Trafodion Blueprint Design Document:
Privilege Management

Title	Privilege Management
Date	September 2014
Author	Trafodion Security Team
Audience	Open Source community
Abstract	This document describes how users and roles are managed in Trafodion.

Document History

Document Version	Date	Changes
0.1	06/26/2014	• Initial revision
0.2	07/07/2014	• Revision after document review by Trafodion Security Team
0.3	07/31/2014	• Added roles
0.4	08/7/2014	• Added component privileges
0.5	09/15/2014	• Corrections from reviews

Legal Notice

TABLE OF CONTENTS

1 Introduction

1.1 Overview

There are two ways privileges will be enforced for Trafodion: pass-through authorization and Trafodion managed authorization. Pass-through authorization accepts grant and revoke statements, converts them to HBase requests and passes them to HBase. HBase is responsible for handling all authorization through their access controller and visibility labels co-processors. Trafodion managed authorization is performed through a new set of classes called Privilege Manager. Privileges are gathered and accessibility is checked at query compile time. With Trafodion managed authorization, all HBase objects are owned by the Trafodion ID.

This proposal describes Trafodion managed authorization through the privilege manager interface. The privilege manager grants and revokes privileges to authorization IDs, and Trafodion enforces these privilege settings when queries are performed. Authorization IDs include identifiers for users, roles and some special IDs such as PUBLIC.

The many different facets of authorization and privilege management will be implemented in several releases. There are also features that have been mentioned in this proposal that will be documented separately and be delivered as they complete.

1.2 Security function descriptions

The following terms are used throughout the document to describe the various roles required to manage Trafodion and the Trafodion ecosystem.

- **Enterprise security administrator** - a term referring to a person or office that manages information security across the customer's enterprise. The enterprise security administrator, among other things, defines and manages users in their service providers. Service providers may be single sign-on (SSO) providers such as Kerberos or directory services such as OpenLDAP. Trafodion supports OpenLDAP.

- **Trafodion security administrator**
  a term referring to a person or office that manages information security in the Trafodion eco-system. The Trafodion security administrator configures Trafodion to be accessible to the authentication service and manages security aspects of the system.

- **Database security administrator**
  a term referring to a person or office that manages security for the database(s) defined in the Trafodion ecosystem. The database security administrator manages roles and users described in the database by assigning users to roles and associating authentication service groups to roles.

- **Database administrator**
  a term referring to a person or office that manages the database(s) in the Trafodion ecosystem. The database administrator is responsible for, among other things, creating schemas and managing the data in the database.

The various security and administrator roles may be managed by separate individuals or offices; alternatively, an individual or office may perform two or more functions. There might even be separate

database administrators for each database defined in the customer's environment. It is up to the customer on how these different administrators are set up and managed.

2 Externals

2.1 Overview

This section discusses some basic terminology and information needed to understand the remainder of the document.

2.1.1 Authentication

When authentication is enabled, every user accessing Trafodion needs to be defined in the OpenLDAP directory service configured for the database. The customer should use their enterprise directory service with which the database communicates. Within the OpenLDAP directory service, users are defined along with their passwords; at authentication time, users pass their name and password; the database sends authentication information to the directory service to verify the user's credentials.

2.1.2 Groups, roles, users, and ownership

2.1.2.1 Groups

Open LDAP directory service groups

A group refers to the OpenLDAP group. OpenLDAP groups define a set of users and give them common access to features and functionalities in the directory.

Platform groups

The platform being used for Trafodion is Linux. Certain database objects such as UDF libraries and java files used by stored procedures are not stored in the database but in platform files. These files are secured using standard Linux security. In Linux, each file has a set of file permissions assigned. File permissions describe read, write, and execute privileges for the user, the group, and everyone else. When group is referenced in context of Linux security it is referring to the group attributes within LInux file permissions.

2.1.2.2 Roles

A role offers the flexibility of implicitly assigning a set of privileges to users, instead of assigning privileges individually. A user can be granted one or more roles. A role can be granted to one or more users. A role can be granted by, or revoked by, a database-user administrator, a role owner, or a member of the role.

Privileges are granted to a role. When a role is granted to a user, the privileges granted to the role become available to the user. If new privileges are granted to the role, then those privileges become available to all users who have been granted the role. When a role is revoked from a user, the privileges granted to the role are no longer available to the user.

