ORACLE
INSURANCE

Oracle Health Insurance Back Office

Data Management Technical Reference

version 3.12

Part number: F75883-01

June 30, 2023

CHANGE HISTORY

Release	Version	Changes
10.15.1.0.0	1.4	• Changes for database 12C • Small textual changes
10.15.3.0.0	1.5	• Added remark for VPD and the created database user for SS/DM • Added advise for parameter parallel_degree_level for subset export as well as disabling archive log mode
10.16.1.0.0	1.6	• Added remark about deferred segment creation • Added OHI Data Marts support
10.16.2.0.0	1.7	• Added changes for OEM 13C • Added work-around for subset definition import bug 19828552
10.16.2.2.0	1.8	• Pre-processing masking script SDM0001S.pl made compulsory (bug 25252355)
10.17.1.0.0	1.9	• Added paragraph 4.2 about masking flex fields
10.17.1.3.0	2.0	• Adjust paragraph 3.4, adding creating the subset rule definition file • Removed workaround for bug 23249155
10.17.2.0.0	2.1	• Updated prerequisites; OEM 13C R2 is now certified
10.18.1.0.0	2.2	• Added masking for OHI Data Marts • Updated prerequisites; Supported version of database plugin changed to 13.2.2.0
10.18.2.0.0	2.3	• No changes except part number.
10.18.2.2.0	2.4	• Slightly adjusted the recommendations for IO calibration
10.18.2.3.0	2.5	• Added pre masking step • Added 2 known issues
10.19.1.0.0	2.6	• Added option for configuration only subset • Some minor textual corrections and changes
10.19.1.1.0	2.7	• Updated prerequisites; OEM 13C R3 is now certified
10.19.2.0.0	2.8	• No changes, republished with new part number.
10.20.1.0.0	2.9	• No changes, republished.
10.20.3.0.0	3.0	• Adapted for impact of DB 19c certification
10.20.6.0.0	3.1	• Small improvements. • Added the step-by-step execution guide appendix.

Release	Version	Changes
10.20.8.0.0	3.2	• Added a note to state that the OHI BO Data Management data subsetting and masking processes cannot be executed on a database instance (PDB) with an active Oracle Database Vault installation. • Updated prerequisites; OEM 13C R4 is now certified
10.21.1.0.0	3.3	• No changes, republished with new part number.
10.21.4.0.0	3.4	• Data masking revised
10.21.5.0.0	3.5	• Modified Datamasking instructions • Added Appendix 2 “Masking a custom database schema”
10.22.1.0.0	3.6	• No changes, republished with new part number.
10.22.6.0.0	3.7	• Added policy number masking rule in Appendix 2 “Masking a custom database schema” • Added description in 4.1 to adjust the application from production to test • Added note in 4.3 and in 4.7 about the need to gather table statistics again
10.22.7.0.0	3.8	• Subsetting revised • Added claim number and invoice reference masking rule in Appendix 2 “Masking a custom database schema”
10.22.8.0.0	3.9	• Corrected sample masking script in paragraph 7.4 • Expanded description for subsetting cleanup
10.23.1.0.0	3.10	• Added custom masking rule for name prefix in appendix 2 • Exclude certain column types from custom masking • Allow repeated masking and subsetting • Added option to keep table statistics • Added option to compress OHI Back Office tables during subset or masking • Added option to make use of NOLOGGING to decrease the time needed to execute a run • New part number.
10.23.2.0.0	3.11	• Use of database parameter PARALLEL_FORCE_LOCAL added • Subset prepare steps modified
10.23.5.0.0	3.12	• Added additional checks and logging in 4.1

Contents

1 Introduction

Welcome to the technical reference guide for the Oracle Health Insurance Back Office suite (OHI BO and OHI DM) Data Management application. This reference guide is intended to help you in using the data subsetting and data masking functionality as provided by the OHI Data Management product for OHI Back Office and OHI Data Marts.

1.1 Subsetting and Masking

This guide describes how you can create a subset of a given OHI BO data store. The subset contains less data than the given data store but at the same time is still fully functional for the OHI BO application (that is: all data integrity and consistency rules are upheld). Using data subsetting you create smaller versions of, for instance, production-size OHI BO databases. These smaller databases can then be provisioned to development teams, test teams, or deployed in user-acceptance environments. Through data subsetting you can significantly reduce the total amount of disk storage required to support the various, and often many, non-production OHI BO environments in your organization.

There is no separate subset process for an OHI DM environment, instead a OHI DM subset environment for an OHI BO subset environment needs to be created through an initial full load of the OHI BO subset.

Next to data subsetting, this guide describes how OHI BO and/or OHI DM data stores can be masked. Due to privacy regulations, organizations are obliged to deal with privacy sensitive data in a secure manner. Production environments usually have stringent data access control and auditing mechanisms in place to ensure that only those who need to access privacy sensitive data can do so. Typically, those accessing the various non-production environments are not authorized to see privacy sensitive data or the data access control and auditing mechanisms are less stringent, or even absent, in these environments. With data masking you can mask (scramble, anonymize, pseudonymize) the privacy sensitive data elements in non-production OHI BO and/or OHI DM environments. Development and test teams that use these masked environments are therefore not able to see privacy sensitive data. The masked data store is still fully functional for the OHI BO and/or OHI DM application (that is: all data integrity and consistency rules are upheld).

Both subsetting and data masking work mainly on the so-call Fact tables (e.g., policies, claims). The Dimension tables (containing setup data) are left largely untouched.

The intended audience for this technical reference guide is the DBA group that administers the various OHI BO and OHI DM environments inside an organization. This technical reference guide contains four chapters.

1. *Introduction*
   The chapter you are currently reading. This chapter introduces you to the data subsetting and data masking packages of the OHI BO application.
2. *High Level Design*
   In chapter 2 you find a high-level design of the data subsetting and data masking processes as they have been designed to operate on a data store of the OHI BO application.
3. *Detailed Process*
   In chapters 3 and 4 you find a step-by-step description of the data subsetting and data masking processes. By following these steps, you can create a subset of the OHI BO application data store and mask this data store.

Subsetting and data masking are implemented in the database of OHI and do not require additional applications.

1.2 Prerequisites

The OHI BO Data Management data masking processes requires one of the following components:

- *A non-production OHI Back Office environment with the appropriate release*
- *A non-production OHI Datamarts environment with the appropriate release*

The OHI BO Data Management data subsetting processes requires the following components:

1. *A non-production OHI BO environment with the appropriate release*

2 High Level Design

This chapter contains a high-level overview of the data subsetting and data masking processes as they apply to the OHI BO application.

2.1 Subsetting Process

Subsetting is performed in-place. The subset process is run on a target database, which is then resized. Subsetting is performed by either executing a stored procedure or scheduling a database job in the target environment.

Figure 2.1 shows a high-level overview of this process.

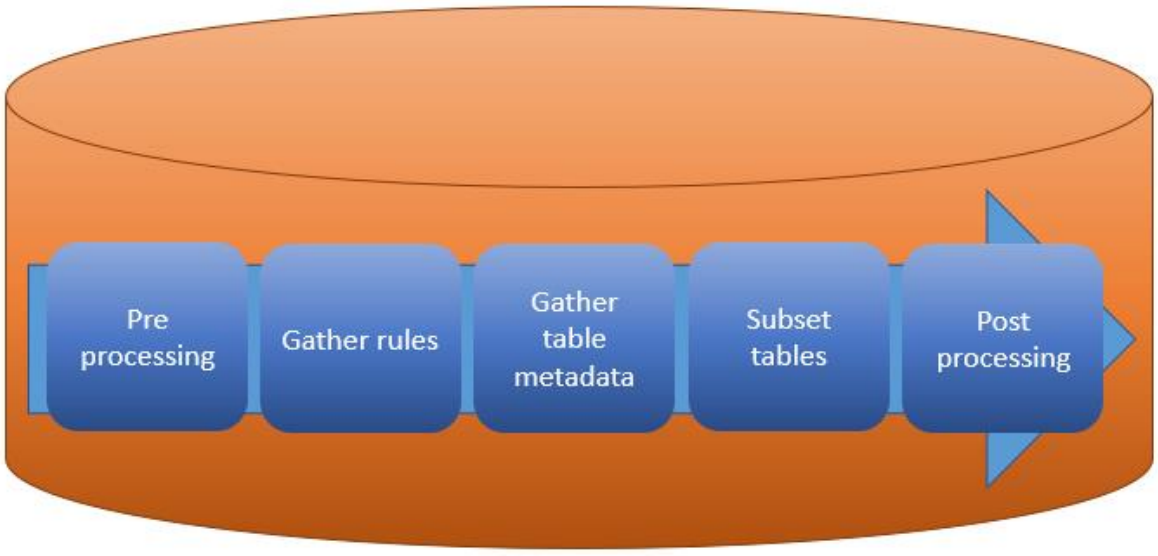


Figure 2.1: High level overview of the subset process.

