tables).

Please note that commit IDs do not indicate a mutual sequence of execution (10 short transactions can take place within the duration of a single, long transaction, and be completed before the long transaction).

7. **Should the commit ID also be logged centrally for all records?**
   The commit ID (commit ID + sequence number per record) can also be included in the central table to facilitate easy establishment of the total composition of a logical transaction.

As you may already have concluded from the above, modification logging by table is provided at record level, and not at column level. Past experience has shown that modification logging at column level has almost no added value, while it does demand considerable additional work in terms of configurability and use of the logged data, but also in terms of logging implementation.

Configurability at column level is more complex, more expensive in terms of realization, and will often be configured as logging at record level because the exact data required cannot be established. Moreover, modification logging at record level can indeed be used to retrieve every modification performed at column level.

For the purpose of simplicity, the above-mentioned settings have been combined into three settings, each with three possible values (explained earlier, see above).

A specific OHI BO log table (for example ALG$BRONNEN, for regular table ALG_BRONNEN) is as follows:

```
MUTATIE_OPERATIE                          NOT NULL VARCHAR2(1)
MUTATIE_ID                                         NUMBER(14)|
COMMIT_ID                                          NUMBER(14)
SEQ_IN_COMMIT                                      NUMBER(10)
O$OMS                                              VARCHAR2(200)
O$LAATSTE_MUTATIE_BRON_ID                          NUMBER(14)
O$LAATSTE_MUTATIE_MOMENT                           DATE
O$LAATSTE_MUTATIE_DOOR                             NUMBER(14)
ID                                                 NUMBER(14)
CODE                                               VARCHAR2(10)
OMS                                                VARCHAR2(200)
CREATIE_BRON_ID                                    NUMBER(14)
CREATIE_MOMENT                                     DATE
CREATIE_DOOR                                       NUMBER(14)
LAATSTE_MUTATIE_BRON_ID                            NUMBER(14)
LAATSTE_MUTATIE_MOMENT                             DATE
LAATSTE_MUTATIE_DOOR                               NUMBER(14)
```

The modification type (Insert, Update or Delete) that resulted in the log record is initially recorded in the first column. The other three columns are used to record the modification ID and the commit ID if applicable, provided configurations have been made for this purpose.

The columns in the table that can be modified by the user or by means of the code are included twice: once as old columns (the name is identical to the column name in the default table, with the exception of the first two characters, an O followed by a dollar sign, $) and again as ‘new’ columns (the column names are identical to those in the default table).

Columns that cannot be modified are only included as ‘new’ columns.

When ‘minimal’ logging is activated for the sole reason of traceability, a record containing all modifiable columns with the values prior to the record modification (regardless of whether the modifiable column concerned has also been modified) is saved in the event of a modification to a modifiable column.

This makes it possible to reconstruct any past situation in combination with the current record (uniquely identified by means of its ID).

The name of a shadow log table is identical to that of the original table. However, the underscore (_) in position 4 is replaced by a dollar sign ($), for example ALG$BRONNEN as opposed to ALG_BRONNEN.

When the composition of a transaction, and the sequence in which modifications are performed across the various tables, needs to be established without large numbers of scans of all log tables, it is possible to use additional logging of references to the log tables in the central overall log table.

The central OHI BO log table ALG_MUTATIE_LOG is structured as follows:

```
MUTATIE_OPERATIE                          NOT NULL VARCHAR2(1)
TAB_ID                                    NOT NULL NUMBER(14)
RECORD_ID                                          NUMBER(14)
COMMIT_ID                                          NUMBER(14)
SEQ_IN_COMMIT                                      NUMBER(14)
MUTATIE_ID                                         NUMBER(14)
MUTATIE_DOOR                                       NUMBER(14)
MUTATIE_MOMENT                                     DATE
```

This table contains a reference to the table containing the table names, which facilitates easy tracing of the table in which the centrally logged modification was performed. The column containing the Record ID refers to the unique number per table issued to each record.

Activation of modification logging

The ALG_LOGGING_PCK package contains a number of public procedures that enable the activation and deactivation of logging/journaling at various levels. Naturally, modification logging can also be activated manually by means of SQL statements in the ALG_TABELLEN table for tables for which it is already supported (i.e. the tables for which there is a shadow table).