A role is identified as follows:

- By the role name described in the CREATE ROLE statement which is either a regular or delimited SQL identifier

- By a role id which is a 32 bit number associated with the role that is used internally in the database for efficient access and storage

Roles are created through a CREATE ROLE command and removed through a DROP ROLE command. At creation time, the user specifies an SQL identifier and the software associates this name with a role id. At DROP ROLE time, the role name and role id become available for re-use. The role cannot be dropped until all privileges granted to the role are revoked.

Privileges on SQL objects may be granted to roles. When a role is granted to a database user, the user inherits authority based on the privileges granted to the role. When a role is revoked from a database user, the role authority is also revoked, but the user retains all directly granted privileges and authority based on other granted roles.

The ANSI SQL standard is limited in how privileges are checked for users. According to the standard, only privileges assigned to the current user and the current role can be used during authorization validation; so if privileges are required from multiple roles to authorize access, it is not possible. Therefore, ANSI has produced a standard called "Role Based Access Control" that describes the fundamentals of role based access control which allows a customer to define what roles are associated with users by default and interactively during a session.

2.1.2.3 Users

In ANSI, a database user is implementation defined; for Trafodion, the user is defined in the directory service and mapped to a name recognized by the database. The enterprise security administrator creates users in the directory service, and the database security administrator registers these directory service users as database users and assigns them a database username via a REGISTER USER command. Usage information for database users is stored in the Trafodion metadata. Only registered database users may access the Trafodion database.

Users are identified as follows:

- By the directory service username specified at logon time
- By an alternate username specified as part of the REGISTER USER command. It must be up to 128 characters long and is treated as an SQL identifier.
- By a user id which is a 32-bit number generated by the database to allow efficient processing while managing relationships.

The directory service username is the name used to logon to Trafodion. This name can take various forms; for example:

- Americas\JSmith
- jsmith@hp.com

When a user is registered in the database, the database security administrator can choose to use a different name since the directory service username is often somewhat cumbersome; for example:

- REGISTER USER “Americas/JSmith”;
- REGISTER USER “Americas/JSmith” as jsmith;

In the first case, both the directory service username and the database username are the same. So if privileges are assigned, the entire name is required: GRANT SELECT ON table1 TO “Americas/JSmith”. In the second case, the directory service username and the database username

are different. So, when privileges are assigned, the database username is required: GRANT SELECT ON table1 TO JSmith.

For purposes of this document, when username is specified unqualified, it is referring to the database username.

ANSI has a concept of a current user and, for a session, specifying a current user is optional as long as a current role exists. For Trafodion, we always require a current user to exist and the current user cannot change during the session.

2.1.2.4 Ownership

Object ownership:

At object create time, information is written into the database about the object, the object owner, and other characteristics. Some objects, such as tables, also create physical objects that store data. In the metadata, all objects are owned by an authorization ID which is the numeric equivalent of the username that created the object. When a physical file is created, it is assigned standard Linux permissions. Linux file permissions and other access control lists (ACLs) are managed by HDFS. The actual permissions defined on the Linux file are not necessarily the same IDs as defined in Trafodion. The different layers in the eco-system perform their own mapping of user IDs. <add a picture of this mapping>

File ownership:

There are files used by the database that are stored on the platform. These include .jar files for stored procedures and library files for user defined functions. <add details on how these files are secured>.

2.1.3 Authorization overview

A schema consists of objects (tables, views, procedures, etc) owned by a user. The owner of an object (grantor) has all privileges on the object and can assign these privileges to other users (grantee). The owner can assign the following DML privileges: EXECUTE for user defined routines, USAGE and UPDATE for libraries, and SELECT, UPDATE, DELETE, INSERT, REFERENCES for all other objects. In addition, the owner can specify WITH GRANT OPTION which allows the grantee to assign their granted privileges to yet more users. Privileges are additive.

The privilege manager supports two special authorization IDs: PUBLIC and _SYSTEM. PUBLIC is an ANSI SQL term which represents all authorization identifiers in the database as grantees. Granting a privilege to PUBLIC means that any existing or future user can perform the operation. PUBLIC may be specified as a grantee of a privilege, but PUBLIC may not be specified as a grantor. _SYSTEM is the special grantor who grants the initial privileges on an object to the object owner. _SYSTEM is not a valid grantor or grantee value for the GRANT and REVOKE commands.

