Informatica® Dynamic Data Masking
9.9.3

Stored Procedure Accelerator Guide for Oracle

Informatica Dynamic Data Masking Stored Procedure Accelerator Guide for Oracle
9.9.3
September 2021

Revision: 1
Publication Date: 2021-09-15

Table of Contents

Preface

Read the *Stored Procedure Accelerator Guide for Oracle* to learn how to perform dynamic data masking on stored procedures and table-valued functions in an Oracle database.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

Introduction to the Stored Procedure Accelerator for Oracle

This chapter includes the following topics:

Stored Procedure Accelerator for Oracle Overview

Use the Stored Procedure Accelerator for Oracle to implement Dynamic Data Masking to mask stored procedures and user-defined table functions in an Oracle database. The accelerator contains predefined Dynamic Data Masking security rule templates and example rules.

The Stored Procedure Accelerator for Oracle is in the Dynamic Data Masking installation directory as an additional component that you can configure to mask stored procedures and user-defined table functions. You can define masking rules based on the stored procedures and sensitive data in the Oracle database.

You can use the Stored Procedure Accelerator for Oracle with Oracle 11g and later.

Stored Procedure Accelerator Components

The accelerator contains a PL/SQL package, two .jar files, and four security rule sets.

You can find the accelerator in the following directory:

```
<Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking
```

The DDM_SP_MASKING PL/SQL package creates temporary tables, verifies whether to mask the result set, and creates masked select statements. You must compile the package in the <DDMADMIN> database schema before you mask stored procedures.

Dynamic Data Masking uses the StoredProcedureMasking.jar and MultipleStatementMasking.jar files in the Java Action security rule to mask stored procedures and user-defined table functions.

You must configure two security rule sets to use the accelerator. The accelerator contains template rule sets and sample HR security rule sets.

Stored Procedure Accelerator Example

You want to mask data called by a stored procedure.

You have a stored procedure named Proc_Dept_Emp, which has the following parameters:

```
emp_id IN NUMBER,
job_id IN NUMBER,
salary_rec OUT SYS_REFCURSOR
```

The SYS_REFCURSOR out parameter of the stored procedure selects the following columns in the Dept_Emp table:

- EMPLOYEE_ID
- Last_Name
- FIRST_NAME
- JOB_ID
- Job_Title
- SALARY
- COMM
- DEPARTMENT_ID

You set up the stored procedure accelerator and create a masking rule set. You have a masking rule that masks the Proc_Dept_Emp stored procedure. The following table describes the columns and masking functions that you define in the masking rule:

Column	Masking Function
.*SALARY	999
.*COMM	\(col)-FLOOR(\(col) / 7)*7
.*Last_Name	substr(\(col),1,2)

Note: When you create the masking rule, you must enter the column name preceded by .*, as in the table.

You send the following request to the database:

```
CALL Proc_Dept_Emp(25,10, ?)
```

The ? is a placeholder for the REF_CURSOR as OUT parameter.

Dynamic Data Masking creates a temporary stored procedure in the temporary schema that you defined in the Dynamic Data Masking ORACLE_DDM_TEMPDB symbol. The temporary stored procedure uses the DDM_SP_MASKING package and sends the call to the following temporary stored procedure:

```
CALL TEMPDB.P1436942931536_2 (25,10,? )
```

The temporary stored procedure executes the original stored procedure. The temporary stored procedure then creates and populates the global temporary table, TEMPDB.T_1436942931536, with the result set data and changes the out REF_CURSOR parameter to the following statement according to the masking rules:

```
SELECT EMPLOYEE_ID , substr(LAST_NAME ,1,2) LAST_NAME , FIRST_NAME , MIDDLE_NAME ,
JOB_ID , MANAGER_ID , HIREDATE , 999 SALARY , COMM -FLOOR(COMM / 7)*7 COMM ,
DEPARTMENT_ID , EMAIL FROM TEMPDB . T_1436942931536 ;
```

CHAPTER 2

Masking Stored Procedures and User-Defined Table Functions

This chapter includes the following topics:

Masking Stored Procedures and User-Defined Table Functions Overview

Dynamic Data Masking uses a Java Action security rule to determine whether an incoming statement is a call to a stored procedure or a SELECT statement that includes a user-defined table function that returns a result set. The Java Action rewrites the SQL statement so that the masking rules can use the result set structures.

If the stored routine is a user-defined table function, the Rule Engine rewrites the SQL statement so that the Java Action receives the altered statement.

The Dynamic Data Masking user defines a temporary Oracle schema in the DefMaskRSSym security rule. To mask stored procedure outputs, Dynamic Data Masking dynamically creates a temporary stored procedure in the temporary schema. The temporary stored procedure creates global temporary tables within the temporary schema based on the structure of the result set. It then populates the table with the data in the temporary schema. Dynamic Data Masking names the tables in the temporary schema in the following way:

```
T_<Timestamp>
```

For example, a table in the temporary schema might have the following name:

```
T_5290365863
```

The timestamp is an automatically generated number.

The temporary stored procedure forms SELECT queries for each table in the temporary schema and masks the queries based on the security rules.

Note: Because the Stored Procedure Accelerator for Oracle uses methods implemented on Oracle 11g, the database must be Oracle 11g or later.

Parameter Data Types

The stored procedures that you mask with the Stored Procedure Accelerator for Oracle must have parameters with recognized data types.

You can use the accelerator to mask stored procedures that contain parameters with the following data types:

Constant Arguments

- DATE
- INTEGER
- VARCHAR
- VARCHAR2

Binding Arguments

- DATE
- INTEGER
- OBJECT TYPE
- RECORD TYPE
- REF CURSOR
- TABLE TYPE
- VARCHAR
- VARCHAR2

MATCH_FUNCTION Symbols

You can create MATCH_FUNCTION and MATCH_FUNCTION_PARAMS symbols and a user-defined function that determine whether to mask a stored procedure.

