

Teradata Vantage™ - Database Administration - 17.20

Release 17.20




June 2022

Copyright and Trademarks

Copyright © 2000 - 2023 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
 NOTICE	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.
 CAUTION	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
 WARNING	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 laws, all designs, specifications, statements, information, recommendations and content (collectively, "content") available from the Teradata Documentation website or contained in Teradata information products is presented "as is" and without any express or implied warranties, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement, which are hereby disclaimed. In no event shall Teradata corporation, its suppliers or partners be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of content, even if advised of the possibility of such damage.

The Content 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 Content 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 the Content at any time without notice.

The Content is subject to change without notice. Users are solely responsible for their application of the Content. The Content does not constitute the technical or other professional advice of Teradata, its suppliers or partners. Users should consult their own technical advisors before implementing any Content. Results may vary depending on factors not tested by Teradata.

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.

Confidential Information

Confidential Information means any and all confidential knowledge, data or information of Teradata, including, but not limited to, copyrights, patent rights, trade secret rights, trademark rights and all other intellectual property rights of any sort.

The Content available from the Teradata Documentation website or contained in Teradata information products may include Confidential Information and as such, the use of such Content is subject to the non-use and confidentiality obligations and protections of a non-disclosure agreement or other such agreements to protect Confidential Information that you have executed with Teradata.

Contents

Chapter 1: Introduction	10
Prerequisites	11
Changes and Additions	11
Chapter 2: Setting Up Your Administrative Environment: All DBAs	12
Logging in to the Operating System	12
User DBC	12
Setting Up the Database Administrator User	14
Creating a Spool Reserve Database	20
Using Viewpoint Alerts	22
Setting Up Teradata Viewpoint Alerts for Space Usage	23
Global Default Parameters	24
Chapter 3: Databases and Users in Teradata: All DBAs	25
Databases and Users	25
The System Users	28
Other System Databases and Users	29
As-A-Service System Users	30
Recommended Hierarchy	31
Teradata Secure Zones Overview	31
Chapter 4: Working with Databases: All DBAs	33
Database Creation	33
Best Practices for Database Creation	33
Dropping a Database or User	38
Transferring Ownership with GIVE	39
Chapter 5: Working with Tables and Views: Application DBAs	40
Choosing a Primary Index	40
Creating Tables	44
Copying a Table	55
Dropping a Table	56
Recreating a Table	57
Specifying Fallback Tables	59
Working with Views	59
Using BTEQ Scripts to Create Database Objects	64
Chapter 6: Working with Stored Procedures and User-defined Functions: Application DBAs	66
Creating Stored Procedures	66

Creating User-defined Functions	77
Chapter 7: Working with Users, Roles, and Profiles: Operational DBAs	82
Overview of Establishing Users	82
User Access Methods	82
Types of Users	82
Assessing Database User Needs	83
Best Practices for Creating Users	84
Creating User Accounts	85
Creating User Profiles	87
Working with Database Users	90
Creating Temporary Passwords for First Login	94
Using Roles to Manage User Privileges	94
Granting Privileges Directly To Users	101
Viewpoint Monitoring Privileges	104
Setting Up Teradata Viewpoint Users	106
Chapter 8: Loading and Exporting Data: Application DBAs	108
Teradata Parallel Transporter	108
Basic Teradata Query for Loading Tables	109
Reading Committed Data While Loading to the Same Table	111
FastLoad Utility	114
FastExport Utility	115
MultiLoad Utility	116
Maximum Number of Concurrent FastLoad, MultiLoad, and FastExport Jobs	117
Teradata Parallel Data Pump	118
Loading Geospatial Data	119
Loading Unicode Data with Unicode Pass Through	119
Interpreting LOAD Utility Status	123
Choosing the Best Utility for Your Purpose	123
Utility Job Performance Analysis and Capacity Planning	126
Chapter 9: Working with Sessions and Accounts: Operational DBAs	130
Session Modes and Transaction Processing	130
Transaction Processing in ANSI Session Mode	131
Transaction Processing in Teradata Session Mode	131
Setting the Session Mode	132
Checking for Logged On Sessions	133
Obtaining Session Information	133
Troubleshooting Problems with Sessions	134
Working with Accounts	135
Creating Accounts	136
Working with System Accounting Views	136
Logging Resource Usage Data with Account String Variables	139
Enabling Account String Expansion	139

Setting the Default Account String to Use ASE	139
ASE Substitution Variables	140
Using ASE With Client Utilities	143
Chapter 10: Managing Space: Operational DBAs	144
Types of Space	144
Global Space Accounting	145
Fixing Issues with Space Accounting	148
Identifying and Correcting System-level Space Problems	152
Using the System Health Portlet to Find System Space Issues	152
Identifying Space Problems Using Viewpoint Space Usage Portlet	153
Identifying Space Issues by Querying System Views	154
Transferring Ownership and Permanent Space	161
Reallocating Perm Space to Resolve Space Issues	162
Increasing the Spool Space Limit for a User	162
Finding and Fixing Skewed Tables by Querying the TableSizeV View	162
Example of Finding Skewed Tables by Querying the TableSizeV View	163
Defining Temporary Space Limits	164
Protecting Transactions by Reserving Cylinders for Perm Space	166
Creating a Macro for Space Usage Reporting	166
Viewing Space Usage Trends for Databases and Users	168
Other Ways to View Space Utilization	168
Managing Data Blocks	169
Managing Cylinders	170
Giving One User to Another	176
Chapter 11: Maintaining the Database: Operational DBAs	178
Database Maintenance Tasks	178
Managing Accumulated Log Data	178
Managing Vproc Status and Initializing AMP Disks with Vproc Manager	183
Maintaining Data Dictionary Tables	184
Housekeeping on an Ad-Hoc Basis	189
Chapter 12: Archiving, Restoring, and Recovering Data: Operational DBAs	192
Archive/Restore Utility Support	192
Chapter 13: Managing Database Resources: Operational DBAs	193
Managing I/O with Cylinder Read	193
Managing the Database Workload with Teradata Active System Management	193
Managing Sessions and Transactions with Query Banding	196
Tracking Object Use and Table Changes	202
Analyzing Trends with Account String Expansion Variables	207
Resource Usage Trend Analysis	207
Chapter 14: Managing Queries: Operational DBAs	210

Redrive Protection for Queries	210
Recommendations for Common Query Problems	214
Identifying Poorly Performing Queries	215
Controlling Poorly Performing Queries	216
Finding and Fixing Skewed Tables	217
Finding and Resolving Lock Contentions	218
Investigating Query Blocks and Delays	219
Chapter 15: Improving Query Performance Using COLLECT STATISTICS: Application DBAs . . .	221
Statistics Collection	221
Automated Statistics Management	221
When To Collect Statistics	222
Collecting Statistics	222
Stale Statistics	223
SUMMARY Statistics	223
Skipping Unneeded Statistics Recollection with the THRESHOLD Option	223
Sampling Statistics with the USING SAMPLE Option	224
Working with Special Cases	226
Chapter 16: Tracking Query Behavior with Database Query Logging: Operational DBAs	228
DBQL Overview	228
Rules Validation	232
SQL Statements That Should Be Captured	232
SQL Logging Statements	233
SQL Logging Considerations	233
SQL Logging by Workload Type	234
SQL Statements to Control Logging	235
Setting DBQL Logging Algorithm	245
DBQL Macros	245
DBQL Tables	245
Querying DBQL Tables	247
DBQL Views	247
Shredding the Lock Plan Information in the XML Lock Log Table, DBQLXMLLockTbl	248
Query Data Storage and Protection	267
Things to Consider When Logging DBQL Data	268
Logging Scenarios	272
Example of OBJECT Data for One Query	276
Example of STEP Data for One Query	277
Examining the Logged Data	277
Zone-level Query Logging	278
Maintaining the Logs	279
Archiving DBQL Data	280
Reviewing or Ending Current Rules	280
Granting DBQL Administrative Privileges to Other Users	283

Chapter 17: Analyzing Requests and Request Plans: Application DBAs	285
Using EXPLAIN to Analyze Request Plans	285
Capturing Request Plan Steps	286
Query Capture Database	288
Creating the Query Capture Database Tables	289
Dropping Query Capture Database Tables	290
Querying QCD Tables	291
Building Baseline Transaction Profiles	293
Chapter 18: Working with System Information and Global Defaults: Operational DBAs	295
Viewing the Software Release and Version	295
Reviewing or Changing System-Wide Parameters	295
International Character Set Settings and Defaults	296
Changing Time Zones, Currencies, and Formats for Date, Time, and Numbers	299
Client Configuration Overview	300
Communicating with Teradata Vantage	302
Mainframe Environment	304
TDP Functionality	308
Network Environment	309
The Teradata Gateway	310
Chapter 19: Troubleshooting: Operational DBAs	313
Tools for Troubleshooting and Administrating	313
Tools for Troubleshooting Client Connections	323
Troubleshooting Spool Space Problems	324
Adjusting for Low Available Free Memory	326
Solving Partitioning and RI Validation Errors	326
Incident Information Checklist	327
Monitoring Transaction Recovery	329
Resolving Orphan or Phantom Spool Issues	331
Repairing Data Corruption	331
Chapter 20: Using Maps to Position Table Data across AMPs: All DBAs	333
Map Overview	333
Scenarios for Using Maps to Move Table Data	334
Enabling Teradata MAPS Architecture	334
Initial Setup for Managing Maps	335
A Simple Process for Changing the Map for a Table	335
Viewpoint Portlet for Analyzing and Optimizing Map Placement	336
Introduction to the Advisor and Mover Tools	336
Enabling Step Logging To Aid in Map Analysis	336
Excluding Objects from Map Reassignments	337
Analyzing Maps	337
ActionsTbl	339

Reviewing Recommended Map Actions	339
Customizing the List of Recommended Map Actions	340
Renaming Maps after a System Expansion	343
Sessions Used for Reassigning Maps	343
Estimating the Time Needed to Reassign Maps	344
Decisions Required before Reassigning Maps	344
Reassigning the Map for a Table	345
Limiting the Resources for Map Reassignments	348
Stopping Map Reassignments	348
Managing Restarts	349
Other Cleanup Tasks	350
Adjusting Space Limits and Skew Settings	351
 Appendix A: Performance Management: All DBAs	 352
 Appendix B: Teradata System Limits	 420
 Appendix C: Handling Teradata Crashdumps: Operational DBAs	 437
 Appendix D: Additional Information	 464

Introduction

Using the Database Administration Content

Teradata Vantage™ - Database Administration covers the basics of database administration, giving you the essential tasks and the procedures for doing them. Sections in this document are either designed for all DBAs or split tasks into two major groupings indicated in the section title: those for application DBAs, who are concerned with the design and performance of queries, and administrative DBAs, who are responsible for the day-to-day operation and maintenance of the Vantage system.

Why Would I Use this Content?

Database administrators use this content to perform a variety of administrative tasks. For example:

- Configure databases, users, roles, and profiles
- Create tables and views
- Create stored procedures and user-defined functions
- Load and export data
- Use Database Query Logging (DBQL) to track query statement counts and response times, and make refinements to workload groups and scheduling, and so on
- Learn about administrative tools to troubleshoot problems

How Do I Use this Content?

If you are configuring a new Vantage system, start at the with [Setting Up Your Administrative Environment: All DBAs](#) and proceed through the rest of the content to set up your system. If Vantage is already configured, select a topic related to what you want to do.

How Do I Get Started?

- Start by [Setting Up Your Administrative Environment: All DBAs](#)
- Learn about [Databases and Users in Teradata: All DBAs](#) and then:
 - [Working with Databases: All DBAs](#)
 - [Working with Users, Roles, and Profiles: Operational DBAs](#)
- [Working with Tables and Views: Application DBAs](#)
- [Loading and Exporting Data: Application DBAs](#)

References to Other Relevant Content

These publications are directly related to the information in this document:

- *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144
- *Teradata Vantage™ - Database Utilities*, B035-1102
- *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100

- *Teradata Vantage™ - Data Dictionary*, B035-1092

It may be helpful to review these publications:

- *Teradata Vantage™ - Database Introduction*, B035-1091
- *Teradata Vantage™ - Database Design*, B035-1094
- *Teradata Vantage™ - SQL Fundamentals*, B035-1141

Prerequisites

You should have some knowledge of:

- Relational database concepts
- Teradata SQL
- Teradata Vantage configuration

It may be helpful to review these publications:

- *Teradata Vantage™ - Database Introduction*, B035-1091
- *Teradata Vantage™ - Database Design*, B035-1094
- *Teradata Vantage™ - SQL Fundamentals*, B035-1141

These publications are directly related to the information in this document and you may find it helpful to have them available for reference:

- *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144
- *Teradata Vantage™ - Database Utilities*, B035-1102
- *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100
- *Teradata Vantage™ - Data Dictionary*, B035-1092
- The suite of Teradata Tools and Utilities client documents

Changes and Additions

Date	Description
June 2022	<ul style="list-style-type: none"> • Minor edits. • Teradata Unity was discontinued as of version 17.05. Use Business Continuity Manager instead.
July 2021	Minor edits.
June 2020	The Database Query Log (DBQL) is enhanced for SQL Engine 17.00 with more complete and accurate logging. DBQL now uses Algorithm 3 by default, which includes collecting statistics on aborted and parallel steps, and results in more accurate resource usage statistics. Upgraded systems that were not previously using Algorithm 3 will not use Algorithm 3 until explicitly configured to do so. See DBQL Overview .

Setting Up Your Administrative Environment: All DBAs

This section describes user DBC and provides a brief overview of space management. The section also describes these initial administrative tasks:

- Logging in to the operating system
- Setting up the database administrator user
- Creating a spool reserve database
- Establishing Viewpoint alerts
- Changing the PasswordExpire parameter

Logging in to the Operating System

Database administrators who need access to system nodes should log on to the operating system as a user who is a member of the tdtrusted group or as the root user.

Root Logons

Root access is required to start the database and install new versions. Users can perform all other Teradata administrative functions, including switching to a new Teradata version once it is installed, as a member of the tdtrusted group.

tdtrusted Logons

The tdtrusted group is created automatically at installation of Vantage. tdtrusted access allows nearly all command-line monitoring and administration of Teradata. The advantages of using tdtrusted include greater security because root access is limited and greater accountability because administrative user actions are recorded under each account rather than as the root user.

Non-root accounts that administer Teradata must be members of the tdtrusted group on all Vantage nodes. Each customer site should determine how to implement this within site-specific account management policies. tdtrusted logons rely on standard UNIX account management facilities.

Related Information

For more information about the users and groups created automatically during Vantage installation, see "Working with OS-Level Security Options" in *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

User DBC

When you first install Analytics Database, it has only one user. This user is called DBC, and from it all other future databases and users in the system are created.

User DBC also initially owns all the space in the entire system. As you create new databases and users, any permanent space you grant them is subtracted from available permanent space in user DBC. Because all space ultimately comes from user DBC, user DBC owns all database objects.

User DBC contains data like a database, but unlike a database it has a password. The usable (PERM) disk space in DBC initially reflects the entire system hardware capacity, minus space for the following system users, databases, and objects:

- All the space available for your databases, users, journals, data tables, indexes, stored procedures, functions, and temporary tables. DBC owns all unallocated space.
- Crashdumps user
- SysAdmin user
- Sys_Calendar database
- TD_SYSFNLIB database
- SQLJ, SYSLIB, and SYSUDTLIB databases for external routines and user-defined types. For more information, see *Teradata Vantage™ - SQL External Routine Programming*, B035-1147.
- SYSSPATIAL database for geospatial data types. For more information, see *Teradata Vantage™ - Geospatial Data Types*, B035-1181.
- DBCExtension database for Global and Persistent (GLOP) sets. For more information, see “DBCExtension Tables” in *Teradata Vantage™ - Data Dictionary*, B035-1092.
- SystemFE user for field engineers
- TDPUSER user
- The system Transient Journal (TJ) which stores the before-change image of every data row involved in a transaction, the row ID of every inserted row, and other control records used by the system to automatically recover data from transaction aborts or to complete transactions.
- The system catalog tables of the Data Dictionary and the suite of user-accessible views defined on those tables (for more information, see *Teradata Vantage™ - Data Dictionary*, B035-1092).

This also includes a suite of special system query logging tables used to store query logging rules and data when you enable the DBQL feature. See [Tracking Query Behavior with Database Query Logging: Operational DBAs](#).

Note:

The setting of DefaultCharacterSet in the DBS Control utility determines the default character set for user DBC. For information on changing the character set for user DBC, see [Default Server Character Sets](#).

NOTICE

Never alter the privileges for user DBC. Changing DBC privileges may cause installation, upgrade, maintenance, or archive procedures to end abnormally and consequently require Teradata Support to correct the problem.