When a user attempts an operation which is protected by a privilege, the authorization algorithm gets the list of the session user's privileges on the objects being accessed. This list is checked to determine if the user has the correct authority to perform the request. If so, the request can be completed; if not, the request fails with a “no privilege” error.

2.1.4 Configuration changes

2.1.4.1 Enabling authorization checks

Enabling authorization is performed by an enhanced traf_authentication_setup script. In addition to enabling authentication, INITIALIZE AUTHORIZATION [, UPGRADE] is called to enable authorization. This script is executed even if authentication has already been enabled. The script will skip enabling authentication if it is already enabled.

2.1.4.2 Configuring authorization IDs

During the installation process, the customer configures the Trafodion ecosystem which includes configuring the directory service. The process for configuring the directory service so it can be used by Trafodion is described on the Trafodion wiki page under the "Installing - Enabling Authentication" section. In addition to configuring the directory service, the database needs to be installed. The following information is needed from the customer to install the database:

- A directory service user that will have Trafodion database root access. When INITIALIZE TRAFODION is performed, a special database user called DB__ROOT is registered and mapped to "TRAFODION". This relationship should be changed to map DB__ROOT to an OpenLDAP user that will be assigned database root privileges. <need a way to map DB__ROOT to an LDAP user>
- Database security administrator – the user that manages database security functions including creating and managing users and roles.

We also configure the two special authorization IDs including PUBLIC and _SYSTEM.

PUBLIC:
is assigned the authentication ID of -1 and is an ANSI SQL term to mean all existing and future users.

_SYSTEM:
is assigned the authentication ID of -2 and is the root of all grants.

One user is preconfigured:

DB__ROOT:
This user is assigned to "TRAFODION" and indicates that the user TRAFODION has root authority for the database. This mapping should be changed to assign DB__ROOT to a valid OpenLDAP user.

2.1.5 Privileges

A privilege provides authority to perform a specific operation on a specific object or component. A privilege can be granted to, or revoked from, a user or role in many ways:

- Implicit privileges are granted to an owner of an object or component when the object or component is created. The owner retains implicit privileges for the object's lifespan.
- Explicit privileges can be granted to, or revoked from, a user or role. Explicit privileges can be granted or revoked by a database user administrator, an object owner, or a user who has been granted the privilege with the WITH GRANT OPTION option.
- The privileges granted to a user can come from various sources. Privileges can be directly granted to a user or they can be inherited through a role. For example, a user gets the SELECT

privilege on table T1 from two different roles. If one of the roles is revoked from the user, the user may still select from T1 via the SELECT privilege granted to the remaining role.

- A user who is granted a role is thereby conferred all privileges of the role. The only way to revoke any such privilege is to revoke the role from the user (or revoke the privilege from the role).

Privileges can be granted and revoked on the following:

- components such as SQL_OPERATIONS
- objects
- (TBD) columns (cells) of objects

2.1.6 Metadata

The metadata tables required for privilege management are created and maintained as Trafodion tables. This set of tables is referred to as privilege manager metadata. Access to Trafodion metadata is needed to get user and object details.

The following commands exist to manage privilege manager metadata:

- INITIALIZE AUTHORIZATION; -> creates metadata for privilege manager
- INITIALIZE AUTHORIZATION, DROP; -> remove privilege metadata
- INITIALIZE AUTHORIZATION, UPGRADE; -> upgrades privilege metadata

2.2 Features

This document covers changes to Trafodion and Privilege Management:

New or changed SQL Statements/functions:

- INITIALIZE AUTHORIZATION [, DROP | , UPGRADE]
- REGISTER USER statement
- ALTER USER statement
- UNREGISTER USER statement
- GRANT and REVOKE privileges statement
- CREATE ROLE
- DROP ROLE
- GRANT and REVOKE role statement
- SHOWDDL statements
- Built-in functions
- GET statement support

2.3 New and changed commands

2.3.1 Initialize Authorization

The INITIALIZE AUTHORIZATION statement creates, drop, or upgrades privilege management metadata tables.