By default, the DefMaskRSSym rule contains ExplicitRuleSet and ORACLE_DDM_TEMPDB symbols. You can optionally define additional symbols that use a PL/SQL function to determine whether to mask the stored procedure.

To use the additional symbols, you must create a decision making function that determines whether to mask the stored procedure. The function must return 0 or 1. If the function returns 1, Dynamic Data Masking masks the stored procedure. If the function returns 0, Dynamic Data Masking does not mask the stored procedure.

You can define the following symbols in the DefMaskRSSym rule:

MATCH_FUNCTION

The MATCH_FUNCTION symbol identifies the name of the decision making function.

In the **Symbol Name** field, enter MATCH_FUNCTION. In the **Symbol Value** field, enter the fully qualified name of the PL/SQL decision making function.

MATCH_FUNCTION_PARAMS

The MATCH_FUNCTION_PARAMS symbol defines the number of arguments in the original stored procedure that you want to pass to the decision making function.

In the **Symbol Name** field, enter MATCH_FUNCTION_PARAMS. In the **Symbol Value** field, enter the number of arguments in the original stored procedure that you want to pass to the decision making function. The count starts from the first argument.

MATCH_FUNCTION Symbols Example

You want to create a user-defined function and Dynamic Data Masking symbols that determine whether to mask a stored procedure.

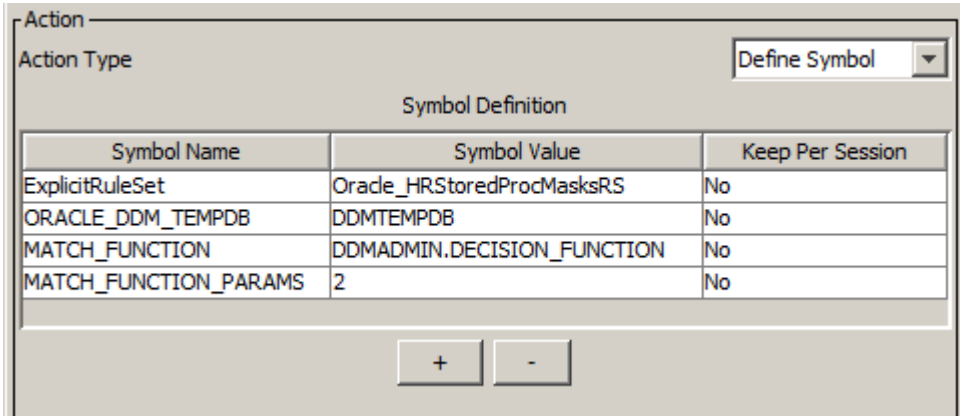
You create the following function that serves as the decision making function for the symbols:

```
create or replace function DDMADMIN.DECISION_FUNCTION(
     I_INST_ID varchar2,
     I_USER_ID varchar2
) return number
is
begin
     if (length(I_INST_ID)+length(I_USER_ID)) > 10
     then
      return 1;
     else
      return 0;
     end if;
end;
```

The function name is DDMADMIN.DECISION_FUNCTION. The function returns 0 or 1 based on whether the stored procedure returns sensitive data.

In the DefMaskRSSym security rule, you add MATCH_FUNCTION and MATCH_FUNCTION_PARAMS symbols. The MATCH_FUNCTION symbol value is the name of the function, DDMADMIN.DECISION_FUNCTION. The MATCH_FUNCTION_PARAMS symbol value is the number of arguments in the unmasked stored procedure that you want to pass to the decision making function. In this example, the symbol value is 2 and Dynamic Data Masking passes the first two arguments of the stored procedure to the decision making function.

The following image shows the rule action of the DefMaskRSSym rule:



Stored Procedure Accelerator Maintenance

Remove tables in the temporary database schema or use the clean up script template to create an Oracle job that cleans up the temporary objects that the accelerator creates in the temporary schema as part of the masking process.

To mask stored procedures, Dynamic Data Masking dynamically creates tables and temporary stored procedures in the Oracle temporary schema that the Dynamic Data Masking user configures. Because the tables are global temporary tables, the tables do not contain data.

When you set up the accelerator, you can create the CLEANUP_TEMP_OBJECTS procedure and schedule the procedure job to drop the tables and stored procedures. If you do not create the procedure, you must periodically manually drop the tables and stored procedures in the temporary schema.

Stored Procedure Accelerator Constraints

Before you use the Stored Procedure Accelerator for Oracle, review the stored procedure constraints that result in errors or unmasked data.

Variables as Arguments in a Procedure Call

You cannot declare a variable inside a PL/SQL block if you use the variable as an argument in a masked stored procedure or stored function.

For example, you want to mask the output of the following stored procedure:

```
dummy_proc_table2(
  in_int integer,
  o_int integer,
  o_table ddmtabletype) ; -- ddmtabletype is a TABLE Type defined at SCHEMA level
```

The following text is a stored procedure call that uses the declared variable v_int as an argument to the stored procedure:

```
Declare
  v_table ddmtabletype;
  v_int integer; -- declaration of variable
begin
  v_int := 12; -- value assigned
   dummy_proc_table2(
                    in_int => v_int, -- variable used
                    o_int => :o_int,
                    o_var => :o_var,
                    v_table => v_table
                   );
end;
```

The request returns unmasked data.

The following text is a stored procedure that passes the value directly to the stored procedure call and returns masked data:

```
Declare
  v_table ddmtabletype;
begin
  dummy_proc_table2(
                    in_int => 12,
                    o_int => :o_int,
                    o_var => :o_var,
```

```
                    v_table => v_table
                   );
end;
```

Blocks Containing Multiple Stored Procedure Calls

If a block contains multiple stored procedure calls, Dynamic Data Masking recognizes the first call, but does not recognize the following calls.

