Oracle Health Insurance Back Office

Object Authorization within

Oracle Health Insurance Back Office

Version 1.8

Part number: E64133-01

May 27, 2015

Change History

Release	Version	Changes
10.12.2.0.0	1.5	• Grants to four views are not dependent anymore on use of General Ledger
10.12.3.0.0	1.6	• Information is added about the 'with grant option' grant for custom code objects used in 'translation views'.
10.14.2.0.0	1.7	• Only minor adjustments
10.15.1.0.0	1.8	• The GRANT_OPTION parameter of the OZG_DIRECT.grt script has been documented. • Removed reference to the GL objects. • Made more clear that custom code objects used in OHI views need to be granted with full DML privileges to the OHI object owner.

Introduction

This document contains a detailed explanation of the procedure employed for roles, 'grants', custom software (including custom code defined within the application as pl/sql definitions) and accounts used for interfaces.

This is based on the principle that the server component of the application must ultimately be fully robust and not permit any corrupting modifications.

Data may therefore only be modified in such a manner that it remains compliant with the business rules. For this a strict database object authorization mechanism needs to be maintained and obeyed.

Robust database

One of the main ideas behind the 'modernization' of the application, as implemented some years ago, was the provision of a 'robust database' that may also be manipulated using software other than the standard reference version. This allows OHI customers to develop their own (user) interface(s) independently from Oracle. This refers to interfaces that are specifically geared towards supporting a specific process.

Because this modernization process could not be completed 'overnight' in one release the consequences were published by means of amendments in various releases. Furthermore, only the database structure was being modernized as part of this process. The application software remained unchanged for the most part.

Bearing this in mind, the database object authorization is explained in further detail below.

In this context, we define a 'robust database' as follows:

1) The inability to implement modifications that do not comply with the integrated business rules.

2) This is regardless of the manner in which the database is accessed, as the check is performed directly on the data within the database.

The consistency and validity of the data can be guaranteed for as long as the DBA ensures that the database can only be accessed via accounts that have no more than the permitted rights. This also applies to the parts of the database structure that have not yet been modernized, provided that the instructions in this document are observed.

This type of robust database does not contain a data or process authorization function to establish whether the user may modify the data concerned. Moreover, there is no access check in relation to the visibility of the data in the database to establish whether the user may view the data concerned.

Multiple 'user groups'

The application has to support multiple types of users in such a manner that the robustness of the modernized parts of the database cannot be compromised.

We distinguish the following types of users and processes:

1. **Interactive standard application users**
   These are the users who perform their activities via the screens.
2. **Standard batch processes**
   These are processes realized as reference software that run in the background and can be 'requested' by the users.
3. **Interface users**
   These are essentially indirect users who interact with the database via a synchronous or asynchronous customized interface (this is only permitted in the modernized parts of the database).

   This may or may not take place via the API/Service Layer.
4. **Customization users**
   Customized software added to the database structure (only permitted for the modernized parts of the structure) can in many cases be used to modify data directly. In effect there is no real difference with respect to interface users, although interface users will not usually perform their activities via a direct database account. In principle, customization activities are usually performed next to the API/Service Layer.

In actual fact, this distinction in terms of types of users and processes is not yet of any real benefit. The reason for this distinction will only become clear when these user groups are examined from a more technical point of view.

Identification, staff and accounts

Users are indicated as staff. Only staff can and may make modifications using the screens. A staff user must be registered within the application with an application user definition (an 'officer' record) and must be active.

Function authorization in screens, etc. is also granted based on the condition that the user is a member of staff. This authorization is done for application user definitions.

The screens require that the user connects to the database using an Oracle database account linked to a unique member of staff, so associated with an active application user definition with the same name.

When a user submits a script request, the batch process concerned will log on using a generic Oracle account (usually Oracle/Unix account 'batch'), after which a check will be performed in the batch process to establish whether a registered member of staff submitted the request.

When a user logs on via a customized part of the system and wishes to perform modifications, they will also have to do so using an application user definition account, so with an account which as a member of staff has been registered in the application.

For interface users it may be the case that there is a generic Oracle account for the sake of optimization, which is used to log on to the database, while it may be desirable and even necessary that a member of staff registered be specified for modifications. Another option could be for each 'interface user' to log on via their own Oracle account.

All of these situations must be supported.

Custom code within OHI software

In release 2009.03 a first implementation is offered of dynamic pl/sql code that can be defined by the customer. This code can be called within certain standard processes of the application.

For this code these restrictions apply:

1. No DDL is allowed.

2. It is only allowed to *query* data from the database (so only 'select statements' are allowed, the update, delete and insert DML statements are not allowed) except for when the pl/sql definition allows DML.

