

Teradata Vantage™ - SQL Data Control Language

Release 17.10




July 2021

Copyright and Trademarks

Copyright © 2000 - 2021 by Teradata. All Rights Reserved.

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. For more information, see [Trademark Information](#).

Product Safety

Safety type	Description
	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
	Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

Third-Party Materials

Non-Teradata (i.e., third-party) sites, documents or communications ("Third-party Materials") may be accessed or accessible (e.g., linked or posted) in or in connection with a Teradata site, document or communication. Such Third-party Materials are provided for your convenience only and do not imply any endorsement of any third party by Teradata or any endorsement of Teradata by such third party. Teradata is not responsible for the accuracy of any content contained within such Third-party Materials, which are provided on an "AS IS" basis by Teradata. Such third party is solely and directly responsible for its sites, documents and communications and any harm they may cause you or others.

Warranty Disclaimer

Except as may be provided in a separate written agreement with Teradata or required by applicable law, the information available from the Teradata Documentation website or contained in Teradata information products is provided on an "as-is" basis, without warranty of any kind, either express or implied, including the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.

The information available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The information available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in this information at any time without notice.

Machine-Assisted Translation

Certain materials on this website have been translated using machine-assisted translation software/tools. Machine-assisted translations of any materials into languages other than English are intended solely as a convenience to the non-English-reading users and are not legally binding. Anybody relying on such information does so at his or her own risk. No automated translation is perfect nor is it intended to replace human translators. Teradata does not make any promises, assurances, or guarantees as to the accuracy of the machine-assisted translations provided. Teradata accepts no responsibility and shall not be liable for any damage or issues that may result from using such translations. Users are reminded to use the English contents.

Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: docs@teradata.com.

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

Contents

Chapter 1: Introduction to SQL Data Control Language	4
Changes and Additions	4
Chapter 2: Database Privileges	5
About Database Privileges	5
Privilege Levels	5
User Privilege Types	7
Non-User Privileges	9
Chapter 3: Statement Syntax	10
Introduction to Statement Syntax	10
GIVE	10
GRANT	13
GRANT (Monitor Form)	14
GRANT (Role Form)	16
GRANT (SQL Form)	19
GRANT CONNECT THROUGH	73
GRANT LOGON	86
GRANT MAP	91
GRANT ZONE	92
GRANT ZONE OVERRIDE	93
REVOKE	94
REVOKE (Monitor Form)	95
REVOKE (Role Form)	98
REVOKE (SQL Form)	100
REVOKE CONNECT THROUGH	119
REVOKE LOGON	123
REVOKE MAP	126
REVOKE ZONE	127
REVOKE ZONE OVERRIDE	128
Appendix A: Notation Conventions	129
Appendix B: Privilege Dictionary	132
Appendix C: Additional Information	154

Introduction to SQL Data Control Language

Teradata Vantage™ is our flagship analytic platform offering, which evolved from our industry-leading Teradata® Database. Until references in content are updated to reflect this change, the term Teradata Database is synonymous with Teradata Vantage.

This document describes the Teradata SQL Data Control Language (DCL) statements used to manage access to the database objects and data in the system. Teradata SQL is an ANSI compliant product. Teradata has its own extensions to the language.

For more information about Vantage security, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

Changes and Additions

Date	Description
July 2021	Minor edits.

Database Privileges

These topics provide an overview of the database privileges and a summary of the types of access control, or security, supported by the privileges.

About Database Privileges

A database privilege is a permission to access or to manipulate a database object or data. Specific privileges are required for nearly everything that can be done in Vantage.

Although privileges are sometimes called access rights, permissions, or authorizations, this book uses the term *privilege*, per the ANSI/ISO SQL:2011 specifications.

Database privileges are used by administrators to control access to database objects and data, and to control the types of actions and activities available to users. For a complete list of privileges, see [Privilege Dictionary](#).

The privileges are used to control which users can:

- Access, create, modify, or delete specific database objects and data
- Execute specific macros, procedures, and UDFs
- Monitor system-wide activity
- Grant privileges to other users

Teradata Data Control Language (DCL) requests grant and revoke the privileges that enable users to perform these actions and activities.

To grant a privilege, the user must have the privilege on the object and the right to grant the privilege.

Privilege Levels

Vantage supports system-level, object-level, row-level, and zone-level privileges.

- System-level privileges, see [System-Level Privileges](#)
- Object-level privileges, see [Object-Level Privileges](#)
 - Database
 - Table
 - Row or column
 - Other object types
- Row-level privileges, see [Row-Level Privileges](#)
- Zone-level privileges, see [Teradata Secure Zones Zone-Level Privileges](#)

System-Level Privileges

System-level privileges apply system-wide and cannot be defined on a database object. Generally, these privileges are used by administrators.

The system-level privileges are as follows:

- CONSTRAINT ASSIGNMENT
- CONSTRAINT DEFINITION
- CREATE MAP
- CREATE ROLE
- CREATE PROFILE
- CREATE ZONE
- CTCONTROL
- DROP MAP
- DROP ROLE
- DROP PROFILE
- DROP ZONE
- MONITOR
 - ABORTSESSION
 - MONRESOURCE
 - MONSESSION
 - SETRESRATE
 - SETSESSRATE
- ZONE OVERRIDE

Object-Level Privileges

Object-level privileges can be granted:

- On database objects (databases, tables, columns, maps, and other objects).
- To other users by the creator or owner of a database object.

For more information about privileges, see the following:

- [Privilege Dictionary](#)
- *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100

After the privileges are granted on the objects, users are able to perform the specific actions on the objects permitted by the privileges.

Row-Level Privileges

Access to database objects, for example, tables and views is primarily based on object-level user privileges. Object-level privileges provide basic access control, but are discretionary, that is, object owners automatically have the right to grant access on any owned object to any other user.

In addition to object-level privileges, you can use Teradata row-level security (RLS) to control user access for each table row, by SQL operation. RLS access rules are based on the comparison of the RLS access capabilities of each user and the RLS access requirements for each row.

Object owners do not have discretionary privileges to grant row access to other users. Only users with security constraint administrative privileges can manage row-level access controls.

Government agencies commonly create security labels (classifications) and use them to define user access capabilities and row access requirements.

Where Information About Row-Level Security Elements is Stored

Information about all of the current row-level security constraints, constraint assignments, and constraint functions is stored in the following Data Dictionary tables:

Information	Data Dictionary Table
Names and values of row-level security constraints	DBC.SecConstraints (names) DBC.ConstraintValues
Constraint functions for row-level security constraints and statement actions	DBC.ConstraintFunctions
Current assignment of row-level security constraint values	DBC.AsgdSecConstraints

Teradata Secure Zones Zone-Level Privileges

You cannot grant privileges on objects in a secure zone to users or roles in another zone. However, you can grant privileges to non-zone users or roles under certain conditions.

Related Information

For more information about privileges in general, see the following sources:

- *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100
- *Teradata Vantage™ - Database Administration*, B035-1093

For more information about row-level security and zone-level privileges, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

User Privilege Types

Implicit Privileges

Privilege	Description
Ownership	Vantage grants implicit privileges on database objects to the owner of the space that contains the objects.

Explicit Privileges

Privilege	Description
Automatic	When a user creates a database object, Vantage automatically grants privileges to: <ul style="list-style-type: none"> • The creator of the object • A newly created user or database
GRANT	You can GRANT privileges: <ul style="list-style-type: none"> • Directly to a user or database • To a role, then GRANT membership in the role to one or more users • To an external role, then map the role to one or more groups of directory users
Inherited	Privileges that a user acquires indirectly: <ul style="list-style-type: none"> • All users automatically have the privileges of PUBLIC, a role-like collection of default privileges. You can also grant or revoke privileges for PUBLIC. • A user inherits all the privileges granted to any roles of which the user is a member. • Directory users inherit the privileges of the database users and external roles to which they are mapped.
Assigned	Security constraints define user access to table rows protected by a corresponding security constraint column. You can assign the security constraints in a CONSTRAINT object to a: <ul style="list-style-type: none"> • User, by specifying the CONSTRAINT object in a: <ul style="list-style-type: none"> ◦ CREATE USER or MODIFY USER statement ◦ CREATE PROFILE or MODIFY PROFILE statement, and then assigning the profile to the user <p>Note: Constraint OVERRIDE privileges, which allow a user to bypass row level security protection, are granted using the GRANT OVERRIDE CONSTRAINT statement.</p> <ul style="list-style-type: none"> • Table, by defining a constraint column that is named for the CONSTRAINT object in a CREATE TABLE or ALTER TABLE statement.

For More Information

For information on syntax and the required privileges for a particular DCL statement, see [Statement Syntax](#).

For information on various user types and associated privilege strategies, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

Checking User Privileges

Before you explicitly grant any privilege to a user or role, you can access Data Dictionary views to determine which privileges are already in effect.

View	Information it Contains
AllRightsV	The explicit and automatic privileges in effect for each user.
AllRoleRights	The explicit privileges in effect for each role.
RoleMembersV	The members (users) for each role.

Vantage does not record privileges that derive from ownership, so they are not maintained in DBC.AllRightsV.

For more information about views, see *Teradata Vantage™ - Data Dictionary*, B035-1092.

Non-User Privileges

In some cases, you might need to assign privileges to a non-user object, for example:

- You can grant privileges to a role based on the database access requirements of a specific group of users, and then grant role membership to each of the users individually.
- If you set up a separate Views database, you must grant the database privileges on the underlying tables of the views contained within that database.

Statement Syntax

Introduction to Statement Syntax

These topics describe the DCL statements that you can use to grant and revoke the privileges supported by Vantage.

The following information is provided for each statement:

- Description
- Syntax
- Compliance with ANSI/ISO SQL standard
- Required privileges, that is, the privileges required to use the statement
- Example of use

For information about how to read a syntax diagram, see [Notation Conventions](#).

GIVE

Transfers ownership of a database or user space to another user. Also transfers all databases and users owned by the transferred database or user.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Other SQL dialects support similar non-ANSI standard statements with names such as the following:

TRANSFER OWNERSHIP

Required Privileges

You must have the DROP DATABASE privilege on the given object, and the CREATE DATABASE privilege on the recipient.

The GIVE statement does not revoke any explicit privileges on the given database or user. No explicit privileges on the given database or user are granted to the new ownership hierarchy as a result of the GIVE statement, nor does the database or user being given receive any explicit privileges.

The recipient of a GIVE statement cannot be owned by the given object; if A owns B, A cannot be given to B.

GIVE Syntax

```
GIVE { database_name | user_name } TO recipient_name [;]
```

Syntax Elements

database_name

user_name

Name of the database or user whose ownership is being transferred.

TO

Introduction to the name of the recipient.

recipient_name

Name of the new immediate owner for the transferred database or user.

For more information about using Teradata Secure Zones with GIVE, see [Using GIVE with Teradata Secure Zones](#).

See *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

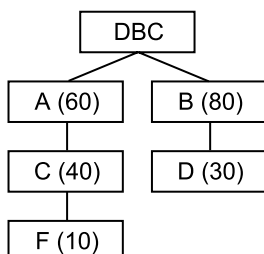
Usage Notes

Transfer of Space Allocation

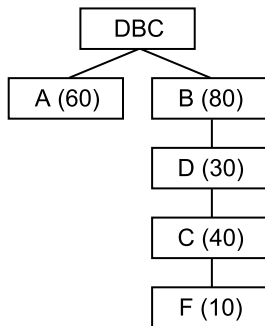
A transfer of ownership also transfers the permanent space allocated to the named database or user. This affects space allocation in the system as follows:

- The aggregate number of permanent space bytes available to the former owners (the owner who submits the GIVE statement plus the owners above this owner in the hierarchy) is reduced by the number of permanent space bytes in the transferred database. This includes total space allocated to the database, plus that of all databases and users owned by the transferred database.
- If the transferred database is dropped, then the number of bytes of permanent space allocated to the new immediate owner of the transferred database is increased by the number of bytes of permanent space in the transferred database. In addition, the aggregate number of bytes of permanent space available to owners above this owner in the hierarchy is increased by the number of bytes of permanent space that had been allocated to the transferred database.

For example, consider the following hierarchy:



If ownership of database *C* is transferred to database *D*, the structure of the hierarchy changes as follows:



When database *C* is transferred:

- Database *F*, which is owned by *C*, is also transferred.
- The number of permanent space bytes allocated to database *A* remains the same, but the aggregate number of permanent space bytes available to database *A* is reduced by 50 (the total number of permanent space bytes allocated to databases *C* and *F*).
- The number of bytes allocated to databases *D* and *B* remains the same.
- The available number of permanent space bytes, however, is increased by 50. That is, if databases *C* and *F* were dropped, the bytes allocated to *C* and *F* are transferred to database *D*.
- *A* no longer has implicit privileges on *C* and *F*.
- *B* and *D* now have implicit privileges on *C* and *F*.
- Explicit privileges held by any of the databases or users do not change.

For example, if *A* had granted itself explicit privileges on *C*, or *F*, or on objects they contain, *A* would still have those explicit privileges after submitting the GIVE statement.

Using GIVE with Teradata Secure Zones

A user with the appropriate privileges can transfer the ownership of a database from a non-zone user to a zone user, or from one zone user to another user within the same zone. However, a user cannot GIVE a zone database or a zone user to a non-zone user, or to another zone user in a different zone.

If you use Teradata Secure Zones in your database system, and you GIVE databases owned by non-zone users to zone users, only the ownership of the given databases change. Non-zone users retain their existing privileges on the given databases and can continue to access them unless you explicitly revoke their privileges.

If you want to move non-zone objects into a zone, you can GIVE a user or database and all their descendants to the zone root. You also need to review all of the non-zone users who have rights on the database or user objects that you moved and grant them zone privileges, if you want them to maintain their access to the objects. Before you grant them zone privileges, you should make them zone guests.

To complete the move, you also need to contact your Professional Services or Teradata Services representative for help in updating certain dictionary tables. The appropriate tables must be updated to

move the profiles and roles used by the users and descendants that you moved and to make the access rights on the objects zone-specific.

NOTICE

To ensure that the system functions properly, do not modify or delete any Data Dictionary tables yourself.

Example of Transferring the Ownership of a Database

The following statement transfers ownership of the finance database from user administrator to user Chin.

```
GIVE Finance TO Chin;
```

GRANT

GRANT establishes explicit privileges for one or more users, proxy users, databases, or roles. GRANT has several forms that differ in function and syntax:

GRANT Form	Purpose
GRANT (Monitor Form)	Performance monitoring of Vantage.
GRANT (Role Form)	Grant role membership to users and other roles.
GRANT (SQL Form)	Grant access to, creation of, or logging of, database objects.
GRANT MAP	Grant existing contiguous or sparse maps to users and roles.
GRANT ZONE	Grant zone guest status to users or roles that do not belong to any zone. GRANT ZONE does not automatically grant users access to database objects within the zone. Zone users must grant privileges to zone guests before access is permitted.
GRANT CONNECT THROUGH	Grant the ability to connect as a proxy permanent or proxy application user through a trusted user.
GRANT LOGON	Grant system logon privileges.

Using GRANT (SQL Form) and GRANT (MONITOR Form)

The GRANT (SQL Form) controls access to, and manipulation of, database objects, while the GRANT (MONITOR form) privilege set relates to monitoring system-wide performance. To grant a user all privileges, including MONITOR, you must perform both of the following requests:

```
GRANT ALL PRIVILEGES ON object
TO user
WITH GRANT OPTION;
```

```
GRANT MONITOR PRIVILEGES
TO user
WITH GRANT OPTION;
```

ALL PRIVILEGES refers only to database-related privileges. MONITOR PRIVILEGES indicates all monitoring-related privileges.

GRANT MONITOR does not have an ON object clause. Because this statement allows a user to impact the entire system, the permissions are implicitly ON PUBLIC.

GRANT ZONE controls access to a specific zone. The WITH GRANT OPTION privilege is not valid for use with GRANT ZONE.

You must specify an ON object clause in a GRANT (SQL Form) request. You cannot specify an ON object clause in any GRANT (MONITOR form) or GRANT ZONE request.

GRANT (Monitor Form)

Grants system-wide performance monitoring privileges.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must have MONITOR privileges to use the monitor form of GRANT.

These privileges should be granted only to those users who are cleared to monitor all applications on all sessions.

There is no lower level of MONITOR privilege: its scope is always global. For example, the database administrator cannot grant user *Addams* the ability to do session-level monitoring of her applications only. Instead, the DBA would have to grant *Addams* the permission to do session-level monitoring of all applications by all sessions.

To determine who is currently using the MONITOR partition, issue the following query:

```
SELECT UserName, IFPNo
FROM DBC.SessionInfoV
WHERE partition = 'MONITOR';
```

The GRANT statement is used only to assign specific privileges.

To transfer ownership of a database or user, see [GIVE](#).

GRANT Syntax (Monitor Form)

```
GRANT {
  MONITOR [ PRIVILEGES | BUT NOT monitor_privilege [,...] ] |
  monitor_privilege [,...]
}
TO {
  { grantee [,...] | PUBLIC } [ WITH GRANT OPTION ] |
  role_name [,...]
} [;]
```

Syntax Elements

MONITOR PRIVILEGES

The named recipients are to receive all MONITOR-related privileges. MONITOR PRIVILEGES does not permit the user to grant the indicated privilege to others without the WITH GRANT OPTION being specified.

MONITOR BUT NOT

The named recipients are to receive all of the grantable privileges except those specified after BUT NOT. If the ability to grant these privileges is to be included, the WITH GRANT OPTION must be specified explicitly.

monitor_privilege

A valid monitoring privilege.

See [Monitor Privileges](#) for a list of the valid monitoring privileges.

grantee

```
[ALL] user_name
```

role_name

ALL

Grants the specified object privilege set to the named database or user and to every database or user owned by that database or user now and in the future.

user_name

The name of a user or database to be granted the specified MONITOR privileges. You can specify up to 25 names.

user_name must be the identifier of a user already defined to the system.

PUBLIC

Indicates that the privileges are to be inherited by all existing and future database users.

WITH GRANT OPTION

Indicates that the grantee receives privileges WITH GRANT OPTION. If this option is not specified, the grantee receives the privilege set without the grant option.

Usage Notes

Monitor Privileges

GRANT MONITOR takes effect immediately when the grantee issues his next statement. It is unnecessary to log out to receive the monitor privilege just granted.

GRANT MONITOR does not have an ON object clause. Because this statement allows a user to impact the entire system, the permissions are implicitly ON PUBLIC.

You can specify any of the following privileges using the GRANT (Monitor Form) statement.

Option	Description
ABORTSESSION	Aborts any outstanding request or ongoing transaction of one or more Vantage sessions and, optionally, logs off the sessions.
MONRESOURCE	Gathers information on the performance and availability of each PE and AMP.
MONSESSION	Gathers information about logged on sessions and overall system usage on a session-by-session basis.
SETRESRATE	Sets the frequency at which processor resource usage data is updated in the system.
SETSESSRATE	Sets the frequency at which session-level performance data is updated in the system.

For information about revoking privileges granted by the Monitor form of GRANT, see [REVOKE \(Monitor Form\)](#).

Security Logging of Access Attempts

To maintain a security log of access attempts, see the information about the BEGIN LOGGING statement in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

GRANT (Role Form)

Grants roles to users or other roles.

Note:

You cannot grant external roles with this GRANT statement. You can only grant individual privileges and database roles to external roles within Vantage.

For more information, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100 and *Teradata Vantage™ - Database Administration*, B035-1093.

ANSI Compliance

This statement is ANSI SQL:2011 compliant.

Required Privileges

To grant a role, you must have the WITH ADMIN OPTION privilege on the role. The following users can grant a role to a user or other role:

- User DBC.
- A user who has been granted the specified role WITH ADMIN OPTION.

The creator of a role is automatically granted the specified role WITH ADMIN OPTION.

- A user who has an *active* role to which the specified role was granted WITH ADMIN OPTION. An active role can be a current role or a nested role of a current role.

A grantor does not need to have any privilege, including WITH ADMIN OPTION, on the grantee to grant a privilege to it, whether the grantee is a role or a user.

GRANT Syntax (Role Form)

```
GRANT role_name [, ...]
  TO { user_name | role_name } [, ...]
  [ WITH ADMIN OPTION ] [;]
```

Syntax Elements

role_name

One or more comma-separated names of roles to grant to specified users or other roles.

The system ignores duplicate role names.

TO***user_name******role_name***

The names of role grantees.

You can specify a maximum of 25 names per GRANT request.

Grantees can be users or roles; however, a role cannot be granted to itself or to PUBLIC.

GRANT does not produce an error if a specified role is already granted to a grantee.

WITH ADMIN OPTION

The role grantees have the privilege to use DROP ROLE, GRANT, and REVOKE statements to administer the specified roles.

To change a privilege previously granted WITH ADMIN OPTION, a GRANT statement must include WITH ADMIN OPTION.

Usage Notes for Roles

Roles are used to define privileges on database objects for multiple users. A user who is assigned a role can access all the objects on which the role and its nested roles have privileges. Users can only be assigned a role that has been granted to them.

You can grant a newly created role to a user or other role before the role has privileges on any database objects.

An unlimited number of roles can be granted to a role or user.

Roles cannot be granted on themselves or on PUBLIC, nor can they be granted any of the following privileges:

- CREATE PROFILE
- CREATE ROLE
- CREATE USER
- CREATE ZONE
- CTCONTROL
- DROP PROFILE
- DROP ROLE
- DROP USER
- DROP ZONE
- ZONE OVERRIDE

If you use Teradata Secure Zones to create secure zones, the role that you grant and the recipients of the role (users or other roles) should be in the same zone.

Roles can only be nested one level deep. Thus, a role that has a nested role cannot also be a nested role. This is a deviation from the ANSI/ISO SQL:2011 standard, which allows multiple nesting levels.

Examples of Using GRANT Role Form

The following statements create roles called services and sales:

```
CREATE ROLE services;
CREATE ROLE sales;
```

To make sales a nested role of services, use the following GRANT statement:

```
GRANT sales TO services;
```

To grant the sales role to user marks, and give marks the privilege to add other members to sales, use the following GRANT statement:

```
GRANT sales TO marks WITH ADMIN OPTION;
```

Related Information

Topic	Reference Source
Granting privileges on database objects to roles and obtaining the CREATE ROLE system privilege	GRANT (SQL Form) .
Revoking privileges granted by the Role form of GRANT	REVOKE (Role Form)
Granting CONNECT THROUGH proxy connection privileges to a permanent or application user with a set of roles	GRANT CONNECT THROUGH
Assigning default roles to users	Information about CREATE USER and MODIFY USER in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
Changing the current role for a session	Information about SET ROLE in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
Granting individual privileges and database roles to external roles within Vantage.	<i>Teradata Vantage™ - Advanced SQL Engine Security Administration</i> , B035-1100 <i>Teradata Vantage™ - Database Administration</i> , B035-1093

GRANT (SQL Form)

Grants one or more explicit privileges on a database, user, proxy logon user, table, hash index, join index, view, procedure, User-Defined Function (UDF), function mapping, User-Defined Method (UDM), User-Defined Type (UDT), or macro to a role, group of roles, user, or group of users or databases.

For a list of database privileges, see [Database Privileges](#).

For information about granting the NONTEMPORAL privilege, see *Teradata Vantage™ - Temporal Table Support*, B035-1182.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must be one of the following to grant privileges on an object using the SQL GRANT statement:

- User DBC.
- An owner of the object.

For details regarding security issues associated with owner privileges, see [Security Considerations With the CREATE MACRO Privilege](#)

- A user possessing each privilege to be granted.

A user can have a privilege by having been granted it explicitly or by inheriting it from a role as a result of creating a view, macro, or procedure.

To grant the following privileges, you must have the CONSTRAINT ASSIGNMENT privilege.

- OVERRIDE DELETE CONSTRAINT
- OVERRIDE DUMP CONSTRAINT
- OVERRIDE INSERT CONSTRAINT
- OVERRIDE RESTORE CONSTRAINT
- OVERRIDE SELECT CONSTRAINT
- OVERRIDE UPDATE CONSTRAINT

Note the following details about the privileges required to execute a GRANT (SQL Form) request.

- A user need not be related to a grantor through ownership to receive a privilege.

A grantor does not need to have any privilege, including WITH ADMIN OPTION, on the grantee to grant a privilege to it, whether the grantee is a role, a user, database, or PUBLIC.

- If a GRANT statement is on a database or user, the privilege applies to all objects, both current and future, created in that space.

If a REVOKE statement later removes the privilege, the privilege is dropped for all objects, regardless of when they were created.

A REVOKE statement at the object level cannot remove a privilege from that object that was granted on the database or user.

- When you specify the WITH GRANT OPTION phrase, the recipient of the privilege can then grant that privilege to other users.

An owner implicitly has the WITH GRANT OPTION privilege on any database, user, or object it owns.

An owner can explicitly grant any or all privileges on any of the following	The Privilege Can Be Granted To
<ul style="list-style-type: none"> a child database a child user a database object 	<ul style="list-style-type: none"> any other database any other user a role PUBLIC <p>You cannot assign row-level security privileges to PUBLIC.</p>

- Any privilege granted automatically or explicitly can be revoked using the REVOKE statement.

See [REVOKE \(SQL Form\)](#).

- Implicit privileges cannot be revoked.

When Privileges Are Granted

GRANT takes effect immediately when the grantee issues their next statement. You do not need to log out to receive a privilege that was just granted to you.

GRANT Syntax (SQL Form)

```

GRANT {
  { ALL [ PRIVILEGES ] |
    [ ALL BUT ] privilege [,...] |
    CTCONTROL
  } ON object |

  map_privilege [,...] |

  role_privilege [,...] |

  profile_privilege [,...] |

  zone_privilege [,...] |

  CONSTRAINT ASSIGNMENT |

  CONSTRAINT DEFINITION
}
TO {

```

```

    { grantee [,...] | PUBLIC } [ WITH GRANT OPTION ] |
    role_name [,...]
  } [;]

```

object

```

  { { database_name | user_name } [ .object_name ] |

    object_name |

    [ PROCEDURE ] [ database_name. | user_name. ] procedure_name |

    SPECIFIC FUNCTION [ database_name. | user_name. ]
      specific_function_name |

    [ FUNCTION ] [ database_name. | user_name. ]
      function_name ( [ function_parameter [,...] ] ) |

    TYPE [SYSUDTLIB.] UDT_name
  }

```

grantee

```

  [ ALL ] { database_name | user_name }

```

function_parameter

```

  [ parameter_name ] data_type

```

data_type

```

  { INTEGER | SMALLINT | BIGINT | BYTEINT | DATE |

    { TIME | TIMESTAMP } [ (fractional_seconds_precision) ] [WITH TIME
    ZONE] |

    INTERVAL YEAR [(precision)] [TO MONTH] |

    INTERVAL MONTH [(precision)] |

    INTERVAL DAY [(precision)]
  }

```

```

    [TO { HOUR | MINUTE | SECOND [(fractional_seconds_precision)] }] |

INTERVAL HOUR [(precision)]
    [TO { MINUTE | SECOND [(fractional_seconds_precision)] }] |

INTERVAL MINUTE [(precision)] [TO SECOND
[(fractional_seconds_precision)]] |

INTERVAL SECOND [ (precision [, fractional_seconds_precision ] ) |

PERIOD (DATE) |

PERIOD ( { TIME | TIMESTAMP } [(precision)] [WITH TIME ZONE] ) |

REAL |

DOUBLE PRECISION |

FLOAT [ (integer) ] |

NUMBER [ ( { integer | * } [, integer]... ) ] |

{ DECIMAL | NUMERIC } [ ( integer [, integer]... ) ] |

{ CHAR | BYTE | GRAPHIC } [ (integer) ] |

{ VARCHAR | CHAR VARYING | VARBYTE | VARGRAPHIC } [ (integer) ] |

LONG VARCHAR |

LONG VARGRAPHIC |

{ BINARY LARGE OBJECT | BLOB | CHARACTER LARGE OBJECT | CLOB }
    ( integer [ G | K | M ] ) |

[SYSUDTLIB.] { XML | XMLTYPE } [ ( integer [ G | K | M ] ) ]
    [ INLINE LENGTH integer ] |

[SYSUDTLIB.] JSON [ ( integer [ K | M ] ) ] [ INLINE LENGTH integer ]
    [ CHARACTER SET { UNICODE | LATIN } | STORAGE FORMAT { BSON |
UBJSON } ] |

[SYSUDTLIB.] ST_GEOMETRY [ (integer [ K | M ]) ] [ INLINE LENGTH

```

```

integer ] |

[SYSUDTLIB.] DATASET [ (integer [ K | M ]) ]
[ INLINE LENGTH integer ] [storage_format] |

[SYSUDTLIB.] { UDT_name | MBR | ARRAY_name | VARRAY_name }
}

```

Syntax Elements

ALL PRIVILEGES

User or database receives all privileges that can be granted on the specified object. GRANT ALL means that all implicit and explicit object privileges owned by the grantor WITH GRANT OPTION that pertain to the type of object, and only those privileges, are granted on the specified database object.

To include the ability for the designated user to GRANT object privileges to other users or databases, specify WITH GRANT OPTION.

Note:

ALL cannot be granted to roles.

An error is returned if the grantor has no privileges WITH GRANT OPTION on the object.

You must use the monitor form of the GRANT statement to grant monitor privileges.

