Trafodion Blueprint Design Document:
ANSI schemas

Title	ANSI schemas
Date	January 26, 2015
Author	Trafodion Security Team
Audience	Open Source community
Abstract	This document describes how schemas are managed in Trafodion.

Document History

Document Version	Date	Changes
1.0	10/28/2014	• Initial revision
1.1	11/6/2014	• Adds description of private and shared schemas
1.2	11/12/2014	• Correction of typos, other clarifications following team review
1.3	11/19/2014	• Update after project review, shared schema owners no longer will have DML access to objects in their schema
1.4	1/26/2015	• Updated to indicate portions that have been completed for release 1.0.

Legal Notice

TABLE OF CONTENTS

1 Introduction

This document describes the addition of ANSI schema support in Trafodion.

1.1 Overview – Existing Trafodion schemas

The term “schema” is used differently in different database offerings, but the common (and ANSI) definition is the one Trafodion follows, specifically that a schema is set of database objects grouped under a common label and managed by a single authorization ID.

Currently Trafodion schemas are only qualification labels for database objects. The CREATE SCHEMA command does nothing. When an object is created, the schema name does not have to refer to an existing schema.

There is no authorization at the schema level. Any database user can create an object in any schema. There is no concept of a schema owner or administrator to allow or deny access to the schema.

Trafodion uses the term database to describe the set of defined schemas. The term catalog is also used, but is being replaced with the preferred term database.

1.2 Requirements and design – ANSI schemas in Trafodion

ANSI schemas require a single authorization ID to own and administer a schema. Trafodion will allow the authorization ID to be a database user or a role. Creation of objects in a schema will require the schema to have previously been created; no longer will objects be allowed to specify a non-existent schema as a label name qualifier.

1.2.1 Private and shared schemas

Trafodion will support two classes of schema, private and shared.

Authority in a shared schema is similar to the current Trafodion behavior. Any user can create an object in a shared schema, and the user owns that object and has full authority on the object. One significant difference from the current Trafodion implementation is that the schema owner will also have full DDL authority on any object created in the schema, regardless of who created the object. If the schema owner is a role, any user granted the role has full DDL authority on all objects in the schema. The schema owner can drop or alter any object in their schema, or perform any utility operation such as update statistics.

By default, in a private schema, only the schema owner can create objects. The schema owner can grant access to objects in their schema. This is the ANSI and Oracle model for schemas and schema authority.

If a user has the component privilege CREATE ON SQL_OPERATIONS (or a specific create operation), they can create objects in a private schema. They will not own the object—ownership will be given to the schema owner. The creator of the object will be granted full DML privileges on the object they create, but the schema owner can revoke any or all of those privileges. Only the schema owner can alter or drop objects in a private schema.

	PRIVATE	SHARED
Who owns objects	Schema owner	Object creator – user that issued create object command.
Who can create objects	Schema owner, users or roles granted create privilege	Any database user
Who can grant access to objects	Schema owner	Object owner/creator
Who can alter objects in schema	Schema owner, users granted alter privilege	Schema owner plus object creator and users granted the alter privilege
Who can drop objects in schema	Schema owner, users granted drop privilege	Schema owner plus object creator and users granted the drop privilege

All Trafodion reserved[1] schemas are private. They are owned by DB__ROOT. No one can explicitly create objects in Trafodion reserved schemas – the set of objects is defined when Trafodion is initialized.

If authorization is not enabled, all non-reserved schemas are shared. If authorization is dropped, all existing non-reserved schemas are converted to shared schemas and the owner is set to DB__ROOT. If authorization is subsequently initialized, all non-reserved schemas remain shared. There is currently no way to convert a shared schema back to private.

Upgrade will convert all existing schemas to shared schemas owned by DB__ROOT. For more information on upgrade, see section 3.4 Upgrade.

1.2.2 Schema-level authority

The schema owner has full DDL authority on all objects in their schema. That means even though another user may have created the object, the schema owner can alter or drop any object.

The schema owner may be a role. In that case, all users who are granted the role have full authority on all objects in the schema owned by the schema owner and DDL authority for objects owned by other users or roles. Note that currently the GRANTED BY clause for object level grants has not been implemented. Until it is implemented, it will not be possible for users to grant object privileges on behalf of a role.

The schema owner may grant access on any object they own to another user. In a shared schema, the object owner may grant access to other users and roles.

DB__ROOT may also grant access on any object to another user or role. For both shared and private schemas, DB__ROOT is granting on behalf of the object owner.