To end up with a database that is smaller than the original (i.e., takes up less disk space), the DBA needs to rename the original tablespaces and create new tablespaces with the original prescribed names, using new, smaller, data files **before** starting the subsetting process. This means the database will temporarily require more disk space.

For the set of tables to be subsetted (based on the subset rules), the metadata is gathered and stored in a processing table. The metadata contains the definition of the table and its indexes, constraints and triggers.
During the subset process this data is processed per table.

The next step is to rename the original table and create a new table, based on the stored metadata of the original table (that now has been renamed). This newly created table is than filled with the data from the original table with the subset rule(s) applied to it.

Next the constraints and indexes are added to the tables and if all steps are successful the original (renamed) tables are dropped. This will free the original, renamed tablespaces and data files. These can then be dropped by the DBA.

Chapter 3 contains a detailed description of the subset process.

2.2 Data Masking Process

Data masking is performed in-place. The masking process is run on a target database, which is then masked. Masking is performed by either executing a stored procedure or scheduling a database job in the target environment.

Figure 2.2 shows a high-level overview of this process.

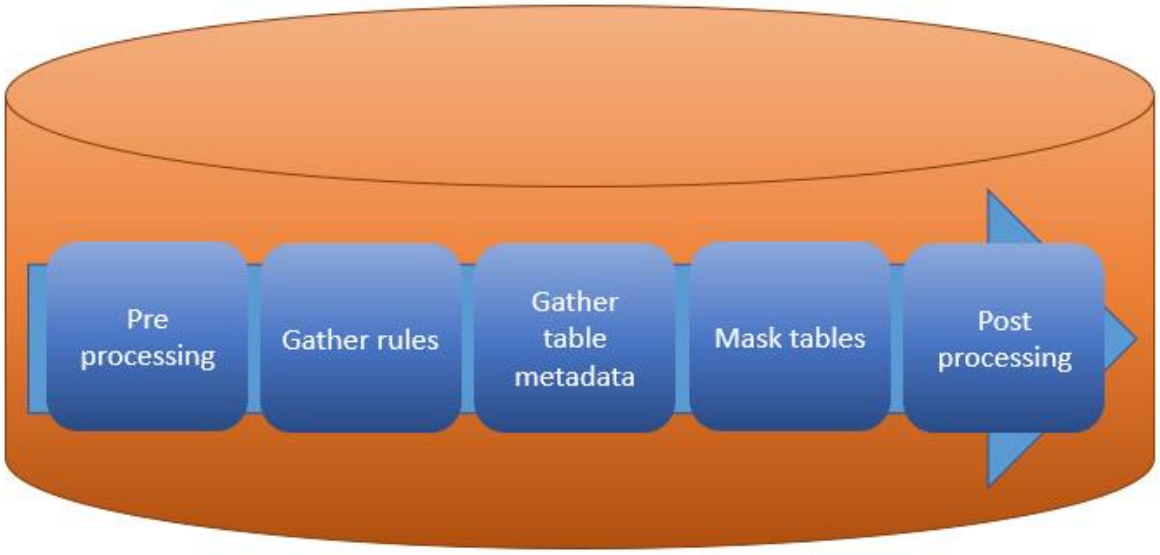


Figure 2.2: High level overview of masking process.

For the set of tables to be masked (based on the masking rules), the metadata is gathered and stored in a processing table. The metadata contains the definition of the table and its indexes, constraints and triggers.
During the masking process this data is processed per table.
If a tablespace name is provided the original table is first moved to this tablespace so no additional space is used in the original tablespace. The specified tablespace can be created temporarily and may be removed afterwards.

The next step is to rename the original table (either in the original tablespace or in the specified tablespace) and create a new table, in the original tablespace, based on the stored metadata of the original table (that has been renamed). This newly created table is than filled with the data from the original table with the masking rules applied to it. Next the constraints and indexes are added to the table and if all steps are successful the original (renamed) table is dropped, and the next table will be processed.

Chapter 4 contains a detailed description of the data masking process.

3 Detailed Process - Subsetting

The subset process executes "in-place" within an existing OHI Back Office environment.

3.1 Preparing to-be-subset Database (Target Database)

Obviously, subsetting should never be executed on a production environment. Therefore, it is required to adjust the value of the application environment parameter in the OHI system parameter record to “Test” instead of “Production” (alg_systeem_parameters.applicatie_omgeving).

In OHI Back Office this can be done by modifying the value in the window SYS1010F “General system parameter” on the tab “Other user settings”.

Before subsetting the target database, it is advisable to create a backup of the database. Depending on the size of the archive log destination file system, it may be advisable to disable archive logging during the subset process, and, when required, re-enable it afterwards and create a new initial full backup.

Recommended database configuration during subsetting

Before running the subset process on the target database, the database should be configured to maximize performance. The following parameters are recommended (and might deviate from parameters required for OHI Back Office at runtime):

- `parallel_max_servers` : default value (>1) or (for example) twice the number of available CPU threads, to enable parallel query during subsetting
- `pga_aggregate_target` : parallel_max_servers * 1GB

Checks and actions before subsetting

The subset program creates an empty copy of the original table for each table that is part of a subset rule and inserts the data based on the rule. This means that the subset program requires a certain amount of free space in the original tablespaces.

One of the goals of a subset is to reduce the size of a test environment. Just deleting data won’t release the storage used by the database. The existing tablespaces and underlying data files must be removed to free up disk space. Because the tablespace names are prescribed by OHI and checked during release installations, we need to end up with the original tablespace names.

That is the reason the subset process requires the following steps:

- **The DBA** renames all tablespaces containing the tables and indexes that occur in a subset rule. These tablespaces are:
  `OZG_FACT_FIN_IND`
  `OZG_FACT_FIN_TAB`
  `OZG_FACT_REL_IND`

```
OZG_FACT_REL_TAB
OZG_FACT_SYS_IND
OZG_FACT_SYS_TAB
OZG_FACT_ZRG_IND
OZG_FACT_ZRG_TAB
```

Arbitrary new names can be chosen, as these tablespaces need to be removed at the end of the subset process.

- **The DBA** creates a new set of (smaller) tablespaces with these original names:
  ```
  OZG_FACT_FIN_IND
  OZG_FACT_FIN_TAB
  OZG_FACT_REL_IND
  OZG_FACT_REL_TAB
  OZG_FACT_SYS_IND
  OZG_FACT_SYS_TAB
  OZG_FACT_ZRG_IND
  OZG_FACT_ZRG_TAB
  ```
- The subset process renames the existing full-size tables and indexes in the renamed tablespaces
- The subset process creates copies of the original tables and indexes with the original names in the new tablespaces. These contain the subset of the data. The subset process deletes the original (renamed) tables and indexes, freeing the original (renamed) tablespaces.
- **The DBA** drops the empty renamed tablespaces and underlying data files, freeing up the disk space

Note: the tables are “moved” to the prescribed tablespace, even if they were originally located in the wrong tablespace.