3. It is not allowed to lock any data when DML is not allowed.

4. It is not allowed to change any package states (i.e. variables within a package) of standard OHI packages.

5. The code should be very efficient in order to prevent noticeable delays and a decreased response time (when performance problems are caused by this code a logged incident will be marked as caused by customer).

6. It is in no way at all allowed to circumvent business rules or authorization rules in the application.

7. Database object access is restricted to the standard (!) object access rights implemented for custom code and as granted likewise to the role OZG_ROL_DIRECT (described below). It is strictly prohibited to grant any additional object privileges or system privileges that provide generic object access (dynamic code is executed through a special account which may only receive the standard OHI object privileges and privileges on custom code objects).

8. Transactions may not be influenced (so no explicit rollback, savepoints, autonomous transactions or whatsoever may be implemented when DML is allowed).

These checks will be enforced where possible and may change in strictness over different application releases. So when you do not follow these rules it may be that in a future release your code will no longer work.

Additional rules will be defined here based on experiences with this functionality.

Non-OHI software

Interface and customized software can in some cases consist of database objects (PL/SQL packages, procedures, tables, views, etc.) incorporated into the database in which the OHI database structure was created. While this is not permitted using the same framework (OHI schema owner account) used to create the objects, a different account (custom code schema) may be used. Moreover, in this custom code direct references may be made to specific (i.e. not all!) OHI objects as long as they are granted through the standard OHI provided granting routines.

Naturally, this situation must be supported.

Beware, when custom code objects in a custom code owner account need to be used in 'translation views', views in OHI that can be defined on a custom code definition, it is important these custom code objects are granted in the correct way. They need to be granted to the OHI owner account including the 'with grant option'. This is required so these views can be granted again.

For the 'OHI system views' (ending on 'SVW') the same applies.

As such views are granted to the internal OHI application role with all DML privileges (insert, update, delete), next to the select privilege, custom code objects used in these views need to be granted to the OHI object owner with these privileges with the 'with grant option'.

Points of attention

The above-mentioned points mean that there must be a grant structure that complies with all of the requirements without in any way jeopardizing the robustness of the application.

For the regular screen users, it will be sufficient if all database objects are granted to a single role, and each Oracle account that must be able to use the screens are able to activate this role. These screens will be 'familiar', as they are part of the reference software. Measures must also be taken to ensure that the users can only query and modify the data via the screens. In order to ensure compatibility with 'old' code, the 'grants' for the user interface users provide extensive rights, which essentially facilitate every type of modification. This includes modifications that cannot be checked by the database side of the application and should normally really not be permitted.

For interface and customized software and users we want to utilize a rights structure that prevents compromising with the robustness layer. Consequently, a much more limited, robust 'grant' structure is required for this purpose.

Nevertheless, the problem is that certain users (staff) may want to use the database in a variety of ways (via the regular user interface, but also via customized or other applications that exchange modifications with the OHI application), in which case we will have to proportionally enable use of another rights structure.

Solution

Recognition of multiple roles and their 'grants' makes it possible to use different rights depending on the purpose.

Consequently, the following roles are used and are mandatory in the database:

1. `OHI_ROLE_ALL`
2. `OZG_ROL`
3. `OZG_ROL_BATCH`
4. `OZG_ROL_SELECT`
5. `OZG_ROL_DIRECT`

OHI_ROLE_ALL and OZG_ROL

All OHI object rights are assigned to the `OHI_ROLE_ALL` role using the `OZGGRANTS.ins` script. The `OHI_ROLE_ALL` role is granted to the application role `OZG_ROL`. This two level role grant mechanism is used as OZG_ROL is a secure application role which is limited in use.

This OHI_ROLE_ALL and OZG_ROL role may *not be granted to any account in the database*, with no exceptions, even not the OHI *batch scheduler* account, for which role OZG_ROL_BATCH is dedicated.

The OZG_ROL role is (only) dynamically activated when a user logs on via the OHI screens. Consequently, this is *not* a default user role, which prevents the user from performing modifications on the OHI data using other tools (e.g. SQL*Plus or SQL Developer).

Users *cannot* activate the OZG_ROL role *themselves* using the commands "`SET ROLE`" or "`dbms_session.set_role`". The role can only be activated using the `ALG_SECURITY_PCK` package, which contains logic for checking whether the role is created using a supported (user) interface. Checks are also performed to establish whether a registered member of the OHI staff is using the package.

This is facilitated by means of the 'public granting' of a small number of objects. There are two packages and tables for which public execution and select rights are granted.