NOTICE

If you use MODIFY USER to change the default character set for user DBC, user-defined functions may no longer work for this user without being recompiled, you must rerun the DIPUDT and DIPDEM scripts, and site-defined functions that use SQL_TEXT must be recompiled.

Setting Up the Database Administrator User

Create the principal database administrator, user DBADMIN (or the name your site uses), and grant space and privileges to the user before proceeding with database implementation. User DBADMIN can then create databases, tables, users, and other objects in the space it owns.

Note:

For larger systems, you may find it useful to create additional administrative users to share database administration duties. See the sections beginning with “Administrative Users” in *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

Creating the Database Administrator Profile

Profiles determine certain system resources available to member users, and can also set password control parameters.

1. From an administrative client, log on as user DBC.
2. Create the profile for the database administrator. To do this, specify values for the following fields and controls:

Do not enter values at this time for fields not listed.

Field	Description
Profile Name	<p>Required. The name of the database administrator profile. Administrative procedures in this publication use the name AdminProfile.</p> <p>Note: User DBADMIN will be the only member of this profile. Because of space and ownership concerns, you must create a separate profile for other administrative users.</p>
Account	<p>Optional. An account string defines the following characteristics for profile member sessions in the database:</p>

Field	Description
	<ul style="list-style-type: none"> • Session priority. This classification is in effect only when TASM or TIWM classification processes do not match the request to a user-defined workload. • Account ID (to assign charges for system time and resources) • Date stamp • Time stamp <p>Recommendation: Define at least one account string per profile. If necessary, you can add an account string later using a MODIFY PROFILE statement.</p> <p>For more about accounts, see Setting Up Your Administrative Environment: All DBAs.</p>
Spool Space	<p>Optional. The spool space specification is only a <i>limit</i> to the amount of space available for intermediate query results or formatted answer sets to queries and volatile tables. The system borrows spool space for a user from unused permanent space anywhere in the system.</p> <p>Recommendation: Set this value equal to the amount of space in the Spool_ Reserve database.</p>
Temporary Space	<p>Optional. Required only when using global temporary tables and other features that require temporary space.</p> <p>Recommendation: Temporary space is not normally needed. Specify temporary space only when you develop a strategy for using the features that need temporary space. If necessary, you can add a temporary space later with a MODIFY USER statement.</p>
Password	<p>Optional. Specify values for password control attributes if profile members cannot use default values.</p>
Other password controls	<p>Optional.</p> <p>Recommendation: Set other password controls globally by updating DBC. SysSecDefaults if required by your site security policy. Be sure to review the default password control values to determine whether they conform to your site security requirements.</p> <p>For information on the default password control values and how to change them, see “Managing Database Passwords,” in <i>Teradata Vantage™ - Analytics Database Security Administration</i>, B035-1100.</p>

For example:

```
CREATE PROFILE "AdminProfile" AS
SPOOL=spool_space
TEMPORARY=NULL
ACCOUNT=(' $H-DBC-MANAGER', ' account__str2', ' account_str3')
DEFAULT DATABASE="All"
PASSWORD =
(EXPIRE=90,MINCHAR=NULL,MAXCHAR=NULL,MAXLOGONATTEMPTS=NULL,LOCKEDUSEREXPIRE=NULL
,
REUSE=NULL,DIGITS=NULL,RESTRICTWORDS=NULL,SPECCHAR=NULL);
```


Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
2	Setting up accounts	Working with Users, Roles, and Profiles: Operational DBAs.
	The default password control values and how to change them	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Creating the Database Administrator User

1. From an administrative client, log on as user DBC.
2. Create the database administrator as follows:

Field	Description
User Name	Required. The name of the database administrator user. All administrative procedures in this publication use the name DBADMIN for the principal administrator.
Owner	The owner of user DBADMIN, which is DBC.
Permanent Space	Required. The space in bytes that contains all objects that user DBADMIN creates or owns. Perm space can also be specified with a constant expression resulting in a numeric value in bytes. This allows you to specify perm space per AMP, which is useful when porting applications from a smaller test system to a production system. See the next example. Because user DBADMIN will create and own nearly all databases and tables, assign it the majority of space on the system. Recommendation: 60-65% of available system (DBC) space.
Password	Required. The temporary password for DBADMIN. Recommendation: Any simple password will work that follows the default system password controls and site policy. Each user is prompted to change the temporary password to a permanent, private password at first logon. Note: Be sure to review the default password control values to determine whether they conform to your site security requirements. For information on password control options, see <i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100.
Spool Space	Optional. The spool space specification is only a limit to the amount of space available for intermediate query results or formatted answer sets to queries and volatile tables. The system borrows spool space for a user from unused permanent space anywhere in the system. Spool space can also be specified with a constant expression resulting in a numeric value in bytes. This allows you to specify spool space per AMP, which is useful

Field	Description
	when porting applications from a smaller test system to a production system. See the example following the table. Recommendation: Spool space specification is not normally needed. Spool is already allocated in AdminProfile.
Default Database	Optional. The user or database containing the space in which Analytics Database stores or searches for new or target objects unless a different database is specified in the transaction SQL. Recommendation: Specify DBADMIN.
FALLBACK	Optional. Specification of FALLBACK automatically creates a duplicate of each table stored in the user space, in addition to the duplicates already created by disk mirroring, to which the system can revert in the event of a failure. Note: You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.
Profile	Required. The name of the profile in which the user has membership. A user can be a member of only one profile. Recommendation: Enter AdminProfile, the profile created for user DBADMIN in Creating the Database Administrator Profile .

For example:

```
CREATE USER "DBADMIN" FROM "DBC"
AS PERM = 2e5 * (hashamp() + 1)
PASSWORD = "temp"
SPOOL = spool_space
PROFILE = AdminProfile
STARTUP = ''
NO BEFORE JOURNAL
NO AFTER JOURNAL;
```

The *hashamp()* function is a database built-in function that returns the highest AMP number in the system configuration. Because AMP numbers start from zero, *(hashamp() + 1)* returns the total number of AMPs in the system. The example expression: *2e5 * (hashamp() + 1)* means allocate 200,000 bytes on each AMP for perm or spool space for the user or database.

Note:

After you create the administrative user DBADMIN, do not log on as user DBC to perform subsequent administrative activities except during activities that only user DBC can perform.

Keep the client application open for use in the [Granting Database Privileges to User DBADMIN](#).

Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resource for Further Information
2	User types and characteristics	"Creating Users and Granting Privileges," in <i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100
	Syntax and options for the CREATE USER statement	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
	Guidelines for managing of space	Managing Space: Operational DBAs
	The default password control values and how to change them	"Managing Database Passwords," in <i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Granting Database Privileges to User DBADMIN

1. Log on to the client tool of your choice as user DBC.
2. Grant object access privileges to system tables and other objects owned by user DBC. For example:

```
GRANT EXECUTE, SELECT ON "DBC" TO "DBADMIN" WITH GRANT OPTION;
```

3. Grant object-level database privileges to the database administrator on all objects subsequently created in DBADMIN space. To do that, specify values for the following fields:

Field	Description
Database Name	The name of the user or database that owns the objects on which the privileges are being granted, in this case DBADMIN .
Object Type	The object on which the privileges are given. Includes all databases and all objects created in the database(s).
To/From User	The name of the user that receives the privileges, in this case DBADMIN .
All	Specifies all rights for all objects created in DBADMIN space.
Grant	Specifies that the user is granted the WITH GRANT OPTION privilege, which means that the user can grant all included privileges to other users.

For example:

```
GRANT EXECUTE, SELECT, INSERT, UPDATE, DELETE, STATISTICS, DUMP, RESTORE,
CHECKPOINT, SHOW, EXECUTE PROCEDURE, ALTER PROCEDURE, EXECUTE FUNCTION,
ALTER FUNCTION, ALTER EXTERNAL PROCEDURE, CREATE OWNER PROCEDURE, CREATE
TABLE, CREATE VIEW, CREATE MACRO, CREATE DATABASE, CREATE USER, CREATE
TRIGGER, CREATE PROCEDURE, CREATE FUNCTION, CREATE EXTERNAL PROCEDURE,
```



```
CREATE AUTHORIZATION, DROP TABLE, DROP VIEW, DROP MACRO, DROP DATABASE, DROP
USER, DROP TRIGGER, DROP PROCEDURE, DROP FUNCTION, DROP AUTHORIZATION ON
"DBADMIN" TO "DBADMIN" WITH GRANT OPTION;
```

4. Grant object-level privileges on DBC tables and views to DBADMIN. For example:

```
GRANT EXECUTE, SELECT, STATISTICS, SHOW ON "DBC" TO "DBADMIN" WITH
GRANT OPTION;
```

5. Grant additional system-level privileges not included with the object-level privileges granted in the previous step:

```
GRANT MONRESOURCE, MONSESSION, ABORTSESSION, SETSESSRATE, SETRESRATE,
REPLCONTROL, CREATE PROFILE, CREATE ROLE, DROP PROFILE, DROP ROLE TO
"DBADMIN" WITH GRANT OPTION;
GRANT UDTTYPE, UDTMETHOD ON SYSUDTLIB TO DBADMIN WITH GRANT OPTION;
```

6. Grant additional system-level privileges not covered by the GRANT shown in the previous step. For example:

```
GRANT CTCONTROL ON "DBADMIN" TO "DBADMIN" WITH GRANT OPTION;
```

7. Grant privileges on Sys_Calendar, which contains tables and views for date-related system functions:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON "Sys_Calendar" TO "DBADMIN" WITH
GRANT OPTION;
```

Note:

You do not need to execute a GRANT LOGON statement for user DBADMIN or any other user created in the database. All users automatically have the right to log on to the database from any connected client, unless a revoke logon statement is issued.

8. Log off as user DBC and log back in as user DBADMIN.
9. When prompted, create a private password for user DBADMIN. This client is now configured for use by user DBADMIN. Conduct all future administrative tasks as user DBADMIN or the administrative username used at your site.

Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resource for Further Information
2	System tables and views	<i>Teradata Vantage™ - Data Dictionary</i> , B035-1092
3	User privileges	"Creating Users and Granting Privileges" in <i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100.

Step	Topic	Resource for Further Information
2 through 6	Syntax and options for GRANT (SQL form) and (Role form)	<i>Teradata Vantage™ - SQL Data Control Language</i> , B035-1149
10	Password format rules	"Managing Database Passwords," in <i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Creating a Spool Reserve Database

Teradata strongly recommends that you give sufficient space under user DBC to be used as spool. There are two ways to do this: set aside permanent space to remain unused or create an empty database to hold the space. The advantage of allocating space under DBC to be used for spool is that the space will also be available for other system uses. However, there is also a risk that you may forget to maintain this reserved space and inadvertently assign it to other users and databases for PERM space usage. This could then result in a physical out-of-spool condition if there is no spool space left when processing queries.

An alternative is to create a database under DBC and allocate permanent space to be unused to hold the reserved spool space. Assigning permanent space to DBC to be permanently unused is reserving this physical space for spool usage. The advantage of creating a database to hold spool space means that space will always be set aside and available for spool usage.

A specific example of why it is important to reserve space for DBC is the use of spool space by system tables. Spool limits are allocated for each user or profile and actual space must be available for spool usage when a query is being processed. Sometimes, however, system tables like Transient Journal are allowed to use more space than is available in DBC. When DBC tries to use more space than it has, the system prevents new logons and refuses any data definition language SQL statements. This happens because logons and DDL statements require the system to add additional rows to DBC tables.

To avoid this situation, reserve plenty of space in DBC. This is even more necessary if you are using access logging or database query logging because the tables produced can use up a lot of space. Monitor the space usage for the logging functions often and regularly.

Be sure to not only allocate enough space to DBC but also to regularly clean up old tables by deleting rows from them or archiving them instead if you need the data for later.

To create a spool reserve database, submit a CREATE DATABASE statement and specify the amount of space you want to keep in reserve as the PERM parameter.

Do not create objects or store data in this database. As long as the reserve database remains empty, its PERM allocation remains available for use as spool space.

For example, assume you created an administrative user named DBADMIN. Since the space of this user is under your control, you can use it to create a child database named Spool_Reserve that never contains tables, as follows:

1. Log on to the database as user DBC.
2. Submit the following statement:


```
CREATE DATABASE Spool_Reserve FROM DBADMIN AS PERM = n ;
```

where n is the number of bytes that is a specific percentage you have decided is appropriate.

3. Quit the session and log off.

If your system is using global space accounting, define the spool reserve database with zero skew.

Note:

If your system allows a database or user to exceed space limits (the GlobalSpaceSoftLimitPercent field in the DBS Control utility is set to a non-zero value), a DBA must ensure that the system keeps this extra space available. When creating the spool reserve database, a DBA should factor in the soft limit percent and set the PERM limit for the spool reserve database higher accordingly. This way, when users or databases exceed their space limits, out-of-physical-space errors are less likely to occur.

For more information on determining the allocation of spool space, see “Sizing Spool Space” in *Teradata Vantage™ - Database Design*, B035-1094. For more information on determining perm space requirements, see “Calculating Total PERM Space Requirements” in *Teradata Vantage™ - Database Design*, B035-1094. For more information on global soft limits, the SKEW option, and global space accounting, see [Global Space Accounting](#).

Guidelines for Reserving Minimum Spool Space

Analytics Database dynamically allocates available permanent space as spool space when necessary. To make sure that reducing permanent space does not impact transaction processing, Teradata recommends that you reserve permanent space for spool requirements.

At minimum, the average uncompressed and non-fallback protected database should reserve 40% of space relative to CURRENTPERM. (For example, if CURRENTPERM is 100 GB, spool space will be 40 GB for a total MAXPERM of 140 GB.)

You may increase the reserve as needed depending on your actual workload. Use the following table to help you estimate your spool space requirements. Note that the percent of spool space suggested is relative to CURRENTPERM.

System Fallback Rate	Compression Rate	Spool Space Reserve Recommended (Percent Relative to CURRENTPERM)
100%	0%	20%
100%	40%	30%
50%	20%	32%
0%	0%	40%
0%	40%	60%

Note:

Because every site is different and can greatly vary, use these numbers as guidelines only.

Using Viewpoint Alerts

Before setting up Viewpoint alerts, make sure you have:

- Installed a Teradata Viewpoint server and connected it to your system.
- Enabled the RSS data collection system on your Vantage system.
- Created the Teradata Viewpoint server logon user (viewpoint) in the database and given it the privileges necessary to logon and execute database monitoring and management functions.
- Configured at least one administrative user to access and administer Teradata Viewpoint and enabled the Viewpoint data collectors.

See the *Teradata® Viewpoint Configuration and Upgrade Guide*, B035-2207 for instructions.

You can configure Teradata Viewpoint to perform alert actions when the database achieves or exceeds certain defined thresholds. You can specify the following alert actions individually or in any combination:

- Write a message to the alert log
- Send an email message to a specified address
- Communicate the alert to a specified third-party application
- Run a specified BTEQ script or other program

Each alert action set has a unique name and is triggered according to the rules for any alert that specifies the action set name.

Setting Up Alert Actions

The alert setup defines alert notification parameters and alert actions.

See *Teradata® Viewpoint User Guide*, B035-2206 for procedure details.

1. From the main Teradata Viewpoint portal, select **Admin > Setup**.
2. Use **Delivery Settings/Email** to configure communication with the host SMTP email server.
3. Select the Vantage system for which you are configuring the alert.
4. Use **Alerts Presets/Action Sets** to set up common alert actions, for example, an email message to all administrative users. Then you can set up monitoring parameters that invoke the associated alert action whenever Teradata Viewpoint detects that a parameter was exceeded.

Note:

Specifying one or more roles in the action sends an email to each role member and is the most efficient way of identifying email recipients. You can list email addresses for any recipients that are not Teradata Viewpoint users.

Creating Alerts

A Teradata Viewpoint alert is based on:

- A threshold specified for a monitoring parameter, for example, when a database exceeds 95% space usage.
- The time period during which the parameter must be over threshold to invoke the alert.
- A definition of who should receive the alert and how the message is delivered.
- A message that the system delivers to the list of users defined in the Action Set, if the threshold and time criteria are met.

Using the **Alert Setup** portlet in Teradata Viewpoint, you can customize the type of alerts you want to receive. For details and options, see *Teradata® Viewpoint User Guide*, B035-2206.

Setting Up Teradata Viewpoint Alerts for Space Usage

You can use Teradata Viewpoint to regularly monitor space usage, and catch most space problems. However, it may be useful to setup a few alerts in areas where space problems can seriously affect database function.

The following table contains some examples of common space usage alerts.

Alert	Setup Parameters	Response to Alert Occurrence
Crashdumps Space The crashdumps database needs enough space to contain possible large data dumps that occur during crashes, and which are used in system recovery.	Alert rules: <ul style="list-style-type: none"> • Only include databases = crashdumps • Current Perm Max% > 70 Note: The threshold is set low because a crashdump can be very large.	<ul style="list-style-type: none"> • Add perm space to the crashdumps database. • Purge the crashdumps database of unneeded data and back up needed data to external media.
DBC Space User DBC contains all system tables and logs. Critical system functions depend on DBC having enough space.	Alert rules: <ul style="list-style-type: none"> • Only include databases = DBC • Current Perm Max% > 80 Note: Use 90% for SystemFE.	Add perm space to the database.

Alert	Setup Parameters	Response to Alert Occurrence
Note: You can set the same alert parameters for system databases DBCMNGR and SystemFE, which perform other critical system functions.		
Set up three alerts to identify when a database exceeds important space thresholds. <ul style="list-style-type: none"> • All DB 90% • All DB 95% • All DB 97% 	Alert rules: <ul style="list-style-type: none"> • All databases • Current Perm Max% > <i>percent</i> Alert Action: <ul style="list-style-type: none"> • Email (Pager for critical alerts) • Do not run twice in 5 minutes Severity: (apply one of the following to the related alert) <ul style="list-style-type: none"> • Medium (90%) • High (95%) • Critical (97%) 	You should allocate additional perm space to a database as soon as possible after perm space usage exceeds 90%. Databases that trigger the High or Critical alert levels before you can respond to a Medium alert are growing very fast and should receive more space immediately. You can investigate which tables are using database perm space.

Global Default Parameters

There are many global defaults that you can change for your system, including international character set settings, time zone, business calendars, and many others. For more information, see [Working with System Information and Global Defaults: Operational DBAs](#).

Databases and Users in Teradata: All DBAs

This section describes concepts important to understanding a Teradata environment:

- The differences between databases and users
- The difference between being the creator, owner, or parent of objects in the database hierarchy
- The system users and databases created at database initialization
- A recommended hierarchy for databases and users

Databases and Users

A database or user is a uniquely named permanent space that can store objects like tables, indexes, procedures, triggers, functions and other databases and users. (Views, macros, roles, and profiles do not consume space; their definitions are stored in the Data Dictionary, which takes up space in DBC.) Each database and each user may optionally contain one permanent journal.

A database is a logical repository for database objects, privilege definitions, and space limits. In Vantage, a user is the same as a database except it is an active repository that can log on with a password and establish a session.

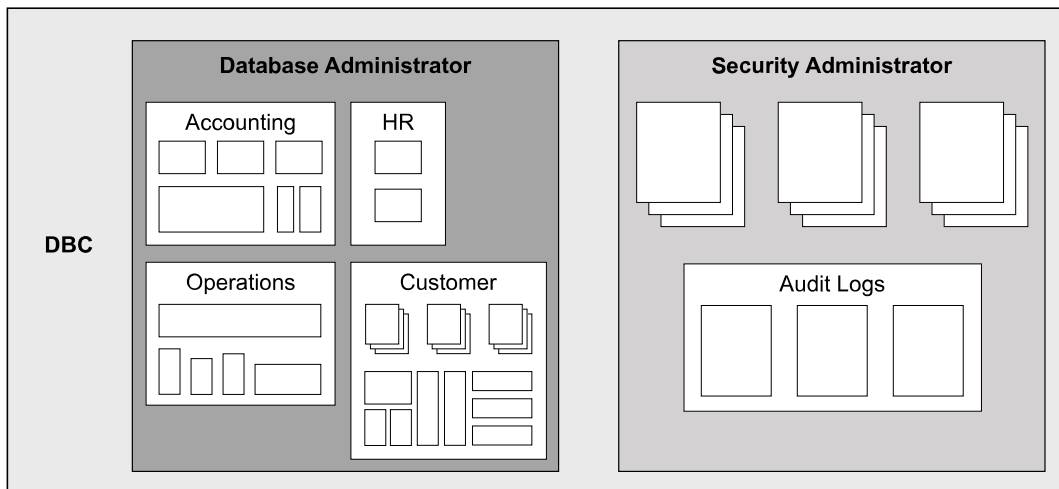
Databases and users may own objects such as tables, views, macros, procedures, and functions. Both may hold privileges. However, only users may log on, establish a Vantage session, and submit requests.

A user performs actions; a database is passive. Users have passwords and startup strings; databases do not.

Creator privileges are associated only with a user because only a user can log on and submit a CREATE statement. Implicit privileges are associated with either a database or a user because each can hold an object and an object is owned by the named space in which it resides.

Space Used by Databases and Users

On a newly installed system, user DBC initially owns all the space in the database. Teradata recommends that you create special administrative users that belong to DBC and create your databases and users from those special administrative users.



As you create objects or insert data rows, space is allocated as needed from the permanent space of the administrative user who is the immediate owner. If the administrative user grants other users the privilege to create more databases and users, those databases and users can only use the amount of space available to them.

When creating additional databases or users, you must always allocate a permanent space limit. (You define space limits at the database or user level, not at the table level.) The `MODIFY DATABASE/USER` statement also allows you to specify the maximum limit of temporary, spool, and permanent space allocated for a database and user.

Every database and user has an upper limit of temporary, spool, and permanent space. For more information on the different types of space, see [Managing Space: Operational DBAs](#).

Another method for ensuring adequate space is to use global space accounting. For details, see [Global Space Accounting](#).

Creating Versus Owning Objects in the Hierarchy

There is a difference between creating and owning objects in the database hierarchy. Owner privileges and creator privileges are different and determine default settings for undefined parameters during creation.

- The creator of an object is the user who submitted the `CREATE` statement. Every object has one and only one creator.

Note:

If the `CREATE` statement is executed within a macro, then the user who executed the macro is the creator of the object. The macro is just a vehicle for that creation.

- The owner of an object is any database or user above the object in the database hierarchy.
- The immediate owner (or parent) of a new user or database is the owner of the space in which the object resides. (In other words, a database or user that has space subtracted from its own permanent space to create a new object becomes the immediate owner of that new object.) The creator is the user

who submitted the CREATE statement, but that user can specify a different database with the FROM database option of the CREATE USER or CREATE DATABASE statement.

The default database is the database of the creating user, but can be a different database if you precede the object name with the database name and a period separator (databasename.objectname) in the CREATE statement.

An object must always have an immediate owner; that is, an immediate owner of one or more objects cannot be dropped. However, the immediate owner of a database or user can be changed.

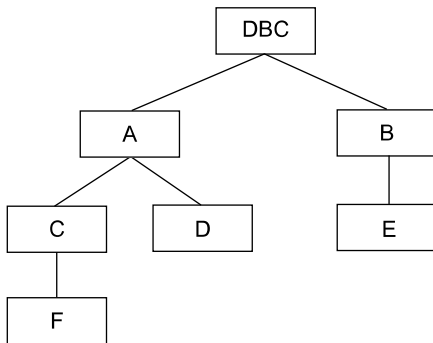
The following table lists the basic rules defining a creator, an owner, and an immediate owner.

Action	Result
You execute a CREATE statement that creates an object (anywhere in the database).	You are the creator of that object. You are not necessarily an owner, or the immediate owner, of that object.
You see an object directly below you in the hierarchy.	You are the immediate owner of that object.
You create an object in your own database.	You are both the creator and the immediate owner of that object.
You create an object in the space of another user or database (assuming you have the privilege that allows you to do so).	<ul style="list-style-type: none"> You are the creator of the object. The other user or database is the immediate owner of the object.
<ul style="list-style-type: none"> UserA owns UserB UserB creates an object in the database of UserB 	<ul style="list-style-type: none"> UserB is the creator of the object. UserB is the immediate owner of the object. UserA is also an owner, but not the immediate owner of the object.
You are using directory server integration. UserC is a directory-based user mapped to database-based UserA. UserC creates a database in UserD.	UserA is recorded as the creator and UserD is recorded as the immediate owner of the new database.

Creating a Database or User

Users other than DBC (or a site administrative user already granted privileges on ALL objects in the database by DBC) must explicitly be granted the CREATE DATABASE and CREATE USER privileges before they can create another user or database, even in their own space. For details, see “Privileges That Must Be Explicitly Granted” in *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

As you create users and databases, a hierarchical relationship evolves.



- DBC owns everything in the hierarchy, and is the immediate owner, or parent, of A and B.
- A owns C, D, and F. A is the immediate owner, or parent, of C and D.
- C is the immediate owner, or parent, of F.
- B is the immediate owner, or parent, of E.

The user who submits the CREATE DATABASE/USER statement is the creator of the database or user. The database or user whose permanent space is used to create a new database or user becomes the immediate owner of that new database or user.

In addition, that owner owns all databases or users below it in the hierarchy, because they are created from its original permanent space. The exception to this is if ownership of a specific database or user is transferred to another database or user.

The creator is not necessarily the immediate owner; a creator is the immediate owner only if the new database or user resides within the database of the creator (and thus is directly below the creator database in the hierarchy). With the appropriate privileges, the creator can create a new database or user somewhere else in the hierarchy. For more information on ownership privileges which are implicit privileges, as well as explicit privileges automatically granted to a creator, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

The System Users

Permanent data disk space includes database and user fixed PERM allocation. The Analytics Database initialization routine creates three system users as follows:

- DBC (at the top of the hierarchy)
- SYSTEMFE (owned by DBC)
- SYSADMIN (owned by DBC)

All permanent space not consumed by SYSTEMFE and SYSADMIN is allocated to user DBC.

System User DBC

The space allocation of system user DBC is used for the following:

DBC Space	Purpose
PERM	Provides ownership of the Teradata production database space, which must accommodate the maximum PERM space allocated to each user and database by its creator. The allocated space is used to store the rows in the data tables created by each user, including the following items: <ul style="list-style-type: none"> • Primary data table rows • Join index table rows • Hash index table rows • LOB and XML subtable rows • Secondary index subtable rows • Optional permanent journal image rows • Optional fallback rows, which replicate the rows of primary data tables, index subtables, hash and join index tables, and permanent journals
	Creates and maintains the WAL log/transient journal, storing a before-image copy of every row affected by data changes in all current sessions and Redo and Undo WAL records.
	Stores in the Data Dictionary the definitions for system tables, indexes, macros, triggers, and views.
	Provides permanent space for the following: <ul style="list-style-type: none"> • Table headers. Note that each materialized global temporary table also requires a minimum of 512 bytes from the PERM space of its containing database or user for its table header. • CRASHDUMPS database.
TEMP	Provides temporary space for data inserted into materialized global temporary tables.
SPOOL	Provides maximum defined SPOOL space to each user and database for the following uses: <ul style="list-style-type: none"> • Dynamic allocation and release of intermediate and final result sets during query processing. • Volatile temporary tables.

System User SYSTEMFE

System user SYSTEMFE is owned by user *DBC.SystemFE* contains special macros and objects. The quantity of PERM space allocated to SYSTEMFE is minimal.

Only Teradata field support personnel normally log on under this user name.

System User SYSADMIN

System user SYSADMIN is owned by *DBC.SysAdmin* contains special views and macros. The quantity of PERM space allocated to SYSADMIN is minimal.

Only Teradata field support personnel normally log on under this user name.

Other System Databases and Users

The following table lists some of the additional users and databases created during the installation process.

System Database /User	Description
CRASHDUMPS	CRASHDUMPS is defined in the Data Dictionary when the DIPCRASH script is run during the DIP phase of installation, migration, or upgrade. This user is created for use by the system for logging internal errors. Adjust the space allocation accordingly and monitor its usage. CRASHDUMPS provides temporary storage of dumps generated by the Parallel Database Extensions (PDE) software. You may need to allocate more PERM space, based on the size of your configuration, to accommodate at least three dumps.
Sys_Calendar	Note: Sys_Calendar is defined in the Data Dictionary when the DIPCAL script is run during the DIP phase of installation, migration, or upgrade. This database contains tables and views related to the system calendar, including the Calendar view, which is a user-accessible tool for date arithmetic. For more information, see “CALENDAR System View” in <i>Teradata Vantage™ - Data Dictionary</i> , B035-1092.
TD_SYSFNLIB	Note: The TD_SYSFNLIB system database is created when the DIPDB or DIPALL scripts are run during the DIP phase of installation, migration, or upgrade. This database contains the non-type-specific embedded services functions offered as part of the database release. For details on embedded services functions, see <i>Teradata Vantage™ - SQL Functions, Expressions, and Predicates</i> , B035-1145. Note: Do not create any database objects in the TD_SYSFNLIB database. The TD_SYSFNLIB database is for internal use only.

Note:

ALL, DEFAULT, and PUBLIC may appear as databases or user names but actually have no content, no tables, no views, and users cannot log on to them. They are defined with no PERM space and are used by the database system software. For more information, see “Special Keywords” in *Teradata Vantage™ - Data Dictionary*, B035-1092.

User DBC not only owns all these system databases/users but also has access to everything in the system because it has privileges on everything as well. Therefore, rather than use DBC to manage the system and risk accidentally changing the system users, create an administrative user to manage the system.

As-A-Service System Users

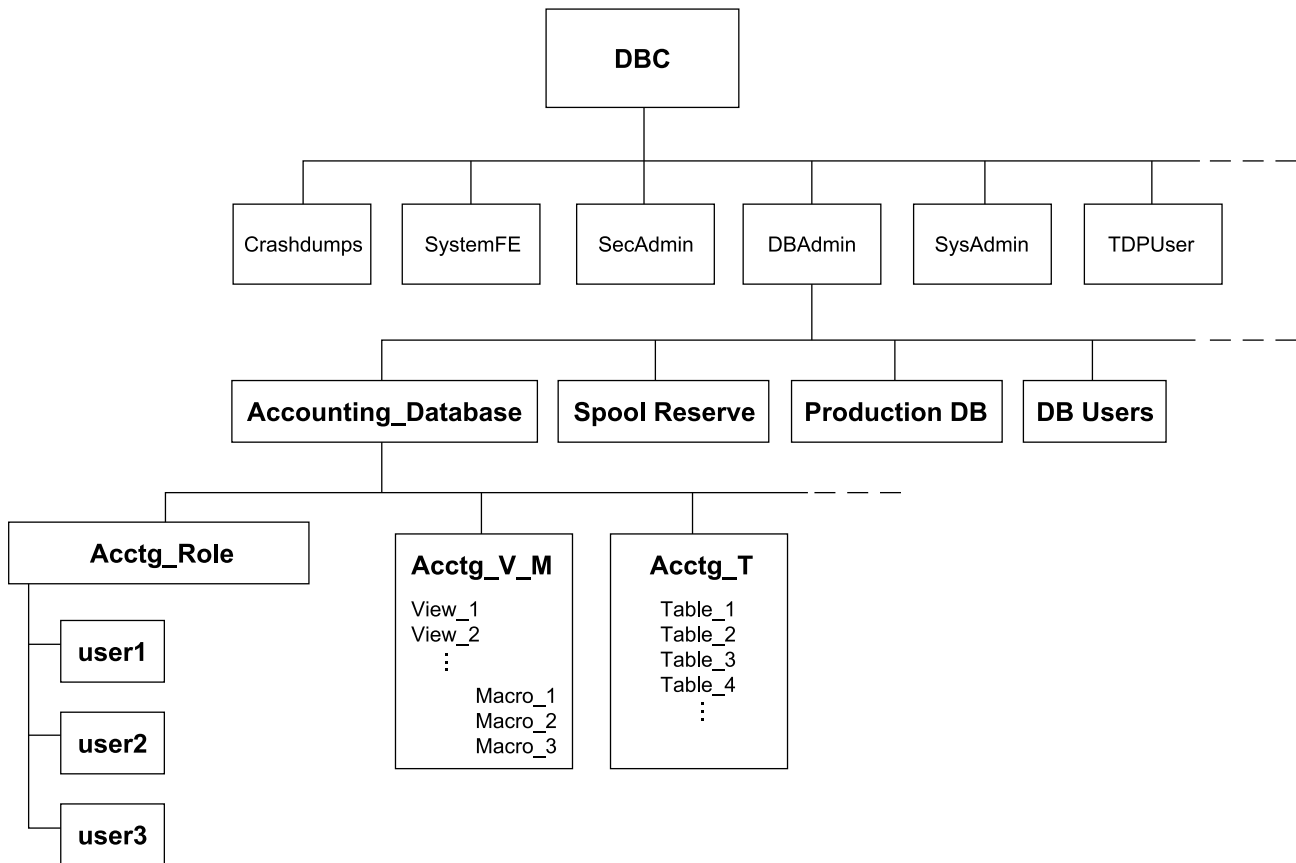
The following table lists pre-installed database users intended for Teradata personnel who are responsible for performing as-a-service administrative tasks on customer systems. The passwords for these users are securely managed with Teradata controlled vaults and services. Like any user, their assigned SQL access

rights can be displayed through Dictionary views. However, the properties of these system users and their rights cannot be dropped or altered by other users, including the DBC user.

System User	Administrative Role
TDaas_Support	Problem and error investigation
TDaaS_Maint	Database software patching and configuration
TDaaS_Monitor	Monitoring resource usage
TDaaS_BAR	Backup and restore of customer databases
TDaaS_DB	Owning database for as-a-service objects such as stored procedures

Recommended Hierarchy

Teradata recommends this type of hierarchy for databases and users:



Teradata Secure Zones Overview

Teradata Secure Zones allows you to create one or more exclusive database hierarchies, called zones, within a single database system in Vantage. Access to the data in each zone and zone administration is handled separately from the rest of the database system in Vantage.

Teradata Secure Zones are useful in situations where the access to data must be tightly controlled and restricted. You can also use Teradata Secure Zones to support some regulatory compliance requirements for the separation of data access from database administration duties.

For example, consider the following use of Teradata Secure Zones. Suppose you have a multinational company or conglomerate enterprise with many subsidiaries. You can create a separate zone for each of the subsidiaries. If your company has divisions in different countries, you can create separate zones for each country to restrict data access to the personnel that are citizens of that country. Your corporate personnel can manage and access data across multiple zones while the subsidiary personnel in each zone have no access to data or objects in the other zones. A system-level zone administrator can manage the subsidiary zones and object administration can be done by either corporate DBAs or zone DBAs, as required.

With Teradata Secure Zones, you can ensure the following:

- Users in one subsidiary have no access or visibility to objects in other subsidiaries.
- Corporate-level users may have access to objects across any or all subsidiaries.

Another typical scenario is the case of cloud companies that host multiple data customers as tenants. Companies that offer cloud-based database services can host multiple tenants within a single Vantage system, using zones to isolate the tenants from each other as if they were running on physically segregated systems. Zone DBAs can administer database objects in their own zone as required. The tenant zones can be managed by a system-level zone administrator, where Teradata acts as the system administrator.

With Teradata Secure Zones, you can ensure the following:

- Users in one tenant's zone have no access or visibility to objects within other zones.
- Users in one tenant's zone cannot grant rights on any objects in the zone to any other users, databases, or roles of other zones within the system.

Working with Databases: All DBAs

This section describes the best practices for database creation and steps through the tasks involved in initial database setup:

- Creating the Tables database
- Creating the Views database
- Granting table access privileges to the Views database

The section also details how to drop or transfer ownership of a database or user.

Database Creation

In Teradata Vantage, you must create database objects to contain other objects such as data tables, indexes, views, and macros, as well as users and other lower level databases. The disk space that you assign to a database determines the size and number of objects it can contain. Databases implicitly own and have privileges on all objects created within their assigned space. Individual objects contained in a database do not require a space allocation, but you can specify space limits for some objects.

Best Practices for Database Creation

To ensure data integrity, Teradata recommends that only administrators and high-level users have the privilege to access data tables. All other database activity should be done in views.

Teradata recommends that you create top-level databases to separate tables and views. You can create additional lower-level databases within the top level databases as required by site security policy and user access needs.

Database	Description
Tables_ Database(s)	<ul style="list-style-type: none"> • Owns perm space. • Contains all tables and data. • Normally accessible only to privileged administrative and batch users for creating and modifying database objects or performing data loads and updates.
Views_ Database(s)	<ul style="list-style-type: none"> • Requires minimal perm space because it does not contain data. • Contains views of the tables in the Tables_Database. • Exists primarily to provide SELECT access to the data for all users. <p>Note: Where security policy requires it, you can designate a layer of views for load and update operations so that even batch users do not need to directly access data tables.</p>

Creating the Tables Database

Use the following procedure to create the Tables Database that will contain all data tables.

1. Log on as user DBADMIN.
2. Specify values for the following database parameters:

Syntax Element	Explanation
Database Name	Required. The name of the master database containing all tables and data. Recommendation: Use a name that provides a hint about the function of the database, for example, Tables, Tables_Database, or TBL_DB.
Owner	Required. The name of the user or database that owns the space in which the object is being created. Recommendation: Enter the name of the primary administrative user, DBADMIN.
Permanent	Required. The permanent space in bytes allocated to contain tables and other objects created in the database. Recommendation: For the Tables database, enter a value or an expression resulting in a numeric value equivalent to 80% of permanent space allocated to user DBADMIN. If you create child databases within the Tables database, divide the Tables database perm space among them according to the size of the tables each database contains. Note: Tables do not have perm space allocations. They automatically take the space required to contain their data from the owning database.
Spool	Recommendation: Not applicable. Queries running against the Tables database draw from the spool space available to the user.
Temporary Space	Use TEMPORARY = <i>n</i> BYTES to define the space allowed by default for users to create global temporary tables within this database. The default is the largest value that is less than the owner temporary space and that is a multiple of the number of AMPs in the system. <i>n</i> must not exceed the owner temporary space. If no default temporary space is defined for a database, the space allocated for global temporary tables created in that database is the maximum temporary space allocated to the immediate owner of the database. The database/user must have adequate perm space to accommodate the global temporary table header on each AMP. Table header size varies by table definition and the maximum size for a table header is 1 MB.
Account	Not applicable. Recommendation: Specify accounts at the profile or user level.
Default Journal	The default location for journal images of tables created in the new or modified database. Recommendation: Specify only if you use journaling. See the syntax elements Before Journal and After Journal in this table.
Comment	Optional. You can enter text to describe the database.

Syntax Element	Explanation
Before Journal	Specifies journaling for “before” change row images as follows: <ul style="list-style-type: none"> • Yes – specifies a single journal. • No – specifies no journal. • Dual – specifies dual journals. This option is not normally specified for the initial database implementation.
After Journal	Specifies journaling for “after” change row images as follows: <ul style="list-style-type: none"> • Yes – specifies a single journal on a different AMP from changed row. • No – specifies no journal. • Dual – specifies dual journals. • Local – specifies a single journal on the same AMP as changed row. The Local option is only available for non-FallBack tables. This option is not normally specified for the initial database implementation.
FallBack	Enabled by default. Specifies that the system will automatically create a duplicate of each table stored in the database space to provide backup in the event of a failure. <p>Note:</p> You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.

Note:

This example is the minimum recommended specification for creating the Tables_Database. You can specify additional options later using the MODIFY DATABASE statement.