Sufficient quota for these new tablespaces should be granted to the application owner and the default tablespace should be set to OZG_FACT_ZRG_TAB for the application owner again.

To verify this the following procedure can be called:

```
set serveroutput on
exec sdm_driver_pck.pre_check(pi_action => 'S');
```

The output looks like the following:

```
Environment is not a production environment
Environment is not yet a subset.
Database is in restricted session mode
Batch scheduler is not active
VPD is not active
Default tablespace is: OZG_FACT_ZRG_TAB


INFO: Tablespace sizes
INFO: ----------------------------------------
INFO: OZG_DIM_FIN_IND  :      0,014 Gb
INFO: OZG_DIM_FIN_TAB  :      0,007 Gb
INFO: OZG_DIM_REL_IND  :      0,011 Gb
INFO: OZG_DIM_REL_TAB  :      0,007 Gb
INFO: OZG_DIM_SYS_IND  :      0,245 Gb
INFO: OZG_DIM_SYS_TAB  :      0,376 Gb
INFO: OZG_DIM_ZRG_IND  :      1,445 Gb
INFO: OZG_DIM_ZRG_TAB  :      1,045 Gb
INFO: OZG_FACT_FIN_IND :      1,133 Gb
INFO: OZG_FACT_FIN_TAB :      1,167 Gb
INFO: OZG_FACT_REL_IND :      0,885 Gb
INFO: OZG_FACT_REL_TAB :      0,622 Gb
INFO: OZG_FACT_SYS_IND :      1,485 Gb
```

```
INFO: OZG_FACT_SYS_TAB :       6,486 Gb
INFO: OZG_FACT_ZRG_IND :       2,057 Gb
INFO: OZG_FACT_ZRG_TAB :       1,634 Gb
INFO: OZG_LOG_IND      :       0,007 Gb
INFO: OZG_LOG_TAB      :       0,102 Gb
INFO: <**total**>      :      18,726 Gb
```

For performance reasons it is best to de-activate all maintenance background jobs to prevent for example a parallel statistics gathering job uses unnecessary resources that delay the subset process.

Note: Please deactivate Virtual Private Database (VPD) when this is activated in the OHI Back Office schema.

NOLOGGING and FORCE LOGGING

To decrease the time needed to create a subset the subset tooling uses the NOLOGGING option to insert the rows in the table and create subsequent indexes on this table. To allow the use of this NOLOGGING option the database should not should not be running in FORCE LOGGING mode. When using the NOLOGGING option a full back up has to be made (before and) after the subset process as a recovery without the required archive- and redo log files is not possible.

PARALLEL_FORCE_LOCAL

When the subset is run on a RAC environment with multiple instances available, the database parameter PARALLEL_FORCE_LOCAL can be set to FALSE. In this way the queries used by the subset can make use of the available instances instead of restricting execution to one instance. The subset needs to be run on a database service which can use these multiple instances in this case.

Afterwards this parameter needs to be set to TRUE when using the OHI Back Office application again.

3.2 Running Subset on Target Database

To start the subset process a database job can be created. To do this execute the following command as the OHI database schema owner:

```
begin
  sdm_driver_pck.job
  ( pi_action                  => 'S'
  , pi_masking_seed            => null
  , pi_masking_tablespace      => null
  , pi_subset_policy_selection => <The name of the policy selection>
  , pi_job_class               => 'DEFAULT_JOB_CLASS'
  , pi_force_query             => 'N'
  , pi_force_dml               => 'Y'
  , pi_force_ddl               => 'Y'
  , pi_parallel_policy_manual  => 'Y'
  , pi_gather_stats            => 'Y'
  , pi_compress_bo             => 'N'
  );
end;
/
```

This will start a database scheduler job with the name OHI_SUBSET and run the SDM_DRIVER_PCK.SUBSET procedure.
It is also possible to start the subset procedure directly in the calling session, but this will require you to keep the executing SQL*Plus session open and alive.

The parameters do have the following impact/effect during the subset process:

- PI_ACTION: S for Subset
- PI_MASKING_SEED: not used for subsetting
- PI_MASKING_TABLESPACE: not used for subsetting
- PI_SUBSET_POLICY_SELECTION: The name of the policy selection that act as the basis for the subset to be created. See appendix 1 for more explanation.
- PI_JOB_CLASS: The scheduler job class this job will use as a resource group.
- PI_FORCE_QUERY: when Y(es) the session is set to FORCE parallel query, when N(o) the session is set to ENABLE parallel query
- PI_FORCE_DML: when Y(es) the session is set to FORCE parallel DML, when N(o) the session is set to ENABLE parallel DML
- PI_FORCE_DDL: when Y(es) the session is set to FORCE parallel DDL, when N(o) the session is set to ENABLE parallel DDL
- PI_PARALLEL_POLICY_MANUAL: when Y(es) the session is set to MANUAL for the parameter PARALLEL_DEGREE_POLICY, when N(o) the session is set to AUTO for the parameter PARALLEL_DEGREE_POLICY.
  Note: This parameter will not be set in an Autonomous Data Warehouse environment.
- PI_GATHER_STATS: When Y(es) table and index statistics will be gathered at the end of the subset run, when N(o) the gather statistics step is omitted in the subset run. A recent set of Statistics is necessary for the performance of the application during normal use.
  When K(eep) the original table and column statistics will be reused. This way a smaller database can act as if it was a full production database in regards to query execution.
  Note: Gathering of statistics will be executed with the same settings as in step 820 of OHIPATCH. The time limit of step 820 does not apply here.
- PI_COMPRESS_BO: when Y(es) the tables in the subset will be compressed using Oracle Advanced Compression if no compression is present. When N(o) is chosen no compression is applied nor removed from the tables in the subset.
  Note: An appropriate licence for Advanced Compression is required when using this option.

All parameters are provided with a default which is the OHI recommended setting.

These settings will make maximum use of the available resources. If it is not possible or if you don't want to use the maximum available resources, you can set these parameters to the other possible value.

3.3 Monitoring the subset run

There is no direct output during the subset run. However, information about the run is stored in a table.

The current action is shown in the first row in the following query which can be run under the OHI schema owner. The schema owner in the queries is the owner of the tables the subset is run for:

```
select mlg.*
from   sdm#log          mlg
where  mlg.schema_owner = '<schema owner>'
order by mlg.id desc;
```

After the initialization phase is done, the overall progress can be monitored by looking at the status of the tables with the following query:

```
select ssd.base_table
,      ssd.status
from   sdm#statement_data ssd
where  ssd.object_type    = 'TABLE'
and    ssd.usage_type     = 'SUBSET'
and    ssd.schema_owner   = '<schema owner>'
order by ssd.id;
```

Tables with status D(one) have been processed successfully.

3.4 Checking the outcome

After the subset job and/or procedure is finished a report can be created to check on the process and possible errors. To create the report the following script can be run under the subset schema owner

```
col spool_tijdstip new_value l_spool_tijdstip noprint
select to_char(sysdate,'YYYYMMDD_HH24_MI_SS') spool_tijdstip
from dual
/

set trimspool on
set trim on
set pages 0
set linesize 1000
set long 10000000
set longchunksize 10000000
set feedback off
set showmode off
set flush off
set verify off
spool SUBSET_REPORT_&l_spool_tijdstip..html

select * from table(sdm_driver_pck.report(pi_full => 'N'))
/
spool off
undefine l_spool_tijdstip
```

This will create an HTML file with the following information

- Provided parameter values
- Instance and session information
- Generic run information
- Top 20 long running actions
- Any errors that occurred

When the parameter pi_full is provided with the value ‘Y’ the report will provide a full report of all executed steps. This option can be requested by OHI Support to investigate issues.

3.5 Processing errors and restarting

If one or more errors occurred the process can be restarted by using the same parameters, after mitigating the cause of the error, e.g., by increasing a tablespace size.
In that case, the process will detect an incomplete previous run and only process the tables that are not yet successfully processed.