See [GRANT \(Monitor Form\)](#).

privilege

One of the privileges listed under the topic [Privilege Dictionary](#) or one of the keywords for a privilege combination.

See [Granting Multiple Privileges With a Single Keyword](#) for a list of the combination privilege keywords.

INSERT, REFERENCES, SELECT, and UPDATE privileges have table-level and column-level options.

See [REFERENCES Privilege](#).

The DELETE privilege applies only to the DELETE DML statement, not to DELETE USER or DELETE DATABASE, which are granted by DROP USER and DROP DATABASE, respectively.

See also [Rules for privilege Keywords](#).

For an explanation of the privileges a specific statement requires, see Required Privileges section in the appropriate volume of the SQL book set.

ALL BUT *privilege*

User is to receive all privileges that can be granted on the specified object except for those specified in the privilege list. As in ALL, only those object privileges owned by the grantor WITH GRANT OPTION are granted.

ALL BUT is a Teradata extension to the ANSI/ISO SQL:2011 standard.

Granting privileges on a database or user is a Teradata extension to the ANSI/ISO SQL:2011 standard.

database_name

user_name

Name of the database or user to which the privilege set is granted.

All objects contained by the specified database or user space are granted the specified privilege set.

database_name.object_name

user_name.object_name

Name of the immediate owning database or user for the specified database object and the name of the database object (table, view, procedure, or macro) to which the privilege set is granted. Only the specified database object is granted the specified privilege set.

Name of the immediately owning database or user and the name of the object (table, view, procedure, or macro) on which the privileges are to be granted. Only the named object is affected.

object_name

Name of a database object (table, join index, hash index, view, procedure, UDF, UDM, UDT, macro, or authorization name) on which the privileges set is granted.

If the form of the privileges option includes a set of column names, then *object_name* must specify either a table or view.

You should always qualify object names when granting privileges because Vantage checks for matching database names before checking for object names.

- If the object name is not qualified and the system finds a database with that name, then Vantage assumes the name is a database name.
- If the object name is not qualified and no database having that name is found, then Vantage assumes the object is within the current default database.
- If the object name is not qualified by either a database name or user name and there is an object with the same name under the current database of the executing user and that

of the grantee, then the object is assumed to be in the current database of the executing user. The only exception to this is for all objects related to UDTs, including UDT-related UDFs and methods, all of which must be contained within the SYSUDTLIB database.

- An unqualified object name is considered to be that of the current database if the name is that of the current database and also the name of a table either within the current database or within the database of the grantee.
- If neither a database nor an object is found with the specified name, then the system aborts the request and returns an error to the requestor.

PROCEDURE

database_name

user_name

procedure_name

Object is a procedure.

PROCEDURE is optional if the privilege being granted contains the PROCEDURE keyword.

See [Usage Notes for Procedure-Specific Privileges](#).

database_name and *user_name* are optional. You can qualify *procedure_name* by its containing *database_name* or *user_name*.

You can grant the following privileges if you specify PROCEDURE:

- If you specify EXECUTE, you grant the EXECUTE PROCEDURE privilege.
- If you specify DROP, you grant the DROP PROCEDURE privilege.

You must grant the ALTER EXTERNAL PROCEDURE and CREATE EXTERNAL PROCEDURE privileges explicitly.

SPECIFIC FUNCTION

database_name

user_name

specific_function_name

Name of the function on which a privilege set is to be granted.

You must specify the SPECIFIC FUNCTION keywords when a privilege is granted on an overloaded function.

database_name and *user_name* are optional. You can qualify *specific_function_name* by its containing *database_name* or *user_name*.

FUNCTION

database_name

user_name

function_name

Name of the UDF on which a privilege set is to be granted.

FUNCTION is optional if the privilege being granted contains the FUNCTION keyword.

The FUNCTION keyword must be specified when a privilege is granted on a UDF by its calling name.

database_name and *user_name* are optional. You can qualify *function_name* by its containing *database_name* or *user_name* if necessary.

parameter_name

data_type

Parenthetical comma-separated list of data types and optional parameter names for the variables to be passed to the UDF. The data types are required in order to differentiate overloaded functions with the same name.

BLOB and CLOB types must be represented by a locator. For a description of locators, see the information about the USING request modifier in *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146. Vantage does not support in-memory LOB parameters: an AS LOCATOR phrase must be specified for each LOB parameter and return value.

You must specify opening and closing parentheses even if no parameters are passed to the function.

parameter_name is optional. If you specify one parameter name, then you must specify names for all the parameters passed to the function. See the information about HELP FUNCTION in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

The data type associated with each parameter is the type of the parameter or returned value. All Teradata data types are valid. Character data can also specify a CHARACTER SET clause.

TYPE SYSUDTLIB.

UDT_name

Name of a UDT on which a privilege set is to be granted.

SYSUDTLIB is optional. If you specify a containing database for *UDT_name*, you must specify *SYSUDTLIB*.

The TYPE privileges that you can grant to a UDT are listed in the following list:

- You can grant UDTMETHOD only on the SYSUDTLIB database.
- You can grant UDTTYPE only on the SYSUDTLIB database.
- You can grant UDTUSAGE on the SUSUDTLIB database, on TYPE, or on both.

map_privilege

Grant create and drop map privileges to users and roles.

You can specify the following privileges:

- CREATE MAP
- DROP MAP
- MAP

MAP is shorthand, not a separate privilege. Specify MAP to grant both the CREATE MAP and DROP MAP privileges.

role_privilege

One of the following role privileges:

- CREATE ROLE
- DROP ROLE
- ROLE

The following request allows user Franklin to create and drop roles:

```
GRANT ROLE
TO Franklin;
```

This form does not have an ON clause. If you specify an ON clause for the role privilege, the system returns an error.

ROLE is shorthand, not a separate privilege. Specify ROLE to grant both the CREATE ROLE and DROP ROLE privileges.

Role privileges cannot be granted to a role or to PUBLIC.

profile_privilege

One of the following privileges:

- CREATE PROFILE
- DROP PROFILE
- PROFILE

PROFILE is shorthand, not a separate privilege. Specify PROFILE to grant both the CREATE PROFILE and DROP PROFILE privileges.

Profile privileges can only be granted to a set of users or to a role. They cannot be granted on an object.

You can assign row-level security privileges to a profile and then map a directory user to that profile. The constraints apply to that directory user even if the user is not mapped to a permanent user. See *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100, for more information.

zone_privilege

One of the following privileges:

- CREATE ZONE
- DROP ZONE
- ZONE

ZONE is shorthand, not a separate privilege. Specify ZONE to grant both the CREATE ZONE and DROP ZONE privileges.

Zone privileges cannot be granted to a role.

See *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

database_name

user_name

role_name

PUBLIC

Name of a database, user, role, or PUBLIC on which the explicitly granted privileges are to be granted. All objects contained by this database or user space are affected.

You cannot grant row-level security privileges to PUBLIC.

database_name

user_name

Name of a database or user on which the privileges are to be granted. All objects in this database or user space are affected.

database_name.object_name

user_name.object_name

Name of the immediately owning database and optionally the name of the object or the name of the user and the name of the object (table, view, procedure, or macro) on which the privileges are to be granted. Only the named object is affected.

CONSTRAINT ASSIGNMENT

System-level CONSTRAINT ASSIGNMENT privilege.

CONSTRAINT ASSIGNMENT enables a user to maintain row-level security constraint object assignments to users, profiles, and tables such as ALTER TABLE, CREATE PROFILE, CREATE TABLE, CREATE USER, MODIFY PROFILE, and MODIFY USER.

You can only grant CONSTRAINT ASSIGNMENT to specific users or roles, not to tables or databases.

You cannot grant CONSTRAINT ASSIGNMENT to PUBLIC.

See [System-Level Privileges for Row-Level Security](#) for more information.

CONSTRAINT DEFINITION

System-level CONSTRAINT DEFINITION privilege.

CONSTRAINT DEFINITION enables a user to create and maintain row-level security constraints, such as ALTER CONSTRAINT, CREATE CONSTRAINT, and DROP CONSTRAINT.

You can only grant CONSTRAINT DEFINITION to specific users or roles, not on tables or databases.

You cannot grant CONSTRAINT DEFINITION to PUBLIC.

See [System-Level Privileges for Row-Level Security](#) for more information.

CTCONTROL

System-level CTCONTROL privilege.

CTCONTROL authorizes a user to grant or revoke the CONNECT THROUGH privilege (see [GRANT CONNECT THROUGH](#)) using the GRANT CONNECT or REVOKE CONNECT statements.

In the ON clause, you can only grant CTCONTROL to users, not to other types of database objects.

You cannot grant CTCONTROL to PUBLIC.

Use care when granting the CTCONTROL privilege, because this privilege applies to all database objects in the system.

See [CTCONTROL Privilege](#).

ALL

user_name

database_name

Name of a database or user that identifies the recipient. *user_name* must be the identifier of a user already defined to the system.

You can specify a maximum of 25 names per GRANT request.

ALL is optional. If you specify ALL, then the object privileges are granted to the named database or user and to every database or user owned by that database or user now and in the future.

ALL *user_name* is a Teradata extension to the ANSI/ISO SQL:2011 standard.

The following statement allows all current and future users created under the personnel database to select data from the department table:

```
GRANT SELECT
ON personnel.department
TO ALL personnel;
```

PUBLIC

Privileges are inherited by all existing and future Vantage users and databases.

For example:

```
GRANT SELECT
ON personnel.department
TO PUBLIC;
```

WITH GRANT OPTION

Grantee receives the granted privileges WITH GRANT OPTION.

This option does not apply to grantees that are roles or who are zone guests in a zone.

A GRANT statement that specifies WITH GRANT OPTION privilege when there is a role or a zone guest in the list of grantees aborts and returns an error message.

WITH GRANT OPTION applies to the indicated privileges only.

In this example, user George grants all privileges that he has on his own user space to user Marston with the following restrictions:

- George can grant only those privileges for which he has WITH GRANT OPTION at the user level.
- George must have the WITH GRANT OPTION privilege on the privileges he grants to Marston.

```
GRANT ALL
ON marston
TO marston
WITH GRANT OPTION;
```

database_name

user_name

Containing database or user for *role_name*, if different from the current database or user.

role_name

Name of an existing role, which can be a Vantage role or an EXTERNAL role.

Note:

Row-level security constraint values cannot be assigned to roles. They can only be assigned to users and profiles.

These privileges can be granted to databases, users, and roles, except for the USER privileges, which cannot be granted to roles:

- CREATE DATABASE
- CREATE FUNCTION
- CREATE MACRO
- CREATE TABLE
- CREATE VIEW
- CREATE PROCEDURE
- CREATE USER (cannot be granted to roles)
- DROP DATABASE
- DROP USER (cannot be granted to roles)

Usage Notes

Granting Multiple Privileges With a Single Keyword

Teradata has a special category of keywords that you can use to grant multiple privileges. For example, the following GRANT request grants both the CREATE DATABASE and DROP DATABASE privileges to *user_name* using the keyword DATABASE:

```
GRANT DATABASE on user_name;
```

The following table lists this class of keywords and indicates the multiple privileges that they confer when they are granted on a user or database:

Keyword	Privileges Conferred
ALL	all implicit and explicit object privileges owned by the grantor WITH GRANT OPTION that pertain to the type of object specified, and only those privileges, are granted on the specified database object.
AUTHORIZATION	<ul style="list-style-type: none"> • CREATE AUTHORIZATION • DROP AUTHORIZATION
CHECKPOINT	<ul style="list-style-type: none"> • ability to execute the CHECKPOINT SQL statement. • ability to execute the HUT CHECKPOINT command.
DATABASE	<ul style="list-style-type: none"> • CREATE DATABASE

Keyword	Privileges Conferred
	<ul style="list-style-type: none"> • DROP DATABASE
DROP	<ul style="list-style-type: none"> • If the ON clause specifies FUNCTION, then the privilege granted is DROP FUNCTION. • If the ON clause specifies PROCEDURE, then the privilege granted is DROP PROCEDURE.
EXECUTE	<ul style="list-style-type: none"> • If the ON clause specifies FUNCTION, then the privilege granted is EXECUTE FUNCTION. • If the ON clause specifies PROCEDURE, then the privilege granted is EXECUTE PROCEDURE. • If the ON specifies nothing, then the privilege granted is EXECUTE MACRO.
FUNCTION	<ul style="list-style-type: none"> • CREATE FUNCTION • DROP FUNCTION
GLOP	<ul style="list-style-type: none"> • CREATE GLOP • DROP GLOP
INDEX	<ul style="list-style-type: none"> • CREATE INDEX • DROP INDEX
MACRO	<ul style="list-style-type: none"> • CREATE MACRO • DROP MACRO
MAP	<ul style="list-style-type: none"> • CREATE MAP • DROP MAP
PROCEDURE	<ul style="list-style-type: none"> • CREATE PROCEDURE • DROP PROCEDURE
PROFILE	<ul style="list-style-type: none"> • CREATE PROFILE • DROP PROFILE
RESTORE	<p>ability to execute the following HUT commands on the specified object:</p> <ul style="list-style-type: none"> • DELETE JOURNAL • ROLLBACK • ROLLFORWARD
ROLE	<ul style="list-style-type: none"> • CREATE ROLE • DROP ROLE
SHOW	<p>the ability to execute the following SQL statements <i>only</i>:</p> <ul style="list-style-type: none"> • HELP <i>database_object</i> • SHOW <i>database_object</i>
STATISTICS	<ul style="list-style-type: none"> • COLLECT STATISTICS • DROP STATISTICS

Keyword	Privileges Conferred
TABLE	<ul style="list-style-type: none"> • CREATE TABLE • DROP TABLE
TRIGGER	<ul style="list-style-type: none"> • CREATE TRIGGER • DROP TRIGGER
USER	<ul style="list-style-type: none"> • CREATE USER • DROP USER
VIEW	<ul style="list-style-type: none"> • CREATE VIEW • DROP VIEW

Data Dictionary Privilege Abbreviations

See [Privilege Dictionary](#) for a complete list of privileges and their abbreviations as maintained in *DBC.AccessRights*. You cannot use these abbreviations in a GRANT (SQL Form) request. You must specify the complete privilege name.

Rules for *privilege* Keywords

The following rules apply to using the *privilege* keywords:

- You can specify any combination of privileges. However, the user submitting the statement must have the specified privileges in order to grant them.
- The recipient of a privilege can perform the corresponding statement on the object for which the privilege was granted.

For example, if *user_1* receives the CREATE TABLE privilege on database *DbTest*, *user_1* can then perform a CREATE TABLE statement in which the new table is directed to *DBTest*, where the target database is resolved either implicitly, as determined by the default database for *user1*, or explicitly with a fully qualified table name.

- The CREATE DATASET SCHEMA privilege can only be granted at the database level. DROP DATASET SCHEMA and WITH DATASET SCHEMA can be granted at the database or at the individual schema level.
- The STATISTICS privilege authorizes COLLECT STATISTICS statements. STATISTICS can be granted at the table and database levels.
- The CHECKPOINT privilege refers to the execution of the SQL statement and the Host Utilities (HUT) command.

The DUMP and RESTORE privileges refer to the corresponding HUT command performed on the specified object.

RESTORE also refers to execution of the HUT commands ROLLBACK, ROLLFORWARD, and DELETE JOURNAL.

- The CREATE forms of DATABASE, FUNCTION, MACRO, TABLE, VIEW, PROCEDURE, or USER, and the DROP forms of DATABASE and USER are allowed only on databases or users.
- DROP TABLE authorizes the ALTER TABLE, CREATE INDEX, and the DROP INDEX statements on the tables.
- DROP MACRO, PROCEDURE, or VIEW includes REPLACE MACRO, PROCEDURE, or VIEW.
- Only DROP MACRO and EXECUTE are allowed on macros.
- Only ALTER PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, and CREATE OWNER PROCEDURE are allowed on procedures.
- CREATE OWNER PROCEDURE is an explicit privilege that must be granted explicitly to a user or database.
- ALTER EXTERNAL PROCEDURE is required to change the execution state of an external procedure between the PROTECTED and NOT PROTECTED states using the ALTER PROCEDURE (External Form) statement.

See *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

- Only ALTER FUNCTION, DROP FUNCTION, and EXECUTE FUNCTION are allowed on UDFs.
- You cannot grant the ALTER FUNCTION privilege on a UDF.
- DUMP, RESTORE, INDEX, and CHECKPOINT are not allowed on views.
- DATABASE, FUNCTION, INDEX, GLOP, MACRO, PROCEDURE, PROFILE, ROLE, TABLE, USER, and VIEW confer CREATE and DROP privileges.

If the DATABASE, FUNCTION, INDEX, GLOP, MACRO, PROCEDURE, PROFILE, ROLE, TABLE, USER, or VIEW keyword is specified without CREATE or DROP, CREATE and DROP are assumed by default.

If the target of USER is a user, then USER confers CREATE and DROP privileges. If the target of USER is a database, then USER confers the privilege to CREATE users within the database.

- INDEX authorizes the CREATE INDEX and DROP INDEX statements on the tables.
- INSERT, REFERENCES, SELECT, and UPDATE have both table-level and column-level options.
- You cannot grant the UPDATE privilege on a GENERATED BY DEFAULT identity column.
- You cannot grant privileges on a trigger, only on the database or table to which the trigger applies.

The CREATE TRIGGER and DROP TRIGGER privileges are granted to a user, either on the specified database or on the subject table of the trigger.

- You can grant CREATE PROFILE, DROP PROFILE, CREATE ROLE, DROP ROLE, CREATE ZONE, DROP ZONE, and ZONE OVERRIDE privileges to users only, not to roles, databases, or PUBLIC.
- CREATE PROFILE, DROP PROFILE, CREATE ROLE, DROP ROLE, CREATE ZONE, DROP ZONE, and ZONE OVERRIDE are all system-level privileges; you grant the privileges to a user, but not on a specific object.

- The ZONE privilege (shorthand for the CREATE ZONE and DROP ZONE privileges) cannot be combined with any other privilege when you use GRANT. Similarly, the ZONE OVERRIDE privilege cannot be combined with any other privilege when you use GRANT. For example, you cannot use syntax such as GRANT ZONE, ZONE OVERRIDE to u1; or GRANT ROLE, ZONE to u1; these commands fail.
- A GRANT or REVOKE request for the CONSTRAINT ASSIGNMENT, CONSTRAINT DEFINITION, and CTCONTROL system privileges can only be submitted by user *DBC* or by a user who has previously been granted the privilege WITH GRANT OPTION.

You cannot grant these privileges to PUBLIC or to EXTUSER.

- The name of the user or role you specify when you grant the CONSTRAINT ASSIGNMENT, CONSTRAINT DEFINITION, or CTCONTROL privilege must be a unique identifier assigned to an existing user or role. It cannot be PUBLIC or EXTUSER.
- The SHOW privilege enables a user to view the definition (using a SHOW *object_type* request), or request help about definitions and other information (using a HELP *object_type* request) for a database object without also being able to change the definition of the object or retrieve rows from it.

SHOW, which must be granted explicitly, can be granted at the object and database levels. The creator of an object does not receive the SHOW privilege automatically on the object it creates. User *DBC* can also grant this privilege on the dictionary tables to other users/databases.

SHOW CONSTRAINT requires CONSTRAINT ASSIGNMENT or CONSTRAINT DEFINITION privilege.

You can use SHOW to provide access to statements such as SHOW TABLE, HELP TABLE, and HELP STATISTICS that require any privilege of any kind. All statements that require any privilege can also use SHOW.

This privilege enables DBAs to grant a user the ability to SHOW a table or view, or do HELP STATISTICS, and so on without having SELECT access to a table.

Differences Between GRANT and GIVE

The GRANT statement is used only to assign specific privileges, while the GIVE statement transfers the ownership of a database or user to another database or user.

For more information, see [“GIVE”](#).

REFERENCES Privilege

The REFERENCES privilege applies at both the table and column levels. It is required on all columns referenced, whether implicitly or explicitly, in a FOREIGN KEY clause of a CREATE TABLE or ALTER TABLE statement. The system grants this privilege automatically, with grant authority, to the creator of a table. Owners acquire the privilege implicitly.

If you specify a column list (a list of parenthetically enclosed, comma-separated column names, for example (column_name_1, ..., column_name_n)) with the keywords INSERT, REFERENCES, SELECT, or UPDATE, and the specified grantee set already has the privilege specified at the table level with the indicated grant authority, then the system accepts the statement but takes no action. This is because having INSERT, REFERENCES, SELECT, or UPDATE privileges at table level also enables that action against all of its columns.

INDEX Privilege

The INDEX privilege only applies at table level. You must have either this privilege or DROP TABLE to perform the CREATE INDEX or DROP INDEX statements. The system implicitly gives this privilege, with grant authority, to the immediate owner of the table at the time of creation.

Column-Level Privileges

The DBC.AccessRights table stores the field ID of a column on which an INSERT, REFERENCES, SELECT, or UPDATE privilege has been granted. The column type is SMALLINT and cannot be null. If the privilege defined by the row is not at the column level then the value of the column is set to zero.

Column-level privileges are held implicitly by the owners of a table. As is always true for implicit privileges, a row is not inserted into DBC.AccessRights for implicit column-level privileges.

Rows are also not generated in DBC.AccessRights for individual columns when the INSERT, REFERENCES, SELECT, or UPDATE privileges are granted at the table level.

Rows are generated in DBC.AccessRights only for privileges granted by a GRANT statement at the column level when the grantee does *not* have the privilege at the table level. The only exception is the case of a user or database that has a privilege at table level but without GRANT authority and who then is granted the same privilege on an individual column of the same table *with* grant authority.

Privileges Required for Modifying Data

Any user who needs to perform any form of update, delete, or insert operations, including the insert and update operations performed by a MERGE request, must be granted the SELECT privilege on all columns that are read by the delete, insert, merge, or update request from which that user needs to read values.

For the INSERT statement, SELECT privileges are required only if a table is to be reflexively grown by INSERT ... SELECT statements performed on itself.

If the column being updated, deleted, or inserted into has a UDT type, then you must also have the UDTUSAGE or UDTTYPE privilege on that UDT.

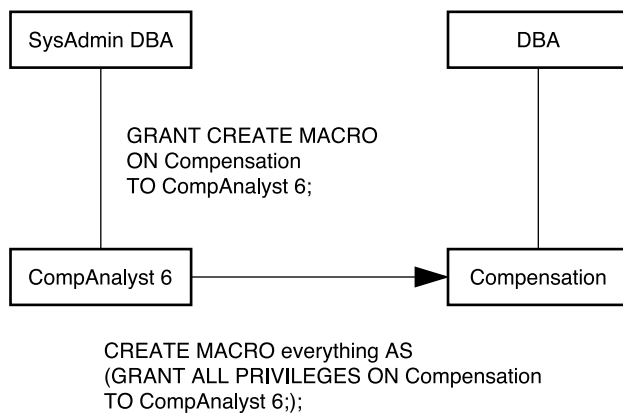
Be careful when granting the delete privilege through a view because data in columns that are not defined in the view (and which might be important to the enterprise) might accidentally be deleted by the user when they delete a row through a view.

Security Considerations With the CREATE MACRO Privilege

Before you grant CREATE MACRO on a database or user, it is extremely important to realize that the recipient of that privilege can create and perform macros that have all the privileges of that database or user. This is because for CREATE MACRO, the privileges are inherited from the immediate owner of the macro, not from its creator.

As a result, the grantee can create macros that contain DCL and DDL statements that are not checked for the privileges of the creator. This means that you are implicitly assigning privileges to the macro creator that they have no explicit, implicit, or automatic privilege to perform. This might not be a desirable result and you should be exceedingly careful when granting this privilege.

For example, consider the scenario presented in the following graphic:



The *compensation* database is owned by user *DBA*.

User *SysAdminDBA*, the system administrator for *compensation*, has privileges on *compensation*, including CREATE MACRO WITH GRANT OPTION, and on all objects owned by *compensation*.

SysAdminDBA can also effectively grant herself any of the following:

- Privileges on objects owned by *compensation*.
- Privileges that *compensation* has WITH GRANT OPTION.
- Any implicit privileges owned by *compensation*.

SysAdminDBA creates user *CompAnalyst6* for an entry level programmer who has been assigned to produce compensation reports for several routine audits performed by various state and federal regulatory agencies. To ensure that *CompAnalyst6* does not have access to critical private employee base salary and bonus information, she has been granted only a restricted set of privileges on objects in the *compensation* database.

To make it easier for *CompAnalyst6* to create the reports, *SysAdminDBA* also grants her the CREATE MACRO privilege on *compensation* as follows:

```
GRANT CREATE MACRO
ON compensation
TO companalyst6;
```

Because the privileges for executing macros in *compensation* derive from *compensation*, *CompAnalyst6* can create and perform macros that report on just the sort of private data she was meant to be restricted from viewing.

For example, *CompAnalyst6* can grant herself full access to all tables in the database through a simple macro and then create any database object or perform a query that reports on salary and bonus data for each employee in the enterprise in the three quick steps outlined in the following procedure:

1. CREATE MACRO everything AS
 (GRANT ALL PRIVILEGES ON compensation
 TO companalyst6;);
2. EXECUTE everything;
3. SELECT *
 FROM salary, bonus;

CompAnalyst6 can also modify data and drop tables in the *compensation* database.

Logging Access Attempts

If you need to maintain a security log of access attempts, you can use the access logging feature to do so.

See the information about BEGIN LOGGING in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Restrictions on Granted Privileges

When you explicitly grant privileges to another user, database, role, or PUBLIC, there are rules that determine whether, how, and on what object the requested privilege can be implemented. The restrictions that apply to explicitly granted privileges are detailed in the following table.

The first column of the table lists the privilege type, the second column describes restrictions if the privilege is granted on a database, user, role, or PUBLIC, and the third column describes restrictions if the privilege is granted on a table, view, function, function mapping, procedure, method, UDT, or macro.

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
ALL	All privileges the grantor can grant on an object. <ul style="list-style-type: none"> Grants UDTMETHOD on the SYSUDTLIB database. 	The following are true only if the grantor owns the privilege: <ul style="list-style-type: none"> Grants EXECUTE and DROP on a macro.

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
	If granted WITH GRANT OPTION, permits grantee to grant others UDTMETHOD, UDTTYPE, and UDTUSAGE, optionally WITH GRANT OPTION.	<ul style="list-style-type: none"> • Grants DROP, DELETE, INDEX, INSERT, REFERENCES, SELECT, UPDATE, RESTORE, CREATE TRIGGER, DROP TRIGGER, and DUMP on a data table. • Grants DROP on hash and join indexes. • Grants DROP, DELETE, INSERT, SELECT, and UPDATE on a view. • Grants INSERT, DUMP, RESTORE, and CHECKPOINT on a journal table. • Grants ALTER PROCEDURE, DROP PROCEDURE, and EXECUTE PROCEDURE privileges on a procedure. • Grants ALTER FUNCTION, DROP FUNCTION, and EXECUTE FUNCTION privileges on a specified UDF. • Grants CREATE GLOP, GLOP, and DROP GLOP to a user. • Grants GLOP MEMBER to a specified user or database. • Grants NONTEMPORAL to a specified user, database, or table. <p>For more information, see <i>Teradata Vantage™ - Temporal Table Support</i>, B035-1182.</p>
ALTER FUNCTION ALTER PROCEDURE	Privilege applies to all external UDFs or internal procedures in the specified space. ALTER FUNCTION is not valid for UDFs.	Privilege applies to the specified external UDFs or procedures. ALTER FUNCTION is not valid for UDFs.
ALTER EXTERNAL PROCEDURE	Privilege applies to all external procedures in the specified space.	Privilege applies to the specified external procedure.
CHECKPOINT	Privilege applies to the journal table in the specified database.	Privilege applies to the named journal table.
CONSTRAINT ASSIGNMENT		<p>A system-level privilege granted to user DBC WITH GRANT OPTION. DBC can grant the privilege to any other user or role defined for your system.</p> <p>CONSTRAINT ASSIGNMENT applies to the following SQL statements used to maintain constraint assignments for profiles, tables, and users.</p> <ul style="list-style-type: none"> • ALTER TABLE

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
		<ul style="list-style-type: none"> • CREATE PROFILE • CREATE TABLE • CREATE USER • MODIFY PROFILE • MODIFY USER • SHOW CONSTRAINT
CONSTRAINT DEFINITION		<p>A system-level privilege granted to user DBC WITH GRANT OPTION. DBC can grant the privilege to any other user or role defined for your system.</p> <p>CONSTRAINT DEFINITION applies to executing the following SQL statements.</p> <ul style="list-style-type: none"> • ALTER CONSTRAINT • CREATE CONSTRAINT • DROP CONSTRAINT
CREATE DATABASE CREATE USER	CREATE granted for the specified space.	Not applicable.
CREATE DATASET SCHEMA	<p>Grants CREATE DATASET SCHEMA on the SYSUDTLIB database.</p> <p>A database-level privilege granted to user DBC WITH GRANT OPTION. DBC must explicitly grant it with or without grant option to any other users or databases that are created. DBC must explicitly grant it without grant option to any role that is created.</p>	Not applicable.
CREATE EXTERNAL PROCEDURE CREATE FUNCTION CREATE MACRO CREATE PROCEDURE CREATE TABLE CREATE VIEW	CREATE granted for the object type for the specified space.	Not applicable.
CREATE GLOP	<p>CREATE granted for the specified user.</p> <p>This privilege is not granted automatically when a user or database is created.</p>	

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
CREATE MAP CREATE PROFILE CREATE ROLE CREATE ZONE	Can only be granted to a user. Cannot be granted on an object.	Can only be granted to a user. Cannot be granted on an object.
CREATE TRIGGER	CREATE granted for the object type for the specified space.	CREATE granted for the specified table.
CTCONTROL	Privilege is a total system privilege and is not granted to or revoked from specific tables or databases. CTCONTROL can only be granted to a user. User DBC can grant CTCONTROL to any other user WITH GRANT OPTION.	
DATABASE FUNCTION GLOP INDEX MACRO MAP PROCEDURE PROFILE ROLE TABLE TRIGGER USER VIEW ZONE	CREATE and DROP granted for the type for the specified space. CREATE MAP and DROP MAP are system-level privileges that can only be granted to users or roles. CREATE MAP and DROP MAP cannot be granted on an object or to PUBLIC. CREATE PROFILE, DROP PROFILE, CREATE ROLE, DROP ROLE, CREATE ZONE, and DROP ZONE can only be granted to users. They cannot be granted on an object.	Not applicable. TRIGGER applies to the table on which a trigger is defined.
DELETE INSERT SELECT UPDATE	Privilege applies to all tables or views in the specified database. UPDATE, INSERT, and SELECT apply to a table or column set of the table. For a grantee to use the granted privileges on a view, the immediate owner of a view must have appropriate privileges on the tables and views referenced by the view.	Privilege applies only to the specified table, view, or columns. UPDATE cannot be granted on a GENERATED ALWAYS identity column. For a grantee to use the granted privileges on a view, the immediate owner of a view must have appropriate privileges on the tables and views referenced by the view.
DROP	Not applicable.	DROP granted for specified: <ul style="list-style-type: none"> Procedures when PROCEDURE is specified before the procedure name or for the specified function. Functions when FUNCTION or SPECIFIC FUNCTION is specified before the function name.