Eventually schema owners will be able to grant authority at the schema level. Schema-level privileges will apply to all objects in a schema. If an object is created in a schema after a schema-level privilege is granted, the privilege will still apply.

[1] Reserved schemas in Trafodion begin with an underscore

2 Externals

This section describes the commands used to manage schemas in a Trafodion database. For existing commands, the changes relative to ANSI schemas are shown in green.

All changes described in the Externals section have been implemented in release 1.0 with the exception of the proposed changes to CREATE ROLE and REGISTER USER. Those changes are not part of release 1.0 and are not currently in progress.

2.1 CREATE SCHEMA

The CREATE SCHEMA command creates a new schema in the database. Previously the CREATE SCHEMA command was a nop; effectively this is a new command.

CREATE [schema-class] SCHEMA *schema-clause*
schema-class is: [PRIVATE | SHARED]
schema-clause is:
{ *schema-name* [AUTHORIZATION *authorizationID*] | AUTHORIZATION *authorizationID* }

schema-class
indicates whether access to the schema is restricted to the authorization ID by default, or if any database user may add objects to the schema. The default class for a schema is PRIVATE. Note, schemas created in 0.9 or earlier are SHARED schemas.

schema-name
is a name for the new schema and is a SQL identifier that specifies a name that is not a current schema name. This parameter is optional, however, if you do not specify a schema name, you must specify the authorization clause. If a schema name is not provided, the authorization ID is used for the schema name. If the authorization ID name matches an existing schema, the command fails.

authorizationID
is the name of the user or role who will own and administer the schema. If this clause is not present, the current user becomes the schema owner.

Semantic Considerations

- Reserved Schema Names
  Schema names that begin with a leading underscore (_) are reserved for internal use.

- AUTHORIZATION Clause
  The AUTHORIZATION clause is optional. If you omit this clause, the current user becomes the schema owner. Note, an authorization ID is assigned to a schema even if authorization is not enabled for the Trafodion database, however no enforcement occurs unless authorization is enabled.

  The schema owner can perform operations on the schema and on objects within the schema.

  For example:
  - Alter DDL of objects
  - Drop the schema
  - Drop objects
  - Manage objects with utility commands such as UPDATE STATISTICS and PURGEDATA.

- Who Can Create a Schema
  The privilege to create a schema is controlled by the component privilege CREATE_SCHEMA for the SQL_OPERATIONS component. By default, this privilege is granted to PUBLIC, but it can be revoked by DB__ROOT.

  When authorization is initialized, the following authorization IDs are granted the CREATE_SCHEMA privilege:
  - PUBLIC
  - DB__ROOT
  - DB__ROOTROLE

  DB__ROOT or anyone granted the DB__ROOTROLE role can grant the CREATE_SCHEMA privilege.

Examples

This example creates a private schema named `MYSCHEMA` owned by the current user:

```
CREATE SCHEMA myschema;
```

This example creates a shared schema and designates user GaryB as the schema owner:

```
CREATE SHARED SCHEMA hockey_league AUTHORIZATION GaryB;
```

This example creates a private schema and designates role DBA as the schema owner: Users with the role DBA can subsequently grant access to the objects in the schema to other users or roles.

```
CREATE PRIVATE SCHEMA Contracts AUTHORIZATION DBA;
```

This example creates a schema named `JSMITH`.

```
CREATE PRIVATE SCHEMA AUTHORIZATION JSmith;
```

2.2 CREATE ROLE

The CREATE ROLE statement creates a SQL role and optionally a schema managed by that role.

```
CREATE ROLE role-name [ WITH ADMIN grantor ] [schema-clause]
  schema-clause is: [schema-class] SCHEMA [schema-name]
  grantor is: database-username
  schema-class is: PRIVATE | SHARED
  schema-name is: SQL-identifier
```

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.

schema-class SCHEMA *schema-name*

if specified, creates a schema with the role as the owner of the schema. If the schema name is omitted, the name of the role is used as the schema name. If a schema with the role name already exists, the command fails. If the schema class is omitted, a private schema is created. The schema clause is optional when creating a role; if omitted, no schema is created, only a role.

Semantic Considerations

- To create a role, you must either be DB__ROOT or have been granted the MANAGE_ROLES privilege for SQL_OPERATIONS.
- _SYSTEM, PUBLIC, NONE, as well as names 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.
- If the SCHEMA clause is specified, any user granted the role can manage the schema and has full DDL authority for all objects within the schema. Note, until the GRANTED BY is implemented, users will not be able to grant object-level privileges on behalf of a role.