For example, you want to mask the output of the following stored procedure:

```
dummy_proc_table2(
  in_int integer,
  o_int integer,
  o_table ddmtabletype) ; -- ddmtabletype is a TABLE Type defined at SCHEMA level
```

The following text is an example of a call that masks the incorrect stored procedure:

```
Declare
  v_table ddmtabletype;
begin
-- first procedure call
  dummy_proc_table_3(
                     in_int => 12,
                     v_table => v_table
                    );

--second procedure call
dummy_proc_table2(
                  in_int => 12,
                  o_int => :o_int,
                  o_var => :o_var,
                  v_table => v_table
                 );
end;
```

Because the first stored procedure in the call is dummy_proc_table_3, the stored procedure accelerator masks the dummy_proc_table_3 procedure, and does not mask the dummy_proc_table2 procedure.

The following text is an example of a call that masks the correct procedure:

```
Declare
  v_table ddmtabletype;
begin
-- first procedure call
  dummy_proc_table2(
                    in_int => 12,
                    o_int => :o_int,
                    o_var => :o_var,
                    v_table => v_table
                   );

--second procedure call
dummy_proc_table_3(
                    in_int => 12,
                    v_table => v_table
                   );

end;
```

Because dummy_proc_table2 is the first procedure in the block, it is masked. However, in the example above, the dummy_proc_table_3 procedure is not masked.

Multi-Level Object/Table Types

Object types must have primitive types attributes.

You cannot use custom or user-defined Object types as attributes inside another Object definition. You cannot have a hierarchy of Object types.

For example, the following text is a command that returns unmasked data because it defines an attribute type inside an attribute and defines a function inside an attribute:

```
create or replace TYPE employee_typ AS OBJECT(
        employee_id NUMBER(6)
  first_name VARCHAR2(20)
  last_name VARCHAR2(25)
  address address_typ // Type inside type
  MEMBER FUNCTION get_idno RETURN SYS_REFCURSOR, // Function inside type
);
```

Positional and Named Parameters

You cannot use both positional and named parameters in the same stored procedure call.

Stored procedure calls can contain positional parameters or named parameters, but cannot have mixed case.

For example, the following text is a call that contains named parameters:

```
call dummy_proc_1(v_int=>'123', v_var=>?, v_ref=>?)
```

The following text is a call that contains positional parameters:

```
call dummy_proc_1(?,?,?)
```

The calls with named and positional parameters are masked. However, the following call is not masked because it contains a mix of named and positional parameters:

```
call oracle_1out_sys_1rs_2constip(?,eid => 1000,ename
=> 'kannan')
```

Overloaded Program Units

You can use overloaded program units that have different numbers of arguments, but you cannot use not overloaded program units that have the same number of arguments with different argument types.

For example, you can use the following package that contains overloaded functions with the same name and a different number of arguments:

```
CREATE OR REPLACE  PACKAGE SP_FUN_PACKAGE IS
/* Number of arguments : 5 */
FUNCTION fun_with_in_args_pkg_table(
in_int number,
in_var varchar2,
in_date date,
v_ref out SYS_REFCURSOR,
v_ref2 out SYS_REFCURSOR)
RETURN pkg_table_type PIPELINED;

/* Number of arguments : 4 */
FUNCTION fun_with_in_args_pkg_table(
in_int number,
in_int1 number,
in_var varchar2,
in_date date,
v_ref out SYS_REFCURSOR)
RETURN pkg_table_type PIPELINED;
END;
```

However, you cannot use the following package, which contains overloaded functions with the same name and same number of arguments:

```
CREATE OR REPLACE  PACKAGE SP_FUN_PACKAGE IS

/* Number of arguments : 4 */
FUNCTION fun_with_in_args_pkg_table(
in_int number,
in_var varchar2,
in_date date,
v_ref out SYS_REFCURSOR)
RETURN pkg_table_type PIPELINED;

/* Number of arguments : 4 */
FUNCTION fun_with_in_args_pkg_table(
in_int number,
in_int1 number,
in_var varchar2,
v_ref out SYS_REFCURSOR)
RETURN pkg_table_type PIPELINED;
END;
```

CHAPTER 3

Stored Procedure Accelerator Setup

This chapter includes the following topics:

Stored Procedure Accelerator Setup Overview

Set up the Stored Procedure Accelerator for Oracle to mask stored procedure and user-defined table function result sets in an Oracle database.

To set up the accelerator, perform the following tasks:

1. Verify the setup requirements.
2. Create the Dynamic Data Masking administrator and grant privileges to the administrator.
3. Compile the DDM_MASKING_SP package and create the temporary schema.
4. Grant privileges to the temporary schema.
5. Optionally, create a decision making function.
6. Create the clean up procedure and schedule the clean up procedure job.
7. Create an Oracle database connection.
8. Create a connection rule.
9. Import the accelerator security rule sets and configure the rules.

10. Define masking rules for stored procedures and user-defined table functions.

Step 1. Verify Requirements

Verify the following requirements before you use the Stored Procedure Accelerator for Oracle:

- You must have Dynamic Data Masking version 9.8.0 or later installed.
- You must have an Oracle database that contains stored procedures or user-defined table functions. If the database does not contain a stored procedure, you can set up the accelerator and create rules, but the rules will not return a match and will not mask data.

Step 2. Grant Dynamic Data Masking Administrator Privileges

Create the Dynamic Data Masking administrator and grant the required privileges.

Grant the following privileges to allow Dynamic Data Masking impersonation on the database:

- `CREATE USER <DDM Admin> IDENTIFIED BY <XXXX>`
- `ALTER USER <DDM Admin> QUOTA UNLIMITED ON USERS`
- `GRANT BECOME USER TO <DDM Admin>`
- `GRANT CREATE SESSION TO <DDM Admin>`
- `GRANT ALTER SESSION TO <DDM Admin>`
- `GRANT SELECT ANY TABLE TO <DDM Admin>`
- `GRANT SELECT ANY DICTIONARY TO <DDM Admin>`
- `GRANT EXECUTE ANY PROCEDURE TO <DDM Admin>`

Grant the following privileges to the Dynamic Data Masking administrator to use the Stored Procedure Accelerator for Oracle:

- `GRANT CREATE ANY PROCEDURE TO <DDM Admin>`
- `GRANT DROP ANY PROCEDURE TO <DDM Admin>`
- `GRANT EXECUTE ANY TYPE TO <DDM Admin>`
- `GRANT SELECT_CATALOG_ROLE TO <DDM Admin>`
- `GRANT GRANT ANY OBJECT PRIVILEGE TO <DDM Admin>`
- `GRANT CREATE ANY TABLE TO <DDM Admin>`
- `GRANT DROP ANY TABLE TO <DDM Admin>`
- `GRANT INSERT ANY TABLE TO <DDM Admin>`

Step 3. Compile the Masking Package

Compile the masking package in the DDMADMIN schema to enable stored procedure masking.