3.6 Cleaning up

The (renamed) tablespaces that held the original data (as described earlier) can be dropped now, to release the extra disk space used and decrease the size of the environment.

Note: make sure you check no objects are left in these tablespaces. The subset process has "moved" all OHI Back Office objects, but you may have custom objects (that should not be present) in these tablespaces. Use this query to detect remaining tables:

```
select * from dba_tables
where tablespace_name like '%PRESUBSET';
```

`Table CREATE$JAVA$LOB$TABLE` can be deleted.

Revert the changes made to the database parameters before using the database as an OHI Back Office runtime environment.

Note: For correct operation of the application, it is necessary to have an actual set of table and index statistics. If gathering statistics is omitted during the subset process (via the parameter PI_GATHER_STATS) these statistics should be gathered manually before using the application. Not having a recent set of statistics could result in a different experience in the performance of the application.

3.7 Repeated subsetting

It is possible to repeat subsetting on the same environment. This can be used to test the subsetting functionality of a newer release after installation on an environment that was subsetted before.

4 Detailed Process - Data Masking

The data masking process executes "in-place" within an existing OHI BO or OHI DM data store.

4.1 Preparing to-be-masked Database (Target Database)

Data masking can be performed on a full-size OHI BO or OHI DM data store, or on a subset OHI BO or OHI DM data store.
Note: A subset of an OHI DM data store can be obtained by performing an initial load from a subset OHI BO environment, see chapter 5. Typically, you would first mask that OHI BO data store before performing an initial load to construct the companion OHI DM data store. Masking an OHI DM data store is typically used for a full sized OHI DM environment.

Obviously, the masking process takes more time on full-size data stores. During masking, tables with sensitive columns are temporarily duplicated (this is further explained in Section 4.3). For this reason, it is necessary to check that tablespaces holding the tables either have enough free space available or are able to grow (auto extend) during the masking process. Upon completion of data masking a database (or tablespace) reorganization of some kind will have to be performed to reclaim the then remaining free space. See Section 4.3 (PI_MASKING_TABLESPACE) for a way to prevent having to execute such a reorganization.

Obviously, masking should never be executed on a production environment. Therefore, it is required to adjust the value of the application environment parameter in the OHI system parameter record to “Test” instead of “Production” (alg_systeem_parameters.applicatie_omgeving).

In OHI Back Office this can be done by modifying the value in the window SYS1010F “General system parameter” on the tab “Other user settings”.

In an OHI DataMarts the environment type can be set by executing the following statement as schema owner:

```
update alg_systeem_parameters
set applicatie_omgeving = 'T'
;
commit;
```

Before masking the target database, it is advisable to create a backup of the database. Depending on the size of the archive log destination file system, it may be advisable to disable archive logging during the masking process, and, when required, re-enable it afterwards and create a new initial full backup.

Recommended database configuration during masking

Before running the masking process on the target, it should be configured to maximize performance. The following parameters are recommended (and might deviate from parameters required for OHI Back Office or OHI Data Marts at runtime):

- `parallel_max_servers`: default value (>1) or (for example) twice the number of available CPU threads, to enable parallel query during masking
- `pga_aggregate_target`: parallel_max_servers * 1GB

Masking an Autonomous Data Warehouse

When masking is performed for OHI Data Marts on an ADW environment it is recommended to use the “high” connection service.

Checks before masking

The masking program processes one table at a time. It creates a copy of the original table and deletes that copy after the table has been masked successfully. This means that the masking program requires a certain amount of free space in the original tablespace or in the specified “temporary” tablespace. If the masking of a table fails, the copy is not removed (to support a quick retry after the cause of the failure has been addressed), and processing continues with the next table.

Therefore, the free space needed will be at least the size of the largest table, plus a reserve for any tables that fail to be processed successfully.

NOTE:

> When the masking is performed on a so-called snapshot clone, all tables that are part of the masking will be recreated and written. Therefore the required amount of disk storage will be at least the amount of all the tables and indexes in the masking set in the snapshot master. Masking a snapshot clone is therefore not efficient in terms of disk storage.

To get an indication of the minimum free space that needs to be available, a pre-check can be executed on the target database before the masking is started.
This check will check some environment settings and will provide a list of the top table in size, per tablespace that is part of the masking set. Also the total size used for the tables and indexes that are part of the masking process will be shown.

The pre-check can be executed by running the following command as schema owner in SQL*Plus:

```
set serveroutput on
exec sdm_driver_pck.pre_check(pi_action => 'M');
```

The output looks like the following:

```
Pre masking checks
INFO : Environment is not a production environment
INFO : Environment is not yet masked
INFO : Database is in restricted session mode
INFO : Batch scheduler is not active
INFO : Back Office parameter 'Default country code' is set to 'NL'
INFO : Value for parameter 'job_queue_processes' is set to 15 with a 'pga_aggregate_limit' of
10Gb
INFO : Largest table to be masked based on size per tablespace
INFO : ------------------------------------------------------------------
INFO : OZG_DIM_FIN_TAB : FSA_BEDRIJFSONDERDELEN         (     0,000 GB)
INFO : OZG_DIM_REL_TAB : RBH_BANKEN                     (     0,000 GB)
INFO : OZG_DIM_SYS_TAB : ALG_GROEPEN                    (     0,000 GB)
INFO : OZG_DIM_ZRG_TAB : ZAS_VERGOEDING_CATEGORIEEN     (     0,133 GB)
INFO : OZG_FACT_FIN_TAB: FSA_VORD_JOURN_REGELS          (     0,245 GB)
INFO : OZG_FACT_REL_TAB: RBH_RELATIES                   (     0,178 GB)
INFO : OZG_FACT_SYS_TAB: ALG#MELDINGEN                  (     0,017 GB)
INFO : OZG_FACT_ZRG_TAB: REF_EIGENSCHAP_WAARDEN         (     1,188 GB)
```

```
INFO : Total allocated size of tables to be masked:                3,378 GB
INFO : Total allocated size of indexes on tables to be masked:     4,690 GB
```

NOLOGGING and FORCE LOGGING

To decrease the time needed to mask an environment the masking tooling uses the NOLOGGING option to insert the rows in the table and create subsequent indexes on this table. To allow the use of this NOLOGGING option the database should not should not be running in FORCE LOGGING mode. When using the NOLOGGING option a full back up has to be made (before and) after the masking process as a recovery without the required archive- and redo log files is not possible.

PARALLEL_FORCE_LOCAL

When the subset is run on a RAC environment with multiple instances available the database parameter PARALLEL_FORCE_LOCAL can be set to FALSE. In this way the queries used by the masking process can make use of the available instances instead of restricting to one instance. The masking process needs to be run on a database service which can use these multiple instances in this case.

Afterwards this parameter needs to be set to TRUE when using the OHI Back Office application again.

JOB_QUEUE_PROCESSES

The database parameter JOB_QUEUE_PROCESSES should be set to at most 10 times the amount of GB set for the parameter PGA_AGGREGATE_LIMIT. If, for example, the pga_aggregate_limit is set to 10GB, the maximum for job_queue_processes is 100.