The ALG_LOGGING_PCK offers the following routines that require a table name as parameter value to implement the functionality:

- DISABLE_LOGGING – disables logging to the central table but leaves logging/journaling records to the shadow table as specified earlier
- DISABLE_JOURNALLING – completely disables any kind of logging for the table, whether it is the central table or the table specific shadow table
- ENABLE_JOURNALLING_BASIC – enable logging to the shadow table in the 'cheapest' (least space requiring) mode
- ENABLE_JOURNALLING_FULL – enable logging to the shadow table while storing the complete record and old and new values
- ENABLE_LOGGING_STMT – enables logging of statement executions on a table to the central log table with transaction id and modification id sequence numbers being assigned
- ENABLE_LOGGING_ROW – enable logging of each involved row that is changed by statement execution to the central log table with transaction id and modification id sequence numbers being assigned

If you like to have a taste of logging 'in between' what is offered above (for example logging of statements to the central log table but without a global modification id being assigned) you should update table ALG_TABELLEN directly.

The best way to determine what is most suited for you is to just start with enabling logging on some tables, enforce some statements that fire and looking at the results in the shadow tables and central log table.

Cached modification logging settings

Beware, when you change settings for the modification logging they are not immediately active:

- Of course you need to commit your changes in order to make them visible for other sessions.

- For performance improvement sessions cache the settings for modification logging. When code associated with a table is initialized the first time immediately the modification logging settings are cached. This may also be a result of a business rule firing for a completely different table that uses a helper function from a related table. In that case the settings for that related table are also initialized. So the only way to be sure the new settings are used is to restart all present sessions. Only when a session reconnects the settings will be read again during code 'warming up' in that session.

Performance and storage space implications

As you may already have concluded, modification logging takes into account the resulting additional load on the system and the extra storage space required as much as possible.

The various settings enable the organization using the system to reasonably weigh the overheads required to record the logging and the effort required for its use. In this respect, easier-to-use modification logging means that it's recording requires additional resources.

It is difficult to provide exact figures regarding the overall impact on the performance of the application, the main reason being that detailed insight is required into the current composition of the system load (load percentage caused by modifications versus other use, modification types, quantities per table, etc.), as well as insight into future logging settings. In reality, actual experience can only be gained in practice.

However, a general comment can be made regarding additional overhead during the recording of a modification. Because a variety of checks are already performed during the recording of the regular modification, and various indexes can be updated, the overhead for the recording of an additional log record will be relatively limited. Depending on the data that is to be logged (and therefore configurable) and the size of the regular modification, the additional load is expected to be between a few percent and a few tens of percents.

As regards the additional storage space, little can be said in advance. To a large extent this depends on the frequency and nature of the modifications and the logging settings, as well the length of time for which the logged data is stored.

There are no default cleaning procedures for this purpose. If required, requests can be submitted for the implementation of default cleaning functionality in a reference release.

Availability of modification logging

The above-mentioned configuration options per table are available for all regular functional tables that are directly available for executing inserts, update or deletes. These tables implement business rule validation through a series of triggers and constraint definitions. The business rule validation mechanism implements also this modification logging functionality.

This means that other tables, which cannot directly be modified, do not offer this functionality. Such tables are typically maintained by the application. These application maintained (or code maintained) tables are often referred as 'technical'

tables in comparison with the user maintainable tables as 'functional' tables. The technical tables can be recognized as having a '#' sign on the fourth position of their name.

The standardized way of granting insert, update, delete and select privileges makes sure only the regular 'functional' tables can be changed.

Impact of release upgrades

When a new OHI release is installed an important requirement is to minimalize the required installation time in order to have a minimal downtime of the application.

For that reason changes in the data structure or in the data as result of an OHI release installation (a single patch, a patch set or a major release) are often optimized. These optimizations may result in the following consequences for the logging data (although they often do not apply):

- ✓ Changes due to a scripted update are not logged
- ✓ The table structure is changed and the current contents is converted in the table (by adding for example a default value for a new column) while the logging table is not updated
- ✓ A table may be newly (re)created where the existing data is converted to the new structure; these changes will not be reflected in the logging table
- ✓ The log table might need to be recreated or dropped; in which situation the old contents are dropped

These consequences do not apply to the central logging table.

Although these consequences might look severe, the heavier the consequence the less often it occurs. And it must be considered that these are completely scripted repeatable operations. For a potential impact of these type of unlogged changes we advise to determine the 'delta' between the situation before and after a release installation to judge whether such modifications are relevant for the derived functionality. Only when there are consequences these must be implemented on the derived environment.

In situations where traceability is key and the rare situation that a log table is cleared we advise to export the table contents before implementing the release installation on the production environment. The saved contents can be used for a custom conversion or later retrieval when necessary.