```
CREATE DATABASE "Tables_Database" FROM "DBADMIN"
AS PERM = 2e7 * (hashamp()+1)
NO BEFORE JOURNAL
NO AFTER JOURNAL;
```

3. You can optionally create additional child databases within the Tables database to group tables by function or department, but it is not required.

Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
3 and 4	Syntax and options for: <ul style="list-style-type: none"> • CREATE DATABASE • DELETE DATABASE • DROP DATABASE • MODIFY DATABASE 	<ul style="list-style-type: none"> • <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144 • <i>Teradata Vantage™ - SQL Data Definition Language Detailed Topics</i>, B035-1184

Creating the Views Database

Use the following procedure to create one or more Views Databases to contain all the views and macros required by the users.

1. Start the client tool of your choice and log on as user DBADMIN.
2. Specify values for the following database parameters.

Syntax Element	Explanation
Database Name	<p>The name of the master database containing user-accessible views. To make data more easily accessible you can create several Views databases according to the needs of various departments or user groups, for example:</p> <ul style="list-style-type: none"> • Views by department, such as Finance_Views and Purchasing_Views. • Views by user type, such as Batch_User_Views and General_User_Views. <p>You can also create Views databases by functional layer, for example:</p> <ul style="list-style-type: none"> • A layer of read-write views where each view mirrors a table, usable for batch loads and updates without direct access to data tables • A layer of read-only views that contain selected columns from one or more tables to combine related information from multiple tables or restrict access <p>For a description of user types, see Types of Users.</p>
Owner	<p>Required. The name of the user or database that owns the space in which the object is being created.</p> <p>Recommendation: Enter the name of the primary administrative user, DBADMIN, previously set up in Setting Up the Database Administrator User.</p>
Permanent Space	<p>The permanent space in bytes or an expression resulting in a numeric value allocated to contain space-consuming objects created in the database.</p> <p>Recommendation: The Views database is not meant to contain data, but it is the best place to locate stored procedures that access the views. You can allocate a small amount of permanent space for stored procedures, for example 100 MB, and then adjust the allocation later using the MODIFY DATABASE statement.</p>
Spool Space	Not applicable. Queries running in the Views database take spool space from the Spool_Reserve database.
Temp Space	Not applicable.
Account	<p>Not applicable.</p> <p>Recommendation: Specify account at the profile or user level.</p> <p>For information on using accounts, see Working with Accounts.</p>
Default Journal	<p>Not applicable.</p> <p>Recommendation: Not applicable, because journaling is not specified for views.</p>
Comment	Optional. You can enter text to describe the database.
Before Journal	Not applicable.
After Journal	Not applicable.

Syntax Element	Explanation
FALLBACK	<p>Optional. Fallback is required only for individual mission-critical tables or tables so large that their size prevents timely backup to tape or external disk, and this database contains only views of the data tables.</p> <p>Note: You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.</p>

Note:

This example is the minimum recommended specification for creating a Views_Database. You can specify additional database options in the initial CREATE DATABASE statement, or add them later using the MODIFY DATABASE statement.

```
CREATE DATABASE "Views" FROM "DBADMIN"
AS PERM = 0
NO BEFORE JOURNAL
NO AFTER JOURNAL;
```

- Grant access privileges to the Views database, as shown in [Working with Table Access Privileges for Views](#). After completing the procedure, return to this section and continue with the remaining tasks.

Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
3	<ul style="list-style-type: none"> CREATE DATABASE DELETE DATABASE DROP DATABASE MODIFY DATABASE 	<ul style="list-style-type: none"> <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144 <i>Teradata Vantage™ - SQL Data Definition Language Detailed Topics</i>, B035-1184
	An overview of SQL procedures and how they are stored in the database	<i>Teradata Vantage™ - SQL Stored Procedures and Embedded SQL</i> , B035-1148

Working with Table Access Privileges for Views

You must grant table access privileges to all views databases to allow the views within each database to access tables in the Tables database.

Note:

When table access privileges are granted to a Views database, the individual views inherit the right to access tables in the Tables database from the parent Views database.

Granting Table Access to Each Views Database

1. Start BTEQ or Teradata Studio and log on as the database administrator user.
2. Run the following requests:
 - For read-only Views databases:

```
GRANT SELECT ON "Tables_Database" TO "database_name" WITH GRANT OPTION
```

- For the updatable Views databases:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON "Tables_Database" TO  
"database_name" WITH GRANT OPTION
```

Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
2	Syntax and options for the GRANT (SQL form) statement	<i>Teradata Vantage™ - SQL Data Control Language</i> , B035-1149
	Database privileges, including those conferred by object ownership	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Dropping a Database or User

Dropping a database or user is a two-step process:

1. Delete all database objects from the specified database or user. For example:

```
DELETE DATABASE used_cars ALL;
```

or

```
DELETE USER marco ALL;
```

2. Drop the empty database or user. For example:

```
DROP DATABASE used_cars;
```

or


```
DROP USER marco;
```

If a journal table exists, first be sure that no data table references it. Then remove it. For example:

```
MODIFY DATABASE used_cars AS DROP DEFAULT JOURNAL TABLE;
```

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

The database or user permanent space that the drop makes available is added to the permanent space of the immediate owner database or user.

Note:

All physical database maintenance is performed automatically. You do not need to restructure or reorganize a database to reuse space. Some physical database maintenance activities required by other database systems (such as eliminating pointer chains and reorganizations) do not apply to Vantage.

Transferring Ownership with GIVE

Transfer databases and users from one immediate owner to another using the GIVE statement, for example:

```
GIVE Finance TO Alice;
```

The GIVE statement also transfers all child databases and users as well as the objects (tables, views, macros, indexes, stored procedures, and triggers) owned by the transferred object.

Rules affecting transfer of ownership include:

- You cannot give an object to children of the object.
- The permanent space owned by the database or user is also transferred.
- To use GIVE, you must have the explicit DROP DATABASE/USER privilege on the object being transferred and explicit DROP/CREATE DATABASE/USER privilege on the receiving object.
- Any explicit privileges granted to others on a transferred user are not automatically revoked. You need to explicitly REVOKE explicit privileges.
- Changing ownership hierarchy impacts implicit privileges. When a database or user has a new owner, the former owner loses implicit privileges on the object (and all objects below it in the hierarchy). The new owner (and those above him in the hierarchy) gains implicit privileges on the object and all objects below it in the hierarchy.

Transferring ownership affects space allocation. Plan carefully and check space distribution and object ownership before using the GIVE statement.

Working with Tables and Views: Application DBAs

This section describes essential tasks and best practices for working with tables and views:

- Choosing a primary index
- Creating tables or converting pre-existing tables from other vendors
- Copying, dropping, and recreating tables
- Creating views
- Using BTEQ scripts to create database objects

Choosing a Primary Index

Defining the primary index (PI) for a table is the most critical aspect of table design.

The system uses the PI to assign each data row to an Access Module Processor (AMP). A well chosen PI balances the distribution across AMPs and helps optimize the performance of queries that access the table. PIs also enable efficient joins and aggregations.

Note:

A query does not use a PI for access unless all columns in the index are in an equality predicate or an IN LIST.

After you identify the PI requirements for a table, you specify the PI as part of the CREATE TABLE statement.

Guidelines for Choosing Primary Index Columns

There are three essential factors to consider when choosing a primary index:

- Uniform data distribution (the most important consideration)
- Optimal access to the data
- The volatility of indexed column values

Use the following guidelines for selecting columns to be used as primary indexes:

- Select columns that consist mainly of unique, distinct values.

This is the most important consideration. Columns with distinct values distribute data rows evenly across all AMPs in the configuration. This maximizes parallel processing by allowing all the processors to participate in processing the data rows for a request.

Avoid using columns with a small number of distinct values that are repeated frequently or columns that have many nulls. This will cause uneven distribution of data rows resulting in some AMPs having more rows to process than others, and will increase the potential for performance bottlenecks.

Recommendation: Select columns having significantly more distinct values than the number of AMPs in the configuration.

- Select columns that are most frequently used in equality predicate conditions.
- Select columns that are most frequently used to access rows.

The PI provides the most efficient method to access data in a table. Therefore, choosing the PI on the most frequently used access path provides for optimal data access. For example, if the table is frequently joined with a specific set of tables, consider defining the PI on the column set that is typically used as the join condition.

Equality conditions permit direct access to the row. Inequality conditions require additional processing.

- Select columns that do not have any of the following data types: BLOB, CLOB, Period, UDT, ST_Geometry, MBR.
- Choose as few columns as possible for the PI to optimize its generality.
- Select primary indexes so that query plans use AMP-local processing and avoid row redistributions.
- Select columns that are not volatile.

Volatile columns force frequent row redistribution because Analytics Database assigns table rows to AMPs based on the PI values.

Note:

Several DBSControl values, including those for AMP buffer size, affect actual row distribution.

Recommendation: Select a PI column set that will never be (preferred) or rarely be updated.

You may encounter conflicts when selecting the optimum primary index for a table. For example, choosing a PI that gives good distribution of data across the AMPs must be balanced against choosing a PI that reflects the most common usage pattern of the table. Also, some applications might favor one type of primary index, while other applications might perform optimally using a different primary index.

Recommendation: Make even data distribution, not data access patterns, the main priority when selecting a primary index. Tables and queries change. Access patterns change. However, data distribution does not. You can always add additional non-primary indexes, such as secondary and join indexes, to facilitate particular applications.

Related Information

Topic	Resources for Further Information
Indexing: <ul style="list-style-type: none"> • Primary indexes • Partitioned primary indexes • Secondary indexes • Join indexes (materialized views) 	<ul style="list-style-type: none"> • <i>Teradata Vantage™ - Database Design</i>, B035-1094 • <i>Teradata Vantage™ - Database Introduction</i>, B035-1091 • <i>Teradata Vantage™ - SQL Fundamentals</i>, B035-1141
No primary index tables	<i>Teradata Vantage™ - Database Design</i> , B035-1094

Topic	Resources for Further Information
Referential integrity	<i>Teradata Vantage™ - SQL Fundamentals</i> , B035-1141

Unique and Nonunique Primary Indexes

You can define the primary index as unique (UPI) or nonunique (NUPI), depending on whether duplicate values are allowed in the indexed column set.

UPIs provide optimal data distribution and are typically assigned to the primary key for a table. However, the primary key is not necessarily always the best candidate for a primary index.

When a table is designed with a NUPI, consider assigning a uniqueness constraint such as PRIMARY KEY or UNIQUE to the primary or other alternate key of the table. This both enforces uniqueness and enhances retrieval for applications where the primary or other alternate key is frequently used as a selection or join criterion.

Row Partitioning

Primary indexes can be row partitioned or nonpartitioned. A partitioned primary index (PPI) permits rows to be assigned to user-defined data partitions on the AMPs. A PPI is defined to be either single-level or multilevel. Multilevel partitioning allows each partition to be subpartitioned.

