Oracle Health Insurance Back Office

Object Authorization within

Oracle Health Insurance Back Office

Version 1.25

Part number: G49637-01

March 25, 2026

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.
10.15.3.0.0	1.9	• Some minor textual changes and addition of a list of supplementary granted pl/sql execute object privileges to a custom code role or schema.
10.16.1.0.0	1.10	• Small change in paragraph that describes how to create the actual list of supplementary granted pl/sql objects.
10.16.2.0.0	1.11	• Parameters of the OZG_DIRECT.grt script have changed. • Litst with supplementary granted (non standard granted) pl/sql objects is updated.
10.17.1.0.0	1.12	• Updated the paragraph containing instructions for granting privileges on custom code objects.
10.17.2.0.0	1.13	• Added description for role OHI_ROLE_EXTRACT and updated some text because of the VPD implementation.
10.18.1.0.0	1.14	• Changed the usage description of script OZG_DIRECT.grt. • Updated list with additional granted objects • Exception for VPD related tables is documented
10.18.2.0.0	1.15	• Republished with different part nr.
10.19.1.0.0	1.16	• No changes. Republished with different part nr.
10.19.2.0.0	1.17	• No changes, republished with different part nr.
10.20.1.0.0	1.18	wChanged to document the new role OHI_ROLE_BATCH. Some textual improvements in the rest of the document.
10.21.1.0.0	1.19	• No changes, republished with new part number.
10.22.1.0.0	1.20	• No changes, republished with new part number.
10.23.1.0.0	1.21	• No changes, republished with new part number.
10.24.1.0.0	1.22	• No changes, republished with new part number.
10.25.1.0.0	1.23	• No changes, republished with new part number.
10.26.1.0.0	1.24	• No changes, republished with new part number.
10.26.3.0.0	1.25	• Replaced ozg_init.env with oraset

Contents

Introduction

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

This is based on the principle that the server component of the application must ultimately be fully robust and not permit any corrupting modifications (modifications that do not adhere to the business rules or authorized changes).

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.

This under the assumption and strict requirement that database system privileges are well maintained by the database administrators and they never allow any account to circumvent this mechanism.

Robust database

One of the main ideas behind the 'modernization' of the application, as implemented during one of the largest application revisions ever, a number of 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, main goal was the database structure to be 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, in whatever way, to implement modifications (!) that do not comply with the integrated business rules and standard object privileges using regular received privileges.

2) This is regardless of the manner in which the database is accessed, as the check is executed directly on top of the data within the database. As finally changes are always implemented through SQL and all granted SQL changes are validated within the database itself this is independent of the way the database is accessed as long as the used database accounts have only 'regular' privileges.

So 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 yet contain a data or process authorization function to establish whether a specific 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.

An exception to this are the columns that have been identified as 'sensitive' and may not be seen by all users for records for specific natural persons. For these columns in these records a technical solution has been implemented with the use of Oracle Virtual Private Database technology, in short VPD. In the first implementation mainly 'person identifying' columns fall in this category (data of birth, social security number, name fields, address, email address, phone nrs, etc.). This functionality is implemented for a limited set of 'sensitive' tables for which this data visibility access rules apply.

Also for this limitation on data access an approach has been implemented which cannot be circumvented when identified functional users do not have the privilege to see values of sensitive columns in these restricted records.

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 typically 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 / accounts**
   These are essentially indirect users or accounts who interact with the database via a synchronous or asynchronous customized interface (this is only permitted in the modernized parts of the database). These interface users usually use a generic functional user identity as the 'external user' of the interface may not be known to OHI Back Office. But when the external user may be a known OHI Back Office user there are functions that support specifying the identity of that user in order to have that API or SQL function act with that identity.

   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. Typically that software may be stored in a custom software account and that account should not be used to act as a user of OHI Back Office.
   In effect there is no real difference with respect to common unknown or known and identified interface users, although interface users will not usually perform their activities via a direct database account but through for example web services using a connection pool.

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

Regular OHI Back Office 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. Normally these users are persons and each person has a personal user account.

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 active 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 technically 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/specifying an application user definition account, so with an account which has been registered in the application as an active member of staff.