Examples

This example creates a role named MANAGER with administration by user JSmith:

```
CREATE ROLE MANAGER WITH ADMIN JSmith;
```

This example creates a role and private schema named SALES:

```
CREATE ROLE SALES SCHEMA;
```

2.3 DROP ROLE

The DROP ROLE statement deletes a SQL role.

```
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
- The role owns any schemas

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 MANAGE_ROLES component privilege.
- Role names beginning with DB__ are reserved and can only be dropped by DB__ROOT.
- You can determine all users to whom a role has been granted with the SHOWDDL ROLE statement.
- 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.

- 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
  - You must drop all schemas the role is a manager for (owner of).

Examples

This example drops the SALES role:

```
DROP ROLE SALES;
```

2.4 DROP SCHEMA

The DROP SCHEMA statement deletes a Trafodion SQL schema from a Trafodion database.

```
DROP SCHEMA schema-name [RESTRICT | CASCADE]
```

schema-name
is the name of the schema to delete.

RESTRICT
If you specify RESTRICT, an error is reported if the specified schema is not empty. The default is RESTRICT.

CASCADE
If you specify CASCADE, objects in the specified schema, and the schema itself, are dropped. Any objects in other schemas that were dependent on objects in this schema are dropped as well.

Semantic Considerations

To drop a schema, one of the following must be true:

- you are the owner of the schema
- you have been granted the role that owns the schema,
- you have been granted the DROP_SCHEMA privilege

Examples

This example drops an empty schema:

```
DROP SCHEMA sales;
```

2.5 GET SCHEMA FOR USER/ROLE

The GET SCHEMAS FOR statement lists all the schemas managed by a specified user or role.

```
GET SCHEMAS [FOR [USER | ROLE] authorization-ID]
```

authorizationID
is the name of a user or role. All schemas owned by this user or role will be displayed. Note, either USER or ROLE may be specified for users or roles. If the FOR clause is omitted, all schemas in the database are displayed.

Examples

```
>>GET SCHEMAS FOR USER daniel;
Schemas for User DANIEL
=======================

DANIEL
LITERATURE
MUSIC

--- SQL operation complete.TBD
```

2.6 REGISTER USER

The REGISTER USER statement defines a user reference in the database and optionally creates a schema managed by that user. 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.

```
REGISTER USER dir-user-name [AS database-user-name] [schema-clause]
  schema-clause is: [schema-class] SCHEMA [schema-name]
  schema-class is: PRIVATE | SHARED
  schema-name is: SQL-identifier
```

dir-user-name is:

The username specified at logon that identifies the user in the directory service

database-user-name is:

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

schema-class SCHEMA schema-name

specifies a schema to be created with the user as the owner of the schema. If the schema name is omitted, the name of the user is used as the schema name. If a schema with the username already exists, the command fails. If the schema class is omitted, a private schema is created. The schema clause is optional when registering a user.

Semantic Considerations:

1) Only DB__ROOT or an authorization ID granted the component privilege MANAGE_USER ON SQL_OPERATIONS can register a user.

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 “jsmith@company.com”. If the AS clause is specified, then this name is the name recognized in the database as the user.

3) Users cannot be registered with database usernames _SYSTEM or PUBLIC or any name that begins with DB__.

Examples

This example registers the user Marion Morrison with database name DUKE and creates a shared schema named SAG and owned by DUKE.

```
REGISTER USER Marion.Morrison@west.com AS DUKE SHARED SCHEMA SAG;
```

2.7 SHOWDDL SCHEMA

The SHOWDDL SCHEMA statement describes details about schema including the authorization ID in its display.

SHOWDDL SCHEMA *schema-name*
schema-name is: SQL-identifier

Examples

```
SHOWDDL SCHEMA myschema;
CREATE PRIVATE SCHEMA MYSCHEMA AUTHORIZATION JSMITH;

SHOWDDL SCHEMA Sales;
CREATE SHARED SCHEMA SALES AUTHORIZATION DBA;
```

Note, the AUTHORIZATION clause is shown even if authorization is not enabled.

2.8 UNREGISTER USER

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.

UNREGISTER USER *user-name* [*drop-behavior*]
drop-behavior is: { 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 DB__ROOT or an authorization ID granted the component privilege MANAGE_USER ON SQL_OPERATIONS can unregister a user.

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, the user cannot be unregistered if there are any objects or schemas 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 USER command fails.
   - If CASCADE is specified, then all objects and schemas owned by the user are dropped. In addition, any objects owned by other users that were able to be created due to privileges granted on objects owned by this user are dropped. Any privileges granted to this user are revoked.

Examples