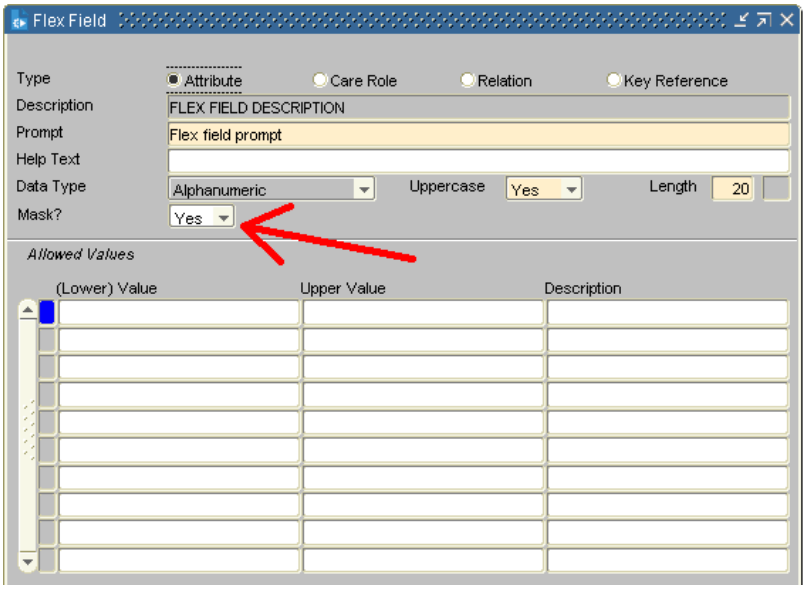
RESTRICTED SESSION MODE

Before the masking can be executed the database must be set in restricted session mode to prevent unwanted access to the schema. When masking a OHI Datamarts environment the OHI Back Office environment acting as the source environment for the OHI Datamarts environment should not be in restricted session mode.

4.2 Masking flex fields

Flex fields are flexible, like the name says, and your organization can determine what they are used for and what is stored in them. As such, it may be that certain flex fields contain sensitive information that should also be masked.

In the OHI BO application it is possible to indicate if the value for a specific flex field should be masked. In window ZRG7019F (Flex field) the indication “Mask?” can be set to “Yes” in order to mask the value for this flex field. If set to “No” (default value) masking will not take place. Masking the OHI DM application will also use this setup in OHI BO.



Note: The value of a flex field can be used to influence the logic of processes like the claims processing. Masking of these flex fields can interfere with this logic and may lead to unpredictable and/or undesirable results. Therefore, the advice is to mask only those flex fields that are actually privacy-sensitive.

4.3 Running Masking on Target Database

To start the masking process a database job can be created. To do this, execute the following command as the OHI database schema owner:

```
begin
  sdm_driver_pck.job
  ( pi_action                   => 'M'
  , pi_masking_seed             => <your 'secret' integer value>
  , pi_masking_tablespace       => 'SCRATCH_TBS'
  , pi_subset_policy_selection => null
  , pi_job_class                => 'DEFAULT_JOB_CLASS'
  , pi_force_query              => 'N'
  , pi_force_dml                => 'Y'
  , pi_force_ddl                => 'Y'
  , pi_parallel_policy_manual   => 'Y'
  , pi_gather_stats             => 'Y'
  , pi_compress_bo              => 'N'
  );
end;
/
```

This will start a database scheduler job with the name OHI_MASKING and run the SDM_DRIVER_PCK.MASK procedure.
It is also possible to start the mask procedure directly in the calling session, but this will require you to keep the executing SQL*Plus session open and alive.

The parameters do have the following impact/effect during the masking process:

- PI_ACTION: M for Masking

- PI_MASKING_SEED: this can have a value between 0 and 4294967295. This will randomize the masking output and prevent identical masked strings for identical source strings for different OHI Back Office customers.
  Note: If you want to mask an OHI Back Office environment and a OHI Data Marts environment that should be coupled together make sure the value of the seed is the same, so make sure to store it in a safe place.
- PI_MASKING_TABLESPACE: when provided each table to be masked will be moved to this tablespace before it is masked. For this an existing but empty tablespace should be provided. If no tablespace is provided the temporary table object is created in the same tablespace as the table to be masked. Providing a tablespace might increase the masking time, because the tables to be masked will first have to be moved to the chosen tablespace, but will have the benefit (after dropping this tablespace) that the size of the original tablespaces is not significantly increased after masking.
- PI_SUBSET_POLICY_SELECTION: not used for data masking
- PI_JOB_CLASS: The scheduler job class this job will use as a resource group. For an Autonomous Data Warehouse environment use the job class best fit for the resource-intensive masking job, e.g., 'HIGH'.
- PI_FORCE_QUERY: when Y(es) the session is set to FORCE parallel query, when N(o) the session is set to ENABLE parallel query
- PI_FORCE_DML: when Y(es) the session is set to FORCE parallel DML, when N(o) the session is set to ENABLE parallel DML
- PI_FORCE_DDL: when Y(es) the session is set to FORCE parallel DDL, when N(o) the session is set to ENABLE parallel DDL
- PI_PARALLEL_POLICY_MANUAL: when Y(es) the session is set to MANUAL for the parameter PARALLEL_DEGREE_POLICY, when N(o) the session is set to AUTO for the parameter PARALLEL_DEGREE_POLICY.
  Note: This parameter will not be set in an Autonomous Data Warehouse environment.
- PI_GATHER_STATS: When Y(es) table and index statistics will be gathered at the end of the masking run, when N(o) the gather statistics step is omitted in the masking run. A recent set of Statistics is necessary for the performance of the application.
  When K(eep) the original table and column statistics will be reused
  Note: Gathering of statistics will be executed with the same settings as in step 820 of OHIPATCH. The time limit of step 820 does not apply here.
- PI_COMPRESS_BO: when Y(es) the tables in the masking set will be compressed using Oracle Advanced Compression if no compression is present. When N(o) is chosen no compression is applied nor removed from the tables in the masking set. This option is only applicable when masking an OHI Back Office database.
  Note: An appropriate licence for Advanced Compression is required when using this option.

All parameters are provided with a default which is the OHI recommended setting, except for the tablespace name.

These settings will make maximum use of the available resources. If it is not possible or if you don't want to use the maximum available resources, you can set these parameters to the other possible value.

4.4 Monitoring the masking run

There is no direct output during the masking run. However, information about the run is stored in a table.

The current action is shown in the first row in the following query which can be run under the OHI schema owner. The schema owner in the queries is the owner of the tables that are (being) masked:

```
select mlg.*
from   sdm#log          mlg
where  mlg.schema_owner = '<schema owner>'
order by mlg.id desc;
```

After the initialization phase is done, the overall progress can be monitored by looking at the status of the tables with the following query:

```
select ssd.base_table
,      ssd.status
from   sdm#statement_data ssd
where  ssd.object_type    = 'TABLE'
and    ssd.usage_type     = 'MASKING'
and    ssd.schema_owner   = '<schema owner>'
order by ssd.id;
```

Tables with status D(one) have been processed successfully.

4.5 Checking the outcome

After the masking job and/or procedure is finished a report can be created to check on the process and possible errors. To create the report the following script can be run under the masked schema owner

```
col spool_tijdstip new_value l_spool_tijdstip noprint
select to_char(sysdate,'YYYYMMDD_HH24_MI_SS') spool_tijdstip
from dual
/

set trimspool on
set trim on
set pages 0
set linesize 1000
set long 10000000
set longchunksize 10000000
set feedback off
set showmode off
set flush off
set verify off
spool MASKING_REPORT_&l_spool_tijdstip..html

select * from table(sdm_driver_pck.report(pi_full => 'N'))
/
spool off
undefine l_spool_tijdstip
```

This will create an HTML file with the following information

- Provided parameter values
- Instance and session information

- Generic run information
- Top 20 long running actions
- Any errors that occurred

When the parameter pi_full is provided with the value 'Y' the report will provide a full report of all executed steps. This option can be requested by OHI Support to investigate issues.

4.6 Processing errors and restarting

If one or more errors occurred the process can be restarted by using the same parameters, after mitigating the cause of the error, e.g., by increasing a tablespace size.
In that case, the process will detect an incomplete previous run and only process the tables that are not yet successfully masked.

4.7 Cleaning up

If you supplied a value for PI_MASKING_TABLESPACE (to move the tables with sensitive columns to an empty tablespace, as described earlier) you can drop that tablespace now, to release the extra disk space used by the masking process.

Revert the changes made to the database parameters before using the database as an OHI Back Office or OHI Data Marts runtime environment.