You can find the masking package in the following location:

```
<Dynamic Data Masking istallation>\Accelerators\StoredProcedureMasking\sql\Oracle
```

The masking package consists of the following files:

- DDM_MASKING_SP_BODY.sql
- DDM_MASKING_SP_SPEC.sql

The masking package performs the following actions:

- Retrieves result set metadata.
- Verifies whether to mask the result set.
- Creates and populates the global temporary tables in the temporary schema.
- Creates masked select statements in the temporary tables.

Compiling the Masking Package

Compile the masking package and create a temporary schema in the DDMADMIN schema.

1. To create the masking package, run the following script:

   ```
   DDM_MASKING_SP_SPEC.sql
   ```

2. To create the body of the package, run the following script:

   ```
   DDM_MASKING_SP_BODY.sql
   ```

3. To create a PUBLIC synonym for the masking package, run the following commands:

   - `CREATE PUBLIC SYNONYM DDM_MASKING_SP FOR <DDMAMDIN>.DDM_MASKING_SP`
   - `GRANT EXECUTE ON DDM_MASKING_SP TO PUBLIC`

Step 4. Grant Temporary Schema Privileges

Grant the following privileges to the temporary schema:

- `GRANT EXECUTE ON DDMADMIN.DDM_MASKING_SP To <TEMP SCHEMA>`
- `GRANT CREATE SESSION TO <TEMP_SCHEMA>`

Step 5. Create a Decision Making Function

Optionally, create a decision making function if you want to use the MATCH_FUNCTION and MATCH_FUNCTION_PARAMS symbols.

You can define additional symbols in the DefMaskRSSym security rule that use a PL/SQL function to determine whether to mask the stored procedure. If you want to use the MATCH_FUNCTION and MATCH_FUNCTION_PARAMS symbols, you must create a PL/SQL function that returns 0 or 1. If the function

returns 1, Dynamic Data Masking masks the stored procedure. If the function returns 0, Dynamic Data Masking does not mask the stored procedure.

Step 6. Create the Clean Up Procedure

To conserve space in the database, use the template for job creation to create an Oracle job that runs periodically.

Dynamic Data Masking creates tables and temporary stored procedures in the temporary schema. The tables are global temporary tables and do not contain data. If you do not run the clean up procedure, you must manually drop the tables and stored procedures in the temporary schema.

Creating the Clean Up Procedure

Create the CLEANUP_TEMP_OBJECTS procedure.

1. Open the following file in Notepad:

   ```
   <Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking\sql\Oracle
   \CLEAN_TEMP_OBJECTS_JOB_CREATION_TEMPLATE.sql
   ```

2. Use SQL Developer to log in to the database as an administrator.
3. Paste the text of the CLEAN_TEMP_OBJECTS_JOB_CREATION_TEMPLATE.sql file into the SQL Worksheet.
4. Run the script to create the CLEANUP_TEMP_OBJECTS procedure.

The CLEANUP_TEMP_OBJECTS has the following attributes:

created_before_seconds

Objects that have a creation time that is before the value of the created_before_seconds attribute are considered for deletion. Specify the time in seconds.

tempdb_schema

The name of the temporary schema where the accelerator creates the temporary objects.

For example, you want to delete the temporary objects in the DDMTEMPDB schema that the accelerator created more than 300 seconds ago. You use the following attribute values:

- `created_before_seconds : 300`
- `tempdb_schema : 'DDMTEMPDB'`

Creating the Clean Up Procedure Job

Schedule the clean up procedure job.

1. Open the following file in Notepad:

   ```
   <Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking\sql\Oracle
   \CLEAN_TEMP_OBJECTS_JOB_CREATION_TEMPLATE.sql
   ```

2. Use SQL Developer to log in to the database as an administrator.
3. Paste the text of the CLEAN_TEMP_OBJECTS_JOB_CREATION_TEMPLATE.sql file into the SQL Worksheet.
4. Search the text for the following keywords and edit the property values based on the user and client requirements:

<SchemaName_where_package_is_created>

The name of the schema in which you created the DDM_MASKING_SP package.

<seconds>

The value of the CLEANUP_TEMP_OBJECTS procedure created_before_seconds argument.

<schema_name>

The name of the temporary schema where the accelerator creates the temporary objects.

5. Run the script.
6. Use the following SQL command to verify that the job was created successfully:

```
select * from ALL_SCHEDULER_JOBS
```

The following text is an example execution script with the edited keyword values in bold:

```
BEGIN
    -- create ddm clean-up job
    DBMS_SCHEDULER.CREATE_JOB (
       job_name           =>  'clean_ddm_temp_object',
       job_type           =>  'STORED_PROCEDURE',
       job_action         => 'DDMADMIN.DDM_MASKING_SP.CLEANUP_TEMP_OBJECTS',
                 number_of_arguments => 2,
       start_date         =>  sysdate + 10/(24*60*60), -- start after 10 seconds
       repeat_interval    =>  'FREQ=HOURLY;BYMINUTE=0',
       end_date           =>  null,
       comments           =>  'DDM Temporary Object Clean-Up');
    -- setting job arguments
    DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
    job_name               => 'clean_ddm_temp_object',
    argument_position      => 1,
    argument_value         => '300'); -- delete TEMP-OBJECTS created before 300 seconds
from JOB-Execution time.
    DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
    job_name               => 'clean_ddm_temp_object',|
    argument_position      => 2,
    argument_value         => 'DDMTEMPDB'); -- schema_name of DDM Temporary Schema (IN
CAPs) from where to delete the temporary objects
       -- enable ddm clean-up job
    DBMS_SCHEDULER.ENABLE('clean_ddm_temp_object');
END;
```

Step 7. Create a Database Connection

Add the Dynamic Data Masking service for Oracle in the Management Console and connect to the database.