```
UNREGISTER USER DUKE;
```

3 Internals

3.1 Syntax/Parser cleanup

As of Trafodion 0.9, the CREATE SCHEMA command syntax contains unused clauses. They will be removed. These include specifying a catalog, re-using a location, collation, and creating objects in the schema. This will result in productions being removed from sqlparser.y and member data and functions being removed from StmtDDLCreateSchema.h.

Data members in StmtDDLCreateSchema to be removed:

```
NAString subvolumeName_;
SchemaName schemaQualName_;
NABoolean allowLocationReuse_;
StmtDDLCreateIndexArray    createIndexArray_;
StmtDDLCreateTableArray    createTableArray_;
StmtDDLCreateViewArray     createViewArray_;
StmtDDLCreateTriggerArray  createTriggerArray_;
StmtDDLGrantArray          grantArray_;
ComBoolean                 isCompoundCreateSchema_;
CharType                   *pCharType_;
StmtDDLAddConstraintCheckArray  addConstraintCheckArray_;
StmtDDLAddConstraintRIArray     addConstraintRIArray_;
StmtDDLAddConstraintUniqueArray addConstraintUniqueArray_;
ElemDDLNode * children_[MAX_STMT_DDL_CREATE_SCHEMA_ARITY];
```

This will also result in the remove of the StmtDDLCreate*Array types and the header files that declare them, as well as their constructor and other function definitions in StmtDDLCreate.cpp and StmtDDLNode.cpp.

Functions in StmtDDLCreateSchema to be removed:

```
StmtDDLAddConstraintCheckArray & getAddConstraintCheckArray();
const StmtDDLAddConstraintCheckArray & getAddConstraintCheckArray() const;
StmtDDLAddConstraintRIArray & getAddConstraintRIArray();
const StmtDDLAddConstraintRIArray & getAddConstraintRIArray() const;
StmtDDLAddConstraintUniqueArray & getAddConstraintUniqueArray();
const StmtDDLAddConstraintUniqueArray & getAddConstraintUniqueArray() const;
Int32 getArity() const;
const NAString &getSubvolumeName() const;
NABoolean getLocationReuse() const { return allowLocationReuse_; };
ExprNode * getChild(Lng32 index);
StmtDDLCreateIndexArray & getCreateIndexArray();
const StmtDDLCreateIndexArray & getCreateIndexArray() const;
StmtDDLCreateTableArray & getCreateTableArray();
const StmtDDLCreateTableArray & getCreateTableArray() const;
StmtDDLCreateViewArray  & getCreateViewArray();
const StmtDDLCreateViewArray  & getCreateViewArray() const;
StmtDDLCreateTriggerArray  & getCreateTriggerArray();
const StmtDDLCreateTriggerArray  & getCreateTriggerArray() const;
StmtDDLGrantArray & getGrantArray();
const StmtDDLGrantArray & getGrantArray() const;
const SchemaName & getSchemaNameAsQualifiedName() const;
SchemaName & getSchemaNameAsQualifiedName();
const ComBoolean isCompoundCreateSchema() const;
```

3.2 New modules

A new module (CmpSeabaseDDLschema.cpp) will be added to implement the CREATE SCHEMA and DROP SCHEMA commands, as well as the describe portion of the SHOWDDL command. There will not be a separate CmpSeabaseDDLschema class. Functions needed to implement CREATE SCHEMA, DROP SCHEMA, and support SHOWDDL SCHEMA will be declared in CmpSeabaseDDL.h.

3.3 Metadata

Creation of a schema will add a row to the objects table. The object name used for this schema object row will be __SCHEMA__. The object type enum COM_SCHEMA_LABEL_OBJECT will be replaced with COM_PRIVATE_SCHEMA_OBJECT and COM_SHARED_SCHEMA_OBJECT. The two character literals will be "PS" and "SS" respectively. The define names for the new literals will be COM_PRIVATE_SCHEMA_OBJECT_LIT and COM_SHARED_SCHEMA_OBJECT_LIT. Note, no existing code is using the current literal string value "SH" for objects and no code is using the current COM_SCHEMA_LABEL_OBJECT enum.

In addition, a new column will be added to the OBJECTS table, schema_owner.

3.4 UPGRADE

A new column, SCHEMA_OWNER, is being added to the OBJECTS table. Upgrade will set the schema owner for all existing objects to DB__ROOT.

All existing schemas will be classified as SHARED schemas. Upgrade will need to scan the objects table for distinct catalog name/schema name pairs.