Syntax Description:

```
INITIALIZE AUTHORIZATION
INITIALIZE AUTHORIZATION, DROP
INITIALIZE AUTHORIZATION, UPGRADE
```

Semantic Considerations:

1) If INITIALIZE AUTHORIZATION is specified, then the privilege manager metadata tables are created in the schema "PRIVMGR_MD". After the tables are created, they are populated with default privileges for existing Trafodion tables. For each object defined in Trafodion that is associated with a physical object, the following grant is effectively performed with the grantor as _SYSTEM:

   GRANT ALL<for object type> ON object TO object_owner WITH GRANT OPTION;

2) When INITIALIZE AUTHORIZATION, UPDATE is performed, dependencies are not updated. For example, prior to the upgrade, user1 creates a table user1_t1 and user2 creates a view user2_v1 that selects from table user1_t1. Once authorization is enabled, user2 would not be able to create view user2_v1 until user1 has granted user2 the correct privilege. <something to consider in the future is to try and find these issues, and possibly perform the associated grants at upgrade time>

3) The DROP option removes all the privilege manager metadata tables. Existing objects will be retained.

4) The UPGRADE option upgrades the privilege management metadata tables.

5) You must be the DB__ROOT user to execute these commands.

2.3.2 Register user statement

The REGISTER USER statement defines a user reference in the database. A database user, identified by an SQL identifier, has a set of privileges defined by the union of the privileges whose grantee is the username.

Syntax Description:

```
REGISTER USER dir-user-name [AS user-name]  [BY grantor]
```

dir-user-name is:
The username specified at logon that identifies the user in the directory service

user-name is:
An SQL identifier that denotes the username as defined in the database. It cannot be the same as an existing user or role name.

grantor is:
An existing user that is the operation is running on the behalf of

Semantic Considerations:

1) Only someone with the user administration authority can register a user. <enhance to describe how user administrator privilege can be managed>

2) The AS clause can be specified to allow the username used by the database to be different than the username defined in the directory service - for example, using jsmith as the database user instead of "Americas\JSmith". If the AS clause is specified, then this name is the name recognized in the database as the user.

3) If [BY *grantor*] is specified, then *grantor* must have the authority to register users.

4) Users cannot be registered with database usernames _SYSTEM or PUBLIC.

Example: REGISTER USER "jsmith@hp.com" AS jsmith;

2.3.3 Unregister user statement

The UNREGISTER USER statement removes a user reference from the database. A user, identified by an SQL identifier, has a set of privileges defined by the union of the privileges whose grantee is the username.

Syntax Description:

```
UNREGISTER USER user-name [drop-behavior]
drop-behavior ::= { RESTRICT | CASCADE }
```

user-name is:
An SQL identifier that denotes the username as defined in the metadata.

drop_behavior:
Describes how to handle objects currently owned by the user

Semantic Considerations:

1) Only someone with the user administrator authority can unregister a user. <enhance to describe how user administrator privilege can be managed>

2) Objects owned, and privileges and roles granted to the user being unregistered are managed according to the *drop_behavior*.

   - If the *drop_behavior* is not specified or RESTRICT is specified, then RESTRICT is assumed; that is, the user cannot be unregistered if there are any objects in the database owned by *user-name* or privileges granted to said user. To enforce this,

the metadata is searched to see if the user owns any object or is granted any privileges. If so, the UNREGISTER fails.

- If CASCADE is specified, then all objects owned by the user are dropped and any objects owned by users that were able to create said object through the granted privilege are removed. To enforce this, the all objects owned by the user are dropped and any privileges granted to the user are revoked.

Example: UNREGISTER USER “jsmith@hp.com”;

2.3.4 Alter user statement

The ALTER USER statement changes attributes associated with the user in the database.

Syntax Description:

```
ALTER USER username {options [, options] …}
options ::= { SET ONLINE | OFFLINE
            | SET EXTERNAL NAME dir-user-name }
```

username is:

An SQL identifier that denotes the database username as defined in the metadata

Semantic Considerations:

1) Only someone with the user administrator authority can change the mapping between database username and *dir-user-name*. <enhance to describe how user administrator privilege can be managed>

2) If a user is marked OFFLINE, authentication attempts fail.

2.3.5 Grant and revoke object privileges

The GRANT PRIVILEGES statement assigns privileges to a user. The REVOKE PRIVILEGES statement removes privileges from a user.