The greatest potential gain in row partitioning a primary index is the ability to access only the rows of the qualified partitions while skipping partitions that do not contain rows meeting the search conditions of a query.

A PPI increases query efficiency by avoiding full-table scans without the overhead and maintenance costs of secondary indexes. A multilevel PPI provides multiple access paths to the rows in the base table and can improve query performance through row partition elimination at each of the various levels or combination of levels.

PPIs are designed to optimize range queries while also providing efficient PI join strategies. A range query requests data that falls within specified boundaries. A common example of this is queries that involve date ranges.

Note:

Column partitioning is not supported for tables with a primary index.

Recommendation: Consider defining a table with a PPI to support either of the following workload characteristics:

- A majority of the queries in the workload specify range predicates on some column, particularly a date column, of the PPI table.
- A majority of the queries in the workload specify an equality predicate on some column of the PPI table, and that column is either not the only column in the primary index column set or not a primary index column.

No Primary Index (NoPI) Tables

You can create tables without a primary index (NoPI). The performance of FastLoad and Teradata Parallel Data Pump (TPump) Array INSERT operations can be enhanced by using NoPI staging tables.

You can use NoPI tables as intermediate work tables where PI access is not a consideration. The system automatically balances the distribution of NoPI tables across AMPs.

NoPI tables are designed for these circumstances:

- Temporary use
- When you have not yet determined a suitable primary index
- Column-partitioning (which cannot be done on tables with a primary index)

In all other circumstances, Teradata recommends that you specify a primary index when you create a table.

Relationship of Primary Indexes, Primary Keys, and Foreign Keys

A primary key defines a column, or columns, that uniquely identify a row in a table. The values defining a primary key for a table:

- Must be unique
- Should not change
- Cannot be null

The following table summarizes the differences between logical primary keys and physical primary indexes:

Primary Key	Primary Index
Important element of logical data model.	Not used in logical data model.
Used to maintain referential integrity.	Used to distribute and retrieve data.
Must be unique to identify each row.	Can be unique or nonunique.
Values should not be changed if you want to maintain data integrity and preserve historical relations among tables.	Values can change.
Cannot be null.	Can be null.
Does not imply access path.	Defines the most common access path.
Required only if referential integrity checks are to be performed.	Defined for most production tables. (Some staging tables may not have a primary index.)

The database uses a unique primary or secondary index to enforce a primary key; therefore, the primary key can affect how Analytics Database distributes and retrieves rows.

Although the primary key is often used as the primary index for a table, it may or may not be the best column set to choose as a primary index. You should base your primary index decision on physical database design considerations that may favor columns other than those of the primary key.

A foreign key identifies table relationships. They model the relationship between data values across tables. A foreign key defines a column, or combination of columns, in a table. The foreign key column(s) must exist in the referenced tables as a primary key or a UNIQUE alternate key.

Analytics Database uses primary and foreign keys to maintain referential integrity between tables. Foreign keys are also used in join conditions.

Creating Tables

Converting Existing Database Tables to Vantage Tables

This procedure uses Oracle as an example to show how you might convert tables created in another database to Vantage.

1. Identify each Oracle schema/user and all database objects (for example, tables, indexes, views, and materialized views) contained in each schema.

Oracle uses schemas to group database objects into functional groupings that are created and owned by users.

In Vantage, database objects are created and organized within the owning database or user:

- A database is a collection of related database objects. A database also contains an allotment of space from which users can create and maintain their own objects, or other users or databases.
- A user is similar to a database, having a space allocation and the capacity to contain objects. However, a user also has a password and can log on to the system, whereas a database cannot.

2. Identify the schemas that have objects you want to migrate to Vantage.
3. Create databases in Vantage that correspond to the Oracle schemas and map the schemas to the corresponding databases.
4. Identify the DDL for each schema object.
5. Map Oracle data types to Teradata data types.
6. Map Oracle table elements to Vantage table elements.

See [Creating Tables in Teradata](#) for a description of common Vantage table options.

Oracle and Vantage tables have some common elements:

- All tables have names.
- All tables contain columns with data types.
- Tables can be partitioned.

In Oracle, you must create tablespaces to store table data. You can partition a table and distribute specific partitions for storage in specific tablespaces.

In Vantage, the system uses the Primary Index (PI) to distribute table rows across units of parallel execution (AMPs). See [Guidelines for Choosing Primary Index Columns](#).

You can achieve further partitioning of the data rows within each AMP by defining a Partitioned Primary Index (PPI) for the table, which is similar to the type of partitioning that Oracle tables perform. However, it is easier to define PPI schemes in Vantage because the partitioning is a logical organization of data within the database file system, and it requires no knowledge of table space definitions or physical disks. See [Row Partitioning](#).

- Constraints can be defined on table columns.
- Referential integrity can be defined.
- Global temporary tables are supported.
- The data block size can be defined to optimize the storage of row data.
- Freespace can be defined to retain room after loading for future data growth.
- Triggers can be defined.

Vantage has a number of additional table options, including:

- Primary index – used for data distribution, data access, table joins, and aggregations.
- FALLBACK option – allows mirroring of table data for recovery scenarios.
-

Note:

You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.

- Journaling options – stores an image of each data row that has been changed to provide data protection in case of a failure.

Also, in Oracle you must enable a table to be accessed with parallel query processes, whereas parallel query process are automatically accepted in Vantage.

7. Translate the DDL for each schema object to Teradata DDL.

The DDL for creating tables differs between Oracle and Vantage primarily in how space utilization is defined for the tables.

In Oracle, database objects are created and stored in tablespaces. Tables can have space utilization parameters describing how they will use space in tablespaces.

In Vantage, database objects are created and stored in databases or users. The database or user definition also defines a space allocation. The system automatically transfers space from a database or user to owned tables as needed, so database table definitions do not include database storage management clauses.

Recommendation: To facilitate loading data from your Oracle table into the new database table, define the column names of the table to be the same as the column names of your Oracle table. You may use the ALTER TABLE statement to rename any of the columns in the database table after loading the data into the table.

For details on using the CREATE TABLE statement, see [Creating Tables in Teradata](#).

Considerations for Defining Tables

A table acquires data attributes when you define its columns in a CREATE TABLE or ALTER TABLE statement with at least a name and a data type phrase.

Data attributes control the internal representation of stored data and determine how that data is presented to a user. You can use the FORMAT phrase to change the external representation returned by a SELECT query.

The table definition directly affects the performance of applications that access that table. Proper planning helps you define a table correctly at the time of creation. The following table lists several things to consider when planning your tables.

Issue	Considerations
Compression (Algorithmic)	Specify the COMPRESS USING <i>udfname</i> option of the CREATE TABLE or ALTER TABLE statement to compress character data by table column and minimize storage. This option allows you to choose your own compression or decompression algorithms. Algorithmic compression works best on seldom-used data. For more information, see <i>Teradata Vantage™ - Database Design</i> , B035-1094.
Compression (Block-level)	Specify the BLOCKCOMPRESSION = <i>block_compression_option</i> of the CREATE TABLE or ALTER TABLE statement to compress data blocks of primary and fallback tables to minimize storage. Software-based block-level compression works best on large volumes of seldom-accessed tables. For more information, see <i>Teradata Vantage™ - Database Design</i> , B035-1094.
Compression (Multivalue)	Specify the COMPRESS phrase to compress up to 255 specific values or NULLs to zero space. (Nullable columns with nulls are automatically compressed, when COMPRESS is specified.) Use ALTER TABLE to immediately add, change, or delete compression on a column, column partition, or a table. For more information, see <i>Teradata Vantage™ - Database Design</i> , B035-1094.
Default value of a column	When you create or alter a table, you can specify a default value for a column by using the DEFAULT function. Specifying a default value allows the system to return a value associated with the column. If you do not specify a value, the default is null. Note: You cannot assign a user-defined type (UDT) as an explicit default value to the column. You can assign the column to return USER, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP built-in values as the default value. For more information, see <i>Teradata Vantage™ - SQL Functions, Expressions, and Predicates</i> , B035-1145. For more information on DEFAULT and NULLs returned to the host, see <i>Teradata Vantage™ - Data Types and Literals</i> , B035-1143.
Default versus NULL for aggregation results	During aggregation, the unspecified value represented by a: <ul style="list-style-type: none"> • NULL, which is ignored in the calculation. • Default value, which is included in the calculation. For example, assume you want the average salary of employees in Department 300 as follows:

Issue	Considerations
	<pre>SELECT DeptNo, AVG(Salary) FROM Employee GROUP BY DeptNo WHERE DeptNo = 300 ;</pre> <p>If a salary is not known, the result differs depending on how the column is defined. If an unspecified value in the Salary column is defined with the following:</p> <ul style="list-style-type: none"> • A DEFAULT value, then the result includes the default value as if it were the actual value. This may be far from the average result if all values were known. • A NULL, then the result is the average of the salaries that are known. NULL is ignored. This may be closer to the actual average. <p>You can do the following to further control how unknowns are handled during aggregation:</p> <ul style="list-style-type: none"> • Exclude substitutions by making NULL the default value for an unknown, then use the NULLIF function definition; for example: <pre>NULLIF(SALARY, defaultvalue)</pre> <p>The <i>defaultvalue</i> is ignored in computing the average.</p> <ul style="list-style-type: none"> • Change an unknown represented by a NULL into the default value, then use the COALESCE function definition. For example: <pre>COALESCE(Salary, defaultvalue)</pre> <p>where <i>defaultvalue</i> is the value to be used in computing the average.</p>
Defining consistent data types and column names	<p>Columns for the same data in different tables should have the same type and name for ease of identification. For example, if the column title is EmpNo in one table, it should be EmpNo in another table. Consistency of data across tables is critical. Some considerations include:</p> <ul style="list-style-type: none"> • Changing data in one table. If you change data in one table, data in other tables may be affected. For example, updating a department number in the Department table also affects the Employee table, which contains a DeptNo column. To maintain data integrity, you can use: <ul style="list-style-type: none"> ◦ A macro, a stored procedure, a trigger, or an application program to update all the tables affected by a change. ◦ RI constraints. (For more information, see <i>Teradata Vantage™ - Database Design</i>, B035-1094 and <i>Teradata Vantage™ - SQL Data Definition Language Detailed Topics</i>, B035-1184.) • Joining the columns of one table with the columns of another table. Ensure the following: <ul style="list-style-type: none"> ◦ Join columns are of the same data type and size. ◦ Data is consistent in both tables. For example, to join two tables on the Employee Name column, the notation must be identical in both tables (for instance, last name first initial, such as "Smith H"). ◦ All row-level security-protected tables referenced in a join contain the same constraints. • Using tables to store large data objects. Create a table with LOB type columns. This includes: <ul style="list-style-type: none"> ◦ Character Large Object (CLOB)- A CLOB column can store character data, such as simple text, HTML, or XML documents.

Issue	Considerations
	<ul style="list-style-type: none"> Binary Large Object (BLOB)- A BLOB column can store binary objects, such as graphics, video clips, files, and documents.
IDENTITY column	<p>IDENTITY is an optional attribute used to generate a number for every row inserted into the table on which it is defined. An identity column does not have to be the first column in the table or defined as an index. The IDENTITY attribute may be specified such that the generated number will be unique.</p>
Joins involving NULLs	<p>If a column used in a join contains either NULLs or generic values, the results might be misleading. For example, assume you want to join the Employee and Department tables to obtain a listing of employees and their workplaces:</p> <pre>SELECT Name, Loc FROM Employee, Department WHERE Employee.DeptNo = Department.DeptNo;</pre> <p>Undesired results may occur. The nulls in each table prevent matches. Employees with an unknown department number (Employee.DeptNo is NULL) and departments without employees are not listed.</p> <p>To list an employee with an unknown department number and a department with no employees, use a different default value to represent unknowns in the DeptNo column, as follows:</p> <ol style="list-style-type: none"> In the Employee table, add a mock “unknown” employee who has a DeptNo equal to the value of the default used for the Department table. To the Department table, add a mock “unknown” department that has a DeptNo equal to the value of the default used for the Employee table. <p>Note:</p> <p>In the example query, a full outer join could be used instead of the inner join to obtain information for non-matching rows when there is a NULL in DeptNo. (Also, as noted previously, you may use the COALESCE function to change a NULL to a value.) However, using outer joins to obtain non-matching rows may be more efficient.</p> <p>For further discussion on the implications of using NULL and for information on how to design efficient joins, see “Joins and Hash Indexes” in <i>Teradata Vantage™ - Database Design</i>, B035-1094.</p>
NULL and the GROUP BY clause	<p>If you use a column defined as NULL for grouping aggregate results, the results may be confusing. For example, assume you submit this query to find the average salary for each department in the organization:</p> <pre>SELECT DeptNo, AVG(Salary) FROM Employee GROUP BY DeptNo;</pre> <p>The results can differ, depending on the definition of the DeptNo column:</p> <ul style="list-style-type: none"> If DeptNo is allowed to be NULL, and two employees have not yet been assigned a department number, then the result lists the computed average for those two employees under a NULL department number. This might be confusing. If DeptNo is defined with DEFAULT and the specified constant is meaningful (such as Unknown), this result is more valid.

Issue	Considerations
	<p>Note:</p> <p>You can convert a NULL into a default value (with NULLIF) and a default value into NULL (with COALESCE), as explained previously for aggregated results.</p>
Nullability of a column	<p>It may not always be possible to specify a value when inserting a new data row. For example, a new employee may not immediately have a job title. You can explicitly define the nullability of a column with a NULL or NOT NULL phrase as follows:</p> <p>If no value is given for a column when a row is inserted, and the nullability of a column is the following:</p> <ul style="list-style-type: none"> • Defined as NOT NULL and a DEFAULT value is not specified, then the INSERT statement returns an error. • Not defined, then a NULL is supplied automatically. <p>Note:</p> <p>It is best practice is specify NOT NULL for a column with a default value (if a default value is appropriate) unless there is a valid reason for allowing the column to be nullable. This saves space and allows for more efficient query plans in some cases. If the column does need to be nullable, be sure to determine, understand, and document the meaning of NULL for this column since null has various interpretations.</p> <p>If an application program requests data from a column (without NULL indicators) that allows nulls and a NULL is found, a substitute value that is compatible with the data type of the column is returned to the application instead of a null. The substitute value (a zero, blank, or zero-length element string) might be misleading, because NULL is an unknown.</p> <p>Note:</p> <p>Never use the phrase NOT IN on nullable columns always replace it with the phrase NOT EXISTS. If NOT IN is used, it will make the database work a lot harder.</p> <p>Also, note that any comparison to NULL results is an UNKNOWN value. NOT EXISTS, however, treats UNKNOWN values as FALSE as shown in EXPLAIN.</p> <p>For example:</p> <p>1 in (1,2,NULL) -> 1=1 or 1=2 or 1=NULL -> TRUE OR FALSE OR UNKNOWN -> TRUE</p> <p>1 not in (1,2,NULL) -> 1<>1 and 1<>2 and 1<>NULL -> TRUE and FALSE and UNKNOWN -> UNKNOWN</p> <p>For more information on unexpected answer sets, see “Behavior of Nulls for NOT IN” in <i>Teradata Vantage™ - SQL Functions, Expressions, and Predicates</i>, B035-1145.</p>
Other considerations	<p>Consider the following:</p> <ul style="list-style-type: none"> • Whether or not to have a primary index. (See the NoPI row in this table.) • The columns to use in the primary index and whether the PI is unique or not. • If the table has a primary index, whether to have one or more levels of row partitioning. • If there is no primary index, whether to have column partitioning and, optionally, one or more levels of row partitioning. • What secondary indexes to define, if any, and whether they are unique. • What constraints are needed on each of the columns. • Row header compression for column partitions that have COLUMN format can reduce storage costs and enhance system performance. However, in some cases, such as wide column partitions that do not benefit from autocompression, ROW format (no row header compression) may be better.

Issue	Considerations
	<ul style="list-style-type: none"> Whether to allow concurrent reads on committed rows in a table while loading. A table must be set up as load isolated to use this feature. A load-isolated table requires more space than a non-isolated table and requires regular maintenance to remove logically deleted rows. For more information, see Reading Committed Data While Loading to the Same Table.
NoPI table	<p>Use a NoPI table when either you want to column-partition the table, a primary index does not add any value, you do not know yet what primary index to specify, or you need a staging table and restrictions on NoPI tables are acceptable.</p> <p>To create a staging table, use a NoPI table which is a table without a PI. Using a NoPI table for staging purposes improves performance when loading data from FastLoad and TPump jobs because the table is not hashed by a PI and the rows do not have to be sorted in any particular order or redistributed to any particular AMP.</p> <p>NoPI tables are useful for applications that load data into a staging table that needs to go through some sort of transformation or standardization. The data must be converted before being stored in another table. You can then apply the data from the staging table to a target table through an INSERT ... SELECT, UPDATE-FROM or MERGE-INTO statement.</p> <p>It is best practice is to explicitly specify a PI or NoPI instead of relying on the default for the PI. The default may not be appropriate and may cause skewing of the data on the AMPs.</p> <p>For more information on how to create a NoPI table, see “CREATE TABLE” in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144.</p>

Creating Tables in Teradata

The following procedure shows how to create tables using BTEQ, but you can also use Teradata Studio.