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
DROP DATABASE DROP USER	DROP granted for the specified space.	Not applicable.
DROP DATASET SCHEMA	Grants DROP DATASET SCHEMA on the SYSUDTLIB database. A database-level or schema-level privilege granted to user DBC WITH GRANT OPTION. DBC must explicitly grant it with or without grant option to any other users or databases that are created. DBC must explicitly grant it without grant option to any role that is created. The creator of a schema is automatically granted this privilege with grant option on the created schema. Dropping a schema is only allowed if the schema is not in use.	Not applicable.
DROP FUNCTION DROP MACRO DROP PROCEDURE DROP TABLE DROP TRIGGER DROP VIEW	DROP granted for the object type for the specified space.	DROP granted for the specified UDF, function mapping, macro, procedure, table, or view. DROP TRIGGER applies to the table on which a trigger is defined.
DROP GLOP	DROP granted for the specified GLOP set to the specified user or database. This privilege is not granted automatically when a user or database is created. This privilege is automatically granted to the creator and owner of the GLOP set.	
DROP MAP DROP PROFILE DROP ROLE DROP ZONE	Can only be granted to a user. Cannot be granted on an object.	Can only be granted to a user. Cannot be granted on an object.
DUMP RESTORE	Privilege applies to all tables in the specified database.	Privilege applies to the named data table or journal table only.
EXECUTE	Privilege applies to all UDFs or macros in the specified database. For the grantee to use the privilege on a UDF or macro, the immediate owner of the macro or UDF must have appropriate privileges on the objects referenced by the macro or UDF.	Privilege applies to the specified macro if the keywords FUNCTION, SPECIFIC FUNCTION, or PROCEDURE are not specified. For the grantee to use the privilege on a UDF or macro, the immediate owner of the macro must have appropriate

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
		<p>privileges on the objects referenced by the macro.</p> <p>Privilege applies to the specified UDFs or procedures if the keywords FUNCTION, SPECIFIC FUNCTION, or PROCEDURE are specified.</p> <p>For a grantee to use the SCRIPT table operator, the database user executing the script query must have the following privileges:</p> <ul style="list-style-type: none"> • The EXECUTE FUNCTION privilege on TD_SYSFNLIB.SCRIPT to enable its use. • The EXECUTE privilege to bind it to an appropriate OS user. SYSUIF.DEFAULT_AUTH is provided as an authorization object target for the EXECUTE privilege.
EXECUTE FUNCTION EXECUTE PROCEDURE	<p>Privilege applies to all UDFs or procedures in the specified space.</p> <p>For the grantee to use the privilege on a procedure, the immediate owner of the procedure must have the appropriate privileges on the objects referenced by the procedure.</p>	<p>Privilege applies to the specified UDFs, function mappings, or procedures.</p> <p>For the grantee to use the privilege on a procedure, the immediate owner of the procedure must have the appropriate privileges on the objects referenced by the procedure.</p>
GLOP MEMBER	<p>Privilege enables an external routine to access the GLOP set specified in the MEMBER OF GLOP SET clause of its definition when that GLOP set is not contained within the containing user or database for the routine.</p> <p>This privilege is not granted automatically when a user or database is created.</p>	
INDEX	Not applicable.	Privilege applies to tables, hash indexes, and join indexes.
NONTEMPORAL	<p>Privilege applies to a table, database, or user.</p> <p>The privilege is not granted automatically and must be explicitly granted by user DBC or by a user who has been granted the privilege WITH GRANT OPTION.</p> <p>For more information, see <i>Teradata Vantage™ - Temporal Table Support</i>, B035-1182.</p>	<p>Privilege applies at the table level.</p> <p>For more information, see <i>Teradata Vantage™ - Temporal Table Support</i>, B035-1182.</p>
OVERRIDE DELETE CONSTRAINT		<p>Object-level privileges that can be granted to a table, database, column, or user.</p> <p>If a row-level security function has not been created for a statement-action</p>

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
OVERRIDE INSERT CONSTRAINT OVERRIDE SELECT CONSTRAINT OVERRIDE UPDATE CONSTRAINT		<p>type, that type of statement can only be executed by a user who has the override privilege required to execute the statement.</p> <p>If a user has the override privilege, the request must specify the values to be assigned to the constraint columns of the target rows.</p> <p>You must have the CONSTRAINT ASSIGNMENT privilege to grant these privileges.</p> <ul style="list-style-type: none"> • If the object is a table, the OVERRIDE privilege is granted on all of its constraint columns. • If the object is a database or user, the OVERRIDE privilege is granted on all of the constraint columns of all of the tables contained within that database or user. <p>A GRANT request that specifies these privileges must also specify the CONSTRAINT object on which the privilege is granted.</p>
OVERRIDE DUMP CONSTRAINT OVERRIDE RESTORE CONSTRAINT	OVERRIDE DUMP CONSTRAINT and OVERRIDE RESTORE CONSTRAINT can only be granted to users and roles.	<p>Object-level privileges that can be granted to a user or a role.</p> <p>You must have the CONSTRAINT ASSIGNMENT privilege to grant these privileges.</p> <ul style="list-style-type: none"> • If the object is a table, it must be defined with at least one row-level security constraint. • If the object is a user, it must contain at least one table that has one or more row-level security constraints. <p>Note:</p> <p>It is not necessary for a database to have a table that contains one or more row-level security constraints.</p>
REFERENCES	Not applicable.	Privilege applies on a table or column of the table.
SHOW	Not applicable.	<p>Privilege applies on a table, hash or join index, or database.</p> <p>Privilege does not allow the grantee the ability to perform any operations on the granted database object other</p>

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
		than to make a HELP or SHOW request against it.
STATISTICS	Not applicable.	Privilege to collect statistics on a table, hash index, join index, or database.
UDTMETHOD	<p>Can only be granted on the SYSUDTLIB database.</p> <p>Privilege applies to all UDTs, methods, and UDFs contained within the SYSUDTLIB database.</p> <p>Effectively grants UDTUSAGE on all UDTs contained within SYSUDTLIB as well as UDTTYPE on the SYSUDTLIB database.</p> <p>Privilege applies to the specified UDT object.</p> <ul style="list-style-type: none"> • Grants ability to access a UDT column in a table or view. • Grants CREATE TYPE, ALTER TYPE, and DROP TYPE including method signatures. • Grants CREATE ORDERING and DROP ORDERING. • Grants CREATE CAST and DROP CAST. • Grants CREATE TRANSFORM and DROP TRANSFORM. • Grants CREATE TABLE for UDT columns. • Grants ability to reference any UDT in a UDF or procedure. • Grants ability to execute all methods. • Grants CREATE METHOD, ALTER METHOD, and REPLACE METHOD. 	Not applicable.
UDTTYPE	<p>Can only be granted on the SYSUDTLIB database.</p> <p>Privilege applies to all UDTs, methods, and UDFs contained within the SYSUDTLIB database.</p> <p>Effectively grants UDTUSAGE on all UDTs contained within the SYSUDTLIB database.</p> <p>Privilege applies to the specified UDT object.</p>	Not applicable.

Privilege	Object (Database, User, Role, PUBLIC)	Table, View, Function, Function Mapping, GLOP, Method, UDT, Procedure, Constraint, or Macro
	<ul style="list-style-type: none"> • Grants ability to access a UDT column in a table or view. • Grants CREATE TYPE and ALTER TYPE only if no method signatures are specified. • Grants DROP TYPE. • Grants CREATE ORDERING and DROP ORDERING. • Grants CREATE CAST and DROP CAST. • Grants CREATE TRANSFORM and DROP TRANSFORM • Grants CREATE TABLE for UDT columns. • Grants ability to reference any UDT in a UDF or procedure. • Grants ability to execute all methods. 	
UDTUSAGE	<p>Can be granted on the following:</p> <ul style="list-style-type: none"> • SYSUDTLIB database. • TYPE. <p>Privilege applies to all UDTs, methods, and UDFs contained within the SYSUDTLIB database.</p>	<p>Privilege applies to the specified UDT object.</p> <ul style="list-style-type: none"> • Grants ability to access a UDT column in a table or view. • Grants ability to execute all methods associated wholly with the specified UDT, but no others.
WITH DATASET SCHEMA	<p>A database-level or schema-level privilege granted to user DBC WITH GRANT OPTION.</p> <p>DBC must explicitly grant it with or without grant option to any other users or databases that are created. DBC must explicitly grant it without grant option to any role that is created.</p> <p>The creator of a schema is automatically granted this privilege with grant option on the created schema.</p>	<p>Gives users permission to associate a created schema with a table column.</p>

Granting Privileges on Global Temporary and Volatile Tables

GRANT always applies to the base global temporary table and never to a materialized instance. Just as with permanent tables, a user must have the appropriate privileges before submitting a GRANT request.

Because Vantage does not check privileges for volatile tables, you cannot GRANT privileges to them.

Verifying Privileges on Views, Macros, and Procedures

Statement Submitted	Privileges That the System Verifies
CREATE MACRO	Any privileges needed to perform the SQL statements in the macro body.
CREATE PROCEDURE (SQL Form)	Any privileges needed to perform the SQL statements in the procedure body.
<ul style="list-style-type: none"> CREATE PROCEDURE (External Form) ALTER PROCEDURE (External Form) 	CREATE EXTERNAL PROCEDURE and any privileges needed to access the specified tables, columns, and views using any of the valid API function calls.
CREATE PROCEDURE ... SQL SECURITY OWNER (both forms)	CREATE OWNER PROCEDURE and either of the following: <ul style="list-style-type: none"> For the SQL form, any privileges needed on the underlying tables, columns, and views to perform the SQL statements in the procedure body. For the external form, any privileges needed on the underlying tables, columns, and views using any of the valid API function calls.
CREATE VIEW	SELECT on the underlying base tables and views.

Vantage also verifies that the appropriate privileges exist on the target objects for any user who attempts to access a view, or perform a macro or procedure. This ensures that a change to a target object does not cause a violation of privileges when the view, macro, or procedure referencing that object is invoked.

CTCONTROL Privilege

CTCONTROL enables a user to grant or revoke the CONNECT THROUGH privilege using the GRANT CONNECT THROUGH or REVOKE CONNECT THROUGH statements. You can only grant CTCONTROL to specific users.

Rules for Granting the CTCONTROL Privilege

The following rules apply to granting the CTCONTROL privilege:

- The user specified by *user_name* must be an existing user of the system. Note that *user_name* *cannot* specify a role. If you attempt to grant CTCONTROL to a role, the request aborts and returns a message to the user.
- User DBC can grant the CTCONTROL privilege to any other user WITH GRANT OPTION.
- To submit a GRANT request for the CTCONTROL privilege to a user, you must either be user DBC or a user who has previously been granted the CTCONTROL privilege WITH GRANT OPTION.

WITH GRANT OPTION permits the grantee to grant the privilege to other users.

For more information about GRANT CONNECT THROUGH, see [GRANT CONNECT THROUGH](#).

Example: Granting the CTCONTROL Privilege

The following example grants user *kate* the privilege to grant the CONNECT THROUGH privilege to user *trusteduser2*:

```
GRANT CTCONTROL
ON trusteduser2
TO kate;
```

To then execute a GRANT CONNECT THROUGH request, *trusteduser2* requires the following privileges:

- CTCONTROL on herself.
- GRANT ... WITH ADMIN OPTION on any role specified
See [GRANT \(Role Form\)](#).
- DROP USER on any PERMANENT users specified as proxy users.

SHOW Privilege

The SHOW privilege enables you to have access to database object definitions and create text without having access to the data contained by the objects on which the privilege is granted. For example, SHOW permits a user to execute HELP and SHOW requests against an object while at the same time not being able to SELECT from it. Any SQL statement that requires *any* privilege (such as the HELP and SHOW statements) to be executed can be granted the SHOW privilege.

SHOW is an explicit privilege. Vantage does not grant the creator of an object this privilege automatically on the created user, database, or database object; SHOW must be granted explicitly. You must have the SHOW privilege WITH GRANT OPTION to be able to grant this privilege explicitly to other users and databases.

User DBC automatically has the SHOW privilege and can grant it on dictionary tables to other users and databases.

Granting Privileges on Queue Tables

GRANT both the SELECT and DELETE privileges on the queue table to the user, database, role, or PUBLIC from which a user performs consume mode SELECT statements.

GLOP Privileges

Vantage does not grant any of the GLOP privileges automatically when a database or user is created. All GLOP privileges must be explicitly granted, even when the privileges have been granted by a user who possesses the WITH GRANT OPTION privilege.

There are three GLOP privileges:

- **CREATE GLOP** enables the user to whom it is granted to perform the **CREATE GLOP SET** statement to create a new GLOP set.
- **DROP GLOP** enables the user to whom it is granted to perform the **DROP GLOP SET** statement to drop an existing GLOP set. **DROP GLOP** is granted automatically to the creator and owner of a GLOP set.
- **GLOP MEMBER** enables an external routine to access a GLOP set that is not contained within its containing user or database. Specifically, **GLOP MEMBER** enables the GLOP set clause referenced in the function, method, or procedure definition to use the GLOP set specified in the **MEMBER OF GLOP SET** clause.

You can also specify the keyword **GLOP** by itself to signify both the **CREATE GLOP** and **DROP GLOP** privileges. See [Multiple Privileges with a Single Keyword](#).

Granting GLOP Privileges

The enterprise wants the *sales_bonus* UDF to access the special *sales* GLOP. User *Joe* creates both the UDF and the GLOP in *SYSLIB*. The following requests must be executed to make this happen:

A user **WITH GRANT OPTION** must execute the following **GRANT** request:

```
GRANT GLOP ON syslib TO Joe;
```

Assuming that *Joe* already has privileges to create a function in *SYSLIB*, *Joe* then executes the following requests:

Joe creates the GLOP set object first to enable a GLOP to be added.

```
CREATE GLOP SET syslib.sales;
```

Joe then calls the procedure *GLOP_Add* to add the *sales* GLOP and associates it with *SYSLIB*.

```
CALL DBCExtension.GLOP_Add('syslib.sales',...);
```

Finally, *Joe* creates the function *sales_bonus* in *SYSLIB*.

```
CREATE FUNCTION syslib.sales_bonus( ... )
  RETURNS DECIMAL(6,2)
  LANGUAGE C
  MEMBER OF GLOP SET syslib.sales
  PARAMETER STYLE SQL
  EXTERNAL;
```

SYSLIB is a public database. For this reason, the Security Administrator instead decides to place the *sales* GLOP in the *finance* database, and to make the Finance department responsible for creating the GLOP data. She also wants the UDF to be contained within the *finance* database.

The following requests must be executed to make this happen:

A user WITH GRANT OPTION must execute the following GRANT request:

```
GRANT GLOP ON finance TO finance;
```

User *finance* must then execute the following requests:

```
CREATE GLOP SET sales;
CALL DBCExtension.GLOP_Add('sales', ... );
GRANT GLOP MEMBER ON finance.sales TO finance;
```

User *Joe* creates the function in the *finance* database. *Joe* must first be granted the privilege to create the UDF in the *finance* database by someone with the privileges to enable him to do that:

```
CREATE FUNCTION finance.sales_bonus( ... )
  RETURNS DECIMAL(6,2)
  LANGUAGE C
  MEMBER OF GLOP SET finance.sales
  PARAMETER STYLE SQL
  EXTERNAL;
```

Privileges and Procedures

Granting Privileges on Procedures

The following table describes the privileges of various types of users or grantors with respect to procedure-specific privileges:

Privilege	Granted to	Granted By
ALTER PROCEDURE and CREATE PROCEDURE	user DBC implicitly.	user DBC.
	other users, roles, databases, or PUBLIC explicitly.	<ul style="list-style-type: none"> • user DBC. • an owner of the user or database where the owner has this privilege WITH GRANT OPTION on itself. • a database or user having this privilege WITH GRANT OPTION on the database or user for which the privilege is being granted.
ALTER EXTERNAL PROCEDURE and	user DBC implicitly.	user DBC.

Privilege	Granted to	Granted By
CREATE EXTERNAL PROCEDURE	other users explicitly.	<ul style="list-style-type: none"> • user DBC. • a database or user having this privilege WITH GRANT OPTION on itself.
DROP PROCEDURE	all users, roles, databases, or PUBLIC explicitly.	default.
EXECUTE PROCEDURE or EXEC PROCEDURE	user DBC implicitly.	user DBC.
	the creator of a procedure automatically.	<p>the creator.</p> <p>Except for user DBC, owners do not implicitly have this privilege.</p> <p>If the immediate owner of the procedure is different from its creator, the owner does not receive this privilege automatically.</p>
	other users, roles, databases, and PUBLIC explicitly.	<ul style="list-style-type: none"> • user DBC. • an owner of the user or database where the owner has this privilege WITH GRANT OPTION on itself. • a database or user having this privilege WITH GRANT OPTION on the database or user for which the privilege is being granted.

Usage Notes for Procedure-Specific Privileges

The following rules apply to privileges specific to procedures:

- CREATE PROCEDURE and CREATE EXTERNAL PROCEDURE are database- or user-level privilege only.
- ALTER PROCEDURE, ALTER EXTERNAL PROCEDURE, DROP PROCEDURE, and EXECUTE PROCEDURE are allowed on databases, users, or specified procedures.
- DROP and EXECUTE can be used as abbreviations for DROP PROCEDURE and EXECUTE PROCEDURE while granting privileges, if you specify the object type PROCEDURE as a qualifier for the procedure name.

If PROCEDURE is not specified before the object name:

Request Made	Effect of the Request
GRANT EXECUTE	Object is assumed to be a macro. If a macro by that name does not exist, an error is returned.
GRANT DROP	Error is returned.

ALTER EXTERNAL PROCEDURE and CREATE EXTERNAL PROCEDURE Privileges

External procedures also require the ALTER EXTERNAL PROCEDURE and CREATE EXTERNAL PROCEDURE privileges. You need the ALTER EXTERNAL PROCEDURE privilege to use the ALTER PROCEDURE (External Form) statement.

The ALTER PROCEDURE (External Form) statement can be used to recompile existing external procedures. The purpose of the ALTER EXTERNAL PROCEDURE privilege is to enable DBAs to change the execution mode or to recompile an existing external procedure in situations where the current library is corrupt or the system has been reloaded. Do *not* grant this privilege to any user other than a DBA.

The ALTER EXTERNAL PROCEDURE privilege enables DBAs to use ALTER PROCEDURE (External Form) requests to change the execution mode for a particular external procedure to execute either directly in unprotected mode (EXECUTE NOT PROTECTED) or as a separate process in protected mode (EXECUTE PROTECTED).

ALTER EXTERNAL PROCEDURE is not an automatic privilege for a user when you create a database or user. The DBA retains the privilege (initially, only user *DBC* holds the privilege implicitly) and assigns it only for external procedures that are completely debugged and production certified, and can have their execution mode changed from EXECUTE PROTECTED to EXECUTE NOT PROTECTED.

Do not grant users the ability to perform the ALTER PROCEDURE (External Form) statement without considering the implications, because doing so could easily compromise the integrity of the system if the privilege is misused.

You can grant the ALTER EXTERNAL PROCEDURE privilege on either a specific external procedure or to an entire database or user.

See *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

CREATE EXTERNAL PROCEDURE Privilege

NOTICE

Do not grant the CREATE EXTERNAL PROCEDURE privilege to any user unless they are explicitly assigned to code an external procedure for your site. Even then, you should restrict this privilege to only your most trusted programmers. You must also ensure that the external procedure is thoroughly tested to verify that it does not compromise the system in any way.

The system does not grant the CREATE EXTERNAL PROCEDURE privilege automatically when you create a user or database. Nor is CREATE EXTERNAL PROCEDURE granted on a database unless an owner explicitly has this privilege on itself WITH GRANT OPTION.

Note:

This functionality is different from the CREATE MACRO privilege, where the system grants CREATE MACRO automatically when you create a database or user, as well as the privilege held implicitly by an owner.

External procedures execute as part of the system when running in unprotected mode, while protected mode external procedures run in a separate process as an ordinary user named tdatuser.

For more information, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Example: Granting the CREATE EXTERNAL PROCEDURE Privilege

The following example grants user *asst_dba* the privilege to create an external procedure:

```
GRANT CREATE EXTERNAL PROCEDURE
ON DATABASE classify
TO asst_dba;
```

Granting Privileges on UDFs

The rules for granting privileges on UDFs are identical to those for macros except for the CREATE FUNCTION and EXECUTE FUNCTION privileges, as documented in the following table, which explains which UDF-related privileges are granted explicitly or not:

Privilege	Usage Notes
ALTER FUNCTION	<ul style="list-style-type: none"> Not granted automatically to the creator when it creates a database or user. Not granted automatically to a database or user when it creates a database or user. Held implicitly only by user DBC. Cannot be granted on a UDF.
CREATE FUNCTION	<ul style="list-style-type: none"> Not granted automatically to the creator when it creates a database or user. Held implicitly only by user DBC.
DROP FUNCTION	<ul style="list-style-type: none"> Granted automatically WITH GRANT OPTION to the creator of a user, database, function, or function mapping. Granted automatically without grant option to a created database or user on itself. Held implicitly on a database or user for an owner of that database or user.
EXECUTE FUNCTION	<ul style="list-style-type: none"> Granted automatically WITH GRANT OPTION to the creator of a function or function mapping. Not granted automatically to the creator when it creates database or user. Held implicitly only by user DBC.

All newly created external UDFs are performed in protected mode by default. This is also true for all newly created methods and external procedures. UDFs do not run in modes.

The ALTER FUNCTION privilege should be restricted to DBAs exclusively. Its intent is to permit a user to perform the ALTER FUNCTION statement, which can be used to change the execution mode of a UDF. Changing the execution mode for a UDF from PROTECTED to NOT PROTECTED, for example, is not an act that should be performed without exercising extreme caution.

For more information see the information about ALTER FUNCTION, ALTER METHOD, and ALTER PROCEDURE in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

CREATE FUNCTION Privilege

If you are granted the CREATE FUNCTION privilege on a database or user, any function you create in that database or user has these privileges WITH GRANT OPTION:

- DROP FUNCTION
- EXECUTE FUNCTION

EXECUTE FUNCTION Privilege

You must specify EXECUTE FUNCTION to grant that privilege on functions or function mappings in a database, except for constraint functions.

The EXECUTE FUNCTION privilege does not need to be granted on constraint functions to users that need to query tables that are protected by row-level security.

Example: Granting the EXECUTE FUNCTION Privilege

This example of the GRANT statement grants the EXECUTE privilege on all functions in SYSLIB:

```
GRANT EXECUTE FUNCTION ON SYSLIB TO user_xyz;
```

You need only specify EXECUTE to grant the EXECUTE FUNCTION privilege on an individual UDF. For example, the following GRANT statement grants the EXECUTE privilege only on the function with the specific function name sales to user_xyz:

```
GRANT EXECUTE ON SPECIFIC FUNCTION SYSLIB.sales TO user_xyz;
```

Example: Granting the EXECUTE FUNCTION Privilege for a Function Mapping

The specified function mapping must exist before you can grant the EXECUTE FUNCTION privilege. This statement grants the EXECUTE FUNCTION privilege on the Interpolator function mapping in the appl_view_db database to user1.

```
GRANT EXECUTE FUNCTION ON appl_view_db.Interpolator TO user1;
```

Example: EXECUTE FUNCTION and Authorization Objects

In one particular case, granting EXECUTE FUNCTION is not sufficient to actually use the function. For example, the script table operator is enabled if a user has the EXECUTE FUNCTION granted to it but it needs an Authorization object bound to it in order to execute the scripts.

For example:

```
GRANT EXECUTE FUNCTION ON TD_SYSFNLIB.SCRIPT TO user_name;
```

```
GRANT EXECUTE ON authorization_name TO user_name;
```

```
GRANT EXECUTE FUNCTION ON TD_SYSFNLIB.SCRIPT TO role_name;
```

```
GRANT EXECUTE ON authorization_name TO role_name;
```

SYSUIF.DEFAULT_AUTH is provided as an authorization object target for the EXECUTE privilege.

DROP FUNCTION Privilege

You must specify DROP FUNCTION to grant that privilege on a database object. If you are granting the privilege on a particular UDF only, then you can specify DROP without also specifying the FUNCTION keyword.

When creators have WITH GRANT OPTION on a function, they can grant the following privileges to another user on that function:

- DROP FUNCTION
- EXECUTE FUNCTION

Users with the DROP FUNCTION privilege on a function can also replace that function. If a user drops the function, however, the user cannot replace the function at a later time without also being granted the CREATE FUNCTION privilege.

UDT-Related Privileges

Initially, only user DBC has UDT privileges on SYSUDTLIB, which user DBC holds implicitly. Any other user or role must be explicitly granted UDT privileges on SYSUDTLIB to execute SQL statements that involve UDTs.

There are three UDT-related privileges:

- UDTMETHOD (see [UDTMETHOD Privilege](#)).

UDTMETHOD allows a user to use, create, drop, or modify any UDT and its methods without restriction.

You can only grant this privilege at the database level, specifically on the SYSUDTLIB database: there are no UDT object-level privileges for UDTMETHOD.

- UDTYPE (see [UDTYPE Privilege](#)).

UDTYPE allows a user to create, alter and drop UDTs in addition to the privileges UDTUSAGE permits.

You can only grant this privilege at the database level: there are no UDT object-level privileges for UDTTYPE.

- UDTUSAGE (see [UDTUSAGE Privilege](#)).

UDTUSAGE allows a user to execute all SQL statements that reference existing UDTs and their existing methods. This means that a user with the UDTUSAGE privilege on a particular UDT can perform the following operations:

- Create a new table that is defined with a column having that UDT type.
- Alter an existing table that is defined with a column having that UDT type.
- Reference that UDT in a query, UDF, or procedures.
- Execute all methods that are associated wholly with the UDT type.

Methods that involve additional UDTs require you to have the UDTUSAGE privilege on any such UDTs as well.

UDTUSAGE does not permit a user to perform any of the following operations:

- Create new UDTs.
- Drop existing UDTs.
- Alter the ordering, casting, or transform behavior of existing UDTs.
- Create new methods.
- Drop or replace existing methods.

Unlike UDTMETHOD and UDTTYPE, you can grant UDTUSAGE on either a specific UDT (as TYPE UDT_name) or on the entire SYSUDTLIB database.

ALL PRIVILEGES and UDTs

When you specify ALL PRIVILEGES on the SYSUDTLIB database, the system grants the UDTMETHOD privilege along with the other privileges ALL PRIVILEGES provides, assuming the creator has the privilege WITH GRANT OPTION.

A user who is granted UDTMETHOD via the ALL PRIVILEGES path WITH GRANT OPTION can then grant others the UDTUSAGE, UDTTYPE, or UDTMETHOD privileges, optionally also with the WITH GRANT option.

UDTUSAGE Privilege

The UDTUSAGE privilege allows a user to use a UDT in a table or view and to execute all methods associated wholly with that UDT. They cannot create or delete UDTs, create, alter, or delete methods, or alter the behavior of a UDT with respect to ordering, casting, or transform functionality.

You can grant UDTUSAGE at either the database or UDT object level.

Level of the Grant	Where UDTUSAGE Must Be Granted
Database	SYSUDTLIB database only.
UDT object	Name of the UDT.

A user with the UDTUSAGE privilege on a specific UDT can perform the following statements and operations:

- CREATE TABLE or ALTER TABLE for tables that contain columns that use that UDT.
- Reference that UDT in queries, UDFs, and procedures that reference the UDT.
- Execute all methods associated wholly with the UDT.