Syntax Description:

```
GRANT
  { object-rqst } TO {grantee [, grantee]… }
  [BY grantor] [WITH GRANT OPTION]

REVOKE [GRANT OPTION FOR]
  { object-rqst } FROM {grantee [, grantee]… }
  [BY grantor] [RESTRICT | CASCADE]

object-rqst ::=
   object-privileges
   ON [object-type] [schema].object

object-privileges ::=
   {object-privilege-list | ALL [PRIVILEGES]}

object-privilege-list ::=
   {object-privilege [, object-privilege]}

object-privilege ::=
    { SELECT
    | DELETE
    | INSERT
    | UPDATE
    | REFERENCES
    | EXECUTE
    | USAGE }

object-type ::=
   {[ TABLE
    | PROCEDURE
    | LIBRARY
    | FUNCTION
    | SEQUENCE_GENERATOR]}

grantee ::= username or role name
grantor ::= username or role name
```

Semantic Considerations

- The GRANT command assigns one or more privilege(s) to the grantee(s).
- The REVOKE command removes one or more privilege(s) from the grantee(s).
- Rules for allowing privileges to be granted:

- For GRANT statements, the WITH GRANT OPTION option allows the grantee to grant the specified privilege(s) to other users or roles.
- For REVOKE statements, the GRANT OPTION FOR option removes the grantee's authority to grant the specified privilege(s) to other users or roles.

For tables and views:

- SELECT - Can use SELECT statement on the object.
- DELETE - Can use DELETE statement on the object.
- INSERT - Can use INSERT statement on the object.
- UPDATE - Can use UPDATE statement on the object.
- REFERENCES - Can create constraints that reference the object.
- ALL PRIVILEGES - Grantee receives all privileges (SELECT, DELETE, INSERT, UPDATE, and REFERENCES) that apply to the object type.

For user defined routines:

- ALL PRIVILEGES - When the object is a stored procedure and ALL is specified, only EXECUTE permission is applied.

For libraries:

- ALL PRIVILEGES - When the object is a stored procedure and ALL is specified, only UPDATE and USAGE permissions are applied.

2.3.6 Create role

The CREATE ROLE statement creates an SQL role.

Syntax Description:

```
CREATE ROLE role-name [ WITH ADMIN grantor ]
  grantor is: database-username
```

role-name:

is a SQL identifier that specifies the new role. *role-name* cannot be an existing role name. *role-name* cannot be a registered database username. However, *role-name* can be a configured directory-service username.

WITH ADMIN *grantor*

specifies a role owner other than the current user. This is an optional clause.

database-username

specifies a registered database username to whom you assign the role owner.

Semantic Considerations

- To create a role, you must have user administrative privileges for roles. You have user administrative privileges for roles if you have been granted the ROLE component privilege.
- _SYSTEM, PUBLIC, NONE, and database usernames beginning with DB__ are reserved. You cannot specify a role-name with any such name.

- You can give role ownership to a user by specifying the user in the WITH ADMIN grantor clause with the grantor as the user. The role owner can perform the following operations:
  - Grant and revoke the role to users
  - Drop the role
- Role ownership is permanent. After the role is created, the ownership of a role cannot be changed or assigned to another user. TDB - add a change role owner request.

2.3.7 Drop role

The DROP ROLE statement deletes a SQL role.

Syntax Description:

```
DROP ROLE role-name
```

role-name:

is an existing role name. The role cannot be dropped if any of the following are true:

- Any privileges are granted to the role
- The role is granted to any users

Semantic Considerations

- To drop a role, you must own the role or have user administrative privileges for the role. You have user administrative privileges for the role if you have been granted the ROLE component privilege.
- _SYSTEM, PUBLIC, and database usernames beginning with DB__ are reserved. You cannot specify a role-name with any such name.
- You can determine all users to whom a role has been granted with the SHOWDDL ROLE statement.
- TDB - How to determine granted privileges on the role where a drop would require objects to be removed.
- Active Sessions for the User - when you drop a role from a user, the effects on any active sessions for the user are undefined. Trafodion recommends that you disconnect such sessions. The user then reconnects to establish new sessions with the updated set of privileges. TBD - query invalidate will fix this.
- Before You Drop a Role
  - You must revoke all privileges granted to the role
  - You must revoke the role from all users to whom it was granted

2.3.8 Grant and revoke role privileges

The GRANT ROLE statement grants one or more roles to a user. The REVOKE ROLE statement removes one or more roles from a user.