1. Logon on to Vantage as user DBADMIN using BTEQ.
2. Create one or more tables using the CREATE TABLE statement. For example:

```
CREATE SET TABLE database_name.table_name,
  (column_name data_type,
   column_name data_type,
   column_name data_type)
UNIQUE PRIMARY INDEX (primary_index_column);
```

MULTISET SET

A MULTISET table allows duplicate rows in compliance with the ANSI/ISO SQL 2011 standard. A SET table does not allow duplicate rows.

If there are uniqueness constraints on any column or set of columns in the table definition, then the table cannot have duplicate rows even if it is declared as MULTISET.

database_name

The name of the database where the table will be created if it is different from the current database.

Recommendation: The database should be the *Tables_Database* or a database located within the *Tables_Database*. See [Database Creation](#).

If *database_name* is not specified, the system creates the table in the default database for the current session.

table_name

The name of the table to be created.

[NO] FALLBACK

Specifies that the system will automatically create a duplicate copy of the current image of every row in this table to provide backup in case of a failure.

You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.

column_name

Specifies the name of one or more columns, in the order in which they and their attributes are to be defined for the table.

Up to 2,048 columns can be defined for a table. Note that other limits or feature restrictions may apply before you reach the column limit. For example, you can have no more than 32 large object (LOB) columns per table.

data_type

Specifies a single data type for each column.

Some data types have different names in Teradata than in other databases. For more information, see *Teradata Vantage™ - Data Types and Literals*, B035-1143.

[UNIQUE|NO] PRIMARY INDEX

Specifies the primary index. A table can have no more than one primary index. If you do not explicitly assign a primary index, Vantage will choose a default primary index (unless you specify NO PRIMARY INDEX).

Recommendation: Explicitly define either PRIMARY INDEX, UNIQUE PRIMARY INDEX, or NO PRIMARY INDEX because the default selected by Vantage may not be optimal for the table.

primary_index_column

Specifies the column(s) that define a primary index. If you specify more than one column, the index is created on the combined values of each column. You can specify up to 64 columns.

Recommendation: Choose columns that primarily have unique values, are frequently accessed, and have values that do not change. See [Guidelines for Choosing Primary Index Columns](#).

Note:

The example is the minimum recommended specification for creating a table. You can specify additional options, or add them later using the ALTER TABLE statement, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

To automate the creation of data tables by using BTEQ scripts, see [Using BTEQ Scripts to Create Database Objects](#).

3. Set privileges on the table so that only designated administrative users can access the table directly.
4. Collect statistics on the newly created, empty table. This defines the columns, indexes, and partitions for a PPI table, as well as the data structures for subsequent collection of statistics and demographics.

Optional Table-level and Column-level Elements

The following are common optional table-level elements you can specify in the CREATE/ALTER TABLE statement. For details about these table options and for a complete description of all table options, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Syntax Element	Explanation
GLOBAL TEMPORARY	<p>Creates a global temporary table. Global temporary tables have a persistent definition but do not have persistent contents across sessions. Each user session can materialize up to 2,000 global temporary tables at a time.</p> <p>To use global temporary tables, create or modify the user or profile to have a temporary space limit by using the TEMPORARY = <i>n</i> BYTES option.</p> <p>If no temporary space limit is defined for a profile, Analytics Database uses the temporary space limit defined for the individual user-creator. If no temporary space is defined for a user, then the space allocated for any materialized global temporary tables referenced by that user is set to the maximum temporary space allocated for the immediate owner.</p>
JOURNAL	<p>Specifies a Permanent Journal (PJ) for the table, which stores an image of each data row that has been changed with an INSERT, UPDATE, or DELETE statement. PJ images can reflect row values as they appeared before the change, after the change, or both.</p> <p>To specify a PJ for all tables in a database, use the JOURNAL clauses in the CREATE DATABASE statement.</p> <p>Recommendation: Specify journaling for tables that need an additional level of data protection after you have analyzed data protection needs.</p>

Syntax Element	Explanation
FREESPACE	<p>Specifies the percent of free space that remains on a cylinder during loading operations. Reserved free space allows tables to expand within their currently allocated cylinders. This can prevent or delay the need for additional cylinders to be allocated, which incurs the overhead of moving data to the new cylinders. Avoiding new cylinder allocations can improve overall system performance. You can control this space at the global and table level.</p> <p>Recommendation: If little or no table expansion is expected (the table is read-only), set FREESPACE to 0. Otherwise, choose a percentage that reflects the growth rate of your table (INSERTs minus DELETes). Common settings are 5 to 15%. The maximum allowed is 75%.</p>
DATABLOCKSIZE	<p>Specifies the maximum data block size for blocks that contain multiple rows. Larger block sizes enhance full table scan operations by selecting more rows in a single I/O. Smaller block sizes are best for transaction-oriented tables to minimize overhead by retrieving only what is needed. You can control the data block size at the global and table level.</p> <p>Recommendation: Use a larger value if the database is used primarily for strategic work (decision support/complex queries). Use a smaller value if the database is used primarily for tactical work (OLTP).</p>
[UNIQUE] INDEX	<p>Specifies secondary indexes for the table. Secondary indexes (SI) allow quick access to table data using an alternate, less frequently used path than the primary index. SIs improve performance by avoiding full table scans.</p>
PARTITION BY	<p>Specifies that the primary index is partitioned based on the value of the specified partitioning expression. A partitioned primary index (PPI) permits rows to be assigned to user-defined data partitions, enabling enhanced performance for range queries while providing efficient PI join strategies. See Row Partitioning.</p>

The following are common optional column-level elements you can specify in the CREATE/ALTER TABLE statement. For details about these column-level options and for a complete description of all table options, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Syntax Element	Explanation
<i>data_type_attributes</i>	<p>If you specify attributes for a column, you must define its data type before you define its attributes. The following attributes are supported:</p> <ul style="list-style-type: none"> • NOT NULL • UPPERCASE • [NOT] CASESPECIFIC • FORMAT • TITLE • NAMED • DEFAULT or WITH DEFAULT • CHARACTER SET • WITH TIME ZONE <p>Recommendation: Teradata highly recommends that you specify the NOT NULL attribute for columns that will never be null.</p>

Syntax Element	Explanation
COMPRESS	<p>Specifies a set of distinct values in a column that is to be compressed to zero space. Column compression can enhance storage capacity, improve response time for table scans, and reduce disk I/O traffic.</p> <p>Recommendation: For best results, use multivalue compression where:</p> <ul style="list-style-type: none"> • Enough rows contain a compressible value (null, zero, blank, or constant value) in the compressed field to exceed the break even point. • The shortened row size eliminates one or more data blocks.
CONSTRAINT	<p>Specifies column-level constraints. The following constraints are supported:</p> <ul style="list-style-type: none"> • UNIQUE • PRIMARY KEY • FOREIGN KEY • CHECK • REFERENCES <p>Recommendation: Specify a name for the constraint specification.</p>

Example of the CREATE TABLE Statement

This table definition creates the Employee table in the database named Tables_Database:

```
CREATE SET TABLE Tables_Database.Employee,
  (Associate_Id      INTEGER,
   Associate_Name    CHAR(25),
   Salary            DECIMAL(8,2),
   DOB              DATE,
   Job_Title         VARCHAR(25),
   Dept_No          SMALLINT,
   Marital_Status    CHAR,
   No_Of_Dependents  BYTEINT)
UNIQUE PRIMARY INDEX (Associate_Id);
```

Using BTEQ script, see [Using BTEQ Scripts to Create Database Objects](#).

Related Information

Topic	Resources for Further Information
An overview of tables, including global temporary and volatile tables	<i>Teradata Vantage™ - SQL Fundamentals</i> , B035-1141
Syntax, options, and required privileges for the following: <ul style="list-style-type: none"> • CREATE TABLE • ALTER TABLE • DROP TABLE 	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144

Topic	Resources for Further Information
• RENAME TABLE	
The default database	<i>Teradata Vantage™ - SQL Fundamentals</i> , B035-1141
Data types and data type attributes	<i>Teradata Vantage™ - Data Types and Literals</i> , B035-1143
<ul style="list-style-type: none"> • Primary indexes • Partitioned primary indexes • Secondary indexes • Join indexes (materialized views) 	Choosing a Primary Index
Using the FREESPACE option to manage performance, and guidelines for calculating a FREESPACE value	FREESPACE table option in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144 and "FreeSpacePercent" DBS Control field in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Using the DATABLOCKSIZE option to manage performance and specifying a DataBlockSize value	DATABLOCKSIZE table option in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144 and "DataBlockSize" DBS Control field in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Compressing column values	<i>Teradata Vantage™ - Database Design</i> , B035-1094
Column-level and table-level constraints	<i>Teradata Vantage™ - Database Design</i> , B035-1094

Copying a Table

Use the AS option of the CREATE TABLE statement to copy some or all of an existing table. The following table describes several options for copying tables.

Statement	Result
CREATE GLOBAL TEMPORARY TABLE AS [tablename/ query_expression] ... WITH NO DATA	<p>Copies an existing table as a global temporary table. Use WITH NO DATA, because global temporary tables are not populated until they are materialized by being referenced in a query.</p> <p>If you want the table to inherit the following:</p> <ul style="list-style-type: none"> • All of the column definitions, then specify <i>tablename</i> WITH NO DATA. • A subset of the column definitions, then specify (<i>query_expression</i>) and WITH NO DATA. <p>The subquery form of CREATE GLOBAL TEMPORARY TABLE AS ... WITH NO DATA does not copy indexes and defines a default primary index which might not be appropriate. Teradata highly recommends specifying explicitly an appropriate PI or specify NO PRIMARY INDEX.</p>
CREATE TABLE AS [tablename/ query_expression] .. . WITH [NO] DATA [AND STATISTICS]	<p>Copies an existing table as a permanent table or a volatile table. You choose what columns you want to copy and whether the table should be populated automatically, as follows. If you want the table to inherit the following:</p> <ul style="list-style-type: none"> • All of the column definitions plus the contents, then specify <i>tablename</i> WITH DATA and (for a volatile table) ON COMMIT PRESERVE ROWS.

Statement	Result
	<ul style="list-style-type: none"> • All of the column definitions but none of the contents, then specify <i>tablename</i> WITH NO DATA. • A subset of the column definitions plus the contents, then specify (<i>query_expression</i>) WITH DATA and (for a volatile table) ON COMMIT PRESERVE ROWS. • A subset of the column definitions but none of the contents, then specify (<i>query_expression</i>) WITH NO DATA. • The statistical histograms of the original table, then specify AND STATISTICS. If USECOUNT is enabled, this option also copies use count information from the original table. See SQL Logging Statements. <p>The subquery form of CREATE TABLE AS ... WITH [NO] DATA does not copy indexes or partitioning and defines a default primary index, which might not be appropriate. Teradata highly recommends specifying explicitly an appropriate PI, Primary AMP index, or specify NO PRIMARY INDEX.</p>

Copying Statistics From a Base to a Target Table

Use the optional AND STATISTICS clause with the CREATE TABLE AS ... WITH DATA statement to copy statistics from a base table to a target table. Using the AND STATISTICS clause saves time from having to collect statistics on the new table.

Note:

You can use the shorthand STATS or STAT in place of STATISTICS.

If you choose not to copy statistics, you can either omit the AND STATISTICS clause or you can specify the AND NO STATISTICS clause. If you wish to copy the table using zeroed statistics, submit the WITH NO DATA AND STATISTICS option instead.

For more information on usage rules and syntax, see CREATE TABLE (AS Clause) in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Dropping a Table

Use one of the following statements to drop a table:

Table You Want To Remove	Statement To Use
Permanent data table	<p>DROP TABLE.</p> <p>Dropping a table does not delete the views, macros, functions or stored procedures that referenced the dropped table. You need to explicitly drop or alter the reference.</p> <p>You cannot drop a table defined with a trigger, hash index, or join index. First drop the index or trigger and then drop the table.</p>
Materialized global temporary table, before session end	DROP TEMPORARY TABLE.

Table You Want To Remove	Statement To Use
	Note: Analytics Database automatically drops all materialized temporary tables at session end.
Volatile table before session end	DROP TABLE.
Global temporary table definition	DROP TABLE.

Recreating a Table

You may need to recreate a data table to:

- Change a default PI to a defined PI for a nonempty table.
- Add or remove a PI for a nonempty table.
- Change the PI to a different PI for a nonempty table.
- Change a NUPI to a UPI in a populated table when there is no USI (however, usually you would just create the USI on the table, and then alter it).
- Redefine the partitioning of a populated table. In some cases, you can still use ALTER TABLE; see “ALTER TABLE” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.
- Change a data type attribute that affects existing data. For rules on changing data types, see “ALTER TABLE” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.
- Define or delete COMPRESS storage attributes for an existing column.
- Add columns that would otherwise exceed the maximum for the number of columns defined during the life of a table.
- Move the table to another database or user.
- Change the groupings of columns into column partitions.

Note:

It is best practice to access a table via a view, especially for tables used by multiple users.

Use the SHOW TABLE or SHOW IN XML TABLE request to display the current table definition, which you can then modify and submit.

Using INSERT ... SELECT

Use the INSERT ... SELECT statement to load the current data rows quickly into a new table.

The procedure for recreating a table is as follows:

1. Select the explicit privileges of the old table with the following query:

```
SELECT username, accessright, grantauthority, columnname, allnessflag
FROM dbc.allrightsV
WHERE tablename = 'Employee' AND databasename = 'Personnel';
```

Note:

Save the output for later use; you will need to recreate the explicit privileges on the new table.

2. Create a new table with a temporary name, such as Temp.

```
CREATE TABLE Temp_Employee
(col1 datatype, col2 datatype...)
```

To display the DDL for the current table, submit a SHOW TABLE or SHOW IN XML TABLE request.

3. If any data types, columns, column attributes, or the column order are not compatible, use an INSERT ... SELECT statement that constructs compatible values. If this is not possible, you may need to load data from an external source.

If the data types are compatible, you can transfer all data rows from the old table to the new table with a single INSERT ... SELECT statement:

```
INSERT INTO Temp_Employee
SELECT *
FROM Employee ;
```

If the tables are not compatible, a more complicated SELECT that computes the values for the target table from columns of the source table is needed.

4. Use SHOW JOIN/HASH INDEX or SHOW IN XML JOIN/HASH INDEX to see the index definitions. If needed, drop the hash index or join index using DROP JOIN INDEX or DROP HASH INDEX and recreate it using CREATE JOIN INDEX or CREATE HASH INDEX.
5. Drop the old Employee table:

```
DROP TABLE Employee ;
```

Note:

When the table is dropped, explicit privileges are also dropped because the Data Dictionary references objects by ID rather than by name.

6. Rename the temporary table:

```
RENAME TABLE Temp_Employee TO Employee ;
```


7. Use the index definitions from step 4 to recreate join and hash indexes that you want to maintain for the new table.
8. Submit GRANT statements to re-establish the explicit privileges you saved in step 1 on the new version of Employee table.

Use the LOGGING ERRORS option to log errors. Errors include duplicate row errors, duplicate primary key errors, CHECK constraint errors, and more. For more information, see INSERT and INSERT ... SELECT in *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.

Specifying Fallback Tables

Fallback provides data protection at the table level by automatically storing a copy of each permanent data row of a table on a different or “fallback” AMP.

If an AMP fails, Analytics Database can access the fallback copy and continue operation. If you cluster your AMPs, fallback also provides for automatic recovery of the down AMP once you bring it back online.

Define fallback using the CREATE/ALTER TABLE commands. Fallback-protected tables occupy twice the permanent space as non-fallback tables and require twice the I/O for inserts, updates and deletes. However, the advantages in continued operation and data integrity are well worth the space.

You can define fallback for the following:

- Primary data table
- Secondary index subtable
- Join index of any type, including aggregate join index
- Hash index
- Databases
- Users

Fallback is beneficial because it does the following:

- Permits access to table data when an AMP is offline.
- Protects the accessibility and integrity of your data if it becomes unavailable due to a software error.
- Adds a level of data protection beyond disk array RAID.
- Automatically applies changes to the offline AMP when it is back online.

Note:

You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.

Working with Views

Database views are created from one or more base tables or from other views. They are virtual tables that you can use to retrieve data from the underlying views or tables. A view does not contain data so it does not use any space. Only the view definitions are stored in the Data Dictionary.

A view may present only a subset of the columns and rows in the base table or tables. In addition, a column defined in a view can be derived and does not need to exist in the underlying base tables. For example, you can display summed or averaged data in a view column.