Methods that involve other UDTs also require the UDTUSAGE privilege on those UDTs.

With regard to nested structured UDTs, be aware that privileges are not automatically inherited from their parents. For example, if you have been granted UDTUSAGE on a top-level structured UDT attribute, you can then specify a column with that type in a create table operation. However, that privilege does not grant you the ability to use observer or mutator methods on any of the structured attributes in the lower layers of that UDT, nor can you invoke any of the methods defined for those lower layered attributes.

To enable that sort of access, you must grant UDTUSAGE explicitly on each individual UDT component of the nested UDT type or grant UDTUSAGE explicitly on the SYSUDTLIB database.

UDTUSAGE is not an automatically granted privilege. A user must either be granted this privilege explicitly or acquire it through a role. Users who are granted UDTUSAGE WITH GRANT option can then grant others the UDTUSAGE privilege, optionally with the WITH GRANT option privilege.

UDTUSAGE is represented by the code *UU* in the *AccessRight* column of the *DBC.AccessRights* table.

User DBC has implicit UDT privileges on the SYSUDTLIB database.

Any other user or role that needs to access the UDT-related objects in SYSUDTLIB must first be granted the appropriate UDT privileges on SYSUDTLIB or a specific UDT in the SYSUDTLIB database explicitly.

The following GRANT request grants the following respective privileges:

- UDTUSAGE on the *SYSUDTLIB* database to the user named *tester1*.
- UDTUSAGE on the UDT named *circle* to the user named *tester3*.
- UDTUSAGE on the UDT named *square* in database *SYSUDTLIB* to the user named *tester4*.

```
GRANT UDTUSAGE
ON SYSUDTLIB
TO tester1;
```

```
GRANT UDTUSAGE
ON TYPE circle
TO tester3;
```

```
GRANT UDTUSAGE
```

```
ON TYPE SYSUDTLIB.square
TO tester4;
```

UDTTYPE Privilege

The UDTTYPE privilege gives a user all the privileges associated with UDTUSAGE as well as permitting that user to execute all SQL statements that reference UDTs without creating new methods or dropping or replacing existing methods.

A user with the UDTTYPE privilege can perform the following statements and operations:

- CREATE TYPE without specifying method signatures.
- DROP TYPE.
- ALTER TYPE without adding or dropping method signatures.
- CREATE ORDERING and DROP ORDERING.
- CREATE CAST and DROP CAST.
- CREATE TRANSFORM and DROP TRANSFORM.
- CREATE TABLE with UDT columns.
- Reference any UDT in an SQL request, UDF, or procedure.
- Perform all methods.

The specified *database_name* in the ON clause must be SYSUDTLIB.

UDTTYPE is not an automatically granted privilege. A user must either be granted this privilege or acquire it through a role. Users who are granted UDTTYPE WITH GRANT OPTION can then grant others either the UDTUSAGE or UDTTYPE privilege, optionally with the WITH GRANT option privilege.

UDTTYPE is represented by the code *UT* in the *AccessRight* column of the DBC.AccessRights table.

The following GRANT request grants the UDTTYPE privilege on database *SYSUDTLIB* to the user named *tester2*:

```
GRANT UDTTYPE
ON SYSUDTLIB
TO tester2;
```

UDTMETHOD Privilege

The UDTMETHOD privilege allows a user to use any UDT and its methods without any restrictions. Its functionally includes both of the following privileges:

- UDTUSAGE on all UDTs in the SYSUDTLIB database.
- UDTTYPE on the SYSUDTLIB database.

A user with the UDTMETHOD privilege can perform these statements and operations:

- CREATE TYPE with or without specifying method signatures.
- DROP TYPE.
- ALTER TYPE with or without adding or dropping method signatures.
- CREATE ORDERING and DROP ORDERING.
- CREATE CAST and DROP CAST.
- CREATE TRANSFORM and DROP TRANSFORM.
- CREATE TABLE with UDT columns.
- Reference any UDT in an SQL statement, UDF, or procedure.
- Execute all methods.
- CREATE METHOD and ALTER METHOD and REPLACE METHOD.

The specified *database_name* in the ON clause must be SYSUDTLIB.

UDTMETHOD is not an automatically granted privilege. A user must be granted it explicitly, or acquire it through ALL PRIVILEGES or through a role. Users who are granted UDTMETHOD WITH GRANT OPTION can then grant others the UDTUSAGE, UDTTYPE, or UDTMETHOD privilege, optionally with the WITH GRANT option.

UDTMETHOD is represented by the code *UM* in the *AccessRight* column of the DBC.AccessRights table.

The following GRANT requests grant the following respective privileges:

- UDTMETHOD on *SYSUDTLIB* to the user named *user_DBA* with grant option.
- UDTMETHOD on *SYSUDTLIB* to the role named *developer_role*.

```
GRANT UDTMETHOD
ON SYSUDTLIB
TO user_DBA WITH GRANT OPTION;
```

```
GRANT UDTMETHOD
ON SYSUDTLIB
TO developer_role;
```

Teradata Row Level Security Privileges

The Teradata Row Level Security feature provides a number of privileges that administrators can use to establish and maintain row-level security for the system. Some of the privileges are system-level privileges and some are object-level privileges.

Initially, only user DBC has row-level security privileges. Any other user must be explicitly granted row-level security privileges to be able to perform the following tasks:

- Create row-level security constraints.
- Assign row-level security constraint values (security credentials) to users and profiles.
- Define row-level security constraints on tables.

- Override (bypass) validation of the row-level security policies contained in the constraint functions applicable to target tables.

The basic types of row-level security privileges are:

- System-level privileges
- Object-level privileges (see [Object-Level Privileges for Row-Level Security](#)).

Note:

Although row-level security credentials are not privileges, they work like required privileges do in other types of access control. When you assign security credentials to users or profiles, you are essentially determining whether the users are able to access table rows that are protected by row-level security. (The security credential assigned to the users must match the security constraint values assigned to the row or rows they are attempting to access.) The exact type or types of access you permit is determined by the row-level security policy defined in the constraint function.

System-Level Privileges for Row-Level Security

Administrators can grant system-level privileges to users or profiles for the purpose of establishing and maintaining row-level security.

These privileges enable users to:

- Create row-level security constraints using SQL requests.
- Define row-level security constraints on tables using SQL requests.
- Assign row-level security constraint values (security credentials) to users and profiles using SQL requests.

The privileges are as follows:

- CONSTRAINT ASSIGNMENT
- CONSTRAINT DEFINITION (see [CONSTRAINT DEFINITION Privilege](#))

See the section on the DBC.AccessRights table in *Teradata Vantage™ - Data Dictionary*, B035-1092 for a list of the two-character abbreviations for these privileges.

CONSTRAINT ASSIGNMENT Privilege

This system-wide privilege enables users to define row-level security constraints on tables and to assign row-level security constraint values to users and profiles using SQL DDL statements. Administrators can grant it to individual users or to profiles.

Vantage automatically grants this privilege to user DBC WITH GRANT OPTION, which enables user DBC to grant it to any other user or role.

The rules and restrictions for granting the CONSTRAINT ASSIGNMENT privilege are as follows:

- You can only grant it to another user or role if you also have the WITH GRANT OPTION privilege.
- You cannot specify a target database object.

- You cannot grant it to PUBLIC.

You must have the CONSTRAINT ASSIGNMENT privilege to use these SQL DDL statements on tables that have row-level security constraints or users and profiles that have security credentials assigned to them. These statements can be used on users or profiles that do not have security credentials assigned to them.

- ALTER TABLE
- CREATE PROFILE
- CREATE TABLE
- CREATE USER
- MODIFY PROFILE
- MODIFY USER
- SHOW CONSTRAINT

The CONSTRAINT DEFINITION privilege also enables you to execute a SHOW CONSTRAINT request.

CONSTRAINT DEFINITION Privilege

This system-wide privilege enables users to create and modify row-level security constraints using SQL DDL statements. Administrators can grant it to individual users or to roles.

Vantage automatically grants this privilege to user DBC WITH GRANT OPTION, which enables user DBC to grant it to any other user or role.

The rules and restrictions for granting the CONSTRAINT DEFINITION privilege are as follows:

- You can only grant it to another user or role if you also have the WITH GRANT OPTION privilege.
- You cannot specify a target database object.
- You cannot grant it to PUBLIC.

You must have the CONSTRAINT DEFINITION privilege to use the following SQL DDL statements to create, modify, or SHOW row-level security constraints:

- ALTER CONSTRAINT
- CREATE CONSTRAINT
- DROP CONSTRAINT
- SHOW CONSTRAINT

The CONSTRAINT ASSIGNMENT privilege also enables you to execute a SHOW CONSTRAINT request.

Object-Level Privileges for Row-Level Security

Administrators can grant these privileges for the purpose of temporarily enabling users to bypass (override) the row-level security policy defined on database objects. Because these privileges enable users to override row-level security policy restrictions, they are referred to as override privileges.

The two basic types of override privileges are:

- DML restriction override privileges
- Archive/Recovery override privileges. see [Archive/Recovery Restriction Override Privileges](#)).

DML Restriction Override Privileges

These object-level privileges enable users to execute DML requests on tables that have row-level security restrictions that prohibit delete, insert, select, and update operations.

Administrators can grant these privileges to temporarily enable users to execute DML requests to perform the prohibited delete, insert, select, and update operations.

These privileges can be granted as follows:

- To users and roles.
- On tables or constraint columns.

The DML restriction override privileges are as follows:

- **OVERRIDE DELETE CONSTRAINT**

When granted on the target table, it enables users to bypass validation of the DELETE security policies contained within each of the DELETE constraint functions applicable to the table.

When granted on a constraint column, it enables users to bypass validation of the DELETE security policy for a specific constraint function.

Note:

Even if this privilege is granted on the target table, a user is not able to delete all rows of the table. This is because a DELETE on the table is also constrained by SELECT constraint functions, which filter out rows that the user is not permitted to access. To delete all rows, a user must be able to bypass all DELETE and SELECT constraint checks.

- **OVERRIDE INSERT CONSTRAINT**

Enables users to submit an INSERT request to specify a value for a column that has the INSERT row-level security constraint.

INSERT requests must contain the value to be assigned to the constraint column. Vantage ensures that the value assigned to a constraint column is one specified by the name:code pairs of the constraint.

- **OVERRIDE SELECT CONSTRAINT**

When granted on a single constraint column, it enables a user to bypass the security policy in the SELECT constraint function associated with that constraint column. This application is used to permit a user to retrieve a single row.

When granted on all constraint columns, it enables a user to retrieve all rows of the table, because the user is able to bypass the security policy in the SELECT constraint functions associated with all constraint columns of the table.

- **OVERWRITE UPDATE CONSTRAINT**

Enables users to submit an UPDATE request that specifies a value in the SET clause for a column that has the UPDATE row-level security constraint.

Note:

You can create a row-level security constraint function (UDF) for each type of DML request that can be executed against tables with row-level security constraints. If a constraint function does not exist for a type of DML request, the request can only be executed by a user who has the override privilege for that type of request.

Rules for Granting These Privileges

You can only grant these privileges as follows:

- to users and roles.
- on databases that contain tables with row-level security.
- on tables with row-level security.
- on constraint columns (columns with row-level security constraints).

Following are additional rules and restrictions for granting the DML restriction override privilege:

- You must have the CONSTRAINT ASSIGNMENT privilege to grant these privileges.
- You cannot grant these privileges WITH GRANT OPTION.

If you attempt to grant one of the privileges WITH GRANT OPTION, the system aborts the request and returns an error.

- The GRANT request must specify the name of the row-level security constraint object on which you want to grant the privilege.
- The target object of the GRANT request must be a database, a table that has row-level security (a table with one or more row-level security constraints), or a constraint column. An error is returned you attempt to grant an OVERWRITE CONSTRAINT privilege on a table that does not have one or more row-level security constraints or a column that does not have a constraint.

Target of Request	Objects the Override Privilege Must Be On
database	all constraint columns of all tables within the specified database.
table	all constraint columns of the specified table.
column	the column.

Archive/Recovery Restriction Override Privileges

These object-level privileges enable users to execute Archive/Recovery utility commands for the purpose of archiving or restoring tables that have row-level security constraints or databases that contain tables with row-level security constraints. Administrators can grant these privileges on databases or tables.

Unlike the DML restriction override privileges, these privileges do not bypass the row-level security policies defined in constraint functions (UDFs).

Note:

These privileges are not sufficient to archive or restore tables that have row-level security constraints or databases that have tables with row-level security constraints. You must also have the DUMP or RESTORE privilege on the table or database you want to archive or restore.

The Archive/Recovery override privileges are as follows:

- **OVERRIDE DUMP CONSTRAINT**
Enables users to execute the Archive/Recovery utility ARCHIVE command to archive databases objects defined with one or more row-level security constraint columns. Required on all targets objects of the ARCHIVE command.
- **OVERRIDE RESTORE CONSTRAINT**
Enables users to execute the Archive/Recovery utility COPY and RESTORE commands on any database object defined with one or more row-level security constraint columns. Required on all targets objects of the COPY and RESTORE commands.

The rules and restrictions for granting the Archive/Recovery override privileges are as follows:

- You must have the CONSTRAINT ASSIGNMENT privilege to grant these privileges.
- You can only grant these privileges to users and roles and on databases and tables.
- You cannot grant these privileges WITH GRANT OPTION.
If you attempt to grant one of the privileges WITH GRANT OPTION, the system aborts the request and returns an error.
- The GRANT request must specify the name of the row-level security constraint object on which you want to grant the privilege.
- The target object of the GRANT request must be a database, table, or constraint column.

Example of Granting Row-Level Security OVERRIDE Privileges to Users and Roles

Assume that all users are granted the INSERT, UPDATE, DELETE and SELECT discretionary access control (DAC) privileges on the *inventory* table, access to which is controlled by the row-level security constraints *classification_level* and *classification_category*.

The following GRANT request grants those privileges to PUBLIC:

```
GRANT INSERT, UPDATE, DELETE, SELECT
ON inventory
TO PUBLIC;
```

Assume that user *top_gun* is granted the necessary privileges to update the *classification_level* and the *classification_category* columns of the *inventory* table.

User *top_gun* is defined as follows:

```
CREATE USER top_gun AS
  PERM=1E6,
  PASSWORD=Top1111GUN;
```

To be able to update the *classification_level* and *classification_category* columns of *inventory*, *top_gun* must have the `OVERWRITE UPDATE` privilege on those constraints.

The following `GRANT` requests grant the `OVERWRITE UPDATE CONSTRAINT` privilege to user *top_gun* on the *classification_level* and *classification_category* constraints.

```
GRANT OVERWRITE UPDATE CONSTRAINT (classification_level)
ON inventory
TO top_gun;

GRANT OVERWRITE UPDATE CONSTRAINT (classification_category)
ON inventory
TO top_gun;
```

Assume you have also created a constraint named *group_membership*. When you initially created this constraint, there was no row-level security policy function specified for the `UPDATE` and `DELETE` statement actions in the definition of the *group_membership* constraint.

To enable the updating and deletion of rows in the *emp_record* table, you must grant the appropriate privileges to a user or role, so you decide to grant those privileges to the role *personnel_clerk*, which was earlier granted to user *sally_jones*. The role *personnel_clerk* already has all discretionary access control privileges on the table.

The `OVERWRITE UPDATE CONSTRAINT` and `OVERWRITE DELETE CONSTRAINT` privileges are granted to *personnel_clerk* by the security administrator, who has the `CONSTRAINT ASSIGNMENT` privilege.

The following `GRANT` request grants the `OVERWRITE UPDATE CONSTRAINT` and `OVERWRITE DELETE CONSTRAINT` privileges for the *group_membership* constraint column to the *personnel_clerk* role on the *emp_record* table.

```
GRANT OVERWRITE UPDATE, DELETE CONSTRAINT (group_membership)
ON emp_record TO personnel_clerk;
```

Granting Privileges to Roles

Roles define privileges on database objects. A database administrator can create different roles for different job functions and responsibilities, grant specific privileges on database objects to the roles, and then grant membership to the roles to users. Users who are members of a role can access all the objects for which the role has privileges. A role that has roles granted to it cannot be granted to a role.

Roles cannot be granted the following privileges:

- CREATE DATABASE
- CREATE ROLE
- CREATE PROFILE
- CREATE USER
- DROP DATABASE
- DROP ROLE
- DROP PROFILE
- DROP USER
- CTCONTROL

Roles cannot be granted on a database or PUBLIC.

To grant role membership to users or other roles, use the GRANT (Role Form) statement.

For more information, see [GRANT \(Role Form\)](#).

The following request grants privileges to a role. In this example, the *finance* role is granted the privilege to SELECT data from the *department* table, which is in the *personnel* database:

```
GRANT SELECT
ON personnel.department
TO finance;
```

All users who are granted membership to the *finance* role also inherit the privilege to SELECT data from the *department* table in the *personnel* database when the role is activated for the user.

Case Study: Granting SELECT on a View

Assume that Allen creates two objects: a table named *Allen.BaseTable*, and a view on that table named *Allen.ViewA*. The system verifies that Allen has SELECT privilege on *Allen.BaseTable* when *ViewA* is created.

Also assume that Allen grants the SELECT privilege on *ViewA* to Bobby. Bobby then creates *Bobby.ViewB*, which references *Allen.ViewA*. The Parser verifies that Bobby has SELECT privilege on *Allen.ViewA* when *Bobby.ViewB* is created.

The following stages summarize this process:

1. Allen creates the base table *Allen.BaseTable*.
2. Allen creates the view *Allen.ViewA* that references *Allen.BaseTable*.
He has the SELECT privilege on *Allen.BaseTable*.
3. Allen grants the SELECT privilege on *Allen.ViewA* to Bobby.
4. Bobby creates *Bobby.ViewB* that references *Allen.ViewA*.

Bobby has the SELECT privilege on *Allen.ViewA*.

Bobby now wants to grant the SELECT privilege on *Bobby.ViewB* to Chuck. Bobby can grant the privilege to Chuck, but Chuck cannot use the views in a query unless Allen grants Bobby the SELECT privilege on *Allen.ViewA* WITH GRANT OPTION.

The same privilege is required when a macro references an object owned by a user other than its creator. For example, assume that the macro *Bobby.MacroB* deletes rows from *Allen.ViewA*.

Other users cannot execute *Bobby.MacroB* unless Bobby has the EXECUTE privilege WITH GRANT OPTION on *Bobby.MacroB*.

Case Study: Granting INSERT and DELETE on a View

Assume that the view *Bobby.ViewB* references the view *Allen.ViewA*, which in turn references *Allen.BaseTable*. When Bobby enters the following statement, the system checks that the privileges in the table following the statement exist.

```
GRANT INSERT, DELETE
ON Bobby.ViewB TO Chuck;
```

The following stages explain this process:

1. Bobby has INSERT WITH GRANT OPTION on *Bobby.ViewB* or on himself.
2. Bobby has DELETE WITH GRANT OPTION on *Bobby.ViewB* or on himself.

Unless revoked, Bobby has these privileges on *Bobby.ViewB* explicitly because they were granted to him automatically when he created the view.

Bobby also had these privileges on himself because they were granted explicitly when user Bobby was created.

Because Bobby owns *Bobby.ViewB*, he also has these privileges implicitly.

Implicit privileges cannot be revoked. For any other type of privilege, however, the system might not find one or more of the necessary privileges. In this case, the system returns an error message to the user submitting the GRANT statement and the statement is not performed.

The breakdown of privilege types for this example is as follows:

Privilege Category	THIS individual ...	HAS this privilege ...	ON this object ...
Automatic	Bobby	INSERT WITH GRANT OPTION	Bobby.ViewB

Privilege Category	THIS individual ...	HAS this privilege ...	ON this object ...
Explicit	Allen	DELETE WITH GRANT OPTION	Allen.BaseTable
		INSERT WITH GRANT OPTION	
	Bobby	DELETE WITH GRANT OPTION	Allen.ViewA
		INSERT WITH GRANT OPTION	

Case Study: SQL Procedures

Assume that *user_1* is to be granted CREATE PROCEDURE privilege on the accounting database. Assume that the grantor is an owner of the database and that she has the explicit CREATE PROCEDURE and EXECUTE PROCEDURE privileges WITH GRANT OPTION on herself. This grant cannot be made unless one of the following conditions is true:

- The grantor has both the CREATE PROCEDURE and EXECUTE PROCEDURE privileges WITH GRANT OPTION on itself and on the user or database WITH GRANT OPTION.
- The grantor is user DBC.

The following GRANT request must be submitted so that *user_1* can create SQL procedures in the *accounting* database:

```
GRANT CREATE PROCEDURE
ON accounting
TO user_1;
```

To grant CREATE PROCEDURE, DROP PROCEDURE, and EXECUTE PROCEDURE privileges to *user_1* on the *accounting* database, submit the following GRANT request:

```
GRANT CREATE PROCEDURE, EXECUTE PROCEDURE, DROP PROCEDURE
ON accounting
TO user_1;
```

If the keyword PROCEDURE is specified without CREATE or DROP in a GRANT request, it confers both CREATE PROCEDURE and DROP PROCEDURE privileges at the user or database level.

The following example assumes that the grantor has the privileges required to grant CREATE PROCEDURE and DROP PROCEDURE privileges to *user_1* on the *accounting* database:

```
GRANT PROCEDURE
ON accounting
TO user_1;
```

Assume that *user_1* needs to be granted EXECUTE PROCEDURE privilege WITH GRANT OPTION on the procedure *daily_updates* in the database *accounting*. Assume that the grantor has the EXECUTE PROCEDURE WITH GRANT OPTION privilege on the procedure or its containing database. The following GRANT request needs to be submitted:

```
GRANT EXECUTE ON PROCEDURE accounting.daily_updates
TO user_1
WITH GRANT OPTION;
```

This request can also be specified using the following syntax:

```
GRANT EXECUTE PROCEDURE ON accounting.daily_updates
TO user_1
WITH GRANT OPTION;
```

Note that the following GRANT request returns an error because the syntax is valid only for macros, and *daily_update* is not a macro:

```
GRANT EXECUTE ON accounting.daily_update
TO user_1
WITH GRANT OPTION;
```

To grant ALTER PROCEDURE, EXECUTE PROCEDURE, and DROP PROCEDURE privileges to *user_1* on the SQL procedure *weekly_update*, the necessary GRANT request looks like either of these requests, assuming the grantor has the privileges to perform the grant:

```
GRANT ALTER PROCEDURE, EXECUTE, DROP PROCEDURE
ON PROCEDURE weekly_update
TO user_1;

GRANT ALL ON PROCEDURE accounting.weekly_update
TO user_1;
```

Examples of Granting Privileges

Example of Granting Privileges to a Group of Users

The following request grants privileges to a group of users. In this example, all users created under the *finance* database are granted the privilege to SELECT data from the *department* table, which is in the *personnel* database:

```
GRANT SELECT
ON personnel.department
TO ALL finance;
```

Example of Granting SELECT on Any Object

The following statement allows user *Moffit* to retrieve information from any object in the *personnel* database:

```
GRANT SELECT
ON personnel
TO moffit;
```

Example of Granting SELECT on One Column of a Table

This example grants the SELECT privilege only on *column_1* of the table named *department* in the *personnel* database to user *Moffit*.

```
GRANT SELECT (column_1)
ON personnel.department
TO moffit;
```

Example of Granting SELECT, INSERT, UPDATE, and DELETE on Any Object

The following statement allows user *Peterson* to perform selects, inserts, updates, and deletes on any object in the *personnel* database:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON personnel
TO peterson;
```

Example of Granting USER Privilege On a Database

The following statement allows user *Phan* to create users within the *personnel* database space, but it does not allow him to modify or drop any users within the *personnel* database except for the ones that he creates.

```
GRANT USER
ON personnel
TO phan;
```

Example of Granting Privileges Related to Triggers

The following request grants CREATE TRIGGER and DROP TRIGGER privileges to user *TrigUser* for any table in database *TrigDB*:

```
GRANT CREATE TRIGGER, DROP TRIGGER
ON TrigDB
TO TrigUser;
```

The following request grants CREATE TRIGGER and DROP TRIGGER privileges to user *TrigUser* for *TrigDB* as the subject table .

```
GRANT CREATE TRIGGER, DROP TRIGGER
ON TrigTable
TO TrigUser;
```

You can also grant the same privileges as in the previous requests using the short form TRIGGER to mean both CREATE TRIGGER and DROP TRIGGER.

For example:

```
GRANT TRIGGER
ON TrigDB TO TrigUser;

GRANT TRIGGER
ON TrigTable
TO TrigUser;
```

Example of Granting Privileges on an External Function

The following GRANT requests grant the following respective privileges:

- ALTER FUNCTION on the specific external function *find_text* to users named *user_dba* and *user_in_house*.
- ALTER FUNCTION on all external UDFs in the *SYSLIB* database to *user_dba*.
- CREATE FUNCTION on the database *document_db* to *user_dba*.
- EXECUTE FUNCTION on the overloaded function *text_process* to user *sammy*.