Syntax Description:

```
GRANT ROLE {role-name [,role-name ]...}   TO grantee

REVOKE ROLE {role-name [,role-name]...}  FROM grantee  [RESTRICT | CASCADE]

grantee is: database-username
```

role-name [,*role-name*] ...
specifies the existing roles to grant or revoke

TO *grantee*
specifies the registered database username to whom to grant the roles.

FROM *grantee*
specifies the user from whom to revoke the roles.

grantee
database-username specifies the registered database username from whom you revoke the roles.

[RESTRICT | CASCADE]
If you specify RESTRICT, the REVOKE ROLE operation fails if any privileges were granted to the role or any objects were created based upon those privileges. If you specify CASCADE, any dependent privileges are removed as part of the REVOKE ROLE operation. The default value is RESTRICT.

Semantic Considerations

- To grant roles to grantees or revoke roles from grantees, you must own the roles or have user administrative privileges for the roles. You have user administrative privileges for roles if you have been granted the ROLE component privilege.
- When you grant or revoke a role to or from a grantee, the effects on any active sessions for the grantee are undefined. Trafodion recommends that you disconnect such sessions. The grantee then reconnects to establish new sessions with the updated set of privileges. TBD - query invalidate will fix this.
- If any errors occur in processing a GRANT ROLE which names multiple roles, then no grants are done. Alternately, if the REVOKE ROLE names multiple roles and any errors occur in processing, no revokes are performed. This behavior differs from the GRANT and REVOKE object privilege statements
- If you attempt to grant a role but a grant with the same role and grantee already exists, the request is ignored and returns a successful operation.

- If RESTRICT (or nothing) is specified, when you revoke a role from a user that has created objects based solely on role privileges, the objects must be dropped before the role can be revoked. However, if you specify CASCADE, the dependent objects are automatically dropped and the role is revoked. Before you issue the REVOKE command, (TBD) determine which objects will be removed.
- All of the specified roles must have been granted to the specified user. If any role has not been granted to any user, the operation returns an error and no roles are revoked.

2.3.9 Register component statement

The REGISTER COMPONENT statement makes a component available to the database.

Syntax Description:

```
REGISTER COMPONENT name [SYSTEM] [DETAIL 'string']

name is a valid ANSI identifier

string is arbitrary text describing the component
```

Semantic Considerations

- If *name* is not a valid regular ANSI identifier a parser error with sqlcode -15001 will be raised. Delimited identifiers are not supported. The only character set permitted is ISO88591 (only 7-bit ASCII characters).
- If a component is designated a SYSTEM component, it can only be unregistered by DB__ROOT, regardless of any other privileges or roles a user may have been granted. In addition, any operations created for the component may only be dropped by DB__ROOT.
- If *string* exceeds 80 characters or is of a character set other than ISO88591 an error with sqlcode <3301> will be raised.
- In the initial release, only DB__ROOT can register a component, otherwise error <1017> (not authorized) will be raised.
- If a component with an identical name already exists in the metadata tables, then error <1055> will be raised. Note that component names are stored in upper case in the metadata tables and are not case sensitive.
- Any single quotes within *string* must be doubled.

2.3.10 Unregister component statement

The UNREGISTER COMPONENT statement removes the component and any privileges assigned to the component.

Syntax Description:

```
UNREGISTER COMPONENT name [RESTRICT | CASCADE];

name is a valid ANSI identifier
```

Semantic Considerations

- If *name* is not a valid regular ANSI identifier a parser error with sqlcode -15001 will be raised. Delimited identifiers are not supported. The only character set permitted is ISO88591 (only 7-bit ASCII characters).
- In the initial release, only DB__ROOT can unregister a component, otherwise error <1017> (not authorized) will be raised.
- If a component with this name does not exist, then error <1004> will be raised. Note that component names are stored in upper case in the metadata tables and are not case sensitive.
- If CASCADE is specified, all operations created for the component are automatically dropped and all privileges granted on the component are automatically removed.
- If RESTRICT is specified, if there are any operations created for the component the command will fail.