For indirect 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 specific member of staff be specified for specific modifications. Another option could be for each 'interface user' to log on via their own Oracle account that uses its own member of staff (potential identical to the used account) if interface users are in fact staff users.

All of these situations must be supported.

Custom pl/sql code within OHI software

In release 2009.03.0.0 (10.9.3.0.0) a first implementation was 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 (is not 'query only').
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 that may only be used by OHI code internally (for the real internal packages this is prevented because they are not accessible by custom dynamic pl/sql code).
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 which may induce additional costs).
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 with name OHI_DPS_USER which may only receive the standard OHI object privileges and privileges for custom code accessible objects).

8. Transactions may not be influenced. So no explicit rollback, savepoints, commits, autonomous transactions or whatsoever may be implemented when DML is allowed. This will be prevented and result in non working custom code, by throwing runtime errors, when not obeyed.

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. In fact, there is a risk that application stability is negatively impacted and support may be limited.

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 same database as in which the OHI database structure was created. While this is not permitted using the same framework (OHI schema owner account and view owner account and accompanying standard accounts for executing dynamic pl/sql code and batch processes) used to create and use the OHI objects, a different account (custom code schema) may be used. Moreover, in these custom code objects 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' or 'system 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 dynamic code account OHI_DPS_USER, probably with grant option (typically this applies to custom code tables, views and stand-alone or packaged functions).

The 'with grant option' might be needed when granting privileges on custom code views used in 'OHI system views' or 'OHI financial translation views', in the situation that custom roles or custom code owners have received privileges on OHI objects with grant option. In such a situation also the mentioned views will be granted with grant option to these custom roles or custom code owners which will only succeed when potential underlying views from a different owner have been granted with grant option.

When custom code objects are accessed in dynamic OHI pl/sql code, privileges on these objects als need to be granted to the OHI dynamic pl/sql user (OHI_DPS_USER).

There might still some very limited situations in OHI where custom code objects can be accessed from within OHI code without using the OHI dynamic pl/sql user. For this to work correct, privileges need to be granted to the OHI schema owner as well as the role OHI_ROLE_ALL.

When in doubt where code is used grant privileges on the custom code objects to the OHI schema owner, view owner, DPS user as well as to the role OHI_ROLE_ALL.

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 (user interface) 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 is able to activate this role. These screens will be 'familiar' and 'trusted', 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 'older' screen based 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 normally are not permitted.

For interface and customized software and users we want to utilize a privileges 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 it is necessary to proportionally enable use of the allowed privileges structure.

Implementation

Recognition of multiple roles and their 'grants' makes it possible to use different privileges 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_SELECT`
4. `OZG_ROL_DIRECT`
5. `OHI_ROLE_EXTRACT`
6. `OHI_ROLE_BATCH`

OHI_ROLE_ALL and OZG_ROL

All OHI object privileges 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 restricted 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 OHI_ROLE_BATCH is dedicated.

The OZG_ROL role is (only) dynamically activated when a user logs on via the OHI Oracle Forms based user interface. 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) when direct access of the database is attempted and possible.

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 running on a known host. 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 OHI objects with very limited privileges. 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 'to English translated' views). There are some exceptions:
    - The modification logging 'shadow' tables and external tables form exceptions to this rule, as only select privileges are granted for these tables.
    - For the 'sensitive data' containing tables, protected through a VPD implementation, only the Select privilege is granted to solely the OHI_ROLE_EXTRACT role. All other access is redirected to the protecting view layer on top of these tables, no form of direct access is granted.
- Select privileges are granted for all views within the dedicated view owner account OHI_VIEW_OWNER and the OHI_DPS_USER account, as well as the sequences as present within the table owner account.
- Execution rights are granted for all stored PL/SQL objects.
- Additional insert, update and delete rights are granted for all views within the OHI_VIEW_OWNER account and not directly being dependent on the DUAL table.

For the rest, the above only occurs for the objects whose names begin with a recognized/known set of three-letter subsystem acronyms.

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'. Tables which have a data access authorization view are not granted, instead the view on top of such tables is granted.

This role therefore gives staff the opportunity to perform selections of all data outside of the user interface, if they receive this role.

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 so be sure this role is very limited granted.