Note: For correct operation of the application, it is necessary to have an actual set of table and index statistics. If gathering statistics is omitted during the masking process (via the parameter PI_GATHER_STATS) these statistics should be gathered manually before using the application. Not having a recent set of statistics could result in a different experience in the performance of the application.

4.8 Repeated data masking

It is possible to repeat data masking on the same environment. This can be used to test the data masking functionality of a newer release after installation on an environment that was masked before.

5 OHI Data Marts subset

OHI Data Marts (OHI DM) is certified to extract data from sub-setted and/or data-masked OHI Back Office (OHI BO) data stores.

The following types of OHI BO data stores are supported:

- subsetted
- subsetted and data-masked
- data-masked

From release 10.18.1.0.0 onwards, OHI DM can be data-masked itself as well. The process to mask OHI DM is described in chapter 4.

When a subset OHI DM environment is needed you should use the approach described in this chapter. Whether or not the source OHI BO environment is already masked determines whether the resulting OHI DM environment is masked. For creating a subset OHI DM environment an initial load is needed from the OHI BO subset environment.

When you need a full size masked OHI DM environment the masking process from the previous chapter can be applied.

Theoretically you can create a subset OHI BO environment and corresponding OHI DM environment and mask them both independently, but this is more time consuming and more error-prone than using a subset masked OHI BO environment to create the corresponding OHI DM environment.

5.1 Preparing OHI Data Marts data-store

An empty OHI DM data store is required to be able to load data from a subsetted and/or data-masked OHI BO data store into OHI DM. The SQL script *OBDRESET.sql* (available within *OZG_TEMPLATES.zip*) is provided to create an empty OHI DM data store by truncating all the necessary OHI DM tables. This script can be run using SQL*Plus while connected as the OBD_OWN account.

We strongly advise to create a clone from an existing OHI DM environment which is at the same OHI patch-level as the OHI BO data store and use this cloned environment to run the SQL script *OBDRESET.sql* against. Make sure the database link SRC_OPENZORG is referring to the correct OHI BO data store.

NOTE: This SQL script should never be used within a production environment!

5.2 Loading OHI Data Marts

Performing loads from a subsetted and/or data-masked OHI BO environment is identical to loading from a normal OHI BO environment. See the OHI Back Office online help for information on loading (topic ‘Loading OHI Data Marts’).

You can proceed with a full initial load from the OHI BO subset environment by not specifying up to which moment to load. Or you choose to first load older data up to a specific date and divide the work in this way, by using additional incremental loads for later periods to load.

6 Appendix 1: Define a policy selection

A policy selection within OHI Back Office is used to determine which rows should be included in the subset.

In this section the setup of the policy selection is explained in more detail.

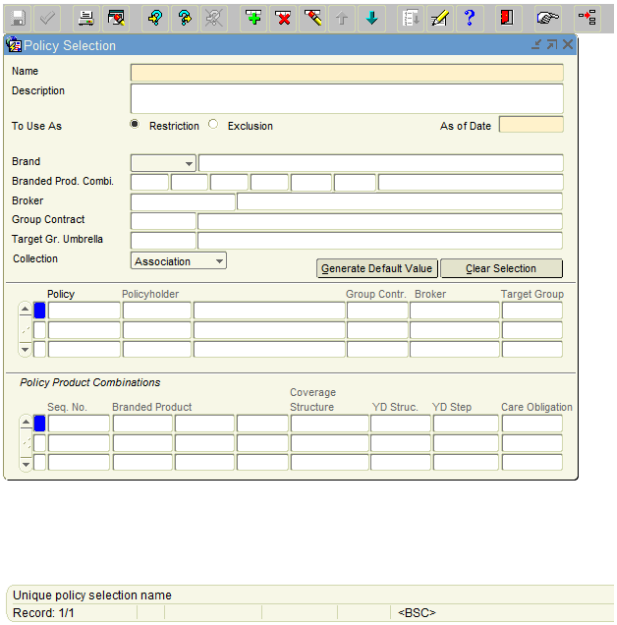
A policy selection is a set of policy numbers. Typically, you should determine a representative set of policies to be included in your subset, as this forms the base of the data that will be included in the subset. All policy related data, including relevant claims, will be incorporated in the subset. So, determining a well-considered policy selection is an important aspect in setting up your subset environment.

Note: Regression and integration tests may require certain types of data or cases to be present. Make sure to include these in your policy selection.

There are two ways you can create a policy selection:

1. Use the OHI Back Office Policy Selection window to setup a policy selection.
2. Use SQL*Plus to directly populate the underlying two tables of a policy selection.

The figure below shows the window ZRG2070F “Policy Selection”. You have to enter a name for the policy selection (this name will be input for one of the steps later on). The description is optional. All other items can be left empty or at their default value. In the second block you enter all required policy numbers.



The Policy Selection window.

The two underlying tables for the Policy Selection screen are depicted in Figure 3.2. They are:

- **VER_POLIS_SELECTIES**, and
- **VER_POL_PER_POS**.

It might be more convenient to use a SQL tool, such as SQL*Plus, to create the necessary rows manually in these tables.

Figure 3.2: Policy selection tables.

For example, the following two insert statements set up a policy selection named SAMPLE_OF_5_PERCENT that holds a random sample of 5% of the total number of policies available in the source database.

```
insert into ver_polis_selecties (naam) values('SAMPLE_OF_5_PERCENT');
insert into ver_pol_per_pos ( pos_id, pol_nr )
select ( select id from ver_polis_selecties where naam = 'SAMPLE_OF_5_PERCENT')
       , nr
from ver_polissen SAMPLE(5);
commit;
```

Of course, you can execute multiple `insert into ver_pol_per_pos` statements with different selections to add policies to the policy selection.

For running these statements, you need an account with (temporary) insert privileges on these tables unless you use the OHI BO table owner account, which is strongly discouraged given the security impact.

7 Appendix 2: Masking a custom database schema

This appendix provides a step-by-step guide to mask a custom database schema that contains data extracted from an OHI environment. This is done in the same way and with the same rules as for the OHI environment.

This guide is not meant to mask any database with any dataset. Only data from an OHI environment in a custom development schema can be masked, with the limitation that no additional masking rules other than those provided and used to mask an OHI environment can be used and are supported.

7.1 Creating the masking rule set

It is assumed that the custom development schema is in the same database as the OHI database schema and that the custom development schema has been granted (directly!!) all privileges that have been granted to the OZG_ROL_DIRECT role.

Note that is is not possible to mask hidden, virtual or identity columns.

Each table and/or column that should be masked should be registered via the procedure sdm_driver_pck.add_custom_rule.

This procedure has the following interface:

```
procedure add_custom_rule
( pi_schema_owner     in sdm#custom_schema_masking.schema_owner%type
, pi_table_name       in sdm#custom_schema_masking.table_name%type
, pi_column_name      in sdm#custom_schema_masking.column_name%type
, pi_masking_rule     in sdm#custom_schema_masking.masking_rule%type
, pi_col_gender       in sdm#custom_schema_masking.col_gender%type default null
, pi_col_first_name   in sdm#custom_schema_masking.col_first_name%type default null
, pi_col_last_name    in sdm#custom_schema_masking.col_last_name%type default null
, pi_col_country      in sdm#custom_schema_masking.col_country%type default null
, pi_col_street       in sdm#custom_schema_masking.col_street%type default null
, pi_col_city         in sdm#custom_schema_masking.col_city%type default null
, pi_col_postal_code  in sdm#custom_schema_masking.col_postal_code%type default null
, pi_col_house_number in sdm#custom_schema_masking.col_house_number%type default null
);
```