```
GRANT ALTER FUNCTION
ON SPECIFIC FUNCTION find_text
TO user_dba, user_in_house;
```

```
GRANT ALTER FUNCTION
ON syslib
TO user_dba;
```

```
GRANT CREATE FUNCTION
ON document_db
TO user_dba, user_in_house;
```



```
GRANT EXECUTE
ON FUNCTION text_process(CHAR,CHAR,INTEGER)
TO sammy;
```

Related Information

For more information about granting privileges on stored procedures, see:

- *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.
- [Privileges and Procedures](#).

For information about revoking privileges granted by the SQL form of GRANT, see [REVOKE \(SQL Form\)](#).

For more information about the NONTEMPORAL privilege and how it can be used, see *Teradata Vantage™ - Temporal Table Support*, B035-1182.

GRANT CONNECT THROUGH

Grants the privilege to connect as a proxy permanent user or proxy application user through the specified trusted user, storing the information in *DBC.ConnectRulesTbl*.

The statement grants the specified trusted user the privilege to do the following:

- Connect as the specified permanent or application user.
- Set roles for the specified permanent or application user.

The specified trusted user is granted the privilege to assert the identity of the proxy user names specified in its defining SET QUERY_BAND request. See the information about SET QUERY_BAND in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must have the CTCONTROL privilege to perform a GRANT CONNECT THROUGH request.

For more information about CTCONTROL, see [CTCONTROL Privilege](#).

You must also have the WITH ADMIN OPTION privilege on each of the roles specified in the WITH ROLE clause of the request.

You must have the DROP USER privilege on each of the users specified as PERMANENT proxy users.

GRANT CONNECT THROUGH Syntax

```
GRANT CONNECT THROUGH trusted_user_name [ WITH TRUST ONLY ]
TO {
```

```

    application_user_name [,...]
    {
        WITH ROLE role_name [...] [ WITH PROFILE profile_name ] |
        WITH PROFILE profile_name
    } |

    PERMANENT permanent_user_name [,...]
    { WITH ROLE profile_name [,...] | WITHOUT ROLE }

} [;]

```

Syntax Elements

trusted_user_name

the name of the user receiving the CONNECT THROUGH privilege.

trusted_user_name must be the name of a permanent user who is already defined in Vantage, but cannot be user DBC.

This grants the specified trusted user the privilege to assert the identity of the specified proxy user names.

Note that *trusted_user_name* should not identify an end user. Its intent is to identify a middle tier application such as a CRM application or Teradata Viewpoint.

If you specify *trusted_user_name* WITH TRUST_ONLY, then you cannot also specify *application_user_name* WITH ROLE or *permanent_user_name* WITH ROLE or WITHOUT ROLE in the same request.

WITH TRUST_ONLY

to require for *trusted_user_name* that all requests to set or remove a Proxy User or Proxy Role must be trusted requests.

This option provides a method of preventing an end user from inserting SET QUERY_BAND requests into the SQL stream and changing the Proxy User or Proxy Role setting without the hosting middle tier application being aware of it.

application_user_name

the name of an application user to whom the proxy logon privilege is to be granted through *trusted_user_name*.

Specifying *application_user_name* is not valid if you also specify the WITH TRUST_ONLY option in the same request.

Application user names can be specified any time that creating a permanent user for every end user of a middle tier application would not be manageable.

Application user names are not defined in Vantage, but they must follow Teradata object naming conventions. An application user name can be anything that represents the client connecting to the middle tier, such as a client name or an ATM identifier.

application_user_name must be unique for the specified *trusted_user_name*.

For more information about *application_user_name*, see [Application Proxy Users](#).

You can specify a maximum of 25 application user names per GRANT CONNECT THROUGH request, but there is no limit to the number of application user names that can be granted logon privileges to a trusted user.

The system adds the names you specify to the CONNECT THROUGH privileges for *trusted_user_name*.

Note that you must specify at least one role in the WITH ROLE clause for each application proxy user that you specify.

permanent_user_name

the name of a permanent user to whom the proxy logon privilege is to be granted through *trusted_user_name*.

Specifying *permanent_user_name* is not valid if you specify the WITH TRUST_ONLY option in the same request.

permanent_user_name must be the name of a permanent Teradata user, but cannot be user DBC.

permanent_user_name must be unique for the specified *trusted_user_name*.

For more information about *permanent_user_name*, see [Permanent Proxy Users](#).

You can specify a maximum of 25 names per GRANT CONNECT THROUGH request, but there is *no* limit to the number of permanent user names that can be granted logon privileges to a trusted user.

The system adds the names you specify to the CONNECT THROUGH privileges for *trusted_user_name*.

WITH ROLE *role_name*

a set of role names to be assigned to the proxy connection.

Specifying WITH ROLE *role_name* is *not* valid if you also specify the WITH TRUST_ONLY option in the same request. This option is only valid when specified for application users or permanent users.

The string that you specify for *role_name* must be the name of an internal role that is already defined in Vantage. An internal role is a role that is managed by *Vantage*, as opposed to an external role, which is directory-managed. See the information about CREATE ROLE in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

In this context, ALL, NONE, and NULL are not valid role names.

You must specify at least one role name for application proxy users, while you can specify `WITHOUT ROLE` for permanent proxy users.

You can specify a maximum of 15 role names per proxy user. Unlike the maxima for permanent and application user names, this is an absolute maximum per proxy user, not a per request maximum.

If a `CONNECT THROUGH` privilege already exists for the specified *trusted_user:permanent_user* or *trusted_user:application_user* name pair, the system adds the roles you specify to the existing `CONNECT THROUGH` privilege.

Note that changes to a `CONNECT THROUGH` privilege are effective immediately, so the next request submitted in a proxy connection following a change to a `CONNECT THROUGH` privilege picks up the new privilege definition.

Note that when roles are dropped, they are removed from the `CONNECT THROUGH` privilege definition, so a rule can be left with no roles defined for it. Proxy users who have privileges that were set with the `WITH ROLE` clause, but later have those roles dropped, will have `PUBLIC` privileges only.

If the addition of new roles causes the number of roles assigned to the privilege to exceed 15, the request aborts and the system returns an error to the requestor. If this happens, you can submit an appropriate `REVOKE CONNECT` request to remove one or more roles from the privilege.

See [REVOKE CONNECT THROUGH](#) for information about how to use this statement.

Vantage activates all of the role names specified in a `WITH ROLE` clause, either by default or when the proxy connection sets `PROXYROLE=ALL`.

If a proxy connection specifies `PROXYROLE=role_name`, the specified role name must be one of the roles granted to the proxy user with the `WITH ROLE` clause. Vantage then sets the role for the proxy connection to this role.

You cannot specify either `NONE` or `NULL` as a *role_name* for this option.

WITHOUT ROLE

to assign the privileges and roles of the specified permanent user to the `CONNECT THROUGH` privilege.

`WITHOUT ROLE` is not valid if you specify the `WITH TRUST_ONLY` option in the same request, nor is it valid for application users.

If a `CONNECT THROUGH` privilege already exists for the specified *trusted_user:permanent_user* with roles, the roles are replaced with the `WITHOUT ROLE` option.

Note that changes to a `CONNECT THROUGH` privilege are effective immediately, so the next request submitted in a proxy connection following a change to a `CONNECT THROUGH` privilege picks up the new privilege definition.

WITH PROFILE *profile_name*

The trusted session is set to the attributes that are defined in the specified *profile_name*. The set of profile attributes are as follows:

- Default Account
- Default Database
- Spool Space
- Temp Space
- Security Constraints
- Query Band

If an application proxy user is assigned a profile, a unique proxy user id is created for the proxy user, and in a proxy connection, the limits for spool space and temporary space are the limits defined in the proxy user's profile.

If a rule exists for the *application_user_name*, you can omit the WITH ROLE clause to assign a profile to the rule. If you add a profile to an existing application proxy user without a profile, the profile is not used in any active connections. Modifying the rule to assign a different profile to the proxy user has the same impact as modifying the current profile.

The rules that determine when changes to parameters in a MODIFY PROFILE request take effect are the same for a trusted session as they are for a regular session. The effect of dropping the profile for a proxy user is the same as the effect of dropping the profile for a user.

CONNECT THROUGH Usage Notes

You should note the following items, when using the CONNECT THROUGH statement:

Granting CONNECT THROUGH to Multiple Trusted Users

A permanent or application user can be granted CONNECT THROUGH privileges through different trusted users with different roles.

Consider the following example requests, both for an application proxy user:

```
GRANT CONNECT THROUGH msi TO debbieg WITH ROLE msirole;
```

```
GRANT CONNECT THROUGH tadmin TO debbieg WITH ROLE tadminrole;
```

After these requests have been successfully submitted, both the *msi* and *tadmin* trusted users have proxy connect privileges for the application user *debbieg*; however, when performing the respective proxy connections, each session for *debbieg* is set to a different role: *msirole* through trusted user *msi* and *tadminrole* through trusted user *tadmin*.

CONNECT THROUGH and Access Logging

The system logs each GRANT CONNECT THROUGH request in the access log when logging has been enabled with BEGIN LOGGING requests such as the following:

```
BEGIN LOGGING ON EACH GRANT;
```

CONNECT THROUGH and Row-Level Security

Proxy users cannot execute SQL requests on row-level security-protected tables.

CONNECT THROUGH and Parameter Markers

Parameter markers are not supported for GRANT CONNECT THROUGH requests.

CONNECT THROUGH and User DBC

You cannot specify user DBC as either the trusted user or as a proxy user in a GRANT CONNECT THROUGH request.

CONNECT THROUGH trusted_user_name WITH TRUST_ONLY

Vantage allows middle-tier applications to categorize an SQL request as trusted or nontrusted, which reduces the risk of users changing a proxy user by injecting SQL code or submitting SQL code via electronic whiteboarding. This implicitly assumes that applications know whether the SQL requests they submit are application-constructed or user-constructed.

When you set the WITH TRUST_ONLY option for a trusted user and a SQL request is flagged as nontrusted, Vantage does not permit SET QUERY_BAND requests to set a new proxy user or to remove the current proxy user.

Vantage enforces this through both client and server software (see [Teradata Client Software Enforcement of Trusted Sessions](#) and [Teradata Server Software Enforcement of Trusted Sessions](#)).

Middle-tier applications that create their own SQL code can run in nontrusted (default) mode, enabling simple backward compatibility.

For more information, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

GRANT CONNECT THROUGH, Trusted Sessions, and User Types

Unless the WITH TRUST_ONLY option is specified, do not use trusted sessions with applications that permit end users to submit or modify SQL requests sent to *Vantage*.

The GRANT CONNECT THROUGH statement is a special version of the SQL form of the GRANT statement. It allows you to grant the CONNECT THROUGH privilege to the specified permanent user or application user through the specified trusted user.

These users are defined in the following table.

Term	Definition
Application user	<p>The name of an application user to which the GRANT CONNECT proxy logon privilege is to be granted.</p> <p>Application user names are not defined in Vantage, but they must follow Teradata object naming conventions.</p> <p>You can specify up to 25 names in a single grant request. The specified names are then added to the grant privileges for the specified trusted user.</p> <p>There is no limit to the number of application user names that can be granted logon privileges to a single trusted user.</p>
Permanent user	<p>A user who is defined to Vantage.</p> <p>In a GRANT CONNECT THROUGH request, this is the name of a user to whom the proxy logon privilege is to be granted.</p> <p>There is no limit to the number of permanent users who can be granted logon privileges to a trusted user.</p>
Trusted user	<p>A permanent user, previously defined to Vantage, who receives the CONNECT THROUGH privilege by means of a GRANT CONNECT THROUGH request.</p> <p>This grants the trusted user the ability to assert the identity of the proxy user specified in the GRANT CONNECT THROUGH request.</p>

Application users and permanent users are collectively referred to as proxy users.

A proxy user is any user who connects to Vantage using the session of a trusted user.

A proxy connection is a Vantage session in which the privileges and profile attributes that are used are those of a proxy user.

Performance management APIs such as MonitorSession and AbortSession identify sessions by their trusted user name.

For enforcement of Teradata Active System Management rules, for a permanent proxy user, rule qualification based on user name, account, and profile is based on the proxy user's name, account, and profile.

For an application proxy user, rule qualification by user name is based on the trusted user's name. If the application proxy user has a profile, qualification by profile is based on the proxy user's profile and qualification by account name on the profile account name. If the application proxy user does not have a profile, qualification based on account and profile is based on the trusted user account and profile.

Proxy User Type	Rights and Session Attributes
permanent user	<ul style="list-style-type: none"> Automatic rights for created objects are granted to the permanent proxy user id. Permanent space charged to the logon user is charged to the permanent proxy user id. The permanent user's profile attributes are automatically assigned to the trusted session. The profile attributes are obtained from the profile assigned to the user or from the user directly. These profile attributes are as follows: <ul style="list-style-type: none"> Default Account Default Database Spool Space

Proxy User Type	Rights and Session Attributes
	<ul style="list-style-type: none"> ◦ Temp Space ◦ Security Constraints ◦ Query Band
application user	<ul style="list-style-type: none"> • No automatic rights are granted for created objects. The effect is the same as if the application proxy user was an unmapped external user. An application proxy user can create table objects within existing databases, based on role privileges, but there is no granting of automatic rights to the application proxy user. The owner of the database can drop the objects created by application users. • The logged on application proxy user does not have permanent space. Any action that charges permanent space to the “logon user” fails. • If the application proxy user has a profile assigned, the session attributes are set to those of the profile. Attributes that are not specified in the profile are set to the values of the trusted user. The following profile attributes are automatically assigned to the trusted session: <ul style="list-style-type: none"> ◦ Default Account ◦ Default Database ◦ Spool Space ◦ Temp Space ◦ Security Constraints ◦ Query Band <p>When the application proxy user has a profile, temp and spool usage is accumulated for the proxy user and the limits are based on the proxy user’s profile.</p> <p>Application proxy users are unique to a trusted user even if they have the same name. Temp and spool usage accumulation is separate for each of these proxy users.</p> <p>If you do not assign a profile to the application proxy user, then the session attributes remain those of the trusted user.</p>

The following built-in functions return information about the proxy connection-related session values:

- USER returns the name of the trusted user for the session.
- CURRENT_USER returns the proxy user name if the user is in a proxy connection; otherwise, CURRENT_USER returns the session user name.
- ROLE returns the current role name for the trusted user.
- CURRENT_ROLE returns the proxy user current role if the user is in a proxy connection; otherwise, it returns the trusted user role name.

See *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145 for details.

If logon restrictions have been set, such as restricting logons by IP address, the system enforces them only for the trusted user logon.

Such restrictions are not enforced when a proxy username is asserted for the session.

Application Proxy Users

An application proxy user name can be anything that represents the client connection with the middle tier application, such as a client name or an ATM identifier. The granularity of application proxy users is largely at your discretion. For example, the term ATM identifier could refer to granularities as different as an identifier for an individual ATM in an ATM network or an identifier for an individual user of any of the ATMs in the network.

The roles that can be active in a proxy connection are those defined in the `CONNECT THROUGH` privilege at the time the proxy connection is made.

You cannot have duplicate application and permanent proxy user names for the same trusted user. For example, consider the following `GRANT CONNECT THROUGH` requests submitted in the order indicated:

```
GRANT CONNECT THROUGH msi TO sbd WITH ROLE finance_role;
```

```
GRANT CONNECT THROUGH msi TO PERMANENT sbd WITH ROLE hr_role;
```

The second request returns a duplicate proxy user name error because the application proxy user named `sbd` already exists as granted through trusted user `msi`.

You must specify at least one role in the `WITH ROLE` clause for each application proxy user.

The following rules apply to the roles assigned to application proxy users:

- Only roles that are defined in the `CONNECT THROUGH` privilege at the time a proxy connection is made can be active in that connection.
All role names specified in the `WITH ROLE` clause are active in the proxy connection by default.
- You cannot set the current role in the proxy connection to `NONE` or `NULL`.
- The privileges for the proxy connection are those for its active roles and `PUBLIC`.

Permanent Proxy Users

A permanent proxy user is an existing permanent user who is defined to Vantage. A `GRANT CONNECT THROUGH` request validates that a permanent proxy user who is specified in that request exists in Vantage. The anticipated use of permanent proxy users is for intranet-type middle tier applications that, for example, might display employee pay stubs or information about available vacation time.

Vantage assigns the name of the permanent proxy user as the creator of any objects created while the proxy connection is in effect.

You cannot have duplicate application and permanent proxy user names for the same trusted user. For example, consider the following `GRANT CONNECT THROUGH` requests submitted in the order indicated:

```
GRANT CONNECT THROUGH crm TO PERMANENT mary WITHOUT ROLE;
```

```
GRANT CONNECT THROUGH crm TO mary WITH ROLE hr_role;
```

The second request returns a duplicate proxy user name error because the permanent proxy user named mary already exists as granted through trusted user crm.

The roles that can be set for a permanent proxy user in a proxy connection are different depending on the WITH ROLES clause in the GRANT CONNECT THROUGH request, as listed in the following table.

IF you specify a ...	THEN ...
WITH ROLE clause in your GRANT CONNECT THROUGH request	<ul style="list-style-type: none"> all role names you specify are active in the proxy connection by default. The roles in the WITH ROLE clause do not need to be granted directly to the user. The GRANT CONNECT THROUGH request grants this privilege by default. the specified roles are the only roles that can be set for the proxy connection. setting the current role to NONE or NULL in the proxy connection is not permitted. the privileges for the proxy connection are those for its active roles and PUBLIC.
WITHOUT ROLE clause in your GRANT CONNECT THROUGH request	<ul style="list-style-type: none"> the default role for the proxy connection is the default role defined for the permanent user. the roles that can be set for the proxy connection are restricted to those granted to the user. In this case, the role in the proxy connection can also be set to NONE or NULL. the privileges for the proxy connection are those granted to the permanent user, its active roles, and PUBLIC.

Software Enforcement of Trusted Sessions

This section describes Teradata client and server software enforcement of Trusted Sessions.

Teradata Client Software Enforcement of Trusted Sessions

Client software enables the enforcement of this security feature by providing code developers with the ability to indicate whether an SQL request is trusted or not. To do this for a CLlv2 application, use the Trusted flag of the CLlv2 Options parcel to specify Y if a request is trusted or N if a request is not.

NOTICE

You should always code your trusted user-based middle tier applications using Parcel Mode Fetch CLlv2 operations only. If you code the application using a CLlv2 Buffer Mode Fetch operation, it becomes possible for nontrusted users to construct their own Options parcels and inject nontrusted SQL code into the application.

Refer to either *Teradata® Call-Level Interface Version 2 Reference for Workstation-Attached Systems*, B035-2418 or *Teradata® Call-Level Interface Version 2 Reference for Mainframe-Attached Systems*, B035-2417, as appropriate, for detailed information about the Options parcel and how CLlv2 applications can be coded using Parcel Mode Fetch operations and the DBCAREA.

Each of the Teradata application APIs provides a mechanism for applications to specify whether requests are to be trusted or not. This mechanism prohibits an SQL request from being upgraded from a nontrusted status to trusted status. Refer to the appropriate Teradata Tools and Utilities documentation to determine how the API you are using for your middle tier application handles this feature.

Teradata Server Software Enforcement of Trusted Sessions

It is assumed that applications that use the trusted sessions feature will submit both trusted and nontrusted requests. It is the application, and not Vantage, that knows which requests can be trusted and which cannot. Vantage just verifies that the SET QUERY_BAND request with a PROXYUSER is a trusted request if the logon user has the WITH TRUST_ONLY option set.

Server software enforces trusted requests by means of GRANT CONNECT THROUGH requests that set trusted users to have the TRUST_ONLY privilege. If you specify the WITH TRUST_ONLY option for a trusted user, Vantage validates that a request to set a Proxy User is trusted; otherwise, it aborts the request and returns an error to the requestor.

Suppose a middle tier application submits the following SET QUERY_BAND request with a PROXYUSER.

```
SET QUERY_BAND = 'PROXYUSER=client787';
```

The following outcomes are possible, depending on whether the request is trusted and what CONNECT THROUGH privileges have been granted to the application user that submits it.

IF the trusted user has ...	AND the Trusted flag of the Options parcel ...	THEN the request is ...
been granted the TRUST_ONLY privilege	is set to Y	trusted. SET QUERY_BAND PROXYUSER requests are permitted.
	is set to N	not trusted. SET QUERY_BAND PROXYUSER requests are aborted.

IF the trusted user has ...	AND the Trusted flag of the Options parcel ...	THEN the request is ...
not been granted the TRUST_ONLY privilege	is set to Y	neither trusted nor not trusted. Vantage ignores the setting of the flag.
	is set to N	neither trusted nor not trusted. Vantage ignores the setting of the flag. This is the default setting.

Dictionary Storage of CONNECT THROUGH Metadata

The dictionary table `DBC.ConnectRulesTbl` contains information on which proxy users can connect through which trusted users and what roles are available to make proxy connections. The table contains one row for every *trusted_user_name:proxy_user_name* combination.

When the system processes a GRANT CONNECT THROUGH request, it writes a row to `DBC.ConnectRulesTbl` for each of the following pairs that you specify:

- Trusted user name:permanent user name
- Trusted user name:application user name

The row persists until either you drop the trusted user or the permanent user.

When you grant WITH TRUST_ONLY to a trusted user, *Vantage* adds a row to `DBC.ConnectRulesTbl` that contains the following information:

- TrustUserId
- ProxyUser=*space_characters*
- TrustOnly=Y

Vantage also updates all rows in `DBC.ConnectRulesTbl` with the value for TrustUserId=*specified_TrustUserID* to set TrustOnly=Y.

When you revoke WITH TRUST_ONLY from a trusted user, *Vantage* updates all rows in `DBC.ConnectRulesTbl` where TrustUserId=*specified_TrustUserID* to set TrustOnly=N.

To provide an audit trail for the management of the rules, *Vantage* retains the row if the privilege is revoked, but does *not* drop the user.

The following list indicates the values for `DBC.ConnectRulesTbl.GrantStatus` when the TRUST_ONLY privilege is granted to a trusted user or not:

- If `DBC.ConnectRulesTbl.GrantStatus` is set to G, then the TRUST_ONLY privilege is granted to *trusted_user*.
- If `DBC.ConnectRulesTbl.GrantStatus` is set to R, then TRUST_ONLY privilege is revoked from *trusted_user*.

Examples of Using GRANT CONNECT THROUGH

Granting CONNECT THROUGH privilege to a Permanent User

The following GRANT CONNECT THROUGH request grants the CONNECT THROUGH privilege to permanent user *sbd* with the assigned proxy connection role *admin* through trusted user *viewpoint*.

```
GRANT CONNECT THROUGH viewpoint
TO PERMANENT sbd
WITH ROLE admin;
```

After this request has been successfully submitted, user *sbd* has proxy connect privileges through the trusted user *viewpoint*, and whenever *sbd* makes a proxy connection, the system assigns him to the *admin* role.

Specifying Roles for a Proxy Connection

All roles specified in the WITH ROLE clause of this example are active by default in the proxy connection.

If no ProxyRole is set for application user *dg120* in the proxy connection, the active roles are *salesrole1*, *salesrole2*, and *salesrole3*.

The proxy connection can be set to one role that is in the WITH ROLE clause. For example, the ProxyRole for application user *dg120* can be set to *salesrole1*, *salesrole2*, or *salesrole3*, but no other roles are permitted.

```
GRANT CONNECT THROUGH dcm
TO dg120, ks392, lm190
WITH ROLE salesrole1, salesrole2, salesrole3;
```

Specifying WITHOUT ROLE for a Proxy Connection

When you set a WITHOUT ROLE clause for a permanent proxy user, as the following request demonstrates, the system uses the privileges and roles granted to that permanent user, and the default proxy role is the default role defined for the proxy permanent user.

The roles that can be set for the proxy user are restricted to the roles granted to the proxy permanent user.

```
GRANT CONNECT THROUGH trm
TO PERMANENT accting
WITHOUT ROLE;
```

Specifying the WITH TRUST_ONLY Option

The WITH TRUST_ONLY option restricts a middle tier application from submitting SET QUERY_BAND requests that set, change, or remove a PROXYUSER or PROXYROLE for the case where a trusted request is required.

The following request restricts trusted *user_name* from submitting SET QUERY_BAND requests from a middle tier application unless the application sets the Trusted field in the Options parcel to Y, which indicates that the request is trusted.

See *Teradata® Call-Level Interface Version 2 Reference for Mainframe-Attached Systems*, B035-2417 or *Teradata® Call-Level Interface Version 2 Reference for Workstation-Attached Systems*, B035-2418 for details about the Options parcel.

This assumes that the middle tier application uses the CLIV2 API. Refer to the appropriate Teradata Tools and Utilities manual for your application to determine the mechanism for specifying this information for that API.

```
GRANT CONNECT THROUGH user_name WITH TRUST_ONLY;
```

Note that if the application does not set the Trusted field in the Options parcel to Y, the system aborts any SET QUERY_BAND request that *user_name* submits.

See

Related Information

Topic	Reference Source
Revoking proxy connections	REVOKE CONNECT THROUGH
Setting query bands to enable proxy connections	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
Setting a nontrusted request	<i>Teradata JDBC Driver Reference</i> , available at https://teradata-docs.s3.amazonaws.com/doc/connectivity/jdbc/reference/current/frameset.html
Administering trusted sessions	<i>Teradata Vantage™ - Database Administration</i> , B035-1093
Security issues related to trusted sessions	<i>Teradata Vantage™ - Advanced SQL Engine Security Administration</i> , B035-1100
The dictionary attributes of trusted sessions	<i>Teradata Vantage™ - Data Dictionary</i> , B035-1092
The Options parcel	<ul style="list-style-type: none"> • <i>Teradata® Call-Level Interface Version 2 Reference for Mainframe-Attached Systems</i>, B035-2417 • <i>Teradata® Call-Level Interface Version 2 Reference for Workstation-Attached Systems</i>, B035-2418

GRANT LOGON

GRANT LOGON does either of the following:

- Gives specific users permission to log on to the database from one or more specific client systems.
- Changes the current system logon defaults.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must have the EXECUTE privilege on the DBC.LogonRule macro to perform GRANT LOGON.

GRANT LOGON Syntax

```
GRANT LOGON ON
  { host_id [,...] | ALL }
  { AS DEFAULT | { TO | FROM } user_name [,...] }
  [ WITH NULL PASSWORD ] [;]
```

Syntax Elements

host_id

A mainframe connection or a workstation network connection that is currently defined to the system by the hardware configuration data. The interface need not be operational.

You can define multiple host IDs per node.

The host ID for the system console is 0. For any other connector, the value for *host_id* ranges from 1 through 32,767.

ALL

Any source through which a logon is attempted, including the system console.

AS DEFAULT

The current default for the specified *host_id* set is to be changed, without residual conditions, as defined in this GRANT LOGON statement. A statement with AS DEFAULT has no effect on the access granted to or revoked from particular user names.

A statement that sets the default for a specific *host_id* takes precedence over a statement that sets the default for ALL client systems.

TO FROM

The clause that specifies the recipient of the GRANT results.

user_name

One or more user names whose current system logon defaults are to be changed.

- You cannot specify the name DBC as a user name in a GRANT LOGON statement. A statement that includes this name returns an error message.
- The product of the number of host IDs times the number of user names cannot exceed 25.

WITH NULL PASSWORD

Permits a logon string that has no password to be accepted from the specified client system community.

The initial default is that all logon requests must include a password. The WITH NULL PASSWORD option, in conjunction with a TDP security exit procedure, negates that default.

This option implies that the user has been authenticated externally and not by the database. See *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100 for more information.

GRANT LOGON Usage Notes

System Privilege Checks

A statement that includes the AS DEFAULT option has no effect on the logon access granted to or revoked from specific user names. A user named in a GRANT LOGON statement can always access the applicable client system even if that client has a default of REVOKE, and a user named in a REVOKE LOGON statement cannot access the applicable client system even if that client has a default of GRANT.

When a GRANT LOGON statement is submitted, the system checks that the requesting user has EXECUTE privilege on the system macro associated with that statement. However, no checks are made on whether the user names defined in the statement apply to users owned by the requesting user. If the submitted statement cannot be verified because, for example, it specifies a non-valid user name or an invalid host ID, no action is taken on the statement.

GRANT LOGON for One or More User Names

When a GRANT LOGON statement is processed for one or more user names, a logon control record is created for each *user_name/host_ID* pair specified. Any existing control record for a particular *user_name/host_ID* pair is replaced. The logon control record created for a particular user name persists until that user is dropped.

See also the information for DROP USER in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Logging On With No Logon Control Record for a User Name

If there is no logon control record for a specific user name, access to Vantage under that name is governed by the current default for the host ID through which the logon is entered.

Logging On as User DBC

Logging on to Vantage as user *DBC* requires a password. If the submitted password is correct, the logon is accepted regardless of the current default for the applicable host ID. This prevents any opportunity to lock out all client systems from Vantage.

You cannot grant logon to user name *DBC*.

Logging On Without a Password

When Vantage is connected to multiple client systems, the initial default is that logon permission is granted to all users from all client systems, and that all logons must include a password.

Logging on without a password requires the following things:

- A GRANT LOGON statement that includes the WITH NULL PASSWORD option must be performed for the defined user name or must be the default for the host ID.
- A security exit installed in the client system must acknowledge that the logon string for this user is valid without a password.

Client Type	Client Applications Where the Security Exit is Located
Mainframe-attached	<ul style="list-style-type: none"> • TDP • CLI
Workstation-attached	CLI

Logging On Using External Authentication

External authentication permits a user to log on to a computer one time and access a database without providing a user name, password, or account name. To enable this, you must explicitly map any directory users who are not already mapped to Vantage users in the directory. If you do not create an explicit mapping between an unmapped user and Vantage, that directory user cannot log on to Vantage.

You can explicitly map a directory user to the following objects:

- EXTUSER.

Mapping to EXTUSER is the most commonly used method.

You cannot assign row-level security privileges to EXTUSER.

Note:

If the AuthorizationSupported property is set to yes in the TdgssUserConfigFile.xml file, and a user attempts to log on to the database, the user is authenticated by the directory. If auto provisioning is turned on, then the user is not logged on as EXTUSER. Instead, the database creates an account for the user in DBC.Dbase and logs the user on using that account. Auto provisioned users are not permanent users and must always authenticate against the directory.

For more information about EXTUSER and auto provisioning, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

- A Profile.
- A Role.
- A Teradata user.

Another way to do this is to grant the user logon privileges with a null password.

The following procedure creates a user who can log on to the system through a gateway that does not have the Append Domain Name option set using the Gtwcontrol utility. This user is already defined as user *rh*h.

1. Create user *rh*h using the following CREATE USER request:

```
CREATE USER rh AS
  PERM = 10000000,
  PASSWORD = rh;
```

2. Grant user *rh*h the following logon privileges using a GRANT LOGON request:

```
GRANT LOGON ON ALL
  TO rh
  WITH NULL PASSWORD;
```

The following procedure creates a Vantage user who can log on to a Teradata system through a gateway that has Append Domain Name set. This user is already defined as user *rh*h and her account is in the *esw2kdev* domain.

1. Create user *rh*h using the following CREATE USER request:

```
CREATE USER "rh@esw2kdev" AS
  PERM = 10000000,
  PASSWORD = rh;
```

2. Grant user *rh*h the following logon privileges using a GRANT LOGON request:

```
GRANT LOGON ON ALL
  TO "rhh@esw2kdev"
  WITH NULL PASSWORD;
```

Related Information

Subject	See ...
revoking logon privileges	REVOKE LOGON
external users	<i>Teradata Vantage™ - Advanced SQL Engine Security Administration</i> , B035-1100
external authentication	<ul style="list-style-type: none"> • <i>Teradata Vantage™ - Advanced SQL Engine Security Administration</i>, B035-1100 • <i>Teradata Vantage™ - Database Administration</i>, B035-1093 • <i>Teradata Vantage™ - Database Utilities</i>, B035-1102

GRANT MAP

Grant existing contiguous or sparse maps to users and roles.

Users or roles who have been granted a map can specify the map in the MAP, DEFAULT MAP, and EXECUTE MAP options of SQL statements.

Sparse maps can only be granted to users and roles within the same secure zone as the sparse map.

Required Privileges

You must have the GRANT MAP WITH GRANT OPTION privilege for the specified map to grant a map privilege.

GRANT MAP Syntax

```
GRANT MAP map_name [,...] TO {
  { user_name | role_name } [,...] |
  PUBLIC |
  user_name [,...] WITH GRANT OPTION
} [;]
```

Syntax Elements

map_name

Name of existing contiguous or sparse map.

You cannot specify TD_DataDictionaryMap or TD_GlobalMap.

user_name

Name of user to be granted the map.

WITH GRANT OPTION

The user who is granted the map can grant the map to other users.

role_name

Name of role to be granted the map.

You cannot grant a map WITH GRANT OPTION to a role.

PUBLIC

Privileges are to be inherited by all existing and future Vantage users.

GRANT ZONE

When using Teradata Secure Zones, grants access to a zone to users or roles that are not zone users.

GRANT ZONE does not automatically grant users access to database objects within the zone. A zone user must grant Discretionary Access Control privileges to zone guests before access is permitted.

Users and roles that are granted zone access using this syntax are called zone guests. Zone users must be created within the zone using the CREATE USER syntax.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must be the creator of the zone to use GRANT ZONE to create zone guests.

Restricted Privileges

Zone creators cannot use GRANT ZONE syntax to make themselves guests in a zone that they created.

Only zone users can grant privileges on database objects within the zone to zone guests, but zone users cannot grant privileges on zone objects to a zone guest using the WITH GRANT OPTION.

A zone guest cannot grant access to zone objects to other users.

GRANT ZONE Syntax