```
SELECT DISTINCT CATALOG_NAME,SCHEMA_NAME FROM OBJECTS;
```

For each distinct pair, a row will be added to the OBJECTS table with these values:

CATALOG_NAME: from rowset
SCHEMA_NAME: from rowset
OBJECT_NAME: __SCHEMA__
OBJECT_TYPE: SS
OBJECT_UID: generated
CREATE_TIME: generated
REDEF_TIME: generated
VALID_DEF: Y
OBJECT_OWNER: 33333[2]
SCHEMA_OWNER: 33333

Currently PUBLIC is granted ALTER, DROP, and CREATE privilege (or in some interim versions, just the CREATE privilege). These privileges will be revoked from PUBLIC and CREATE_SCHEMA will be granted to PUBLIC.

The new column will be added as part of the current 3.0 upgrade effort. The 3.0 upgrade code will also call a function to upgrade PrivMgr data. Initially it will be a stub, but prior to the official release it will be implemented to add the additional schema object rows and set the correct PUBLIC privileges. A script will be provided that performs these same steps for any internal or external users who install and upgrade between the initial 3.0 version availability and the delivery of this task.

[2] 33333 is the reserved ID for DB__ROOT.

3.5 Implementation notes

These are just notes on changes that are required internally to implement the externals and semantic considerations described in Externals. This section may not be complete and is not intended to be exhaustive, however, any additions or suggestions are appreciated.

Any objects created in a private schema will be owned by the schema owner, not the user who created the object. The creator will be granted all privileges with grant option on the object they created; the grantor will be the owner of the schema, not the system.

Objects created in shared schemas will continue to be owned by the user who created them, with the initial grantor the system as before.

Although roles can own schemas, there is a dependency on the GRANTED BY clause for object-level grants. If this is not implemented, it will not be possible to grant privileges on objects owned by a role.

CREATE ROLE and REGISTER USER code in CmpSeabaseDDLauth.cpp will handle the new optional AUTHORIZATION clause. Common code will be called (perhaps in PrivMgrObjects.cpp) to add the schema object row to the OBJECTS table.

If the schema owner is a role, and the role has been granted to the user, then that user's privileges will automatically include privileges granted to the role. No changes to privileges for DML operation is anticipated.

The GET SCHEMAS command is already implemented. Code in ExeExeUtilGet.cpp will be updated to implement the user/role specific query. The output label will say "Schemas for authorization ID authID". A new QueryType enum, SCHEMAS_FOR_ROLES_ will be added to ComTbdExeUtil.h in addition to the existing SCHEMAS_FOR_USERS_. Both will go to the same case/query, the only difference being the addition of the ROLE keyword being allowed in the GET syntax defined in GenRelExeUtil.cpp. It has been requested that the USER/ROLE keywords be optional. Also, the ability to get only shared or private schemas. Get commands are not particularly flexible or user friendly; they don't report errors well or offer many options. Typically options require essentially a new command/query. Not a requirement for this blueprint, but in the future we should consider the best way to deliver information on schemas and other objects.

DROP SCHEMA will delete the row for the schema from the OBJECTS table after all objects in the schema are successfully dropped (CASCADE) or if schema is empty (RESTRICT).
Create authority will move from isDDLOperationAuthorized() to verifyDDLCreateOperationAuthorized(). Both functions will be located in CmpSeabaseDDLcommon.cpp. Function isDDLOperationAuthorized() needs to handle the case of the schema owner performing a DDL operation on an object in their shared schema.

DDL Create commands require one of:

1) User ID DB__ROOT
2) Authorization is not enabled
3) User is schema owner
4) Schema is a shared schema
5) User has CREATE* component privilege
6) Eventually, schema-level DDL privileges as well.

The CREATE ON SQL_OPERATIONS privilege will no longer be granted to PUBLIC when authorization is initialized. PUBLIC will be granted CREATE SCHEMA.

3.6 Futures notes

Can we add GIVE SCHEMA at the same time? Would require updating OBJECT_PRIVILEGES. Could be rejected if new user is in the grant tree for any objects in the schema. Would need to notify query invalidation. Perhaps ALTER SCHEMA is a better command name.

Potential to add LDAP groups. Roles and groups association, with groups managed outside of Trafodion. Any effect on ANSI schemas as described above? What happens when a group is removed? Loss of role is one thing, but if data is associated with role, there needs to be a way to reauthorize access. Perhaps GIVE SCHEMA (or ALTER SCHEMA) is a requirement to support groups.