The parameters

- **PI_SCHEMA_OWNER**: The custom development database schema that holds the table that is to be masked. It is possible to register tables from multiple database schemas, but each masking run will only process tables from one schema and run from that schema.
- **PI_TABLE_NAME**: The table name of the table to be masked or truncated
- **PI_COLUMN_NAME**: The column name which should be masked; must be left empty when the table should be truncated
- **PI_MASKING_RULE**: The masking rule(*) that should be applied to the column or table
- **PI_COL_GENDER**: Column name (or expression) of the column containing the gender value
- **PI_COL_FIRST_NAME**: Column name (or expression) of the column containing the first name value
- **PI_COL_LAST_NAME**: Column name (or expression) of the column containing the last name value
- **PI_COL_COUNTRY**: Column name (or expression) of the column containing the country value

- **PI_COL_STREET**: Column name (or expression) of the column containing the street value
- **PI_COL_CITY**: Column name (or expression) of the column containing the city value
- **PI_COL_POSTAL_CODE**: Column name (or expression) of the column containing the postal code value
- **PI_COL_HOUSE_NUMBER**: Column name (or expression) of the column containing the house number value

(*) Available masking rules for the parameter PI_MASKING_RULE are:

- **TRUNCATE**: truncate the table, in other words all rows are removed leaving an empty table
- **NULL_VALUE**: the column value is removed
- **TRANSLATE**: replaces the content with dummy data depending on the datatype of the column.
- **TRANSLATE_UNIQUE**: replaces the content of a varchar column with dummy characters, but will make sure the content will keep its uniqueness
- **DATE**: adjust the date to the 1st, 11th or 21st of the month, can be used e.g., for a date of birth
- **REL_NR**: replaces the relation number with a new calculated relation number
- **POL_NR**: replaces the policy number with a new calculated policy number
- **DCR_NR**: replaces the claim number with a new calculated claim number
- **KENMERK**: replaces an invoice reference number with a new number
- **BSN**: replaces the existing social security number with a new calculated social security number
- **PHONE**_1: masks a phone number
- **PHONE**_2: masks a phone number with a different pattern
- **FAX**: masks a fax number
- **BANK**: replaces the bank account number with a new calculated bank account number (IBAN)
- **LASTNAME**: replaces the last name of a person with a new last name
- **PREFIX_NAME**: replaces the lastname prefix of a person with a new prefix
- **FIRSTNAME**: replaces the first name of a person with a new first name
- **FIRSTNAME_LETTER**: replaces the first letter of the first name of a person with a new first letter based on the new first name
- **EMAIL**: replaces the email of a person with a new email based on the new first and last name of that person
- **STREET**: replaces the street part of an address with a fake street
- **CITY**: replaces the city part of an address with a fake city
- **COUNTRY**: replaces the country part of an address with a fixed one if this is not the Dutch country code
- **HOUSENUMBER**: replaces the house number of an address
- **POSTAL_CODE**: replaces the postal code part of an address with a fake postal code
- **POSTAL_CODE_LETTER**: replaces the character part of a Dutch postal code with a fake character part
- **POSTAL_CODE_NUMBER**: replaces the numerical part of a Dutch postal code with a fake numerical part

7.2 The use of the PI_COL_* parameters

Masking is based on the data in the given row. Sometimes the name of the column itself is enough, sometimes additional data is required.
For instance a gender is needed to differentiate between male and female first names.
The table below shows which additional columns are needed for each masking rule. If the rule is not mentioned in this table, no additional columns are required.

Some of the required column parameters may seem unnecessary, e.g., a pi_col_postal_code for the column with/for the postal code, but these are required for the correct implementation.

Rule	Additional Parameters
FIRSTNAME	PI_COL_GENDER
FIRSTNAME_LETTER	PI_COL_GENDER, PI_COL_FIRST_NAME
PREFIX_NAME	PI_COL_LAST_NAME
EMAIL	PI_COL_GENDER, PI_COL_FIRST_NAME, PI_COL_LAST_NAME
STREET	PI_COL_COUNTRY, PI_COL_POSTAL_CODE, PI_COL_HOUSENUMBER If addresses outside of the Netherlands are also present in the table, also the following are required: PI_COL_STREET, PI_COL_CITY
CITY	PI_COL_COUNTRY, PI_COL_POSTAL_CODE, PI_COL_HOUSENUMBER If addresses outside of the Netherlands are also present in the table, also the following are required: PI_COL_STREET, PI_COL_CITY
COUNTRY	PI_COL_COUNTRY, PI_COL_POSTAL_CODE, PI_COL_HOUSENUMBER If addresses outside of the Netherlands are also present in the table, also the following are required: PI_COL_STREET, PI_COL_CITY
HOUSENUMBER	PI_COL_COUNTRY, PI_COL_POSTAL_CODE, PI_COL_HOUSENUMBER

	If addresses outside of the Netherlands are also present in the table, also the following are required: PI_COL_STREET, PI_COL_CITY
POSTAL_CODE	PI_COL_COUNTRY, PI_COL_POSTAL_CODE, PI_COL_HOUSENUMBER If addresses outside of the Netherlands are also present in the table, also the following are required: PI_COL_STREET, PI_COL_CITY
POSTAL_CODE_LETTER	PI_COL_COUNTRY, PI_COL_POSTAL_CODE, PI_COL_HOUSENUMBER If addresses outside of the Netherlands are also present in the table, also the following are required: PI_COL_STREET, PI_COL_CITY
POSTAL_CODE_NUMBER	PI_COL_COUNTRY, PI_COL_POSTAL_CODE, PI_COL_HOUSENUMBER If addresses outside of the Netherlands are also present in the table, also the following are required: PI_COL_STREET, PI_COL_CITY

The PI_COL columns can contain the actual column name as well as an expression.

7.3 Execution

Masking of a custom development schema is done with the same interface as for OHI itself.
The procedure sdm_driver_pck.job, or sdm_driver_pck.mask, should be executed under the custom development schema owner. To be able to run the job, the owner of the custom schema must have the privileges ‘CREATE JOB’ and ‘EXECUTE ON DBMS_SCHEDULER’.

7.4 Example script

The following script is an example showing a possible setup.
NB. The procedure SDM_DRIVER_PCK.DEL_CUSTOM_SCHEMA will remove all rules for the given custom development schema.

This example assumes your script is the “master” that is maintained, and the rules are re-created whenever there is a change in your custom schema that impacts the masking.