1. Log in to the Dynamic Data Masking Management Console.
2. Select the Dynamic Data Masking Server in the Management Console tree and click **Tree** > **Add DDM Services**.

   The **Add DDM Services** window appears.
3. Select DDM for Oracle and click **OK.**

   The DDM for Oracle node appears in the Management Console tree.
4. Select the Management Console tree root node and click **Tree** > **Add Database**.

   The **Add Database** window appears.
5. Select the Oracle database type and configure the following database connection parameters:

DDM Database Name

Logical name defined for the target database.

Instance name

Instance name for the target database.

Listener Address

Server host name or TCP/IP address for the target database.

Listener Port

TCP/IP listener port for the target database.

Service Name

Service name for the target database. Dynamic Data Masking determines the target database based on the service name or SID in the client connection request.

DBA Username

User name for the database user account to log in to the Oracle database.

DBA Password

Password for the database user.

6. Click **Test Connection** and verify that Dynamic Data Masking is connected to the database.
7. Click **OK**.

   The database node appears in the Management Console tree.

Step 8. Create a Connection Rule

Create a connection rule that directs SQL requests to the StoredProcTmpl rule set.

1. Select the Dynamic Data Masking service node for Oracle that you created in the Management Console tree and click **Tree** > **Connection Rules**.

   The **Rule Editor** opens.
2. In the **Rule Editor**, select the DDM for Oracle node in the tree and select **Action** > **Append Rule**.

   The **Append Rule** window opens.
3. In the **Append Rule** window, configure the following parameters:

   Rule Name

   Enter the name of the connection rule.

   Rule Matcher

   Select the Current Target Database matcher.

   Database

   Enter the name of the Oracle database.

   Rule Action

   Select the Use Rule Set action. The Use Rule Set action sends the request to a security rule set.

Rule Set Name

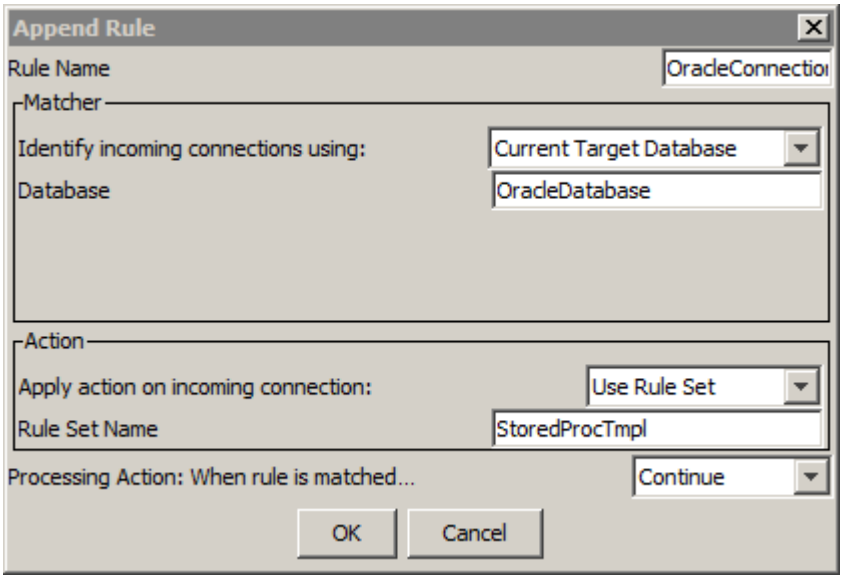
Enter the name of the rule set that you want to direct the request to. To use the template rule set, enter StoredProcTmpl.

The rule set name that you enter is the name of the rule set that contains the Define Symbol and Java Action security rules. If you want to use the HR example rule set, enter HRStoredProcRS. If you want to create your own rule set, enter the name of the rule set that you create.

Processing Action

Select the Continue processing action.

The following image shows the connection rule:



4. Click **OK**.

   The rule appears in the **Rule Editor**.

5. Select **File** > **Update Rules** to save the connection rule.
6. Select **File** > **Exit** to close the **Rule Editor**.

Step 9. Import the Security Rule Sets

Import the security rule sets and configure the rules to mask stored procedures and user-defined table functions.

You can import the security rule set templates and alter them based on the stored procedures and user-defined table functions in the database. You can import the HR rule sets to use as an example.

When you import the rule set that contains the masking rules, you must alter the rules based on the columns that the stored procedure or user-defined table function accesses. Create a separate security rule in the rule set for each stored procedure or user-defined table function.

Importing the StoredProc Rule Set

Import the StoredProcTmpl security rule set into the Management Console.

1. Select the Management Console tree root node and click **Tree** > **Add Rule Set**.

   The **Add Rule Set** window opens.

2. Enter StoredProcTmpl as the rule set name and click **OK**.

   The StoredProcTmpl rule set node appears in the Management Console tree.

   Note: If you want to use the HR example rule sets and you directed the connection rule to the HRStoredProcRS rule set, you must enter HRStoredProcRS as the rule set name.

3. Select the StoredProcTmpl node and click **Tree** > **Security Rule Set**.

   The **Rule Editor** opens.

4. In the **Rule Editor**, click **Action** > **Import**.

   The **Import** window opens.

5. Navigate to the following directory:

   ```
   <Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking\rules\Oracle
   ```

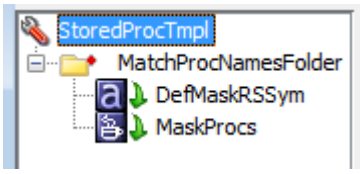
6. Select the Oracle_StoredProcRSTmpl.xml file and click **Import**.