2.3.11 Create component privilege statement

The CREATE COMPONENT PRIVILEGE statement adds a new component operation to the metadata.

Syntax Description:

```
CREATE COMPONENT PRIVILEGE priv_name AS 'priv_abbr' ON name
[SYSTEM] [DETAIL 'string'] ;

priv_name is a valid ANSI identifier

priv_abbr is an abbreviation of priv_name that must be two characters long, in
ISO8591 character set.

name is a valid ANSI identifier which specifies the component

string is arbitrary text describing the component privilege
```

Semantic Considerations

- If *priv_name* is not a valid ANSI identifier a parser error with sqlcode -15001 will be raised. Delimited identifiers are not supported. The only character set permitted is ISO88591 (only 7-bit ASCII characters).
- If *priv_name* or *priv_abbr* already exists in the metadata table *for this component* then error <1055> will be raised. The same *priv_name* or *priv_abbr* can exist for another component.
- If <*priv_abbr*> is not a 2 character ISO88591 string a parser error with sqlcode -15001 will be raised.
- If this statement is issued by any user other than DB__ROOT then an error with sqlcode <1017> will be raised.
- If a component of this name does not exist, then error <1004> will be raised. Note that component names are stored in upper case in the metadata tables and are not case sensitive.

2.3.12 Drop component privilege statement

The DROP COMPONENT PRIVILEGE statement removes the component operation from the metadata.

Syntax Description:

```
DROP COMPONENT PRIVILEGE <priv_name> ON <name> [RESTRICT | CASCADE];

priv_name is a valid ANSI identifier

name is a valid ANSI identifier which specifies the component
```

Semantic Considerations

- If *priv_name* is not a valid ANSI identifier a parser error with sqlcode -15001 will be raised.
- If *priv_name* does not exist in the metadata table *for this component* then error <1004> will be raised.
- If this statement is issued by any user other than DB__ROOT then an error with sqlcode <1017> will be raised.
- If a component of this name does not exist, then error <1004> will be raised. Note that component names are stored in upper case in the metadata tables and are not case sensitive.
- If CASCADE is specified, all grants of this privilege to users and roles are automatically removed. If RESTRICT is specified, if there are any active grants the command fails.

2.3.13 Grant and revoke component privilege statements

The GRANT COMPONENT privilege statement associates one or more component privileges to an authorization ID. You can also grant the privilege(s) WITH GRANT OPTION. The REVOKE COMPONENT statement removes one or more privileges from an authorization ID. At revoke time, all privileges granted WITH GRANT OPTION are effectively removed. That is, revoke behavior is always CASCADE. This may change in the future.

Syntax Description:

```
GRANT COMPONENT PRIVILEGE priv_name_list ON name TO authID
[WITH GRANT OPTION];

REVOKE [GRANT OPTION FOR] COMPONENT PRIVILEGE priv_name_list ON name FROM
user-role-name [CASCADE];

priv_name_list is a list of priv_name's

name is a valid ANSI identifier which specifies the component

priv_name is a valid ANSI identifier
```

Semantic Considerations

- If *priv_name*, *name*, or *user-role-name* is not a valid ANSI identifier a parser error with sqlcode -15001 will be raised.
- If any names in the *priv_name_list* do not exist in the metadata table **for this component** then error <1004> will be raised.
- If a component of with name <name> does not exist, then error <1004> will be raised.

- If a user or role name with name *user-role-name* does not exist then error <1008> will be raised.
- If this statement is issued by any user without WITH GRANT OPTION then an error with sqlcode <1017> will be raised.
- If all of the privileges have been granted or revoked, then error TBD will be raised.
- If some of the privileges have been granted or revoked, they are silently ignored and the statement proceeds.
- The grantee is the *authID*.
- Typically the grantor is the use executing the command, but the grantor may be overridden with the BY clause. Also, if the current user is DB__ROOT, but the component operation was created by another user, the grantor is the creator of the component operation.
- Currently only CASCADE behavior is supported.

2.3.14 SHOWDDL statement

The SHOWDDL command displays the DDL syntax used to create an object as it exists in metadata. The statement has been enhanced to return USER descriptions in the form of GRANT statements. GRANT statements are only provided when authorization is enabled.