OZG_ROL_DIRECT

Selection rights for all directly accessible views, tables or table access replacing views 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 or views, 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, typically when stored pl/sql code objects like packages, etc. have to be created, the `OZG_DIRECT.grt` script (e.g. `OZG_DIRECT.grt`) must be used. This script is created in the `$OZG_BASE` directory every time `OZGGRANTS.ins` is run (which is run during OHIPATCH step 120).

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 be run while connected as the OHI object owner, using sqlplus, connected through the <ohienv>_install wallet entry or by providing the username and password (this is a mandatory requirement for a successful run of the script).

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 'Y' 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:

```
connect /@<ohienv>_install
set serveroutput on
start $OZG_BASE/OZG_DIRECT.grt SVS_OWNER_FIN Y
```

Output of the script will be spooled to a file named OZG_DIRECT.grt.<env> where <env> is determined by the value set for $TWO_TASK, usually set by running oraset.

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, SVL (can be granted separately) or DOM.
- Supplementary execution privileges for objects used in function based indexes and some general use 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 insert' 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 (so 'sensitive data' containing tables are excluded). The 'column-level' grants prevent columns that may not be modified are modified by any interfacing application. While in certain cases the columns may be modified by OHI itself as the result of business rules. When such columns are still assigned a (modified) value via the corresponding API (table), the value is ignored.

The supplementary objects which are referenced in the third bullet are listed below:

API, Domain (DOM) and Service Layer (SVL) objects are granted execute privileges by default.

Object type	Object name	Allowed grant(s)
FUNCTION	ALG_EET_INDEX_EDE	EXECUTE
FUNCTION	ALG_EET_INDEX_EET	EXECUTE
FUNCTION	FSA_VDG_INDEX_COUR	EXECUTE
FUNCTION	FSA_VDG_INDEX_SALDRN	EXECUTE
FUNCTION	FSA_VDG_INDEX_TUP_ST	EXECUTE
FUNCTION	FSA_VDG_INDEX_VERREK	EXECUTE
FUNCTION	FSA_VPG_INDEX_COUR	EXECUTE
FUNCTION	FSA_VPG_INDEX_SALDRN	EXECUTE
FUNCTION	FSA_VPG_INDEX_TUP_ST	EXECUTE
FUNCTION	GEB_DCR_INDEX_GRP_CODE_STATUS	EXECUTE
FUNCTION	GEB_DED_INDEX_MER_CODE_SPEC	EXECUTE
FUNCTION	GEB_DED_IND_MER_CODE_SPEC_DERI	EXECUTE
FUNCTION	GEB_DRC_INDEX_KENMERK	EXECUTE
FUNCTION	GEB_LHL_INDEX_AANL	EXECUTE
FUNCTION	INT_PBT_INDEX_BOEKINGSNOTA	EXECUTE
FUNCTION	INT_PBT_INDEX_BORDEREL	EXECUTE
FUNCTION	RBH_ERK_INDEX_REK	EXECUTE
FUNCTION	RBH_REL_INDEX_CONCAT_NAAM	EXECUTE
FUNCTION	RBH_REL_INDEX_STATUS	EXECUTE
FUNCTION	REF_DPE_INDEX_DBC	EXECUTE
FUNCTION	REF_EWE_INDEX_CHAR	EXECUTE
FUNCTION	REF_EWE_INDEX_EGE	EXECUTE
FUNCTION	REF_EWE_INDEX_NUM	EXECUTE
FUNCTION	VER_PMN_INDEX_REG	EXECUTE
FUNCTION	VER_PMU_IDX_BEEINDIGINGSBRIEF	EXECUTE
FUNCTION	VER_PMU_INDEX_MERK	EXECUTE
FUNCTION	VER_PMU_INDEX_POL_TE_VWK	EXECUTE
FUNCTION	VER_PMU_INDEX_TE_VWK	EXECUTE