   The MatchProcNamesFolder rule folder appears in the **Rule Editor**.

   Note: If you want to use the HR example rule sets, select the Oracle_HRStoredProcRS.xml file.

7. Expand the MatchProcNamesFolder rule folder to view the DefMaskRSSym and MaskProcs rules.

   The following image shows the security rule set rules:

8. Click **File** > **Update Rules** to save the security rules.

Configuring the StoredProc Rules

Configure the StoredProc rules in the **Rule Editor**.

1. Select the MatchProcNamesFolder rule and click **Action** > **Edit**.

   The **Edit Rule** window opens.

2. The Text field of the matcher contains a template that you can use to enter the stored procedure names. Replace "proc name 1," "proc name 2," and "proc name 3" with stored procedure or user-defined table function names.

   For example, you might enter the following text in the Text field:

   ```
   .*Proc_Dept_Emp.*|.*Func_Job_Location.*
   ```

   The following image shows the MatchProcsNamesFolder rule matcher parameters:

3. Click **OK**.

   The **Rule Editor** closes.

4. Click **File** > **Update Rules** to save the security rule.
5. Select the DefMaskRSSym rule and click **Action** > **Edit**.

   The **Edit Rule** window opens.

6. Create the following symbols:

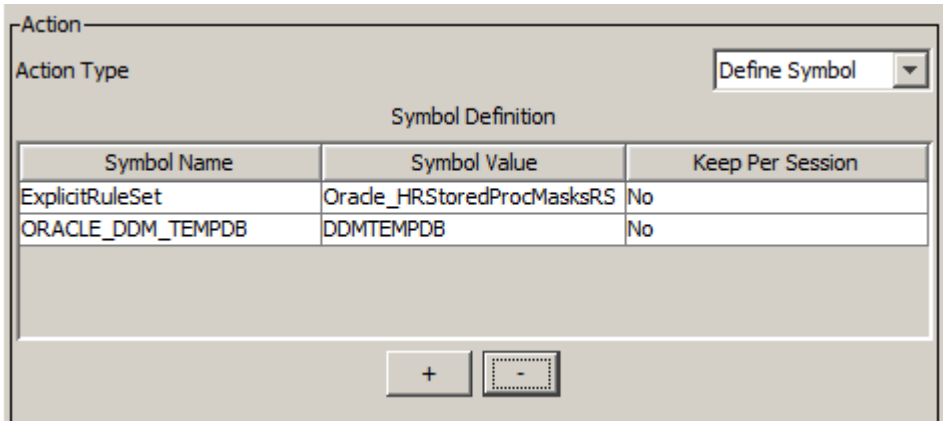
   ExplicitRuleSet

   In the **Symbol Name** field, enter ExplicitRuleSet. In the **Symbol Value** field, enter the name of the rule set that contains the masking rules. For the template rule sets, enter StoredProcMasksTmpl. If you want to use the HR example rule set, enter HRStoredProcMasksRS. In the Keep Per Session field, enter No.

   ORACLE_DDM_TEMPDB

   In the **Symbol Name** field, enter ORACLE_DDM_TEMPDB. In the **Symbol Value** field, enter the name of the Oracle database schema where Dynamic Data Masking will create the temporary objects. In the Keep Per Session field, enter No.

   The following image shows the DefMaskRSSym rule action properties:



7. Optionally, if you have a function that you want to use to determine whether to mask the stored procedure, create the following additional symbols:

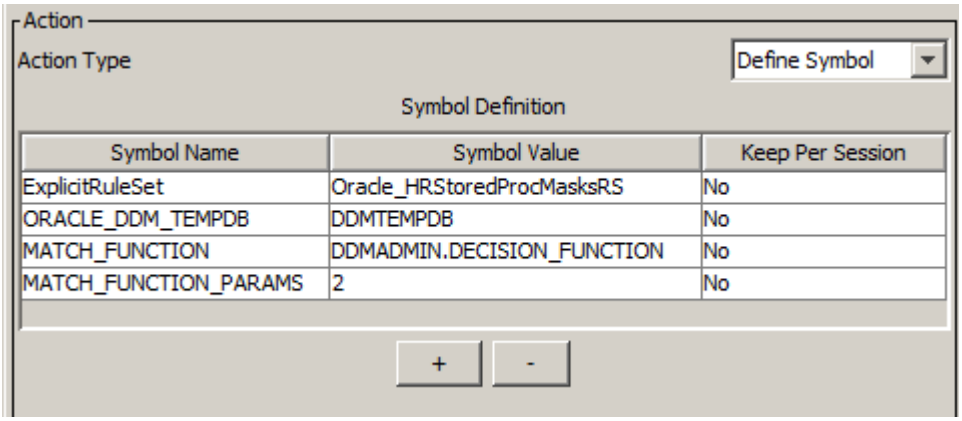
   MATCH_FUNCTION

   In the **Symbol Name** field, enter MATCH_FUNCTION. In the **Symbol Value** field, enter the fully qualified name of the PL/SQL decision making function. The function must return 0 or 1. If the function returns 1, Dynamic Data Masking masks the stored procedure. If the function returns 0, Dynamic Data Masking does not mask the stored procedure.

   MATCH_FUNCTION_PARAMS

   In the **Symbol Name** field, enter MATCH_FUNCTION_PARAMS. In the **Symbol Value** field, enter the number of arguments in the original stored procedure that you want to pass to the decision making function. The count starts from the first argument.

   The following image shows the DefMaskRSSym rule action with the additional symbols:



8. Click **OK**.

   The **Rule Editor** closes.

9. Click **File** > **Update Rules** to save the security rule.

10. Select the MaskProcs rule and click **Action** > **Edit**.

    The **Edit Rule** window opens.

11. In the rule action Class Path field, enter the file path of the stored procedure accelerator .jar files, separated by a semicolon (;).