The following privileges are granted/updated for the OHI_ROLE_ALL role:

- Select, insert, update and delete privileges are granted for all tables and their associated 1 to 1 tables ('translation' and 'translated' views). The modification logging 'shadow' tables and external tables form exceptions to this rule, as only select privileges are granted for these tables.

- Select privileges are granted for all views, as well as the sequences.

- Execution rights are granted for all stored PL/SQL objects.

- Additional insert, update and delete rights are granted for all views not directly dependent on the DUAL table.

All objects whose names begin with the letters 'API' (the 'API objects') form a general exception to the above-mentioned rules.

For the rest, the above only occurs for the objects whose names begin with a recognized three-letter subsystem acronym, the exception being the `CG$ERRORS` package.

OZG_ROL_BATCH

This role must *not be granted to any account in the database*, with exception of the OHI *batch scheduler* account, to which this role *must be granted*. All privileges granted to OHI_ROLE_ALL are granted to OZG_ROL_BATCH, enabling the batch account to perform its requested mutations. On installing a patch, in one of the installation steps, the following checks are performed:

- The existence of OZG_ROL_BATCH;

- It being granted to the batch account;

- It not being granted to any other account.

OZG_ROL_SELECT

The selection rights to the 'selectable OHI objects' are granted to the `OZG_ROL_SELECT` role using the `OZGGRANTS.ins` script. This includes all tables, views and sequences whose names begin with a recognized three-letter acronym and not 'API'.

This role therefore gives staff the opportunity to perform selections of data outside of the user interface.

This role can and may be granted to an interface and/or users of customized applications who only require, or are only allowed to have query rights. Of course privacy regulations should be adhered to when granting this role.

OZG_ROL_DIRECT

Selection rights for all tables and modification rights for tables that are robust are granted to `OZG_ROL_DIRECT` using the `OZGGRANTS.ins` script. With regard to modification rights for these tables, column-level inserts and 'update grants' are used to prevent unauthorized column inserts and updates. The table and functional API objects are also granted to this role. Other database objects are therefore not (!) granted in order to prevent compromising with the robustness layer.

This role can be granted to interface and/or customization users.

When these *'direct access grants'* must be allocated directly to an account, the `OZG_DIRECT.grt.<ENV>` script (e.g. `OZG_DIRECT.grt.prod`) must be used. This script is created (in the `$OZG_BASE` directory) every time `OZGGRANTS.ins` is run.

This can be necessary if a customized owner account is created with customized stored procedures, functions or packages that use the objects. In such cases, 'direct' grants are required, as this type of stored code cannot be created based on 'volatile grants' that only are present when a role is active, which is not the case when a user is logged out, for example.

The script should typically run as the OHI object owner, using sqlplus. To follow what is done enable serveroutput before calling the script. The script will ask for values for 2 variables, GRANTEE and GRANT_OPTION. The first one is obvious, typically the name of a custom code owner account should be passed. In case custom views may be created based on OHI tables or views and these views need to be granted again, you should specify 'with grant option' for the GRANT_OPTION parameter.

When you have run the script without specifying a value for the GRANT_OPTION parameter and later on you do need the grant option privileges, please first revoke the grants from the custom code owner account before calling the script again.

An example for using this in sqlplus:

```
set serveroutput on
start $OZG_BASE/OZG_DIRECT.grt.<ENV> SVS_OWNER3 "with grant option"
```

A more detailed description of the rights granted:

- Select privileges for all tables whose names begin with API.
- Execution privileges for all packages whose names begin with API.
- Execution rights supplementary to objects used in indexes and some general objects.
- Select privileges like the privileges granted to `OZG_ROL_SELECT` role for all of the remaining tables, views and sequences.
- Delete grants, column-level inserts and 'column-level update grants' to all regular application tables (and associated 1 to 1 views, as mentioned before) modernized in line with a 'robust' structure. The 'column-level grants' help prevent columns that may no longer be modified as the result of an

application from being modified (in certain cases the column may be modified as the result of business rules). When such columns are still assigned a (modified) value via the API (table), the value is ignored.

Installation & migration

Installation

See the `OZGI001S.sql` script for instructions on how to create the above-mentioned four roles.

Migration

When an environment is not utilizing the *** ORACLE-REQUIRED *** secured role `OZG_ROL` and wishes to activate this role, the role must be modified as follows under SYS:

```
  alter role ozg_rol identified using <OHI Back Office
owner>.alg_set_gui_role_prc;
```

e.g.

```
  alter role ozg_rol identified using ozg_owner.alg_set_gui_role_prc;
```

The `OZG_ROL` role must subsequently be revoked by means of a revocation for all database accounts.