Views are generally used to:

- Allow users to access only the rows and columns they need in a table.
- Enforce security by restricting table access and updates.
- Provide well-defined, well-tested, high-performance access paths to data.
- Provide logical data independence, which minimizes the need to modify your applications if you restructure base tables.
- Format the output of the data for reports.
- Specify locking, such as specifying a less restrictive ACCESS lock.

To ensure data integrity and system security, Teradata recommends the following:

- Set privileges so that general users cannot access or modify data tables directly.
- Create views which allow general users to access or modify only the table data they need. Use views to restrict users to performing specific queries and update functions.
- Create views in a Views Database.

You can use the following utilities to create views in Teradata:

- BTEQ
- Teradata Studio

BTEQ and Teradata Studio allow you to submit CREATE VIEW DDL statements to Analytics Database.

Creating Views Procedure

The following procedure shows how to create views using BTEQ:

1. Log on to Teradata Vantage as user DBADMIN using BTEQ.
2. Create one or more views using the CREATE VIEW statement.

For example:

```
CREATE VIEW database_name.view_name (column_name [,...] ) AS SELECT_clause;
```

database_name

The name of the database where the view will be created if it is different from the current database.

Recommendation: The database should be a Views Database or a database located within a Views Database. See [Database Creation](#).

If *database_name* is not specified, the system creates the view in the default database for the current session.

view_name

The name of the view to be created.

column_name

The name of a view column. If more than one column is specified, list their names in the order in which each column is to be displayed for the view.

LOCKING_clause

The specified lock is placed on the underlying base table set each time the view is referenced in an SQL statement.

Recommendation: Create views with a LOCKING ... FOR ACCESS modifier to allow concurrent access of the data to read-only users and users who will modify the data. This prevents deadlocks and increases performance.

ACCESS locking is a tool that can be used to reduce the isolation of a user request allowing that request to access valid, but uncommitted data that is subject to change.

This allows users that are interested in a broad, statistical snapshot of the data, but that do not require precise results, to attain access to data while modifications to that data are occurring. This can greatly improve performance, because those users do not have to wait unnecessarily for the write operation to complete.

SELECT_clause

The SELECT statement which obtains the data for the view. The following are some of the options and clauses you can use:

Option	Description
DISTINCT	Returns only one row from any set of duplicate rows in the SELECT result.
ALL	Returns all rows, including duplicate rows in the SELECT result. This is the default.
TOP <i>n</i> or TOP <i>m</i> PERCENT [WITH TIES]	Restricts a view to only <i>n</i> rows or <i>m</i> percent of the rows in an underlying base table. If you do not specify an ORDER BY clause, then a TOP <i>n</i> clause only specifies that any <i>n</i> base table rows be returned, not the TOP <i>n</i> . This option provides a fast method to obtain a restricted, non-statistically random sample of table rows.
FROM	Specifies the set of base tables or views from which data for the view are selected.
WHERE	Specifies a conditional expression by which rows for the view are selected.
GROUP BY	Groups result rows by the values in one or more columns or by various extended GROUP BY operations on specified column expressions.

Option	Description
HAVING	Specifies a conditional expression by which the groups defined by a GROUP BY clause are selected.
QUALIFY	Specifies a conditional ordered analytical function by which rows for the view are selected.
WITH CHECK OPTION	Integrity constraint option that restricts the rows in the table that can be affected by an INSERT or UPDATE statement to those defined by the WHERE clause. This option only pertains to updatable views. Recommendation: Specify a WITH CHECK OPTION clause in all your updatable view definitions to prevent unauthorized modification of data rows.
ORDER BY	Specifies the order in which result rows are to be sorted. You can only specify an ORDER BY clause for a view definition if you also specify either the TOP <i>n</i> or the TOP <i>m</i> PERCENT option.

Note:

The example lists the major options for creating a view. You can modify your view definition by using the REPLACE VIEW statement. To automate the creation of views by using BTEQ scripts, see [Using BTEQ Scripts to Create Database Objects](#).

3. Set privileges on the view as follows:

- Provide general users read-only access so they can use the view to query data.
- Provide privileged users read and write access so they can use the view to update data in the underlying tables.

Example: Creating a View to Limit User Access

The following statement creates a view of the Employee table so that it provides access only to the names and job titles of the employees in department 300:

```
CREATE VIEW Views_Database.Dept300 (Associate_Name, Job_Title) AS
  SELECT Associate_Name, Job_Title
  FROM Tables_Database.Employee
  WHERE Dept_No = 300
  WITH CHECK OPTION;
```

The WITH CHECK OPTION prevents using this view to insert a row into the Employee table, or to update any row of the Employee table where Dept_No is not 300.

Example: Creating a View to Generate a Report

The following statement creates a view which calculates the minimum, maximum, and average salary of each department and displays only those rows with an average salary of \$35,000 or higher.

Because the view definition specifies an ACCESS lock for the table, the view returns valid, but uncommitted data that is subject to change, particularly if the view selects data at the same time another user attempts to modify the salary data in the Employee table.

This view is designed for users who need quick access to data, but do not need precise results.

```
CREATE VIEW Views_Database.Dept_Salary (Dept_No, MinSal, MaxSal, AvgSal)
AS LOCKING TABLE Tables_Database.Employee FOR ACCESS
SELECT Dept_No, MIN(Salary), MAX(Salary), AVG(Salary)
FROM Tables_Database.Employee
GROUP BY Dept_No
HAVING AVG(Salary) >= 35000;
```

Related Information

Related topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
2	Syntax, options, and required privileges for: <ul style="list-style-type: none"> CREATE VIEW/REPLACE VIEW DROP VIEW RENAME VIEW 	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
	The default database	<i>Teradata Vantage™ - SQL Fundamentals</i> , B035-1141
	The SELECT statement, including usage of the options and clauses	<i>Teradata Vantage™ - SQL Data Manipulation Language</i> , B035-1146
	<ul style="list-style-type: none"> Rules for creating, replacing, and using views Updatable views WITH CHECK OPTION clause 	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
	<ul style="list-style-type: none"> Deleting rows using views (DELETE, Basic /Searched Form) Updating rows using views (UPDATE) 	<i>Teradata Vantage™ - SQL Data Manipulation Language</i> , B035-1146
	Specifying locking in views	<i>Teradata Vantage™ - SQL Request and Transaction Processing</i> , B035-1142
	CREATE RECURSIVE VIEW and REPLACE RECURSIVE VIEW	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144

Getting View Column Information

The DBC.ColumnsV[X] views provide complete information for table columns but provide only limited information for view columns. For view columns, DBC.ColumnsV[X] provides a NULL value for ColumnType, DecimalTotalDigits, DecimalFractionalDigits, CharType, ColumnLength, and other attributes related to data type.

To obtain complete information about view columns, issue a SELECT request on either DBC.ColumnsQV[X] or DBC.ColumnJQV[X]. A SELECT request on DBC.ColumnsQV[X] or DBC.ColumnsJQV[X] can also be joined with other views and tables.

- DBC.ColumnsQV[X] provides the same information as DBC.ColumnsV[X] for all database objects. DBC.ColumnsQV[X] additionally provides complete information for view columns.
- DBC.ColumnsJQV[X] provides the same information as DBC.ColumnsV[X] for tables, NoPI tables, and views, and DBC.ColumnsJQV[X] additionally provides complete information for view columns. DBC.ColumnsJQV[X] does not provide information for database objects other than tables, NoPI tables, and views. The filter improves performance for requests that return information about tables and views only.

The following request lists all the VARCHAR columns of the view 'Views_Database.Dept300':

```
SELECT * FROM DBC.ColumnsQV WHERE DatabaseName='Views_Database' AND
TableNames='Dept300' AND ColumnType = 'CV';
```

Using BTEQ Scripts to Create Database Objects

Basic Teradata Query Utility (BTEQ) is a general-purpose, command-based application that allows users to do the following:

- Perform administrative tasks, including creation of database objects such as databases, tables, and views, using SQL DCL, DDL, and DML requests.
- Format reports for both print and screen output.

You can automate the creation of tables, views, and other database objects using BTEQ scripts.

The following example procedure uses a BTEQ script to create a sample table:

1. Create a text file named Create_Emp_Table.sql with the following contents:

```
.LOGON tdpid/UserName,Password
CREATE SET TABLE Tables_Database.Employee,
(Associate_Id INTEGER,
Associate_Name CHAR(25),
Salary DECIMAL(8,2),
DOB DATE,
Job_Title VARCHAR(25),
Dept_No SMALLINT,
```



```

Marital_Status CHAR,
No_Of_Dependents BYTEINT)
UNIQUE PRIMARY INDEX (Associate_Id);
.quit

```

tdpid

The name by which the system is known to the network.

UserName

The name used to log on to Teradata, such as DBADMIN.

Password

The password associated with *UserName*.

2. Create a batch file named Create_Emp.bat with the following contents:

```
bteq < Create_Emp_Table.sql > Create_Emp_Table.log 2>&1
```

Example:

```
2>&1
```

This example redirects the error messages to Create_Emp_Table.log to be printed along with the output messages.

3. Place Create_Emp.bat and Create_Emp_Table.sql in the same folder.
4. Execute Create_Emp.bat to create the Employee table in the database, Tables_Database.

Create_Emp.bat invokes BTEQ using Create_Emp_Table.sql as the input file. BTEQ executes the commands and statements in the Create_Emp_Table.sql file and writes its output and any errors to the Create_Emp_Table.log file.

Related Information

Topic	Resources for Further Information
Starting and exiting BTEQ	<i>Basic Teradata® Query Reference</i> , B035-2414
Running batch jobs to submit BTEQ commands and Teradata SQL	<i>Basic Teradata® Query Reference</i> , B035-2414

Working with Stored Procedures and User-defined Functions: Application DBAs

This section provides real-world examples of creating stored procedures and user-defined functions.

Creating Stored Procedures

Example 1: Compiling and Executing a Stored Procedure with a Cursor

In most circumstances, it is not optimal to use a cursor for high-volume table updates. For details, see: <https://downloads.teradata.com/blog/georgecoleman>.

However, cursor logic is required to generate a sequence of dynamic SQL statements. This example assumes that there is a file called `Sales_Dept_Grant.sql`. The file contains this procedure:

```
REPLACE PROCEDURE Sales_Dept_Grant ( )
BEGIN
Declare GrantStmt VARCHAR(77) Default '';
For UserList as UserList CURSOR FOR
    Select User_Name, Admin_Ind From Sales_Dept_Users
Do
    CASE UserList.Admin_Ind
    WHEN 0 Then
        Set GrantStmt = 'GRANT SELECT on Sales_Dept to ' ||
UserList.User_Name;
    ELSE
        Set GrantStmt = 'GRANT ALL on Sales_Dept to ' ||
UserList.User_Name || '
With Grant Option';
    END CASE;
    CALL DBC.SysExecSql( :GrantStmt );
END For;
END;
```

If you are in the directory that contains this file, invoke BTEQ and enter these commands:

```
.LOGON <IP-addr>/<User-ID>
[Enter the Password]
.COMPILE FILE=Sales_Dept_Grant.sql /* This compiles the procedure */
CALL Sales_Dept_Grant();          /* This invokes the procedure */
```


If you are not in the directory that contains the file, then include the path of the file in the compile command:

```
.COMPILE FILE=C:\Teradata\Procs\Sales_Dept_Grant.sql
```

Note:

To compile the procedure from other client tools, copy and paste the contents of Sales_Dept_Grant.sql into the command window. Only BTEQ uses the .COMPILE command.

Testing Background

This example was tested using these test tables:

```
CT Sales_Dept (Acct varchar(44), Sale_Date DATE, Gross DEC(18,2), Net DEC(18,2);

insert into Sales_Dept values ('Acme Bearings','2013-02-13',12345.00,1234.50)
;insert into Sales_Dept values ('Horse Hockey','2013-03-23',57890.00,5789.00)
;insert into Sales_Dept values ('My Jaxx','2013-04-02',678930.00,67893.00)

CT Sales_Dept_Users (User_Name varchar(32), Admin_Ind BYTEINT);

insert into Sales_Dept_Users values ('GenUser1',1)
```

Example 2: Creating a Stored Procedure to Update Accrued Interest

SQL To Create a Savings Table

The following code, Savings.sql, creates a savings table and inserts test data into the table:

```
DROP TABLE Savings;
CREATE TABLE Savings
,NO BEFORE JOURNAL
,NO AFTER JOURNAL
(
  AccountType          CHAR(1) NOT NULL
,AccountNumber         CHAR(20) NOT NULL
,OpeningBalance        DECIMAL(9,3)
,AccruedInterest       DECIMAL(9,3)
,LastYearEnd           DATE
,LastAccrualDate       DATE
)
UNIQUE PRIMARY INDEX( AccountType,AccountNumber );
INSERT INTO Savings ( 'S', '1200121',10000.00,0,'2012-12-31','2013-01-01');
```



```

INSERT INTO Savings ('S', '1200122', 20000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('S', '1200123', 30000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('S', '1200124', 40000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('S', '1200125', 50000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('C', '1200121', 1000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('C', '1200122', 2000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('C', '1200123', 3000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('C', '1200124', 4000.00, 0, '2012-12-31', '2013-01-01');
INSERT INTO Savings ('C', '1200125', 5000.00, 0, '2012-12-31', '2013-01-01');

```

Stored Procedure to Update Accrued Interest

The following stored procedure, SP_Accrue_Interest.sql, uses a cursor to select qualified records and update accrued interest.

```

REPLACE PROCEDURE SP_Accrue_Interest(IN Arg_AccontType CHAR(20), OUT Proc_Msg
VARCHAR(80))

BEGIN

DECLARE pv_AccountType    CHAR(1);
DECLARE pv_AccountNumber  CHAR(20);
DECLARE pv_CalcBase       DEC(9,2);
DECLARE pv_Interest       DEC(9,2);
DECLARE pv_ProcCount       INT;

DECLARE calc_cursor CURSOR FOR
SELECT  AccountType, AccountNumber, (OpeningBalance+AccruedInterest)
        AS BaseAmount FROM Savings
WHERE   (AccountType = :Arg_AccontType
OR      'A'          = :Arg_AccontType)
AND     LastAccrualDate< DATE;

OPEN    calc_cursor;

FETCH   calc_cursor INTO pv_AccountType, pv_AccountNumber, pv_CalcBase;

SET     pv_ProcCount = 0;
SET     Proc_Msg     = 'Processed ' || pv_ProcCount || ' records';

Calc_Loop:
WHILE (SQLCODE =0) DO

    SET     pv_ProcCount = pv_ProcCount + 1;

```



```

SET      Proc_Msg      = 'Processed ' || pv_ProcCount || ' records';

SET      pv_Interest   = pv_CalcBase * 0.010;

UPDATE   Savings
SET      AccruedInterest = AccruedInterest + :pv_Interest
        ,LastAccrualDate = DATE
WHERE    AccountType    = :pv_AccountType
AND      AccountNumber  = :pv_AccountNumber;

FETCH    calc_cursor INTO pv_AccountType, pv_AccountNumber,pv_CalcBase;

END WHILE;
CLOSE calc_cursor;
END;

```

BTEQ Script to Call the Stored Procedure

The following BTEQ script, Bld_judf.btg, calls the stored procedure and performs SELECTs from the savings table to test the stored procedure.

```

.SET WIDTH          240

.logon TDSsystem/MyUserId,MyPass

.REMARK 'Choosing database';

DATABASE Test_Database;

.REMARK 'Building & populating savings table';
.RUN FILE=../Savings.sql

.REMARK 'Building stored procedure';
.COMPILE FILE=../SP_Accrue_Interest.sql

.REMARK 'Testing stored procedure';

SELECT * FROM Savings  ORDER BY 1,2;

CALL SP_Accrue_Interest('C',"Messg");
SELECT * FROM Savings  ORDER BY 1,2;

CALL SP_Accrue_Interest('S',"Messg");
SELECT * FROM Savings  ORDER BY 1,2;

```



```

CALL SP_Accrue_Interest('C',"Messg");
SELECT * FROM Savings ORDER BY 1,2;

CALL SP_Accrue_Interest('S',"Messg");
SELECT * FROM Savings ORDER BY 1,2;

.REMARK 'Rebuilding & populating savings table';
.RUN FILE=./Savings.sql

SELECT * FROM Savings ORDER BY 1,2;

CALL SP_Accrue_Interest('A',"Messg");
SELECT * FROM Savings ORDER BY 1,2;

CALL SP_Accrue_Interest('A',"Messg");
SELECT * FROM Savings ORDER BY 1,2;

.REMARK 'Finished testing stored procedure';

.LOGOFF
.EXIT 0

```

Linux Code To Call the BTEQ Script

This Linux code calls the BTEQ script:

```
bteq bld_iudfs.btq 2>&1 | tee bld_iudfs.btq.i1.out
```