    You can find the .jar files in the following directory:

    ```
    <Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking\lib
    ```

    For example, you might enter:

    ```
    C:\Program Files\Informatica\DDM\Accelerators\StoredProcedureMasking\lib
    \StoredProcedureMasking.jar;C:\Program Files\Informatica\DDM\Accelerators
    \StoredProcedureMasking\lib\MultipleStatementMasking.jar
    ```

    Note: The Class Name must be ProcMaskerOracle.

12. Click **OK**.

    The **Rule Editor** closes.

13. Click **File** > **Update Rules** to save the security rules.

14. Click **File** > **Exit** to close the **Rule Editor**.

Importing the StoredProcMasks Rule Set

Import the StoredProcMasks masking rule set into the Management Console.

1. Select the Management Console tree root node and click **Tree** > **Add Rule Set**.

   The **Add Rule Set** window opens.
2. Enter StoredProcMasksTmpl as the rule set name and click **OK**.

   The StoredProcMasksTmpl rule set node appears in the Management Console tree.

   Note: If you want to use the HR example rule sets, enter HRStoredProcMasksRS as the rule set name.
3. Select the StoredProcMasksTmpl node and click **Tree** > **Security Rule Set**.

   The **Rule Editor** opens.
4. In the **Rule Editor**, click **Action** > **Import**.

   The **Import** window opens.
5. Navigate to the following directory:

   `<Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking\rules\Oracle`
6. Select the Oracle_StoredProcMasksRSTmpl.xml file and click **Import**.

   The ProcMasks1 rule appears in the **Rule Editor**.

   Note: If you want to use the HR example rules, select the Oracle_HRStoredProcMasksRS.xml file.
7. Click **File** > **Update Rules** to save the security rules.

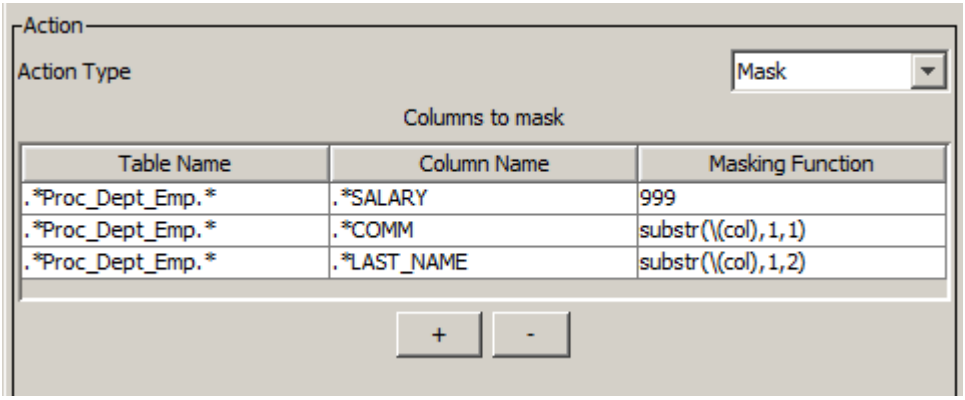
Configuring the StoredProcMasks Rule

Configure the StoredProcMasks rule in the Rule Editor.

1. Select the ProcMasks1 rule and click **Action** > **Edit**.

   The **Edit Rule** window opens.
2. Edit the rule properties based on the stored procedure that you want to mask. For example, the HR rule set has a MaskProcDeptEmp rule with masking functions defined for each column.

   The following image shows the rule action parameters in the HR MaskProcDeptEmp example rule:



3. Click **OK**.

   The **Rule Editor** closes.
4. Click **File** > **Update Rules** to save the security rules.

5. Click **File** > **Exit** to close the **Rule Editor**.

Step 10. Define Masking Rules

Define masking rules for each stored procedure and user-defined table function.

For each stored procedure and user-defined table function in the database, create a masking rule in the StoredProcMasks rule set. In the rule, you can add a row in the Mask rule action for each column that the stored procedure or user-defined table function outputs. For information about masking rules, see the *Dynamic Data Masking User Guide*.

CHAPTER 4

Stored Procedure Accelerator Rules

This chapter includes the following topics:

Stored Procedure Accelerator Rules Overview

The Stored Procedure Accelerator for Oracle contains template rules and sample rules that you can edit or copy to create rules based on the stored procedures and user-defined table functions in the database.

To use the accelerator, you must create a connection rule and two rule sets. The connection rule directs requests to the first rule set. The first rule set contains rules to set up the accelerator. The second rule set contains masking rules. You can import the accelerator rule sets and configure the rules based on the stored routines in the database.

You can find the .xml files that contain the accelerator rule sets in the following location:

```
<Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking\rules\Oracle
```

The Oracle_StoredPRocMasksRSTmpl.xml and Oracle_StoredProcRSTmpl.xml files contain the template rule sets. You can import the template rule sets and configure the rules to mask stored procedures and user-defined table functions.

The Oracle_HRStoredProcRS.xml and Oracle_HRStoredPRocMasksRS.xml files contain the example HR rule sets. You can view the example rules to see how a complete rule set looks.

Connection Rule

A Dynamic Data Masking connection rule directs the SQL request to the StoredProcTmpl security rule set.

You must create a connection rule in the Dynamic Data Masking Management Console to use the accelerator. Configure the connection rule to identify the incoming connection. For example, you can identify the incoming connection by the database type. Select the Use Rule Set action and define the StoredProcTmpl

rule set name or the name of the rule set that you create that contains the accelerator setup rules. Select the Stop if Applied processing action. If the client makes a request to the database, the Rule Engine applies the rule set.

StoredProc Rule Set

The StoredProc rule set contains three rules that identify requests, define the ExplicitRuleSet and ORACLE_DDM_TEMPDB symbols, and call the accelerator .jar files.

The StoredProc template rule set is the Oracle_StoredPRocRSTmpl.xml file in the accelerator directory. To use the template rules, create a rule set with the name StoredProcTmpl and import the rules into the rule set.