FUNCTION	VER_POL_INDEX_STATUS	EXECUTE
FUNCTION	VER_PTL_INDEX_DATUM_INGANG	EXECUTE
FUNCTION	VER_PTL_INDEX_PLT_ID	EXECUTE
FUNCTION	VER_PTL_INDEX_TE_UPD	EXECUTE
PACKAGE	ALG_BATCH_PCK	EXECUTE
PACKAGE	ALG_BOP_PCK	EXECUTE
PACKAGE	ALG_BOP_S%	EXECUTE
PACKAGE	ALG_BOP_U%	EXECUTE
PACKAGE	ALG_BOP_W%	EXECUTE
PACKAGE	ALG_CONTEXT_PCK	EXECUTE
PACKAGE	ALG_CSV_PARSER_PCK	EXECUTE
PACKAGE	ALG_DPS_INSTALL_PCK	EXECUTE
TYPE	ALG_EDE_PAYLOAD_TP	EXECUTE
PACKAGE	ALG_EVENT_INTERFACE_PCK	EXECUTE
PACKAGE	ALG_LOGGING_PCK	EXECUTE
PACKAGE	ALG_MAF_PCK	EXECUTE
PACKAGE	ALG_OHI_SERVICES_PCK	EXECUTE
PACKAGE	ALG_OUTPUT_PCK	EXECUTE
PACKAGE	ALG_SAV_CAPI	EXECUTE
PACKAGE	ALG_SCRIPT_PCK	EXECUTE
PACKAGE	ALG_SUD_PCK	EXECUTE
PACKAGE	ALG_TAB_PCK	EXECUTE
PACKAGE	ALG_TRACE_PCK	EXECUTE
PACKAGE	ALG_TRANSLATE_PCK	EXECUTE
PACKAGE	CG$ALG_SCRIPT_AANVRAGEN	EXECUTE
PACKAGE	COM_DPS_INTERFACE_PCK	EXECUTE
PACKAGE	FIN_FPM_VARS_PCK	EXECUTE
PACKAGE	FSA_BUR_UTIL_PCK	EXECUTE
TYPE	RBH_ADRESVELDEN_TP	EXECUTE
PACKAGE	RBH_RPM_VARS_PCK	EXECUTE
PACKAGE	SDM_ADM_DRV_PCK	EXECUTE
TYPE	SIC_OBJECT_PCK	EXECUTE
PACKAGE	SYS_ALG_EI_PCK	EXECUTE
PACKAGE	SYS_BEP_PAD_PCK	EXECUTE
PACKAGE	SYS_DML_PCK	EXECUTE
PACKAGE	SYS_GEN_PCK	EXECUTE
PACKAGE	SYS_MESSAGE_HANDLING_PCK	EXECUTE
PACKAGE	VER_CONTEXT_PCK	EXECUTE
PACKAGE	VER_GBA_PCK	EXECUTE
PACKAGE	ZRG_AUR_PCK	EXECUTE
PACKAGE	ZRG_DML_INTERFACE_PCK	EXECUTE
PACKAGE	ZRG_FORMULE_BEDRAG	EXECUTE
PACKAGE	ZRG_ZPM_VARS_PCK	EXECUTE

This is a momentarily list.

For creating a current list please execute, a database account with direct granted privileges can run the stored procedure SYS_GEN_PCK.WRITE_HTML_GRANTABLES and provide a writable database directory name for the single parameter for this procedure.

The example below uses the database directory OZG_TMP:

```
begin
  sys_gen_pck.write_html_grantables('OZG_TMP') ;
end;
```

This will create a .zip file in that folder. The .zip file contains an HTML file with the current list of supplementary granted objects.

OHI_ROLE_EXTRACT

Selection rights directly for all tables, in order to circumvent the view layer which is implemented for some tables to potential hide sensitive data that may not be accessed, by the querying account.

BEWARE: This role should be used very carefully as it does bypass the data authorization for sensitive data containing columns implemented in the database by means of Virtual Private Database technology. Only use this when your organization allows to circumvent this authorization and regular select privileges cannot be used.

OHI_ROLE_BATCH

This role is only needed for the OHI Back Office account which technically executes the batch and background jobs. The role should only be granted to the OHI table owner account and the OHI batch executing account. Only a limited set of object privileges is granted to this role. The batch account typically is named BATCH but as the name is customer configurable it may differ.

When the batch scheduler needs to execute custom batches please grant execute privileges on the relevant objects to this role or directly to the batch account.

Installation & migration

Installation

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

Migration

When an environment is not correctly utilizing the *** OHI-REQUIRED *** secure application role `OZG_ROL`, for example due to a migration action, 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.