Syntax Description:

```
SHOWDDL { [PROCEDURE] object-name
        | COMPONENT component-name}
        | USER username
        | ROLE role-name}

object-name is  [schema-name.]object-name
username is a SQL identifier
```

Semantic Considerations

1) The PRIVILEGES option displays privileges granted to the object.

2) The SHOWDDL USER *username* displays the REGISTER USER statement.

3) The SHOWDDL ROLE *role-name* option displays the CREATE ROLE statement and the users that have been granted the role.

4) The SHOWDDL COMPONENT *component-name* option displays the REGISTER COMPONENT statement.

2.3.15 Built-in functions

The USER and CURRENT_USER functions return the database username associated with the specified user ID or the current user. The AUTHNAME function returns the authorization ID associate with the specified authorization name. The value returned is a string data type VARCHAR(128).

Syntax Description:

```
USER[user-id]
CURRENT_USER
AUTHNAME auth-id

user-id is the 32-bit number associated with the user
auth-id is the 32-bit number associated with the authorization name
```

Semantic Considerations:

If *user-id* is specified, the database username for the specified *user-id* is returned. If *user-id* is not specified, then the database username for the current *user-id* is returned. The current *user-id* is set at session logon time.

The USER, CURRENT_USER, and AUTHNAME functions can only be specified in the top level of a SELECT statement.

2.3.16 Get statements

The GET command displays information from the metadata.

Syntax Description:

```
GET USERS
GET ROLES
GET ROLES FOR USER user-name
GET USERS FOR ROLE role-name
GET COMPONENTS
GET COMPONENT PRIVILEGES ON component-name
GET COMPONENT PRIVILEGES ON component-name FOR authID
```

Semantic Considerations:

The following GET commands are supported related to users:

1) GET USERS returns registered users
2) GET ROLES returns available roles
3) GET ROLES FOR USER returns all the roles granted to the specified user.

4) GET USERS FOR ROLE returns all the users granted the specified role.
5) GET COMPONENTS returns registered components
6) GET COMPONENT PRIVILEGES ON *component-name* returns operations created for the specified component.
7) GET COMPONENT PRIVILEGES ON *component-name* FOR *authID* returns operations created for the specific *authID*.

2.4 Metadata tables

TABLE: AUTHS

Why: This table defines authorization IDs

Who: The AUTHS table is created as part of INITIALIZE TRAFODION. A row is written to the table when a REGISTER USER statement is executed. UNREGISTER USER removes data from this table. ALTER USER updates the contents of the associated row.

TABLE: OBJECT_PRIVILEGES

Why: This table records the privileges which have been granted to perform DML on an object.

Who: The OBJECT_PRIVILEGES table is created as part of INITIALIZE AUTHORIZATION. A row is written or updated when a GRANT PRIVILEGES statement is executed. The REVOKE PRIVILEGES statement changes or removes data from this table.

TABLE: COMPONENTS

Why: This table contains the list of defined components.

Who: The COMPONENTS table is created as part of INITIALIZE AUTHORIZATION. A row is written when REGISTER COMPONENT statement is executed. The UNREGISTER COMPONENT statement removes a row from this table.

TABLE: COMPONENT_OPERATIONS

Why: This table contains the list of component privilege operations for a component

Who: The COMPONENT_PRIVILEGES table is created as part of INITIALIZE AUTHORIZATION. A row is written when a CREATE COMPONENT PRIVILEGE statement is executed. The DROP COMPONENT PRIVILEGE statement removes a row from this table.

TABLE: COMPONENT_PRIVILEGES

Why: this table contains privileges that have been granted and revoked against a component privilege type of a component.

Who: The COMPONENT_PRIVILEGES table is created as part of INITIALIZE AUTHORIZATION. A row is written the first time a GRANT COMPONENT PRIVILEGE statement is executed against the component for a user. The REVOKE COMPONENT PRIVILEGE statement removes a row from this table when the last privilege is revoked for a user.

TABLE: ROLE_USAGE

Why: This table records the roles which have been granted to authorization IDs.

Who: The ROLE_USAGE table is created as part of INITIALIZE AUTHORIZATION. A row is written or updated when a GRANT ROLE statement is executed. The REVOKE ROLE statement changes or removes data from this table.