For this example, the following custom development tables are used:

```
create table SVS_PERSONEN
( ID                    NUMBER not null,
```

```
  BSN                   VARCHAR2(20) not null,
  ACHTERNAAM            VARCHAR2(240),
  VOORNAAM              VARCHAR2(240),
  GEBDATUM              DATE not null,
  GESLACHT              VARCHAR2(1) not null,
  REKENINGNUMMER        VARCHAR2(40),
  DEC_CODE              VARCHAR2(30),
  EMAIL                 VARCHAR2(240),
  UZOVI                 VARCHAR2(36),
  MERK                  VARCHAR2(5) not null,
  ZV_PAKKET             VARCHAR2(5),
  AV_PAKKET             VARCHAR2(5),
  NOTITIES              VARCHAR2(4000)
);
ALTER TABLE SVS_PERSONEN ADD CONSTRAINT SVS_PSN_PK PRIMARY KEY (ID);
ALTER TABLE SVS_PERSONEN ADD CONSTRAINT SVS_PSN_UK1 UNIQUE (BSN);

create table SVS_ADRESSEN
( ID                    NUMBER not null,
  DATUM_INGANG          DATE not null,
  BSN                   VARCHAR2(20) not null,
  PC_LETTER             VARCHAR2(2),
  PC_NR                 NUMBER(4),
  HUISNR                NUMBER(5),
  LAND                  VARCHAR2(2),
  BUITENLAND_STRAAT     VARCHAR2(35),
  BUITENLAND_POSTCODE   VARCHAR2(15),
  BUITENLAND_PLAATS     VARCHAR2(35),
  NOTITIES              VARCHAR2(4000)
);
ALTER TABLE SVS_ADRESSEN ADD CONSTRAINT SVS_ADR_PK PRIMARY KEY (ID);
ALTER TABLE SVS_ADRESSEN ADD CONSTRAINT SVS_ADR_FK_PSN FOREIGN KEY (BSN) REFERENCES
SVS_PERSONEN (BSN);

create table SVS_TEKSTEN
( ID                    NUMBER not null,
  BSN                   VARCHAR2(20) not null,
  TEKST                 VARCHAR2(4000)
);
ALTER TABLE SVS_TEKSTEN ADD CONSTRAINT SVS_TKN_PK PRIMARY KEY (ID);
ALTER TABLE SVS_TEKSTEN ADD CONSTRAINT SVS_TKN_FK_PSN  FOREIGN KEY (BSN) REFERENCES
SVS_PERSONEN (BSN);

INSERT INTO SVS_PERSONEN( ID, BSN, ACHTERNAAM, VOORNAAM, GEBDATUM, GESLACHT, REKENINGNUMMER,
DEC_CODE, EMAIL, UZOVI, MERK, ZV_PAKKET, AV_PAKKET, NOTITIES )
VALUES(1,'826897083','Jansen','Jan',to_date('01-06-1980','DD-MM-
YYYY'),'M','NL69INGB0123456789','KLT13405','jansen_j@example.org','0000','DDZ','ZVNAT','ALLIN'
,'Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit
anim id est laborum.');

INSERT INTO SVS_ADRESSEN( ID, DATUM_INGANG, BSN, PC_LETTER, PC_NR, HUISNR, LAND,
BUITENLAND_STRAAT, BUITENLAND_POSTCODE, BUITENLAND_PLAATS, NOTITIES ) VALUES(1,to_date('01-01-
1990','DD-MM-YYYY'), '826897083', 'CB', '1071' , 29, 'NL',NULL,NULL,NULL,NULL );

INSERT INTO SVS_TEKSTEN( ID, BSN, TEKST ) VALUES (1,'826897083','Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.');

INSERT INTO SVS_TEKSTEN( ID, BSN, TEKST ) VALUES (2,'826897083','Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat');

COMMIT;

begin
  -- clear the rules for the schema SVS_OWNER
  sdm_driver_pck.del_custom_schema( pi_schema_owner => 'SVS_OWNER' );

  --table SVS_TEKSTEN should be made empty
  sdm_driver_pck.add_custom_rule
  ( pi_schema_owner     => 'SVS_OWNER'
```

```
, pi_table_name       => 'SVS_TEKSTEN'
, pi_column_name      => null
, pi_masking_rule     => 'TRUNCATE'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'BSN'
, pi_masking_rule     => 'BSN'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'ACHTERNAAM'
, pi_masking_rule     => 'LASTNAME'
);

-- Gender in OHI is stored as 1 or 2; example shows a conversion if
-- the custom data gender column has another domain set for the GENDER
sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'VOORNAAM'
, pi_masking_rule     => 'FIRSTNAME'
, pi_col_gender       => 'DECODE(GESLACHT,''M'',1,''V'',2,GESLACHT)'

);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'GEBDATUM'
, pi_masking_rule     => 'DATE'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'REKENINGNUMMER'
, pi_masking_rule     => 'BANK'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'EMAIL'
, pi_masking_rule     => 'EMAIL'
, pi_col_gender       => 'GESLACHT'
, pi_col_first_name   => 'VOORNAAM'
, pi_col_last_name    => 'ACHTERNAAM'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'DEC_CODE'
, pi_masking_rule     => 'TRANSLATE_UNIQUE'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_PERSONEN'
, pi_column_name      => 'NOTITIES'
, pi_masking_rule     => 'TRANSLATE'
);

-- NOTE: although the SVS_PERSONEN.BSN is masked by a rule above, all other BSN columns
-- should be masked in the same way; there is no automatic detection of the same type of
```

```
-- columns or foreign key relation dependencies
sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_ADRESSEN'
, pi_column_name      => 'BSN'
, pi_masking_rule     => 'BSN'
);
sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_ADRESSEN'
, pi_column_name      => 'PC_LETTER'
, pi_masking_rule     => 'POSTAL_CODE_LETTER'
, pi_col_country      => '''NL'''
, pi_col_postal_code  => 'PC_NR||PC_LETTER'
, pi_col_house_number => 'HUISNR'
);

-- NOTE: although a country code column is present in this table (column LAND)
-- this example uses a literal value for the country code
-- In this specific example we 'know' that it is a Dutch (NL) specific postal code
-- NOTE: In this case the postal code is in the table separated into two columns.
-- you can use a concatenation to make these two columns act as one for the masking rule
sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_ADRESSEN'
, pi_column_name      => 'PC_NR'
, pi_masking_rule     => 'POSTAL_CODE_NUMBER'
, pi_col_country      => '''NL'''
, pi_col_postal_code  => 'PC_NR||PC_LETTER'
, pi_col_house_number => 'HUISNR'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_ADRESSEN'
, pi_column_name      => 'BUITENLAND_STRAAT'
, pi_masking_rule     => 'STREET'
, pi_col_country      => 'LAND'
, pi_col_street       => 'BUITENLAND_STRAAT'
, pi_col_city         => 'BUITENLAND_PLAATS'
, pi_col_postal_code  => 'BUITENLAND_POSTCODE'
, pi_col_house_number => 'HUISNR'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_ADRESSEN'
, pi_column_name      => 'BUITENLAND_POSTCODE'
, pi_masking_rule     => 'POSTAL_CODE'
, pi_col_country      => 'LAND'
, pi_col_street       => 'BUITENLAND_STRAAT'
, pi_col_city         => 'BUITENLAND_PLAATS'
, pi_col_postal_code  => 'BUITENLAND_POSTCODE'
, pi_col_house_number => 'HUISNR'
);

sdm_driver_pck.add_custom_rule
( pi_schema_owner     => 'SVS_OWNER'
, pi_table_name       => 'SVS_ADRESSEN'
, pi_column_name      => 'BUITENLAND_PLAATS'
, pi_masking_rule     => 'CITY'
, pi_col_country      => 'LAND'
, pi_col_street       => 'BUITENLAND_STRAAT'
, pi_col_city         => 'BUITENLAND_PLAATS'
, pi_col_postal_code  => 'BUITENLAND_POSTCODE'
, pi_col_house_number => 'HUISNR'
);
```

```
  -- NOTE: besides a literal or concatenation also simple sql functions like NVL are supported
  sdm_driver_pck.add_custom_rule
  ( pi_schema_owner     => 'SVS_OWNER'
  , pi_table_name       => 'SVS_ADRESSEN'
  , pi_column_name      => 'HUISNR'
  , pi_masking_rule     => 'HOUSENUMBER'
  , pi_col_country      => 'LAND'
  , pi_col_street       => 'BUITENLAND_STRAAT'
  , pi_col_city         => 'BUITENLAND_PLAATS'
  , pi_col_postal_code  => 'NVL(PC_NR||PC_LETTER,BUITENLAND_POSTCODE)'
  , pi_col_house_number => 'HUISNR'
  );

  sdm_driver_pck.add_custom_rule
  ( pi_schema_owner     => 'SVS_OWNER'
  , pi_table_name       => 'SVS_ADRESSEN'
  , pi_column_name      => 'LAND'
  , pi_masking_rule     => 'COUNTRY'
  , pi_col_country      => 'LAND'
  , pi_col_street       => 'BUITENLAND_STRAAT'
  , pi_col_city         => 'BUITENLAND_PLAATS'
  , pi_col_postal_code  => 'NVL(PC_NR||PC_LETTER,BUITENLAND_POSTCODE)'
  , pi_col_house_number => 'HUISNR'
  );

  sdm_driver_pck.add_custom_rule
  ( pi_schema_owner     => 'SVS_OWNER'
  , pi_table_name       => 'SVS_ADRESSEN'
  , pi_column_name      => 'NOTITIES'
  , pi_masking_rule     => 'NULL_VALUE'
  );
  -- NOTE: Don’t forget to commit the transactions
  commit;
end;
/
```