```
GRANT ZONE zone_name [,...] TO { user_name | role_name } [,...] [;]
```

Syntax Elements

zone_name

The name of the zone. You can specify up to 25 names in a comma-separated list.

The zone must already exist.

user_name

role_name

The name of the user or the role. You can specify up to 25 names in a comma-separated list.

Zone guests can only be users or roles. The users or roles must already exist outside the zone.

You cannot make a user or a role in another zone into a zone guest.

Related Information

For information about revoking zone access, see [REVOKE ZONE](#).

For information about implementing Teradata Secure Zones, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

GRANT ZONE OVERRIDE

When using Teradata Secure Zones, grants a user who does not belong to any zone the ability to collect dictionary data system-wide and across zones. This privilege applies only to access to DBC tables. It does not include data table access.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

Only user DBC receives this privilege during system initialization and only user DBC can grant the zone override privilege.

Restricted Privileges

You cannot use the WITH GRANT OPTION when granting this privilege.

GRANT ZONE OVERRIDE Syntax

```
GRANT ZONE OVERRIDE TO user_name [,...] [;]
```

Syntax Elements

user_name

The name of a user to whom you want to grant DBC table access across all zones to. You can specify up to 25 names in a comma-separated list.

Usage Notes

Users who are not zone users but who have the ZONE OVERRIDE privilege and also have EXECUTE privilege on DBQLAccessMacro can enable and disable query logging on all database objects in a zone.

Users with ZONE OVERRIDE privileges who also have SELECT privileges on V views have system-wide access.

Related Information

For information about revoking ZONE OVERRIDE privileges, see [REVOKE ZONE OVERRIDE](#).

For information about implementing Teradata Secure Zones, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

REVOKE

REVOKE rescinds explicit privileges from one or more users, proxy users, databases, or roles. The privileges might have been conferred either automatically or by a previous GRANT statement.

REVOKE has forms that differ in function and syntax:

REVOKE Form	Purpose
REVOKE (Monitor Form)	Revoke performance monitoring.
REVOKE (Role Form)	Revoke role membership to users and other roles.
REVOKE (SQL Form)	Revoke access to, creation of, or logging of, database objects.
REVOKE LOGON	Revoke system logon privileges.
REVOKE MAP	Revoke existing contiguous or sparse maps from users and roles.
REVOKE ZONE	Revoke zone guest status from users or roles.
REVOKE CONNECT THROUGH	Revoke the ability to connect as a proxy permanent or proxy application user through a trusted user.

ANSI Compliance

REVOKE (Role Form) and REVOKE (SQL Form) are ANSI/ISO SQL:2011-compliant. The other forms of REVOKE are extensions to the ANSI/ISO SQL:2011 standard.

Privileges That REVOKE Can and Cannot Remove

The REVOKE statements operate on explicit privileges recorded in the DBC.AccessRights table. Only those explicit privileges that have been granted automatically or explicitly can be revoked.

In most cases, implicit privileges only permit the user holding them to grant them to others, not to perform the SQL statements that correspond to those privileges. Generally, you must have explicit privileges to perform access protected SQL statements. The main exceptions to this rule are privileges on views, macros, and procedures.

Note:

Implicit privileges are determined by ownership and cannot be revoked. You can affect implicit privileges by using the GIVE statement to change ownership. For more information, see [GIVE](#).

REVOKE (SQL Form) and REVOKE (Monitor Form) Differences

The SQL and MONITOR forms of REVOKE are separate statements. To revoke all privileges for a user, including MONITOR, the grantor must perform both of the following statements:

```
REVOKE ALL PRIVILEGES ON object
FROM user_name;

REVOKE MONITOR PRIVILEGES
FROM user_name;
```

Both statements assume that you have the appropriate privileges, either implicitly or explicitly, WITH GRANT OPTION.

REVOKE (Monitor Form)

Revokes system-wide performance monitoring privileges.

REVOKE MONITOR takes effect immediately when the revoked users issue their next statement.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

For REVOKE (Monitor), the user submitting a REVOKE statement must have all the privileges that are to be revoked with GRANT option.

REVOKE Syntax (Monitor Form)

```
REVOKE [ GRANT OPTION FOR ]
{ MONITOR [ PRIVILEGES | BUT NOT monitor_privilege [,...] ] |
  monitor_privilege [,...]
}
{ TO | FROM } { revokee [,...] | PUBLIC } [;]
```

revokee

```
[ALL] user_name
```

Syntax Elements

GRANT OPTION FOR

Indicates that only the grant authority is removed from the specified privileges for the specified grantees.

REVOKE GRANT OPTION FOR revokes the privilege of the recipient to grant, but does not revoke the stated privileges themselves.

MONITOR PRIVILEGES

Revokes from the specified user all privileges, except MONITOR, that can be granted on the specified object, and that are possessed WITH GRANT OPTION by the user executing the REVOKE.

To revoke all privileges for a user, including MONITOR, you must perform at least two statements as follows:

```
REVOKE ALL PRIVILEGES ON object FROM user_name;
```

```
REVOKE MONITOR PRIVILEGES FROM user_name;
```

ALL PRIVILEGES means all *database* privileges.

If you specify the ALL PRIVILEGES option, then only the privileges possessed with grant authority are revoked from the grantee.

MONITOR PRIVILEGES indicates *all* monitoring privileges.

The ANSI/ISO SQL:2011 standard requires ALL to be followed by PRIVILEGES.

MONITOR BUT NOT

Revokes all of the MONITOR privileges except those specified by *monitor_privilege*.

monitor_privilege

The monitoring privileges that are to be revoked.

You can specify as many of the following privileges as you wish, separated by commas:

- ABORTSESSION
- CTCONTROL
- MONRESOURCE
- MONSESSION
- SETRESRATE
- SETSESSRATE

See [Monitor Privileges](#) for the definitions of these monitoring privileges.

ALL

Indicates that the monitoring privileges are to be revoked from the named database or user, and every database or user owned by that database or user now and in the future.

ALL *user_name* is a Teradata extension to the ANSI/ISO SQL:2011 standard.

user_name

The database or user from whom monitoring privileges are to be revoked. You can specify up to 25 names.

PUBLIC

Indicates that the monitoring privileges are to be revoked from all currently defined and future Vantage users.

ALL DBC is equivalent to PUBLIC.

Usage Notes

Rules for Using REVOKE (Monitor Form)

Keyword	Meaning
ALL PRIVILEGES	database-related privileges are affected.
MONITOR PRIVILEGES	system monitoring privileges are affected.

If you violate either one of the following restrictions, a failure response occurs:

- Use REVOKE MONITOR to specify an ON *object* clause.
- Use any other type of REVOKE statement and do not specify an ON *object* clause.

The REVOKE statement takes effect as soon as the recipient issues their next statement.

Revocable Privileges

- Implicit privileges cannot be revoked.
- Revoking the GRANT OPTION for a privilege from a user immediately revokes the privilege of that user to grant that privilege. However, it does not affect privileges that have already been granted by that user to others.
- The user who revokes a privilege from another need not be the grantor of that privilege. If a user receives the same privilege from two or more grantors, any grantor with the necessary privileges can revoke that privilege from the user and from other grantees.
- If a privilege that was granted to ALL *user_name* is revoked from *user_name* , then the privilege is not granted automatically to future users owned by *user_name*.

Example of Revoking All Privileges From a User

This statement pair revokes all SQL privileges on the accounting database and all monitor privileges on the system from a user named ted. This example assumes that the revoker has the necessary privileges either implicitly or explicitly WITH GRANT OPTION.

```
REVOKE ALL PRIVILEGES ON accounting
FROM ted;
```

```
REVOKE MONITOR PRIVILEGES
FROM ted;
```

Example of Revoking Specific Privileges From a User

This statement revokes the ABORTSESSION, MONSESSION, and SETSESSRATE monitor privileges from user pls. This example assumes that the revoker has the necessary privileges either implicitly or explicitly WITH GRANT OPTION.

```
REVOKE ABORTSESSION, MONSESSION, SETSESSRATE FROM pls;
```

For information about granting monitor privileges, see [GRANT \(Monitor Form\)](#).

REVOKE (Role Form)

Revokes a role from users or other roles.

ANSI Compliance

This statement is ANSI SQL:2011 compliant.

Required Privileges

To revoke a role, you must have the WITH ADMIN OPTION privilege on it. The following users can revoke role membership:

- User DBC.
- A user who was granted the specified role WITH ADMIN OPTION.
A role is automatically granted to the creator of the role WITH ADMIN OPTION.
- A user who has an active role to which the specified role was granted WITH ADMIN OPTION. An active role can be either a current role or a nested role of a current role.

The Effects of Revoking A Role

Roles define privileges on database objects. A user who activates a role inherits all the privileges for the role and its nested roles. A user can only activate a role that has been granted to that user.

Users can undergo role changes within their organization. An administrator can revoke a role when users no longer require access to the objects that the role has privileges to. An administrator can also revoke the WITH ADMIN OPTION privilege on a role when users no longer require the privilege to grant the role to users or other roles.

Authorized users can revoke the WITH ADMIN OPTION privilege on a role from the creator of the role.

The effect of revoking a role is immediate. Users who are logged on with the revoked role as the current role or a nested role of the current role lose the privileges that the role defines.

Users who have a default role set to the revoked role do not receive errors or warnings the next time they log on. However, the system does not use the obsolete default role for privilege validation. If the role is again granted to users, the default role again becomes the current role the next time users log on.

REVOKE Syntax (Role Form)

```
REVOKE [ ADMIN OPTION FOR ] role_name [,...]
      { TO | FROM } { user_name | role_name } [,...] [;]
```

Syntax Elements

ADMIN OPTION FOR

Indicates that the roles and users who were granted this role lose the privilege to use GRANT, REVOKE, and DROP ROLE statements to administer the specified role.

If you do not specify ADMIN OPTION FOR in the REVOKE request, the system revokes the specified role from the roles or users to which it was granted.

role_name

One or more comma-separated names of roles that are being revoked.

You can specify a maximum of 25 names per REVOKE request.

The system ignores duplicate role names.

TO
FROM
user_name
role_name

The names of roles or users or both from which the role or the ability to administer the role is being revoked.

The system does not return errors for users or roles that were not previously granted the specified role.

The TO keyword is a Teradata extension to ANSI/ISO SQL:2011.

Example of Revoking One Role and Granting Another

If user *marks* gets a promotion from sales to management, the DBA can use the following statements to revoke the sales role and grant the management role:

```
REVOKE sales
FROM marks;

GRANT management
TO marks;
```

Related Information

Subject	Reference Source
granting roles to users and other roles	GRANT (Role Form)
assigning default roles to users	Information about CREATE USER and MODIFY USER in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
changing the current role for a session	Information about SET ROLE in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144

REVOKE (SQL Form)

Revokes one or more explicit privileges on a database, user, table, view, procedure, UDF, function mapping, or macro from a role, group of roles, user, or group of users or removes the GRANT option from explicit privileges.

REVOKE takes effect immediately when revoked users issue their next request.

For information about revoking the NONTEMPORAL privilege, see *Teradata Vantage™ - Temporal Table Support*, B035-1182.

ANSI Compliance

This statement is ANSI SQL:2011 compliant, but includes non-ANSI Teradata extensions.

Required Privileges

To revoke a privilege, you must first have the privileges to grant it. You must either own the database object, or someone must first grant the privilege, either automatically or explicitly, to you, either directly or by means of a role, using WITH GRANT OPTION.

If the object is a view or macro, the submitting user also must have the applicable privileges, WITH GRANT OPTION, on the objects referenced by the view or macro.

REVOKE Syntax (SQL Form)

```
REVOKE [ GRANT OPTION FOR ] {
  { ALL [ PRIVILEGES ] | [ ALL BUT ] privilege [,...] } ON object |
  map_privilege [,...] |
  role_privilege [,...] |
  profile_privilege [,...] |
  zone_privilege [,...]
}
{ TO | FROM }
{ user_revoker [,...] | PUBLIC | role_revoker [,...] } [;]
```

object

```
{ { database_name | user_name | role_name | PUBLIC } |
  { database_name. | user_name. } object_name |
  object_name |
  PROCEDURE [ database_name. | user_name. ] procedure_name |
  SPECIFIC FUNCTION [ database_name. | user_name. ]
```

```

specific_function_name |

FUNCTION [ database_name. | user_name. ] function_name
    ( [ function_parameter [, ...] ] ) |

TYPE [SYSUDTLIB.] UDT_name
}

```

user_revokee

```
[ALL] user_name
```

role_revokee

```
[ database_name. | user_name. ] role_name
```

function_parameter

```
[ parameter_name ] data_type
```

data_type

```

{ INTEGER | SMALLINT | BIGINT | BYTEINT | DATE |

  { TIME | TIMESTAMP } [ (fractional_seconds_precision) ] [WITH TIME
ZONE] |

  INTERVAL YEAR [(precision)] [TO MONTH] |

  INTERVAL MONTH [(precision)] |

  INTERVAL DAY [(precision)]
    [TO { HOUR | MINUTE | SECOND [(fractional_seconds_precision)] }] |

  INTERVAL HOUR [(precision)]
    [TO { MINUTE | SECOND [(fractional_seconds_precision)] }] |

  INTERVAL MINUTE [(precision)] [TO SECOND
[(fractional_seconds_precision)]] |

  INTERVAL SECOND [ (precision [, fractional_seconds_precision] ) ] |

```

```

PERIOD (DATE) |

PERIOD ( { TIME | TIMESTAMP } [(precision)] [WITH TIME ZONE] ) |

REAL |

DOUBLE PRECISION |

FLOAT [ (integer) ] |

NUMBER [ ( { integer | * } [, integer]... ) ] |

{ DECIMAL | NUMERIC } [ ( integer [, integer]... ) ] |

{ CHAR | BYTE | GRAPHIC } [ (integer) ] |

{ VARCHAR | CHAR VARYING | VARBYTE | VARGRAPHIC } [ (integer) ] |

LONG VARCHAR |

LONG VARGRAPHIC |

{ BINARY LARGE OBJECT | BLOB | CHARACTER LARGE OBJECT | CLOB }
  ( integer [ G | K | M ] ) |

[SYSUDTLIB.] { XML | XMLTYPE } [ ( integer [ G | K | M ] ) ]
  [ INLINE LENGTH integer ] |

[SYSUDTLIB.] JSON [ ( integer [ K | M ] ) ] [ INLINE LENGTH integer ]
  [ CHARACTER SET { UNICODE | LATIN } | STORAGE FORMAT { BSON |
UBJSON } ] |

[SYSUDTLIB.] ST_GEOMETRY [ (integer [ K | M ]) ] [ INLINE LENGTH
integer ] |

[SYSUDTLIB.] DATASET [ (integer [ K | M ]) ]
  [ INLINE LENGTH integer ] [ storage_format ] |

[SYSUDTLIB.] { UDT_name | MBR | ARRAY_name | VARRAY_name }
}

```

Syntax Elements

GRANT OPTION FOR

Only the WITH GRANT OPTION authority is removed from the specified privilege set for the specified grantees for the corresponding explicit privileges they have on the specified database object.

REVOKE GRANT OPTION FOR revokes the ability to grant the specified privilege set to others, but does not revoke the explicit privileges themselves from the specified users or roles.

This option does not apply to grantees that are roles.

ALL PRIVILEGES

Revoke from the specified user or role all explicitly granted non-MONITOR database, but not table, privileges that can be granted on the specified object, and that are held, either implicitly or explicitly, WITH GRANT OPTION by the user executing the REVOKE.

PRIVILEGES is optional.

REVOKE ALL does not revoke the INDEX and REFERENCES privileges. To revoke these privileges, you must do so explicitly.

To revoke all explicit database privileges for a user, including MONITOR, the revoker must perform at least two statements, shown as follows:

```
REVOKE ALL PRIVILEGES ON object
FROM user_name;

REVOKE MONITOR PRIVILEGES
FROM user_name;
```

ALL PRIVILEGES includes all explicit database privileges.

If you specify ALL PRIVILEGES, then only the explicit privileges held with grant authority are revoked from the grantee.

ANSI/ISO SQL requires ALL to be followed by the keyword PRIVILEGES.

ALL BUT

Revoke all explicit database privileges from the specified database object, except those specified in the privilege set, that can be granted on the specified object and that are held, either implicitly or explicitly, WITH GRANT OPTION by the user performing the REVOKE statement.

privilege

One or more of the privileges listed in [Privilege Dictionary](#).

INSERT, REFERENCES, SELECT, and UPDATE have separate table- and column-level options. See [GRANT \(SQL Form\)](#).

You cannot grant the UPDATE privilege on a GENERATED ALWAYS identity column.

The ANSI/ISO SQL:2011 standard does not support REVOKE on a database or user, or ALL BUT. These are Teradata extensions to the ANSI/ISO SQL-2011 standard.

You can specify any combination of privileges; however, the user submitting the request must itself have all of the specified privileges, either implicitly or explicitly, WITH GRANT OPTION.

If DATABASE, FUNCTION, MACRO, PROCEDURE, PROFILE, ROLE, TABLE, TRIGGER, USER, VIEW, or ZONE is specified without CREATE or DROP, both CREATE and DROP are revoked.

If you specify CHECKPOINT, then the privilege is revoked both for the SQL statement and for the HUT commands DUMP and RESTORE.

If you specify either DUMP or RESTORE individually, then the system revokes the privilege to perform the corresponding HUT command only.

If you specify RESTORE, then the system also revokes the privileges to perform the HUT commands ROLLBACK, ROLLFORWARD, and DELETE JOURNAL.

You cannot revoke privileges on a trigger, only on its containing database or its subject table.

map_privilege

Revoke create and drop map privileges from users and roles.

You can specify the following privileges:

- CREATE MAP
- DROP MAP
- MAP

MAP is shorthand, not a separate privilege. Specify MAP to revoke both the CREATE MAP and DROP MAP privileges.

role_privilege

One of the following privileges:

- CREATE ROLE
- DROP ROLE
- ROLE

ROLE is shorthand for both CREATE ROLE and DROP ROLE, not a separate privilege.

Role privileges can only be revoked from a set of users or a role. They cannot be revoked from an object.

profile_privilege

One of the following privileges:

- CREATE PROFILE
- DROP PROFILE
- PROFILE

PROFILE is shorthand for both CREATE PROFILE and DROP PROFILE, not a separate privilege.

Profile privileges can only be revoked from a set of users or a role. They cannot be revoked from an object.

zone_privilege

One of the following privileges:

- CREATE ZONE
- DROP ZONE
- ZONE

ZONE is shorthand for both CREATE ZONE and DROP ZONE, not a separate privilege.

Zone privileges can only be revoked from a set of users or a zone. They cannot be revoked from an object.

For information about implementing Teradata Secure Zones, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

database_name

user_name

role_name

PUBLIC

Name of a database, user, or role on which the explicitly granted privileges are to be revoked, or PUBLIC. All objects contained by this database or user space are affected.

database_name.object_name

user_name.object_name

Name of the immediately owning database and optionally and the name of the object or the user name and the name of the object (table, view, procedure, join index, or macro) on which explicitly granted privileges are being revoked.

Only the specified object, and not all objects in the database or user space, is affected by revoking the privilege set.

object_name

Name of the table, view, join index, procedure, function, function mapping, macro, or authorization name from which explicitly granted privileges are to be revoked.

You should always qualify object names when revoking privileges because the system checks database names before it checks object names.

- If the object name is not qualified and the system finds a database with that name, Vantage assumes it is a database name.
- If the object name is not qualified and no database having that name is found, Vantage assumes it is an object within the current default database.
- If neither a database nor an object is found with the specified name, the system returns an error to the requestor.

PROCEDURE

database_name

user_name

procedure_name

Name of the procedure from which privileges are to be revoked.

database_name and *user_name* are optional. The procedure name can be qualified by its containing database or user when necessary.

SPECIFIC FUNCTION

database_name

user_name

specific_function_name

Specific name of the UDF from which privileges are to be revoked.

database_name and *user_name* are optional. The specific function name can be qualified by its containing database or user when necessary.

FUNCTION

database_name

user_name

function_name

Name of the UDF from which privileges are to be revoked.

database_name and *user_name* are optional. The specific function name can be qualified by its containing database or user when necessary.

parameter_name***data_type***

Parenthetical comma-separated list of data types and optional parameter names for the variables to be passed to the UDF. This is used to uniquely identify overdetermined function names.

BLOB and CLOB types must be represented by a locator. For a description of locators, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146. Vantage does not support in-memory LOB parameters: an AS LOCATOR phrase must be specified for each LOB parameter and return value.

You must specify opening and closing parentheses, even if no parameters are passed to the function.

The data type associated with each parameter is the type of the parameter or returned value. All Teradata data types are valid. Character data can also specify a CHARACTER SET clause that specifies the server character set used for the parameter.

TYPE**SYSUDTLIB.*****UDT_name***

Name of a UDT on which a privilege set is to be revoked.

SYSUDTLIB. is optional.

The various TYPE privileges can be revoked as follows:

- You can revoke UDTMETHOD only from the SYSUDTLIB database.
- You can revoke UDTTYPE only from the SYSUDTLIB database.
- You can revoke UDTUSAGE from the SUSUDTLIB database, from TYPE, or from both.

TO**FROM**

Recipient of the REVOKE statement action, which can be a set of users, ALL users, a role, or PUBLIC.

You can specify a maximum of 25 names per REVOKE request.

For compatibility, ANSI/ISO SQL requires you to specify FROM rather than TO.

ALL user_name

User from whom explicit privileges are revoked. You can specify up to 25 user names.

ALL user_name specifies that the privileges are to be granted to or revoked from the specified user, and every user owned by that user now and in the future.

If you do not specify *ALL user_name*, then the revocation does not cascade through the hierarchy.

ALL *user_name* is a Teradata extension to ANSI/ISO SQL.

PUBLIC

The explicit privileges are to be revoked from all currently defined Vantage users and are not to be granted to future users.

ALL DBC is equivalent to PUBLIC.

database_name

user_name

Containing database or user for *role_name* if something other than the current database or user.

role_name

Name of a role from which privileges are revoked.

You can specify up to 25 role names.

Usage Notes

Rules for *privileges* Keywords

The following rules apply to using the privileges for the REVOKE (SQL Form) statement:

- You can specify any combination of privileges appropriate for the corresponding database objects. However, the user submitting the statement must have those privileges, either implicitly or explicitly and WITH GRANT OPTION, on all of the specified objects.
- The CHECKPOINT privilege applies to performing both the SQL statement and the Host Utilities (HUT) Archive/Recovery commands DUMP and RESTORE.

The DUMP and RESTORE privileges refer to the corresponding HUT command performed on the specified object.

RESTORE also refers to execution of the following HUT commands:

- ROLLBACK
- ROLLFORWARD
- DELETE JOURNAL

If you specify CHECKPOINT, then the system revokes the privilege for the SQL statement and for the Archive/Recovery utility commands DUMP and RESTORE.

The DUMP and RESTORE privileges refer individually to the corresponding HUT commands performed on the specified object.

- CREATE DATABASE, FUNCTION, MACRO, PROCEDURE, TABLE, VIEW, or USER, and DROP DATABASE or USER are allowed only on databases or users.
- DROP TABLE includes ALTER TABLE.
- DROP MACRO, DROP PROCEDURE, or DROP VIEW include REPLACE MACRO, REPLACE PROCEDURE, or REPLACE VIEW, respectively.
- Only DROP and EXECUTE are allowed on specified procedures, UDFs, function mappings, or macros.

If the object is a UDF or procedure, then you must precede its name with FUNCTION or PROCEDURE, as appropriate.

If you do not specify one of those keywords, then the system assumes that the specified object name references a macro. If a macro with that name does not exist, the statement returns an error.

- Only ALTER FUNCTION, CREATE FUNCTION, DROP FUNCTION, EXECUTE FUNCTION, and FUNCTION are allowed on external UDF.
- ALTER FUNCTION is not a valid privilege for SQL UDFs.
- Only ALTER PROCEDURE, DROP PROCEDURE, and EXECUTE PROCEDURE are allowed on SQL procedures.
- Only ALTER EXTERNAL PROCEDURE, DROP PROCEDURE, and EXECUTE PROCEDURE are allowed on an external procedure.
- DUMP, RESTORE, and CHECKPOINT are not allowed on views.
- If you revoke RESTORE, then the privilege to perform the following utility commands is also revoked:
 - ROLLBACK
 - ROLLFORWARD
 - DELETE JOURNAL.
- DATABASE, FUNCTION, JOIN INDEX, MACRO, PROCEDURE, PROFILE, ROLE, TABLE, VIEW, and USER confer both CREATE and DROP privileges on the respective database objects.

If the DATABASE, FUNCTION, GLOP, JOIN INDEX, MACRO, PROCEDURE, PROFILE, ROLE, TABLE, USER, VIEW, or ZONE keyword is specified without CREATE or DROP, then CREATE and DROP are revoked.

- ANSI/ISO SQL:2011 supports the following privileges only:
 - DELETE
 - EXECUTE
 - INSERT
 - REFERENCES
 - SELECT
 - TRIGGER
 - UPDATE

The other privileges are extensions to the ANSI/ISO SQL:2011 standard.

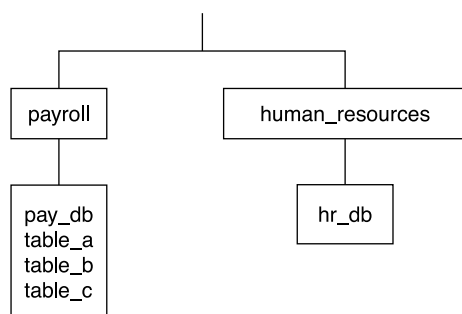
- In most cases, the privilege keyword agrees with the keyword of a Teradata SQL statement. However, the following operations privileges do not correspond to any SQL statements:
 - DATABASE
 - DUMP
 - MACRO
 - PROCEDURE
 - RESTORE
 - TABLE
 - USER
 - VIEW
- INSERT, REFERENCES, SELECT, and UPDATE have both table- and column-level options.
- You can revoke CREATE PROFILE, DROP PROFILE, CREATE ROLE, DROP ROLE, CREATE ZONE, and DROP ZONE privileges from users only, not from roles or databases.
- CREATE MAP, DROP MAP, CREATE PROFILE, DROP PROFILE, CREATE ROLE, DROP ROLE, CREATE ZONE, and DROP ZONE are system privileges. You can revoke the privileges from a user, but not on a specific object.
- The STATISTICS privilege enables collecting and dropping statistics.

Privileges Level for a Revoke

The rows in *DBC.AccessRights* correlate privileges to the level in the system space hierarchy at which they were created. As a result, it is possible to submit a valid REVOKE statement that does not revoke the privileges you specify because the revocation was not requested at the correct hierarchical level.

For example, if privileges on an object were granted at the database or user level, and you later submit a REVOKE request to rescind those privileges on a particular table in that database or user space, the request completes without returning a message, but no rows are deleted from *DBC.AccessRights* because there are no rows correlating those privileges with that table for the specified database or user. Instead, the rows in *DBC.AccessRights* correlate the privileges with the containing user or database.

Consider the following space hierarchy:



Suppose user payroll logs onto the system and grants the SELECT privilege on *pay_db* to user *human_resources* and all its descendants.

The BTEQ LOGON command and GRANT request look like this:

```
.LOGON tdpid/payroll,password
GRANT SELECT
ON pay_db
TO ALL human_resources;

Grant accepted.
```

User *payroll* later decides to revoke the SELECT privilege on *table_c* in database *pay_db* from *human_resources* and its descendants. To revoke this privilege, user payroll submits the following REVOKE request:

```
REVOKE SELECT
ON table_c
FROM ALL human_resources;

Revoke accepted.
```

Payroll now believes that the SELECT privilege on *table_c* has been revoked for *human_resources* and its descendants, but it has not been revoked because there were never any rows in *DBC.AccessRights* granting that privilege to *human_resources* on *table_c*. The row granting the SELECT privilege on the entire *pay_db* database to *human_resources* remains in *DBC.AccessRights*, so the originally granted SELECT privilege remains in effect.

Rules for Revoking Privileges

- Implicit privileges are governed by ownership and cannot be revoked. You can affect implicit privileges by using the GIVE statement to change ownership.
For more information, see [GIVE](#).
- Any combination of privileges can be revoked by a user who has those privileges, either implicitly or explicitly, WITH GRANT OPTION.
- ZONE (includes both the CREATE ZONE and DROP ZONE privileges) cannot be combined with any other privilege when you use REVOKE. Similarly, the ZONE OVERRIDE privilege cannot be combined with any other privilege.
- The system does not automatically revoke privileges previously granted by a user after that user is dropped from the system.
- Revoked privileges do not cascade through the hierarchy unless you specify the ALL *user_name* option.

Conversely, if a privilege that was granted to ALL users and databases is revoked from *user_name*, the privilege is not granted automatically to future users and databases that are owned by *user_name*.

- If the object is a view, procedure, or macro, the requesting user also must have WITH GRANT OPTION and all other applicable privileges on the objects referenced by that view, procedure, or macro.
- If a REVOKE statement removes explicit privileges that were granted at the database or user level, the privileges are revoked for all objects, regardless of when they were created.

A REVOKE statement at the object level cannot remove a privilege from that object that was granted at the database or user level.

See [Privileges Level for a Revoke](#).

- If a user receives the same privilege from one or more grantors, any user who has the necessary privileges can revoke that privilege from the user and from other grantees. A person who revokes a privilege from another does not have to be the grantor of that privilege.
- If a privilege was granted to PUBLIC, the privilege can only be revoked from PUBLIC, not from individual users.
- Revocation of a column-level privilege is only allowed if there is a row in DBC.AccessRights for the columns on which the privilege is to be revoked. If the user has INSERT, REFERENCES, SELECT, or UPDATE privileges at the table level, revoking those privileges on individual columns is not allowed.

Revoking Privileges on Global Temporary and Volatile Tables

REVOKE always applies to the base global temporary table and never to a materialized instance. Just as with permanent tables, a user must have the appropriate prerequisite privileges before submitting a REVOKE statement.

Because no privilege checking is done for volatile tables, you cannot REVOKE any privileges on them.

GRANT/REVOKE Order and Duration of Privileges

If a GRANT ALL ON object TO PUBLIC statement is issued by any user on an object that is lower in the hierarchy than user *DBC*, all other users inherit privileges on that object, including users created after the GRANT request is issued.

If user *DBC* then issues a REVOKE ALL ON object_name FROM DBC, users created after the REVOKE request was issued are not granted privileges on that object. However, all previously created users retain the privileges until a REVOKE ALL ON object_name FROM PUBLIC is issued.

For example:

```
.LOGON dbc.password

CREATE USER sys_admin AS
```

```
PERM=900000 PASSWORD=sys_admin;
```

```
GRANT ALL
ON sys_admin
TO sys_admin
WITH GRANT OPTION;
```

```
.LOGON sys_admin,sys_admin
```

```
CREATE USER dept AS
PERM=500000 PASSWORD=dept;
```

```
GRANT ALL
ON dept
TO dept
WITH GRANT OPTION;
```

```
.LOGON DEPT,DEPT
```

```
CREATE USER user_1 AS
PERM=100000 PASSWORD=user_1;
```

```
.LOGON USER1,USER1
```

```
CREATE TABLE table_1 (
  column_1 INTEGER,
  column_2 INTEGER)
PRIMARY INDEX(column_1);
```

```
INSERT table_1(1,2);
```

```
INSERT table_1(3,4);
```

```
GRANT ALL ON table_1
TO ALL DBC;
```

```
.LOGON dept,dept
CREATE USER user_2 AS
PERM=100000 PASSWORD=user_2;
```

```
.LOGON user_2,user_2
SELECT *
FROM user_1.table_1;
```

The rows of *table_1* are returned as expected.