The StoredProc example HR rule set is the Oracle_HRStoredProcRS.xml file in the accelerator directory. To use the example rules, create a rule set with the name HRStoredProcRS and import the rules into the rule set.

You can also create a rule set with any name and create rules in the rule set based on the template rules and example rules.

MatchProcNamesFolder Rule

The MatchProcNamesFolder rule identifies requests that call stored procedures and user-defined table functions and directs them through the rule set.

The first rule in the StoredProc rule set is the MatchProcNamesFolder rule. The rule identifies requests to restrict the number of requests that go through the masking rules, which improves performance. The database might have stored procedures and user-defined table functions that you do not want Dynamic Data Masking to affect the results of, such as maintenance stored procedures. Configure the MatchProcNamesFolder rule so that those stored procedures are not a match for the rule.

You can use any matcher to identify incoming requests that call stored procedures or user-defined table functions. For example, if the names of the stored procedures in the database begin with Proc_, you can use a regular expression to identify requests that contain Proc_.

The MatchProcNamesFolder has the following parameters:

Rule Name

The rule name is MatchProcNamesFolder.

Matcher

The rule uses the Text matcher. You can change the matcher based on how you want to identify stored procedures.

Text

You can use regular expressions in the Text parameter to identify stored procedures.

Rule Action

The rule uses the Folder action. The Folder action creates a rule folder.

Processing Action

The rule uses the Stop if Matched action.

DefMaskRSSym Rule

The DefMaskRSSym rule defines the ExplicitRuleSet and ORACLE_DDM_TEMPDB symbols and directs the request to the masking rule set.

The DefMaskRSSym rule uses the Define Symbol action to define the ExplicitRuleSet and ORACLE_DDM_TEMPDB symbol values.

The symbol value of the ExplicitRuleSet symbol is the name of the rule set that contains the masking rules. For the template rules, you can enter StoredProcMasksTmpl. If you want to use the HR example rule sets, enter HRStoredProcMasksRS.

The DefMaskRSSym rule has the following properties:

Name

The name of the rule is DefMaskRSSym.

Matcher

The rule uses the Any matcher.

Rule Action

The rule uses the Define Symbol rule action. Do not change the rule action. The Define Symbol rule action creates the following symbols:

- ExplicitRuleSet. The symbol name is ExplicitRuleSet. Do not change the symbol name. The symbol value is the name of the rule set that contains the masking rules. The Keep Per Session value is No.
- ORACLE_DDM_TEMPDB. The symbol name is ORACLE_DDM_TEMPDB. Do not change the symbol name. The symbol value is the name of the schema where Dynamic Data Masking stores the temporary objects in the Oracle database. The Keep Per Session value is No.

Processing Action

The rule uses the Continue processing action to direct the request to the next rule in the rule set.

Optionally, you can define MATCH_FUNCTION and MATCH_FUNCTION_PARAMS symbols in the DefMaskRSSym rule.

MaskProcs Rule

The MaskProcs rule specifies the location of the stored procedure accelerator .jar files.

The MaskProcs rule uses the Any matcher so that the rule applies to all requests. It uses the Java Action rule action to identify the Stored Procedure .jar files. The Class Name must be ProcMaskerOracle.

The MaskProcs rule has the following parameters:

Name

The name of the rule is MaskProcs.

Matcher

The rule uses the Any matcher so that the rule applies to all requests.

Rule Action

The rule uses the Java Action rule action. Do not change the rule action.

Class Path

Enter the file paths of the accelerator .jar files, separated by a semicolon (;). You can find the .jar files in the following location:

```
<Dynamic Data Masking installation>\Accelerators\StoredProcedureMasking\lib
```

For example, you might enter:

```
C:\Program Files\Informatica\DDM\Accelerators\StoredProcedureMasking\lib
\StoredProcedureMasking.jar;C:\Program Files\Informatica\DDM\Accelerators
\StoredProcedureMasking\lib\MultipleStatementMasking.jar
```

Class Name

The class name is ProcMaskerOracle. Do not change the class name.

Processing Action

The rule uses the Continue processing action.

StoredProcMasks Rule Set

The StoredProcMasks rule set contains masking rules for stored procedures.

You can add as many masking rules to the StoredProcMasks rule set as required. You must create a separate rule for each stored procedure and user-defined table function. The predefined rule set contains a rule that shows you how to mask a stored procedure.

The StoredProcMasks template rule set is the Oracle_StoredPRocMasksRSTmpl.xml file in the accelerator directory. To use the template rule, create a rule set with the name StoredProcMasksTmpl and import the rules into the rule set.

The StoredProcMasks example HR rule set is the Oracle_HRStoredProcRS.xml file in the accelerator directory. To use the example rule, create a rule set with the name HRStoredProcMasksRS and import the rules into the rule set.

You can also create a rule set with any name and create masking rules in the rule set based on the template rules and example rules.

Note: You must identify the StoredProcMasks rule set name in the DefMaskRSSym rule when you define the symbol value. If you change the name of the StoredProcMasks rule set, update the Value field in the DefMaskRSSym rule.

ProcMasks1 Rule

The ProcMasks1 rule masks a stored procedure.

You can configure the ProcMasks1 rule to mask a stored procedure. Add a row in the rule action for each column that the procedure outputs. Create a rule for each stored procedure.

The ProcMasks1 rule has the following values:

Name

The name of the rule is ProcMasks1. You can change the name of the rule based on the name of the stored procedure.

Matcher

The rule uses the Any matcher.

Rule Action

The rule uses the Mask rule action.

Table Name

Enter the name of the stored procedure, preceded and followed by .*. For example, you might enter:

```
.*Proc_EMPLOYEE.*
```

Column Name

The name of the column that you want to mask, preceded by .*. For example, you might enter:

```
.*SALARY
```

Masking Function

The masking function that you want to use to mask the column data.

Processing Action

The rule uses the Continue processing action.

INDEX