```
.LOGON dbc,password

REVOKE ALL ON user_1.table_1
FROM dbc;

.LOGON dept,dept

CREATE USER user_3 AS
PERM=100000 PASSWORD=user_3;

.LOGON user_3,user_3

SELECT *
FROM user_1.table_1;
```

An error is returned because *user_3* was created after the REVOKE was issued.

```
.LOGON user_2,user_2
SELECT *
FROM user_1.table_1;
```

The contents of *table_1* are returned as expected.

Revoking Privileges on a Function Mapping

This statement revokes the EXECUTE FUNCTION privilege on the Interpolator function mapping in the *appl_view_db* database from user1.

```
REVOKE EXECUTE FUNCTION ON appl_view_db.Interpolator FROM user1;
```

Revoking Privileges on Procedures

The following rules apply to privileges specific to procedures:

- CREATE PROCEDURE is only a database or user level privilege.
- ALTER PROCEDURE, DROP PROCEDURE, and EXECUTE PROCEDURE apply to databases, users, or specified procedures.
- DROP and EXECUTE can be used as abbreviations for DROP PROCEDURE and EXECUTE PROCEDURE while rescinding privileges if you specify PROCEDURE *procedure_name*.
- The following happens if PROCEDURE is not specified before *object_name*:

Request	Description
REVOKE EXECUTE	Object is assumed to be a macro. If a macro with that name does not exist, an error is returned.
REVOKE DROP	Error is returned.

Examples of Revoking Privileges On SQL Procedures

The EXECUTE PROCEDURE privilege of *user_2* on the SQL procedure named *procedure_name* in *database_name* must be revoked. Assume that all the specified database objects exist and that the grantor owns both the database and the SQL procedure.

You can submit any of the following REVOKE requests to revoke the specified privileges:

```
REVOKE EXECUTE
ON PROCEDURE database_name.procedure_name
FROM user_2;
REVOKE EXECUTE PROCEDURE
ON database_name.procedure_name
FROM user_2;
REVOKE EXECUTE PROCEDURE
ON PROCEDURE database_name.procedure_name
FROM user_2;
```

Submit the following request to revoke CREATE PROCEDURE, DROP PROCEDURE, and EXECUTE PROCEDURE privileges simultaneously from *user_2* on the database named *database_name*:

```
REVOKE CREATE PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE
ON database_name
FROM user_2;
```

To revoke ALTER PROCEDURE, EXECUTE, and DROP privileges simultaneously from *user_2* on the SQL procedure named *procedure_name* in database *database_name*, you can perform either of the following requests:

```
REVOKE ALTER PROCEDURE, EXECUTE, DROP
ON PROCEDURE database_name.procedure_name
FROM user2;
REVOKE ALL
ON PROCEDURE database_name.procedure_name
FROM user2;
```

If you specify `PROCEDURE` without also specifying either `CREATE` or `DROP` in a `REVOKE` request, Vantage drops both the `CREATE PROCEDURE` and `DROP PROCEDURE` privileges on *database_name*.

For example, the following request drops both `CREATE` and `DROP PROCEDURE` privileges on *database_name* from *user_2*:

```
REVOKE PROCEDURE
ON database_name
FROM user_2;
```

Revoking UDT-Related Privileges

The following examples provide a representative sample of how to revoke UDT-related privileges:

```
REVOKE UDTUSAGE
ON SYSUDTLIB
FROM tester1;

REVOKE UDTMETHOD
ON SYSUDTLIB
FROM User_DBA;

REVOKE UDTMETHOD
ON SYSUDTLIB
FROM Developer_Role;
```

Related Information

See the following topics for information about UDT-related privileges:

- [UDT-Related Privileges](#)
- [ALL PRIVILEGES and UDTs](#)
- [UDTUSAGE Privilege](#)
- [UDTTYPE Privilege](#)
- [UDTMETHOD Privilege](#)

Revoking Privileges From Roles

Revoking a privilege from a role does not necessarily revoke the privilege from all of its role members. A member who has been granted the same privilege on an individual basis or through another role retains that privilege until it is explicitly revoked.

The following request revokes the privilege from the *finance* role to change the *personnel* database:

```
REVOKE UPDATE, DELETE, INSERT
ON personnel
FROM finance;
```

Revoking the CTCONTROL Privilege

If you specify the GRANT OPTION FOR option with a REVOKE request on the CTCONTROL privilege, the specification removes only the ability of the recipient to grant the privilege to others. The CTCONTROL privilege itself is not revoked when you specify GRANT OPTION FOR. Instead, you must submit a REVOKE request on the user that does not specify GRANT OPTION FOR.

The following example revokes from user *kate* the privilege to grant the CONNECT THROUGH privilege to any Vantage user:

```
REVOKE CTCONTROL FROM kate;
```

Note that this request revokes the CTCONTROL privilege from *kate*, not just her ability to grant CTCONTROL to other users, which would be the outcome of the following similar appearing request:

```
REVOKE GRANT OPTION FOR CTCONTROL FROM kate;
```

Examples of Using Revoke (SQL Form)

Revoking the INSERT Privilege

This request revokes the INSERT privilege from *UserA* on the *employee* table:

```
REVOKE INSERT
ON personnel.employee
FROM UserA;
```

Revoking Access That Requires Database Privileges

The following request revokes from *UserA* any access to any object that requires database, but not table, privileges in the *personnel* database:

```
REVOKE ALL PRIVILEGES
ON personnel
FROM UserA;
```

Using ALL BUT When Revoking Privileges

The following request leaves *UserA* with table privileges and at least the SELECT privilege on the Department table:

```
REVOKE ALL BUT SELECT
ON Personnel.Department
FROM UserA;
```

The statement does not revoke database-level privileges or table-level privileges that *UserA* might have had on *personnel*; therefore, *UserA* might still have the ability to perform other requests such as an INSERT into *personnel.department*.

The following request leaves *UserA* with the SELECT privilege and all table-level privileges on every object in the *personnel* database:

```
REVOKE ALL BUT SELECT
ON personnel
FROM UserA;
```

Related Information

Topic	Reference Source
granting privileges	GRANT (SQL Form) .
a list of the supported privileges	Privilege Dictionary .
a complete list of privileges and their abbreviations as maintained in DBC.AccessRights	<i>Teradata Vantage™ - Advanced SQL Engine Security Administration</i> , B035-1100 Note: You cannot use these abbreviations in a REVOKE (SQL Form) request. Instead, you must specify the complete privilege name.

REVOKE CONNECT THROUGH

Revokes an existing proxy CONNECT THROUGH privilege from a permanent user or application user.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must have the CTCONTROL privilege (see [CTCONTROL Privilege](#)) to perform a REVOKE CONNECT THROUGH request.

If the request specifies a WITH ROLE clause, you must also have the WITH ADMIN OPTION privilege on each of the roles specified in the clause.

REVOKE CONNECT THROUGH Syntax

```

REVOKE CONNECT THROUGH trusted_user_name {

  { { TO | FROM }

    { application_user_name [,...]
      [ WITH ROLE role_name [,...] ]
      [ WITH PROFILE profile_name ] |

      PERMANENT permanent_user_name [,...]
      [ WITH ROLE role_name [,...] ]
    }
  } |

  WITH TRUST ONLY
} [;]

```

Syntax Elements

trusted_user_name

The name of the trusted user whose CONNECT THROUGH privilege is being revoked.

WITH TRUST_ONLY

Removes the TrustOnly flag from the *trusted_user_name*.

This option is valid only when submitted for a trusted user.

The TrustOnly option requires that a trusted user must submit SET QUERY_BAND requests that set or update a proxy user from a trusted request.

application_user_name

The name of an application user from whom the proxy logon privileges granted through *trusted_user_name* are being revoked.

If you do not specify a WITH ROLE clause, the request revokes the connect privilege for the specified application user.

You can specify a maximum of 25 names in a single revoke request.

permanent_user_name

The name of a permanent user from whom the proxy logon privileges granted through *trusted_user_name* are being revoked.

If you do not specify a WITH ROLE clause, the request revokes the connect privilege for each permanent user from the trusted user.

You can specify a maximum of 25 names in a single revoke request.

role_name

A list of role names to be removed from the CONNECT THROUGH privilege granted to *trusted_user_name*.

If you remove all roles that have been granted the CONNECT THROUGH privilege for the specified permanent or application user, then the system revokes the entire privilege for the specified permanent or application user.

Similarly, if you do not specify a WITH ROLE clause, then the system revokes the entire privilege for the specified permanent or application user.

profile_name

Removes the profile from the *application_user_name* proxy user for *trusted_user_name* but leaves the rule granted. Removing the profile or revoking the entire rule removes the associated rows for the proxy user from the DBC.databasespace table.

Usage Notes for REVOKE CONNECT THROUGH

Role Persistence Through Active Proxy Connections

Changes to a CONNECT THROUGH privilege definition are effective immediately, so the next request submitted in a proxy connection following a change to a CONNECT THROUGH privilege uses the new definition.

When roles are dropped, they are also removed from the CONNECT THROUGH privilege definition, so it is possible for a rule to be left with no roles defined for it. When you drop all of the roles that were assigned to a CONNECT THROUGH privilege definition, Vantage grants the affected proxy users PUBLIC privileges only.

CONNECT THROUGH and Parameter Markers

Parameter markers are not supported for REVOKE CONNECT requests.

Dictionary Storage of CONNECT THROUGH Metadata

The dictionary table `DBC.ConnectRulesTbl` contains information on which proxy users can connect through which trusted users and what roles are available to make proxy connections. The table contains one row for every *trusted_user_name:proxy_user_name* combination.

When the system processes a GRANT CONNECT THROUGH request, it writes a row to `DBC.ConnectRulesTbl` for each of the following pairs that you specify:

- Trusted user name:permanent user name
- Trusted user name:application user name

The row persists until either you drop the trusted user or the permanent user.

When you grant WITH TRUST_ONLY to a trusted user, *Vantage* adds a row to `DBC.ConnectRulesTbl` that contains the following information:

- TrustUserId
- ProxyUser=*space_characters*
- TrustOnly=Y

Vantage also updates all rows in `DBC.ConnectRulesTbl` with the value for TrustUserId=*specified_TrustUserID* to set TrustOnly=Y.

When you revoke WITH TRUST_ONLY from a trusted user, *Vantage* updates all rows in `DBC.ConnectRulesTbl` where TrustUserId=*specified_TrustUserID* to set TrustOnly=N.

To provide an audit trail for the management of the rules, *Vantage* retains the row if the privilege is revoked, but does *not* drop the user.

The following list indicates the values for `DBC.ConnectRulesTbl.GrantStatus` when the TRUST_ONLY privilege is granted to a trusted user or not:

- If `DBC.ConnectRulesTbl.GrantStatus` is set to G, then the TRUST_ONLY privilege is granted to *trusted_user*.
- If `DBC.ConnectRulesTbl.GrantStatus` is set to R, then TRUST_ONLY privilege is revoked from *trusted_user*.

Example of Revoking the CONNECT THROUGH Privilege

The following REVOKE CONNECT request revokes the CONNECT THROUGH privilege that had been granted to permanent user *sbd* with the role *admin* through trusted user *pls*:

```
REVOKE CONNECT THROUGH pls FROM PERMANENT sbd WITH ROLE admin;
```

Related Information

For more information on ...	See ...
Granting proxy connections	GRANT CONNECT THROUGH
Setting query bands to enable proxy connections	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
Administering trusted sessions	<i>Teradata Vantage™ - Database Administration</i> , B035-1093
Security issues related to trusted sessions	<i>Teradata Vantage™ - Advanced SQL Engine Security Administration</i> , B035-1100
The dictionary attributes of trusted sessions	<i>Teradata Vantage™ - Data Dictionary</i> , B035-1092

REVOKE LOGON

Does either of the following:

- Revokes permission to log on to the database from one or more specific client systems.
- Changes the current system logon defaults.

See [GRANT LOGON](#) for information about granting logon privileges.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must have the EXECUTE privilege on the *DBC.LogonRule* macro to perform REVOKE LOGON.

No checks are made on whether the database names or user names defined in the statement apply to users owned by the requesting user.

If the system cannot verify the submitted statement because it specifies a nonvalid *user name* or a nonvalid host ID, then an error is returned.

REVOKE LOGON Syntax

```
REVOKE LOGON ON
  { host_id [,...] | ALL }
  { AS DEFAULT |
    { TO | FROM } user_name [,...]
  } [;]
```

Syntax Elements

host_id

An integer that identifies a mainframe connection or a workstation connection that is currently defined to the system by the hardware configuration data. The interface need not be operational.

The value for the database console is 0.

For any other connector, the value for *host_id* ranges from 1 through 32,767.

ALL

Any source through which a logon is attempted, including the database console.

AS DEFAULT

Indicates that the current default for the specified host ID set is to be changed, without residual conditions, as defined in this REVOKE LOGON request.

A request with AS DEFAULT has no effect on the access revoked from or granted to particular user names. A request that sets the default for a specific host ID takes precedence over a request that sets the default for ALL client systems.

TO FROM

Keywords introduced to override the current default for the specified *database_name_list* or *user_name set* on the specified *host_ID* set.

user_name

One or more user names from whom logons are to be revoked.

The name *DBC* cannot be specified as a *user_name* in a REVOKE LOGON request. Any request that specifies the name *DBC* as a user aborts and returns an error message to the requestor.

The product of the number of host IDs and the number of database and user names cannot exceed 25.

Usage Notes

Default Logon Permissions

When Vantage is connected to multiple client systems, the initial default is that logon permission is granted to all users from all host IDs, and that all logons must include a password.

- The GRANT LOGON and REVOKE LOGON statements control which users have access from which client system connections.
- A REVOKE LOGON statement inhibits only future logon attempts; it does not affect users who are currently logged on.

Effects of REVOKE LOGON Requests

When you submit a REVOKE LOGON request for one or more user names, Vantage creates a logon control record for each *user_name/host_ID* pair specified.

Any existing control record for a particular pair is replaced. The logon control record set created for a particular user name exists until that user is dropped.

Any attempt to revoke logon privileges for user *DBC* aborts the request and returns an error. If you attempt to log on to the system as user *DBC* and you submit the correct password, then the logon is accepted regardless of the current default for the applicable host ID. This prevents any opportunity to lock out all clients from user *DBC*.

AS DEFAULT Option

A request that includes the AS DEFAULT option has no effect on the logon access granted to or revoked from specific user names.

The following rules apply to the AS DEFAULT option:

Statement	Privileges Assumed
REVOKE LOGON	The user cannot access the applicable client even if that client has an assigned default of GRANT.
GRANT LOGON	The user can always access the applicable client even if that client has an assigned default of REVOKE.

Note in the following examples that these two requests do *not* have the same semantics:

```
REVOKE LOGON ON ALL AS DEFAULT;
```

```
REVOKE LOGON ON ALL FROM "DEFAULT";
```

The first request revokes logon privileges on any source through which a logon is attempted, including the Vantage console, without residual conditions. Note that the string AS DEFAULT in this request is a keyword phrase and does not identify a user.

In the second request, Vantage treats "DEFAULT" as a string literal that identifies a user, not as a keyword, and the request revokes logon privileges on all logon sources for the *user* named DEFAULT. Because

DEFAULT is a dummy user name and cannot log on to Vantage, the second request, though syntactically valid, essentially has no effect.

REVOKE MAP

Revoke existing contiguous or sparse maps from users and roles.

Sparse maps can only be revoked from users and roles within the same secure zone as the sparse map.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

You must have been granted the map WITH GRANT OPTION to revoke the map from other users, databases, roles, or PUBLIC.

REVOKE MAP Syntax

```
REVOKE [ GRANT OPTION FOR ] MAP map_name [...]
{ TO | FROM }
{ { user_name | role_name } [,...] | PUBLIC } [;]
```

Syntax Elements

GRANT OPTION FOR

Revoke the ability to grant the map to other users.

Specifying this option only revokes the ability to grant the map to other users.

map_name

Name of existing contiguous or sparse map.

You cannot specify TD_DataDictionaryMap or TD_GlobalMap.

user_name

Name of user for whom the map is to be revoked.

role_name

Name of role for which the map is to be revoked.

PUBLIC

Privileges are to be revoked for all existing and future Vantage users.

REVOKE ZONE

When using Teradata Secure Zones, revokes zone access from zone guests. Zone guests are users or roles that do not belong to the zone but have previously been granted zone access.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

Only the creator of the zone can revoke zone access from a zone guest.

REVOKE ZONE Syntax

```
REVOKE ZONE zone_name [...]
  { TO | FROM }
  { { user_name | role_name } [,...] | PUBLIC } [;]
```

Syntax Elements

zone_name

The name of the zone.

The zone must already exist.

user_name

role_name

The names of the zone guests that you want to revoke zone access from. You can specify up to 25 names in a comma-separated list.

Example of Revoking Zone Privileges

This example revokes the privileges from a zone guest named *guest_jim01*.

```
REVOKE ZONE widgets_inc_zone FROM guest_jim01 ;
```

Related Information

For information about granting zone access, see [GRANT ZONE](#).

For information about implementing Teradata Secure Zones, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

REVOKE ZONE OVERRIDE

When using Teradata Secure Zones, revokes the privilege to bypass zone restrictions when accessing rows in DBC tables.

ANSI Compliance

This statement is a Teradata extension to the ANSI SQL:2011 standard.

Required Privileges

Only user DBC can revoke the ZONE OVERRIDE privilege.

REVOKE ZONE OVERRIDE Syntax

```
REVOKE ZONE OVERRIDE { TO | FROM } user_name [,...] [;]
```

Syntax Elements

user_name

The names of the users whose access to DBC table data that pertain to all zones you want to revoke. You can specify up to 25 comma-separated names.

Example of Revoking the ZONE OVERRIDE Privilege

This example revokes the ZONE OVERRIDE privilege from a user named *Amy_F39674*.

```
REVOKE ZONE OVERRIDE FROM Amy_F39674 ;
```

Related Information

For information about granting the zone override privilege, see [GRANT ZONE OVERRIDE](#).

For information about Teradata Secure Zones and how it can be used, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

Notation Conventions

How to Read Syntax

This document uses the following syntax conventions.

Syntax Convention	Meaning
KEYWORD	Keyword. Spell exactly as shown. Many environments are case-insensitive. Syntax shows keywords in uppercase unless operating system restrictions require them to be lowercase or mixed-case.
<i>variable</i>	Variable. Replace with actual value.
<i>number</i>	String of one or more digits. Do not use commas in numbers with more than three digits. Example: 10045
[x]	x is optional.
[x y]	You can specify x, y, or nothing.
{ x y }	You must specify either x or y.
x [...]	You can repeat x, separating occurrences with spaces. Example: x x x See note after table.
x [, ...]	You can repeat x, separating occurrences with commas. Example: x, x, x See note after table.
x [<i>delimiter</i> ...]	You can repeat x, separating occurrences with specified delimiter. Examples: <ul style="list-style-type: none"> If <i>delimiter</i> is semicolon: x; x; x If <i>delimiter</i> is { , OR }, you can do either of the following: <ul style="list-style-type: none"> x, x, x x OR x OR x See note after table.

Note:

You can repeat only the immediately preceding item. For example, if the syntax is:

```
KEYWORD x [...]
```

You can repeat x. Do not repeat KEYWORD.

If there is no white space between x and the delimiter, the repeatable item is x and the delimiter. For example, if the syntax is:

```
[ x, [...] ] y
```

- You can omit x: y
- You can specify x once: x, y
- You can repeat x and the delimiter: x, x, x, y

Character Shorthand Notation Used in This Document

This document uses the Unicode naming convention for characters. For example, the lowercase character 'a' is more formally specified as either LATIN CAPITAL LETTER A or U+0041. The U+xxxx notation refers to a particular code point in the Unicode standard, where xxxx stands for the hexadecimal representation of the 16-bit value defined in the standard.

In parts of the document, it is convenient to use a symbol to represent a special character, or a particular class of characters. This is particularly true in discussion of the following Japanese character encodings:

- KanjiEBCDIC
- KanjiEUC
- KanjiShift-JIS

These encodings are further defined in *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

Character Symbols

The symbols, along with character sets with which they are used, are defined in the following table.

Symbol	Encoding	Meaning
a-z A-Z 0-9	Any	Any single byte Latin letter or digit.
<u>a-z</u> <u>A-Z</u> <u>0-9</u>	Any	Any fullwidth Latin letter or digit.

Symbol	Encoding	Meaning
<	KanjiEBCDIC	Shift Out [SO] (0x0E). Indicates transition from single to multibyte character in KanjiEBCDIC.
>	KanjiEBCDIC	Shift In [SI] (0x0F). Indicates transition from multibyte to single byte KanjiEBCDIC.
T	Any	Any multibyte character. The encoding depends on the current character set. For KanjiEUC, code set 3 characters are always preceded by <code>ss3</code> .
!	Any	Any single byte Hankaku Katakana character. In KanjiEUC, it must be preceded by <code>ss2</code> , forming an individual multibyte character.
<u>Δ</u>	Any	Represents the graphic pad character.
Δ	Any	Represents a single or multibyte pad character, depending on context.
ss 2	KanjiEUC	Represents the EUC code set 2 introducer (0x8E).
ss 3	KanjiEUC	Represents the EUC code set 3 introducer (0x8F).

For example, string “TEST”, where each letter is intended to be a fullwidth character, is written as **TEST**. Occasionally, when encoding is important, hexadecimal representation is used.

For example, the following mixed single byte/multibyte character data in KanjiEBCDIC character set:

LMN<TEST>QRS

is represented as:

D3 D4 D5 0E 42E3 42C5 42E2 42E3 0F D8 D9 E2

Pad Characters

The following table lists the pad characters for the various character data types.

Server Character Set	Pad Character Name	Pad Character Value
LATIN	SPACE	0x20
UNICODE	SPACE	U+0020
GRAPHIC	IDEOGRAPHIC SPACE	U+3000
KANJISJIS	ASCII SPACE	0x20
KANJI1	ASCII SPACE	0x20

Privilege Dictionary

Privileges

Below is a description of each of the columns in the privileges table that follows:

Column	Description
Privilege	Name of privilege.
Abbreviation in System Views	Two-letter code used in system views to represent each privilege granted on a particular object. The AccessRight column in the following views lists privileges for users and roles: <ul style="list-style-type: none"> • DBC.AllRightsV • DBC.UserGrantedRightsV • DBC.UserRightsV • DBC.AllRoleRightsV • DBC.UserRoleRightsV
Automatically Granted: Creators	When you create a user or database, the system automatically grants you privileges on the created database or user.
Automatically Granted: Created User or Database	When you create a user or database, it automatically gets certain privileges on itself.
Explicitly Granted: Creators	When you create a user, you do not automatically receive certain privileges on the user.
Explicitly Granted: Created User or Database	Although the system automatically grants many database privileges, some privileges are only available to users if you explicitly grant them. You must have the WITH GRANT OPTION privilege on these privileges before you can grant them. When you create a user, it does not automatically get certain privileges on itself.
Group	Functional area of privilege.

The table below lists information about each of the privileges.

Privilege	Abbreviation in System Views	Automatically Granted		Explicitly Granted		Privilege Category
		Creators	Created User or Database	Creators	Created User or Database	
ABORT SESSION or ABORTSESSION	AS	No	No	Yes	Yes	Monitor

Privilege	Abbreviation in System Views	Automatically Granted		Explicitly Granted		Privilege Category
		Creators	Created User or Database	Creators	Created User or Database	
ALTER EXTERNAL PROCEDURE	AE	No	No	Yes	Yes	
ALTER FUNCTION	AF	No	No	Yes	Yes	
ALTER PROCEDURE	AP	No	No	Yes	Yes	
ANY	AN	Yes	Yes	No	No	Indicates a HELP or SHOW statement for which at least one privilege, but no specific privilege, is required.
CHECKPOINT	CP	Yes	Yes	No	No	
CONNECT THROUGH		No	No	Yes	Yes	Trusted session- related
CONSTRAINT ASSIGNMENT	SA	No	No	Yes	Yes	System- level
CONSTRAINT DEFINITION	SD	No	No	Yes	Yes	System- level
CREATE AUTHORIZATION	CA	Yes	Yes	No	No	
CREATE DATABASE	CD	Yes	No	No	Yes	
CREATE DATASET SCHEMA	C1	No	No	Yes	Yes	Dataset Schema
CREATE EXTERNAL PROCEDURE	CE	No	No	Yes	Yes	
CREATE FUNCTION	CF	No	No	Yes	Yes	
CREATE GLOP	GC	No	No	Yes	Yes	Global and Persistent

Privilege	Abbreviation in System Views	Automatically Granted		Explicitly Granted		Privilege Category
		Creators	Created User or Database	Creators	Created User or Database	
						(GLOP) Data
CREATE MACRO	CM	Yes	Yes	No	No	
CREATE MAP	MC	No	No	Yes	Yes	System- level
CREATE OWNER PROCEDURE	OP	No	No	Yes	Yes	
CREATE PROCEDURE	PC	No	No	Yes	Yes	
CREATE PROFILE	CO	No	No	Yes	Yes	System- level
CREATE ROLE	CR	No	No	Yes	Yes	System- level
CREATE SERVER	CS	No	No	Yes	Yes	
CREATE TABLE	CT	Yes	Yes	No	No	
CREATE TRIGGER	CG	Yes	Yes	No	No	
CREATE USER	CU	Yes	No	No	Yes	
CREATE VIEW	CV	Yes	Yes	No	No	
CREATE ZONE	CZ	No	No	Yes	Yes	System- level
CTCONTROL	TH	No	No	Yes	Yes	Trusted session- related
DELETE	D	Yes	Yes	No	No	
DROP AUTHORIZATION	DA	Yes	Yes	No	No	
DROP DATABASE	DD	Yes	No	No	Yes	
DROP DATASET SCHEMA	D1	Yes	No	Yes	Yes	Dataset Schema
DROP FUNCTION	DF	Yes	Yes	No	No	
DROP GLOP	GD	No	No	Yes	Yes	Global and Persistent

Privilege	Abbreviation in System Views	Automatically Granted		Explicitly Granted		Privilege Category
		Creators	Created User or Database	Creators	Created User or Database	
						(GLOP) Data
DROP MACRO	DM	Yes	Yes	No	No	
DROP MAP	MD	No	No	Yes	Yes	System- level
DROP PROCEDURE	PD	Yes	Yes	No	No	
DROP PROFILE	DO	No	No	Yes	Yes	System- level
DROP ROLE	DR	No	No	Yes	Yes	System- level
DROP SERVER	DS	No	No	Yes	Yes	
DROP TABLE	DT	Yes	Yes	No	No	
DROP TRIGGER	DG	Yes	Yes	No	No	
DROP USER	DU	Yes	No	No	Yes	
DROP VIEW	DV	Yes	Yes	No	No	
DROP ZONE	DZ	No	No	Yes	Yes	System- level
DUMP	DP	Yes	Yes	No	No	
EXECUTE	E	Yes	Yes	No	No	
EXECUTE FUNCTION	EF	No	No	Yes	Yes	
EXECUTE PROCEDURE	PE	No	No	Yes	Yes	
GLOP MEMBER	GM	No	No	Yes	Yes	Global and Persistent (GLOP) Data
INDEX	IX	No	No	Yes	Yes	Table-level
INSERT	I	Yes	Yes	No	No	
MONITOR RESOURCE or MONRESOURCE	MR	No	No	Yes	Yes	Monitor

Privilege	Abbreviation in System Views	Automatically Granted		Explicitly Granted		Privilege Category
		Creators	Created User or Database	Creators	Created User or Database	
MONITOR SESSION or MONSESSION	MS	No	No	Yes	Yes	Monitor
NONTEMPORAL	NT	No	No	Yes	Yes	Temporal
OVERRIDE DELETE	OD	No	No	Yes	Yes	Security Constraint Override
OVERRIDE DUMP	OA	No	No	Yes	Yes	Security Constraint Override
OVERRIDE INSERT	OI	No	No	Yes	Yes	Security Constraint Override
OVERRIDE RESTORE	OR	No	No	Yes	Yes	Security Constraint Override
OVERRIDE SELECT	OS	No	No	Yes	Yes	Security Constraint Override
OVERRIDE UPDATE	OU	No	No	Yes	Yes	Security Constraint Override
REFERENCES	RF	No	No	Yes	Yes	Table-level
RESTORE	RS	Yes	Yes	No	No	
RETRIEVE /SELECT	R	Yes	Yes	No	No	
SET RESOURCE RATE or SETRESRATE	SR	No	No	Yes	Yes	Monitor
SET SESSION RATE or SETSESSRATE	SS	No	No	Yes	Yes	Monitor
SHOW	SH	No	No	Yes	Yes	
STATISTICS	ST	Yes	Yes	No	No	Also for creators of tables

Privilege	Abbreviation in System Views	Automatically Granted		Explicitly Granted		Privilege Category
		Creators	Created User or Database	Creators	Created User or Database	
UDT METHOD or UDTMETHOD	UM	No	No	Yes	Yes	UDT
UDT TYPE or UDTTYPE	UT	No	No	Yes	Yes	UDT
UDT USAGE or UDTUSAGE	UU	No	No	Yes	Yes	UDT
UPDATE	U	Yes	Yes	No	No	
WITH DATASET SCHEMA	W1	Yes	No	Yes	Yes	Dataset Schema
ZONE OVERRIDE	ZO	No	No	No	No	System- level

Multiple Privileges with a Single Keyword

Teradata Vantage provides a special category of keywords that you can use to grant multiple privileges. For example, the following request uses the keyword `DATABASE` to grant both the `CREATE DATABASE` and `DROP DATABASE` privileges on `user_name1` to `user_name2`:

```
GRANT DATABASE ON user_name1 TO user_name2;
```

The following keyword grants enter multiple privileges for the grantee in the `DBC.AccessRights` table:

Keyword	Included Privileges
ALL	All implicit and explicit object privileges that the grantor owns, and on which the grantor has WITH GRANT OPTION, on the object specified in the ON clause.
CHECKPOINT	The privilege to execute the: <ul style="list-style-type: none"> CHECKPOINT SQL statement. HUT CHECKPOINT command.
DATABASE	<ul style="list-style-type: none"> CREATE DATABASE DROP DATABASE
FUNCTION	<ul style="list-style-type: none"> CREATE FUNCTION DROP FUNCTION
GLOP	<ul style="list-style-type: none"> CREATE GLOP DROP GLOP

Keyword	Included Privileges
INDEX	<ul style="list-style-type: none"> • CREATE INDEX • DROP INDEX
MACRO	<ul style="list-style-type: none"> • CREATE MACRO • DROP MACRO
MAP	<ul style="list-style-type: none"> • CREATE MAP • DROP MAP
OVERRIDE	<ul style="list-style-type: none"> • OVERRIDE INSERT • OVERRIDE SELECT • OVERRIDE UPDATE • OVERRIDE DELETE • OVERRIDE DUMP • OVERRIDE RESTORE
PROCEDURE	<ul style="list-style-type: none"> • CREATE PROCEDURE • DROP PROCEDURE
PROFILE	<ul style="list-style-type: none"> • CREATE PROFILE • DROP PROFILE
RESTORE	<p>The privilege to execute the following HUT commands:</p> <ul style="list-style-type: none"> • DELETE JOURNAL • ROLLBACK • ROLLFORWARD
ROLE	<ul style="list-style-type: none"> • CREATE ROLE • DROP ROLE
SHOW	<p>Only the following variations of the HELP and SHOW commands:</p> <ul style="list-style-type: none"> • HELP <i>database_object</i> • SHOW <i>database_object</i>
TABLE	<ul style="list-style-type: none"> • CREATE TABLE • DROP TABLE
TRIGGER	<ul style="list-style-type: none"> • CREATE TRIGGER • DROP TRIGGER
USER	<ul style="list-style-type: none"> • CREATE USER • DROP USER <p>Note: If the target of a GRANT USER is a user, then USER confers CREATE and DROP privileges. If the target of a GRANT USER is a database, then USER confers the privilege to CREATE users within the database.</p>

Keyword	Included Privileges
VIEW	<ul style="list-style-type: none"> • CREATE VIEW • DROP VIEW
ZONE	<ul style="list-style-type: none"> • CREATE ZONE • DROP ZONE

Required DBC Privileges

DBC needs privileges, for example, DELETE, UPDATE, and CREATE, on certain Data Dictionary objects, to enable the system to maintain the Data Dictionary. The system setup process grants these privileges by default.

NOTICE

Never revoke DBC privileges. Re-granting DBC privileges requires assistance from the Teradata Support Center.

Default PUBLIC Privileges

By default, the system grants several default privileges to PUBLIC on specific Data Dictionary objects, and all valid database users automatically have PUBLIC privileges. You can grant and revoke PUBLIC privileges according to your site security policy, but you should keep a list of the default privileges so you can re-grant them to PUBLIC if necessary.

Until you make changes, you can query the system to view the default PUBLIC privileges:

```
SELECT DatabaseName, TableName, AccessRight FROM DBC.AllRights WHERE
UserName='PUBLIC' ORDER BY 2, 1;
```

Default PUBLIC Privileges Required by Client Applications

The following privilege...	Granted to PUBLIC on...	Affects the following tools and utilities...
SELECT	DBC.Databases	NetBackup™ Teradata Extension
SELECT	DBC.DBCInfo	Teradata System Emulation Tool (Teradata SET)
SELECT	DBC.TDStats	Teradata Viewpoint Stats Manager. See <i>Teradata® Unity™ Installation, Configuration, and Upgrade Guide for Customers</i> , B035-2523.
SELECT	DBC.Tables	NetBackup™ Teradata Extension, Teradata SET

The following privilege...	Granted to PUBLIC on...	Affects the following tools and utilities...
INSERT, SELECT, DELETE, UPDATE	SysAdmin.FastLog ^a	FastExport, FastLoad, MultiLoad
EXECUTE	SysAdmin.FastLogIns ^a	FastExport, FastLoad, MultiLoad
SELECT	SysAdmin.FastLogRestartV ^a	FastExport, FastLoad, MultiLoad
EXECUTE	SysAdmin.FastLogUpd ^a	FastExport, FastLoad, MultiLoad
SELECT	SysAdmin.FastLogV ^a	FastExport, FastLoad, MultiLoad
^a Denotes an internal table containing FastLoad information.		

Privileges Needed for Database Administration

The following topics describe the privileges required by administrators to manage common database objects and processes.

This is a privilege quick reference. For details, see the information about required privileges for each SQL statement in:

- *Teradata Vantage™ - SQL Data Control Language*, B035-1149
- *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144

Databases

To...	the user must have the...
create a database	CREATE DATABASE privilege on the immediate owner database.
modify or drop a database	DROP DATABASE privilege on the database. Note: A database must be empty before it can be dropped.

DATASET SCHEMA

Action	User Privilege Required
Create a DATASET schema	CREATE DATASET SCHEMA privilege on the containing database or user.
Drop a DATASET schema	DROP DATASET SCHEMA privilege on the database or user or on the individual schema-level. Note: To be dropped, the schema must also not be in use.

Action	User Privilege Required
WITH DATASET SCHEMA Provide users with permission to associate a created schema with a column of a table	WITH DATASET SCHEMA privilege on the database or user or on the individual schema-level. The creator of a schema is automatically granted this privilege with grant option on the created schema.

Geospatial Data Types

Action	Privilege Required
Grant the use of geospatial data types with their associated functions, metadata tables, and procedures	<ul style="list-style-type: none"> • GRANT ALL ON SYSSPATIAL TO SYSSPATIAL WITH GRANT OPTION; • GRANT SELECT ON DBC TO SYSSPATIAL WITH GRANT OPTION; • GRANT EXECUTE PROCEDURE ON SYSSPATIAL TO DBC WITH GRANT OPTION; • GRANT UDTMETHOD ON SYSUDTLIB TO SYSSPATIAL; • GRANT UDTTYPE ON SYSUDTLIB TO SYSSPATIAL; • GRANT CREATE FUNCTION ON SYSUDTLIB TO SYSSPATIAL;
Grant the use of the GeoSequence date type	<ul style="list-style-type: none"> • GRANT UDTUSAGE ON SYSUDTLIB TO <i>geouser</i>; • GRANT EXECUTE FUNCTION ON SYSSPATIAL TO <i>geouser</i>; • GRANT SELECT ON SYSPATIAL TO <i>geouser</i>; • GRANT EXECUTE PROCEDURE ON SYSPATIAL TO <i>geouser</i>;

GLOP

To...	the user must have the...
create a GLOP set	CREATE GLOP privilege on the containing database or user.
drop a GLOP set	DROP GLOP privilege on the database or user containing GLOP_set_name. Note: The DROP GLOP privilege is automatically granted to the creator and owner of a GLOP set.
allow a user/database to specify access to a GLOP set that is from a database other than the one that contains the external routine referencing the GLOP set	GLOP MEMBER privilege If the GLOP is in the same database as the external routine, access to the GLOP is automatically granted. The database where the external routine resides must have either CREATE GLOP, DROP GLOP or GLOP MEMBER privileges on the GLOP set

Hash Index and Join Index

To...	the user must have the...
create a hash or join index	CREATE TABLE privilege on the database in which the hash index or join index is created.
drop a hash index	DROP TABLE or INDEX privilege on the indexed base table or its containing database.
drop a join index	DROP TABLE or INDEX privilege on each of the covered tables or their containing databases.

Macros

To...	the user must have the...
create a macro	<p>CREATE MACRO privilege on the database or user in which the macro is to be created. The creator automatically gets the DROP MACRO and EXECUTE privileges WITH GRANT OPTION. The immediate owner of the macro:</p> <ul style="list-style-type: none"> • is the database in which it exists, not the user who created it • determines the macro access privileges, not the macro • must have all the appropriate privileges for executing the macro, including WITH GRANT OPTION
drop a macro	DROP MACRO privilege on the macro.
rename a macro	<p>DROP MACRO privilege on the macro.</p> <p>In addition, the user renaming a macro must have the privileges for all statements the macro performs.</p>
execute a macro	<p>EXECUTE privilege on the macro.</p> <p>In addition, the immediate owner of the macro (the database in which the macro resides) must have the necessary privileges on objects named in the request set for statements that are contained in the macro.</p>
replace a macro	<ul style="list-style-type: none"> • privileges for all statements the macro performs • privileges that depend on whether the macro being replaced already exists. <ul style="list-style-type: none"> ◦ If the macro already exists, the DROP MACRO privilege on the macro or its containing database or user. ◦ If the macro does not already exist, the CREATE MACRO privilege on the macro or its containing database or user <p>Note:</p> <p>Once a macro has been replaced, its immediate owner is the database in which it exists, not the user who replaced it. The immediately owning database must have all the appropriate privileges for executing the macro, including WITH GRANT OPTION.</p>
delete a macro	DROP MACRO privilege on the specified macro

Profiles

To...	the user must have the...
create a profile	CREATE PROFILE privilege (system level).
modify or drop a profile	DROP PROFILE privilege (system level).

Proxy Users and Trusted Users

To...	the user must have the...
execute a GRANT CONNECT THROUGH statement to: <ul style="list-style-type: none"> • Define a trusted user (application logon user) • Identified associated proxy users and roles • Specify the WITH TRUST ONLY option to prevent the use of SET QUERY BAND statements that can change proxy user role assignments 	CTCONTROL privilege (system level).
use the SET QUERY BAND statement to set a proxy user role	GRANT CONNECT THROUGH privilege that specifies the user, role, and trusted user application

Roles and External Roles

To...	the user must have...
create a role or external role	the CREATE ROLE privilege (system level).
drop a role or external role	one of the following privileges: <ul style="list-style-type: none"> • DROP ROLE privilege (system level) • role membership WITH ADMIN OPTION
grant privileges to a role, or grant role membership	The role granted WITH ADMIN OPTION. Note: The creator of a role is automatically granted the specified role WITH ADMIN OPTION.

Security Constraints

To...	the user must have the...
create, alter, or drop CONSTRAINT objects	CONSTRAINT DEFINITION privilege (system level).
<ul style="list-style-type: none"> • assign constraints to, and remove them from, users and profiles 	CONSTRAINT ASSIGNMENT privilege (system level).

To...	the user must have the...
<ul style="list-style-type: none"> define and remove security constraint table columns in tables, views and indexes grant and revoke security constraint OVERRIDES 	For managing security constraints for users, tables, views and indexes the user must also have the necessary CREATE/REPLACE/ MODIFY privileges on the object.
execute the SHOW CONSTRAINT statement	CONSTRAINT DEFINITION or CONSTRAINT ASSIGNMENT privilege.
bypass enforcement of security constraint UDFs	OVERRIDE INSERT, OVERRIDE SELECT, OVERRIDE UPDATE, OVERRIDE DELETE for the security constraint, on the object being accessed.

Statistics

To...	the user must have...
collect/drop statistics on a permanent or global temporary table, a hash index, or a join index	the STATISTICS privilege on the table, global temporary table, hash index or join index.
collect/drop statistics on a permanent row level security table	the STATISTICS and OVERRIDE SELECT privileges on the table.
collect/drop statistics on a volatile table or a materialized global temporary table	No privileges required
use HELP STATISTICS to display summary or detail information for collected statistics for the specified data table or hash or join index	<ul style="list-style-type: none"> Any privilege on the table_name, join_index_name, or hash_index_name Ownership of the table_name or join_index_name
use SHOW STATISTICS to display summary or detail information for collected statistics for the specified data table or hash or join index	if the <i>values</i> option is not specified: <ul style="list-style-type: none"> Any privilege on the table_name, join_index_name, or hash_index_name Ownership of the table_name or join_index_name if the <i>values</i> option is specified: <ul style="list-style-type: none"> SELECT on the object being reported

SQL and External Procedures

For information about privilege requirements and options of SQL and external procedures, see:

- Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144
- Teradata Vantage™ - SQL External Routine Programming*, B035-1147

Tables

To...	the user must have the...
create a table (including error table)	CREATE TABLE privilege on the database or user in which the table is created.
alter or drop a table	DROP TABLE on the table or the database that contains the table. Additional privileges are required for altering a table to add or change a PI, security constraint column, or UDT column.
drop or rename a table	DROP TABLE on the table or on the database containing the table
archive a table	DUMP privilege on the database or table or you must be the owner of the table. If the table has security constraint columns, you must also have OVERRIDE DUMP privilege.
copy a table	the following privileges: <ul style="list-style-type: none"> • CREATE MACRO • CREATE TABLE • CREATE VIEW • RESTORE Note: Copying of security constraint tables is not supported.
restore a table	CREATE TABLE privilege on the table being restored, or be an owner of the table. If the table has security constraint columns, you must also have OVERRIDE RESTORE privilege.

Triggers

Creating or replacing a trigger does not grant trigger-related privileges to either the creator or the immediate owner of that trigger.

Note:

You cannot grant privileges on a trigger, only on the database or table to which the trigger applies.

To...	the user must have the...
create a trigger	<ul style="list-style-type: none"> • CREATE TRIGGER on both of the following: <ul style="list-style-type: none"> ◦ The database in which the trigger is created ◦ Either the subject table or its containing database • SELECT on any column referenced in a WHEN clause or a triggered SQL statement subquery • INSERT, UPDATE, or DELETE on the triggered SQL statement target table (depending on the triggered action).

To...	the user must have the...
	<ul style="list-style-type: none"> privileges that would normally be required to execute the individual triggered SQL statements
replace a trigger	<ul style="list-style-type: none"> DROP TRIGGER on the subject table or the database The exception is when you use REPLACE TRIGGER when no target trigger exists and you instead create a new trigger, in which case you need CREATE TRIGGER privilege on both of the following: <ul style="list-style-type: none"> The database in which the trigger is created Either the subject table or its containing database SELECT on any column referenced in a WHEN clause or a triggered SQL statement subquery Depending on the triggered SQL statement, INSERT, UPDATE, or DELETE on the triggered SQL statement target table privileges that would normally be required to execute the individual triggered SQL statements
drop a trigger	DROP TRIGGER privilege on the subject table or the database containing the table
execute a trigger	<p>privileges required for executing triggering statements. In addition, the immediate owner of the trigger must have:</p> <ul style="list-style-type: none"> CREATE TRIGGER on the subject table or the database containing the table. SELECT on any column referenced in the WHEN clause of the CREATE TRIGGER statement, or any column in a triggered action statement that requires read access for the execution of the statement.

Users

To...	the user must have the...
create a user	CREATE USER privilege.
modify or drop a user	DROP USER privilege.

User-Defined Functions (UDFs)

To...	the user must have...
create a function that calls a UDF from the SYSUDTLIB database	<p>the CREATE FUNCTION privilege, on the containing database and on the SYSUDTLIB database.</p> <p>If you grant a user the CREATE FUNCTION privilege on a database, any function a user creates automatically has DROP FUNCTION, EXECUTE FUNCTION, and WITH GRANT OPTION on the function.</p>
<ul style="list-style-type: none"> change the execution mode of a function recompile or re-link a function 	<p>the ALTER FUNCTION privilege.</p> <p>A UDF running in protected mode runs in a process created to run as the user called "tdatuser." This user has no special operating system</p>

To...	the user must have...
	<p>privileges. It is up to the site to decide what resources “tdatuser” is to have access to by setting access privileges to system resources as required by the site.</p> <p>Note: If you specify EXECUTE NOT PROTECTED, and the function fails during execution, the database could restart.</p>
use the SHOW FUNCTION statement to display CREATE/REPLACE FUNCTION text and the source code	<p>at least one privilege on the function or the database containing the function to display the CREATE/REPLACE FUNCTION text. If the user has the DROP FUNCTION privilege, the SHOW FUNCTION statement also displays the C source.</p>
drop a UDF	the DROP FUNCTION on the function or the containing database or user.
execute a function	<p>the EXECUTE FUNCTION privilege on the database or specific function. To grant the user privileges to execute a specific user-defined functions, submit:</p> <pre>GRANT EXECUTE ON SPECIFIC FUNCTION <i>function_name</i> TO <i>username</i>;</pre> <p>To grant execute privileges on all UDFs in a database:</p> <pre>GRANT EXECUTE FUNCTION ON <i>dbname</i> TO <i>username</i>;</pre>
use the HELP FUNCTION to get: <ul style="list-style-type: none"> the function name a list of parameters the data types of the parameters whether the function is used to compress or decompress character or graphic data any comments associated with the parameters for SQL, scalar, aggregate, and table functions 	at least one privilege on the function or the database containing the function.
rename a function	the DROP FUNCTION privilege on the function or the containing database, and the CREATE FUNCTION privilege on the database containing the function.
replace an existing function	the DROP FUNCTION privilege on the function or on the database containing the function.

Note:

If you specify a UDT as an input parameter or the function result, the current user must have one of the following privileges:

- UDTUSAGE on the SYSUDTLIB database.
- UDTUSAGE on the specified UDT.

For additional information on creating UDFs, see *Teradata Vantage™ - SQL External Routine Programming*, B035-1147.

UDTs and UDMs

To...	the user must have the...
create, alter, or drop a UDT	UDTTYPE is only granted at the database level. This privilege includes all the abilities of UDTUSAGE plus the ability to create, alter, and drop UDTs. Users can also create or drop cast, ordering, or transform properties. Users with this privilege cannot, however, create new methods or drop and replace existing methods.
create, alter, or drop a UDM	UDTMETHOD. This privilege includes all the abilities of UDTTYPE plus ability to use, create, drop or alter any UDT and its methods without any restrictions.
use a UDT or UDM in a table or view, and execute all associated methods	UDTUSAGE. You can grant this privilege at both the database and object level. It is not an automatic privilege and a user must be granted this privilege or acquire it through a role. A user granted UDTUSAGE WITH GRANT OPTION can grant it (optionally also with the WITH GRANT OPTION) to others. UDTUSAGE allows users to execute all SQL statements that reference existing UDTs and their existing methods. It does not permit creation of new UDTs, altering the ordering, casting, or transform behavior of existing UDTs, or creating new methods.
create or alter a cast operation for a UDT	CREATE CAST and REPLACE CAST privileges.

Views

The system automatically grants the following privileges on a newly created view to its creator:

- DELETE
- DROP VIEW
- INSERT
- SELECT
- UPDATE

To...	the user must have the...
create a view	CREATE VIEW privilege.

To...	the user must have the...
drop or replace a view	DROP VIEW privilege.

Teradata Secure Zones

The DBC user or a user who already has the CREATE ZONE, DROP ZONE, and WITH GRANT OPTION privileges must explicitly grant these privileges to a zone creator.

To ...	the user must have the ...
create a zone and create a zone root	CREATE ZONE privilege
drop a zone	DROP ZONE privilege
add or drop a zone root	ALTER ZONE privilege
assign a primary DBA to a zone	CREATE USER AS DBA or MODIFY USER AS DBA
grant zone access to zone guests	GRANT ZONE
revoke the zone access of zone guests	REVOKE ZONE

Determining Privileges for a User

You can create an AllUserRights macro to list all the privileges a user has on a specific database. The macro gets information from the DBC.AllRightsV and DBC.AllRolesRightsV views, and spells out the two character privilege code it finds in the AccessRightDesc field of the views.

Sample Macro for Determining User Privileges

```
create macro database_name.AllUserRights (UserName char(128)) as (
locking row for access select
    UserName      (varchar(128))
  ,AccessType     (varchar(128))
  ,RoleName       (varchar(128))
  ,DatabaseName   (varchar(128))
  ,TableName      (varchar(128))
  ,ColumnName     (varchar(128))
  ,AccessRight
  ,case
    when accessright='AE' then 'ALTER EXTERNALPROCEDURE'
    when accessright='AF' then 'ALTER FUNCTION'
    when accessright='AP' then 'ALTER PROCEDURE'
    when accessright='AS' then 'ABORT SESSION'
```

```

when accessright='CA' then 'CREATE AUTHORIZATION'
when accessright='CD' then 'CREATE DATABASE'
when accessright='CE' then 'CREATE EXTERNAL PROCEDURE'
when accessright='CF' then 'CREATE FUNCTION'
when accessright='CG' then 'CREATE TRIGGER'
when accessright='CM' then 'CREATE MACRO'
when accessright='CO' then 'CREATE PROFILE'
when accessright='CP' then 'CHECKPOINT'
when accessright='CR' then 'CREATE ROLE'
when accessright='CS' then 'CREATE SERVER'
when accessright='CT' then 'CREATE TABLE'
when accessright='CU' then 'CREATE USER'
when accessright='CV' then 'CREATE VIEW'
when accessright='CZ' then 'CREATE ZONE'
when accessright='C1' then 'CREATE DATASET SCHEMA'
when accessright='D' then 'DELETE'
when accessright='DA' then 'DROP AUTHORIZATION'
when accessright='DD' then 'DROP DATABASE'
when accessright='DF' then 'DROP FUNCTION'
when accessright='DG' then 'DROP TRIGGER'
when accessright='DM' then 'DROP MACRO'
when accessright='DO' then 'DROP PROFILE'
when accessright='DP' then 'DUMP'
when accessright='DR' then 'DROP ROLE'
when accessright='DS' then 'DROP SERVER'
when accessright='DT' then 'DROP TABLE'
when accessright='DU' then 'DROP USER'
when accessright='DV' then 'DROP VIEW'
when accessright='DZ' then 'DROP ZONE'
when accessright='D1' then 'DROP DATASET SCHEMA'
when accessright='E' then 'EXECUTE'
when accessright='EF' then 'EXECUTE FUNCTION'
when accessright='GC' then 'CREATE GLOP'
when accessright='GD' then 'DROP GLOP'
when accessright='GM' then 'GLOP MEMBER'
when accessright='I' then 'INSERT'
when accessright='IX' then 'INDEX'
when accessright='MC' then 'CREATE MAP'
when accessright='MD' then 'DROP MAP'
when accessright='MR' then 'MONITOR RESOURCE'
when accessright='MS' then 'MONITOR SESSION'
when accessright='NT' then 'NONTEMPORAL'
when accessright='OD' then 'OVERRIDE DELETE POLICY'
when accessright='OI' then 'OVERRIDE INSERT POLICY'

```

```

        when accessright='OP' then 'CREATE OWNER PROCEDURE'
        when accessright='OS' then 'OVERRIDE SELECT POLICY'
        when accessright='OU' then 'OVERRIDE UPDATE POLICY'
        when accessright='PC' then 'CREATE PROCEDURE'
        when accessright='PD' then 'DROP PROCEDURE'
        when accessright='PE' then 'EXECUTE PROCEDURE'
        when accessright='R'  then 'RETRIEVE/SELECT'
        when accessright='RF' then 'REFERENCES'
        when accessright='RS' then 'RESTORE'
        when accessright='SA' then 'SECURITY CONSTRAINT ASSIGNMENT'
        when accessright='SD' then 'SECURITY CONSTRAINT DEFINITION'
        when accessright='ST' then 'STATISTICS'
        when accessright='SS' then 'SET SESSION RATE'
        when accessright='SR' then 'SET RESOURCE RATE'
        when accessright='TH' then 'CTCONTROL'
        when accessright='U'  then 'UPDATE'
        when accessright='UU' then 'UDT Usage'
        when accessright='UT' then 'UDT Type'
        when accessright='UM' then 'UDT Method'
        when accessright='W1' then 'WITH DATASET SCHEMA'
        when accessright='ZO' then 'ZONE OVERRIDE'
    else''
        end (varchar(26)) as AccessRightDesc
    ,GrantAuthority
    ,GrantorName (varchar(128))
    ,AllnessFlag
    ,CreatorName (varchar(128))
    ,CreateTimeStamp
from
(
select
    UserName
    ,'User' (varchar(128)) as AccessType
    ,'' (varchar(128)) as RoleName
    ,DatabaseName
    ,TableName
    ,ColumnName
    ,AccessRight
    ,GrantAuthority
    ,GrantorName
    ,AllnessFlag
    ,CreatorName
    ,CreateTimeStamp
from dbc.allrights

```

```

where UserName = :username
    and CreatorName not = :username
union all
select
    Grantee as UserName
    , 'Member' as UR
    , r.RoleName
    , DatabaseName
    , TableName
    , ColumnName
    , AccessRight
    , null (char(1)) as GrantAuthority
    , GrantorName
    , null (char(1)) as AllnessFlag
    , null (char(1)) as CreatorName
    , CreateTimeStamp
from dbc.allrolerights r
join dbc.rolemembers m
    on m.RoleName = r.RoleName
where UserName = :username
union all
select
    User as UserName
    , m.Grantee as UR
    , r.RoleName
    , DatabaseName
    , TableName
    , ColumnName
    , AccessRight
    , null (char(1)) as GrantAuthority
    , GrantorName
    , null (char(1)) as AllnessFlag
    , null (char(1)) as CreatorName
    , CreateTimeStamp
from dbc.allrolerights r
join dbc.rolemembers m
    on m.RoleName = r.RoleName
where m.grantee in (select rolename from dbc.rolemembers where
grantee = :username)
) AllRights
order by 4,5,6,7; );

```


where *database_name* is the name of a database in your system, for which the macro checks user privileges. For example, if you create the macro in the DBAdmin database, you identify the macro as DBAdmin.AllUserRights.

Note:

This macro returns all privileges granted to a user either directly or through a role. It does not return implicit (ownership) privileges.

Executing the Privilege Check Macro

After you create the macro, you can execute it to check access privileges for a particular user with the command:

```
execute database_name.AllUserRights ('username');
```

database_name

The name of the macro, and also identifies database for which the macro checks user privileges.

username

The name of a permanent database user for which the macro checks privileges. The user that executes the macro must have EXECUTE privileges on the database that contains the macro.

Additional Information

Teradata Links

Link	Description
https://docs.teradata.com/	Search Teradata Documentation, customize content to your needs, and download PDFs. Customers: Log in to access Orange Books.
https://support.teradata.com	One-stop source for Teradata community support, software downloads, and product information. Log in for customer access to: <ul style="list-style-type: none">• Community support• Software updates• Knowledge articles
https://www.teradata.com/University/Overview	Teradata education network
https://support.teradata.com/community	Link to Teradata community