Output of the Stored Procedure

```

BTEQ 14.00.00.02 Wed Apr 24 09:07:08 2013
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
.SET WIDTH 240
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
.logon TDSys/MyUserId,
*** Logon successfully completed.
*** Teradata Database Release is 14.10i.00.434
*** Teradata Database Version is 14.10i.00.434
*** Transaction Semantics are BTET.
*** Session Character Set Name is 'ASCII'.
*** Total elapsed time was 1 second.
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
.REMARK 'Choosing database';
Choosing database
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
DATABASE Felix_Test;

```



```

*** New default database accepted.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
.REMARK 'Populating savings table';
Populating savings table
+-----+-----+-----+-----+-----+-----+-----+
+-----+
.RUN FILE=../SavingsData.sql
+-----+-----+-----+-----+-----+-----+-----+
+-----+
DELETE FROM Savings ALL;
*** Delete completed. 10 rows removed.
*** Total elapsed time was 2 seconds.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200121',10000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 3 seconds.
+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200122',20000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.
+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200123',30000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200124',40000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200125',50000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200121', 1000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200122', 2000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200123', 3000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200124', 4000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

```



```

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200125', 5000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
*** Warning: EOF on INPUT stream.
+-----+-----+-----+-----+-----+-----+-----+
+-----+
.REMARK 'Testing internal UDF';
Testing internal UDF

+-----+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 6 seconds.
AccountType  AccountNumber  OpeningBalance  AccruedInterest
LastYearEnd  LastAccrualDate
-----
C            1200121            1000.000            .000
12/12/31      13/01/01
C            1200122            2000.000            .000
12/12/31      13/01/01
C            1200123            3000.000            .000
12/12/31      13/01/01
C            1200124            4000.000            .000
12/12/31      13/01/01
C            1200125            5000.000            .000
12/12/31      13/01/01
S            1200121            10000.000           .000
12/12/31      13/01/01
S            1200122            20000.000           .000
12/12/31      13/01/01
S            1200123            30000.000           .000
12/12/31      13/01/01
S            1200124            40000.000           .000
12/12/31      13/01/01
S            1200125            50000.000           .000
12/12/31      13/01/01
+-----+-----+-----+-----+-----+-----+-----+
+-----+
CALL SP_Accrue_Interest('C',"Messg");
*** Procedure has been executed.
*** Total elapsed time was 2 seconds.
Proc_Msg
-----
Processed          5 records
+-----+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 1 second.
AccountType  AccountNumber  OpeningBalance  AccruedInterest
LastYearEnd  LastAccrualDate
-----
C            1200121            1000.000            10.000
12/12/31      13/04/24
C            1200122            2000.000            20.000
12/12/31      13/04/24
C            1200123            3000.000            30.000
12/12/31      13/04/24

```



```

C          1200124          4000.000          40.000
12/12/31   13/04/24
C          1200125          5000.000          50.000
12/12/31   13/04/24
S          1200121          10000.000          .000
12/12/31   13/01/01
S          1200122          20000.000          .000
12/12/31   13/01/01
S          1200123          30000.000          .000
12/12/31   13/01/01
S          1200124          40000.000          .000
12/12/31   13/01/01
S          1200125          50000.000          .000
12/12/31   13/01/01
+-----+-----+-----+-----+-----+-----+-----+
+-----+
CALL SP_Accrue_Interest('S',"Messg");
*** Procedure has been executed.
*** Total elapsed time was 1 second.
Proc_Msg
-----
Processed          5 records
+-----+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 1 second.
AccountType AccountNumber OpeningBalance AccruedInterest
LastYearEnd LastAccrualDate
-----
C          1200121          1000.000          10.000
12/12/31   13/04/24
C          1200122          2000.000          20.000
12/12/31   13/04/24
C          1200123          3000.000          30.000
12/12/31   13/04/24
C          1200124          4000.000          40.000
12/12/31   13/04/24
C          1200125          5000.000          50.000
12/12/31   13/04/24
S          120012          10000.000          100.000
12/12/31   13/04/24
S          120012          20000.000          200.000
12/12/31   13/04/24
S          120012          30000.000          300.000
12/12/31   13/04/24
S          12001          40000.000          400.000
12/12/31   13/04/24
S          12001          50000.000          500.000
12/12/31   13/04/24
+-----+-----+-----+-----+-----+-----+-----+
+-----+
CALL SP_Accrue_Interest('C',"Messg");
*** Procedure has been executed.
*** Total elapsed time was 1 second.
Proc_Msg
-----
Processed          0 records
+-----+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 1 second.
AccountType AccountNumber OpeningBalance AccruedInterest
LastYearEnd LastAccrualDate
-----

```



```

-----
C      1200121      1000.000      10.000
12/12/31      13/04/24
C      1200122      2000.000      20.000
12/12/31      13/04/24
C      1200123      3000.000      30.000
12/12/31      13/04/24
C      1200124      4000.000      40.000
12/12/31      13/04/24
C      1200125      5000.000      50.000
12/12/31      13/04/24
S      1200121      10000.000     100.000
12/12/31      13/04/24
S      1200122      20000.000     200.000
12/12/31      13/04/24
S      1200123      30000.000     300.000
12/12/31      13/04/24
S      1200124      40000.000     400.000
12/12/31      13/04/24
S      1200125      50000.000     500.000
12/12/31      13/04/24
+-----+-----+-----+-----+-----+-----+-----+
+-----+
CALL SP_Accrue_Interest('S',"Messg");
*** Procedure has been executed.
*** Total elapsed time was 1 second.
Proc_Msg
-----
Processed          0 records
+-----+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 4 seconds.
AccountType  AccountNumber  OpeningBalance  AccruedInterest
LastYearEnd  LastAccrualDate
-----
C      1200121      1000.000      10.000
12/12/31      13/04/24
C      1200122      2000.000      20.000
12/12/31      13/04/24
C      1200123      3000.000      30.000
12/12/31      13/04/24
C      1200124      4000.000      40.000
12/12/31      13/04/24
C      1200125      5000.000      50.000
12/12/31      13/04/24
S      1200121      10000.000     100.000
12/12/31      13/04/24
S      1200122      20000.000     200.000
12/12/31      13/04/24
S      1200123      30000.000     300.000
12/12/31      13/04/24
S      1200124      40000.000     400.000
12/12/31      13/04/24
S      1200125      50000.000     500.000
12/12/31      13/04/24
+-----+-----+-----+-----+-----+-----+-----+
+-----+
.REMARK 'Populating savings table';
Populating savings table
+-----+-----+-----+-----+-----+-----+-----+
+-----+
.RUN FILE=../SavingsData.sql
+-----+-----+-----+-----+-----+-----+-----+
+-----+

```



```

DELETE FROM Savings ALL;
*** Delete completed. 10 rows removed.
*** Total elapsed time was 2 seconds.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200121',10000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200122',20000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200123',30000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200124',40000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('S','1200125',50000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200121', 1000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200122', 2000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200123', 3000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200124', 4000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
INSERT INTO Savings ('C','1200125', 5000.00,0,'2012-12-31','2013-01-01');
*** Insert completed. One row added.
*** Total elapsed time was 1 second.

+-----+-----+-----+-----+-----+-----+-----+
+-----+
*** Warning: EOF on INPUT stream.

```



```

+-----+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 1 second.
AccountType  AccountNumber  OpeningBalance  AccruedInterest
LastYearEnd  LastAccrualDate
-----
C             1200121         1000.000         .000
12/12/31      13/01/01
C             1200122         2000.000         .000
12/12/31      13/01/01
C             1200123         3000.000         .000
12/12/31      13/01/01
C             1200124         4000.000         .000
12/12/31      13/01/01
C             1200125         5000.000         .000
12/12/31      13/01/01
S             1200121        10000.000         .000
12/12/31      13/01/01
S             1200122        20000.000         .000
12/12/31      13/01/01
S             1200123        30000.000         .000
12/12/31      13/01/01
S             1200124        40000.000         .000
12/12/31      13/01/01
S             1200125        50000.000         .000
12/12/31      13/01/01
+-----+-----+-----+-----+-----+-----+-----+
+-----+
CALL SP_Accrue_Interest('A',"Messg");
*** Procedure has been executed.
*** Total elapsed time was 1 second.
Proc_Msg
-----
Processed          10 records
+-----+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 1 second.
AccountType  AccountNumber  OpeningBalance  AccruedInterest
LastYearEnd  LastAccrualDate
-----
C             1200121         1000.000         10.000
12/12/31      13/04/24
C             1200122         2000.000         20.000
12/12/31      13/04/24
C             1200123         3000.000         30.000
12/12/31      13/04/24
C             1200124         4000.000         40.000
12/12/31      13/04/24
C             1200125         5000.000         50.000
12/12/31      13/04/24
S             1200121        10000.000        100.000
12/12/31      13/04/24
S             1200122        20000.000        200.000
12/12/31      13/04/24
S             1200123        30000.000        300.000
12/12/31      13/04/24
S             1200124        40000.000        400.000
12/12/31      13/04/24
S             1200125        50000.000        500.000
12/12/31      13/04/24
+-----+-----+-----+-----+-----+-----+-----+

```



```

+-----+
CALL SP_Accrue_Interest('A',"Messg");
*** Procedure has been executed.
*** Total elapsed time was 1 second.
Proc_Msg
-----
Processed          0 records
+-----+-----+-----+-----+-----+-----+
+-----+
SELECT * FROM Savings ORDER BY 1,2;
*** Query completed. 10 rows found. 6 columns returned.
*** Total elapsed time was 1 second.
AccountType  AccountNumber  OpeningBalance  AccruedInterest
LastYearEnd  LastAccrualDate
-----
C            1200121          1000.000          10.000
12/12/31      13/04/24
C            1200122          2000.000          20.000
12/12/31      13/04/24
C            1200123          3000.000          30.000
12/12/31      13/04/24
C            1200124          4000.000          40.000
12/12/31      13/04/24
C            1200125          5000.000          50.000
12/12/31      13/04/24
S            1200121         10000.000         100.000
12/12/31      13/04/24
S            1200122         20000.000         200.000
12/12/31      13/04/24
S            1200123         30000.000         300.000
12/12/31      13/04/24
S            1200124         40000.000         400.000
12/12/31      13/04/24
S            1200125         50000.000         500.000
12/12/31      13/04/24
+-----+-----+-----+-----+-----+-----+
+-----+
.REMARK 'Finished testing internal UDF';
Finished testing internal UDF
+-----+-----+-----+-----+-----+-----+
+-----+
.LOGOFF
*** You are now logged off from the DBC.
+-----+-----+-----+-----+-----+-----+
+-----+
.EXIT 0
*** Exiting BTEQ...
*** RC (return code) = 0

```

Related Information

For more information about creating stored procedures, see *Teradata Vantage™ - SQL Stored Procedures and Embedded SQL*, B035-1148.

Creating User-defined Functions

Example: Creating a User-defined Function To Verify a Date

The following is an example of creating an external C language UDF.

SQL Definition

The following code in my_yyyymmdd_to_date2.sql creates the UDF.

```

---- UDF Build DDL/SQL--
REPLACE FUNCTION my_yyyymmdd_to_date2
(
    InputDate      VARCHAR(8) CHARACTER SET LATIN
)
RETURNS DATE
LANGUAGE C
SPECIFIC my_yyyymmdd_to_date2
NO SQL
DETERMINISTIC
PARAMETER STYLE SQL
CALLED ON NULL INPUT
EXTERNAL NAME 'CS!my_yyyymmdd_to_date2!./my_yyyymmdd_to_date2.c'
;

```

C Function Definition

The following user-defined function, my_yyyymmdd_to_date2.c, validates whether the argument character string in YYYYMMDD format is a valid date. It returns either a date or returns a NULL if the string is not a valid date.

```

/*
my_yyyymmdd_to_date2.c
Teradata User Defined Function (UDF)
Calling
-----
my_yyyymmdd_to_date2(date_str);
SELECT my_yyyymmdd_to_date2('20130423') AS ValidDate;
Parameters
-----
date_str
    Character string containing date to be validated
UDF Compilation
-----

```



```

REPLACE FUNCTION my_yyyymmdd_to_date2
(
    InputDate VARCHAR(8)
)
RETURNS DATE
LANGUAGE C
NO SQL
DETERMINISTIC
PARAMETER STYLE SQL
EXTERNAL NAME 'CS!my_yyyymmdd_to_date2!./my_yyyymmdd_to_date2.c'
;

*/
/*      Must define SQL_TEXT before including "sqltypes_td      */
#define SQL_TEXT Latin_Text
#include "sqltypes_td.h"
#include "stdio.h"
#include "string.h"
#define IsNull -1
#define IsNotNull 0
#define NoSqlError "00000"
#define YYYYMMDD_LENGTH 8
#define ERR_RC 99
void my_yyyymmdd_to_date2
(
    VARCHAR_LATIN      *InputDateString
    ,DATE               *result
    ,int                *inputDateStringIsNull
    ,int                *resultIsNull
    ,char               sqlstate[6]
    ,SQL_TEXT           extname[129]
    ,SQL_TEXT           specificname[129]
    ,SQL_TEXT           error_message[257]
)
{
    char input_integer[30];
    int  year_yyyy;
    int  month_mm;
    int  day_dd;
    char day_char[3];
    char month_char[3];
    char year_char[5];
    int  in_len,i;
    /* Return Nulls on Null Input */
    if ((*inputDateStringIsNull == IsNull))

```



```

{
    strcpy(sqlstate, "22018") ;
    strcpy((char *) error_message, "Null value not allowed.") ;
    *resultIsNull = IsNull;
    return;
}
in_len = strlen(InputDateString);
if ( in_len != YYYYMMDD_LENGTH )
{
    *result = ( 1 * 10000 ) + ( 12 * 100) + 1;
    *resultIsNull = IsNull;
    strcpy((char *) sqlstate, "01H01");
    strcpy((char *) error_message,
        "InputDateString is of wrong length, must be in YYYYMMDD format");
    return;
}
if ( in_len != YYYYMMDD_LENGTH )
{
    *result = ( 1 * 10000 ) + ( 12 * 100) + 2;
    return;
}
strcpy(input_integer , (char *) InputDateString);
for (i = 0; i<in_len; i++)
{
    if (input_integer[i] < '0' || input_integer[i] > '9')
    {
        *result = ( 1 * 10000 ) + ( 1 * 100) + 3;
        return;
    }
    else
    {
        input_integer[i] = tolower(input_integer[i]);
    }
}

sprintf(year_char,"%c%c%c%c",input_integer[0],input_integer[1],input_integer[2],
    input_integer[3]);
sprintf(month_char,"%c%c",input_integer[4],input_integer[5]);
sprintf(day_char,"%c%c",input_integer[6],input_integer[7]);
year_yyyy  = atoi(year_char);
month_mm   = atoi(month_char);
day_dd     = atoi(day_char);
/* Format output_date in internal Teradata format ((YEAR - 1900) * 10000 )
+

```



```

        (MONTH * 100) + DAY          */
        *result = (( year_yyyy - 1900 ) * 10000 ) + ( month_mm * 100) + day_dd;
    }

```

BTEQ Script To Call the User-defined Function

This BTEQ script calls the user-defined function and tests it by selecting from the my_yyyymmdd_to_date2.sql table:

```

.SET WIDTH          240
.logon TDSsystem/MyUserId,MyPass
.REMARK 'Building External UDF';
DATABASE Test_Database;
.RUN FILE=../my_yyyymmdd_to_date2.sql
.REMARK 'Testing External UDF';
SELECT Test_Database.my_yyyymmdd_to_date2('20130422')      AS ValidDate;
SELECT Test_Database.my_yyyymmdd_to_date2('201304')        AS ValidDate;
SELECT Test_Database.my_yyyymmdd_to_date2(NULL)            AS ValidDate;
.REMARK 'Finished testing External UDF';
.LOGOFF
.EXIT 0

```

Linux Command To Call the BTEQ Script

The following Linux commands call the BTEQ script:

```
bteq bld_xudfs.btq 2>&1 | tee bld_xudfs.btq.i1.out
```

Related Information

For more information about creating user-defined functions, see “UDF Code Examples” in *Teradata Vantage™ - SQL External Routine Programming*, B035-1147.

Working with Users, Roles, and Profiles: Operational DBAs

This section describes the strategies that an administrator can use to control user access. Specific topics include:

- Assessing user needs to develop a user management strategy
- Creating users and accounts
- Creating user profiles
- Using roles to manage privileges
- Granting privileges directly to users
- Creating Teradata Viewpoint users

Overview of Establishing Users

To establish users for the first time, perform all the tasks in this section completely and in the order presented.

1. Group potential database users by functional group and determine database access needs. See [Assessing Database User Needs](#).
2. Set up accounts to track resource usage and assign workload priorities. See [Creating User Accounts](#).
3. Set up profiles to define resources, account assignments, and password control parameters for groups of users. See [Creating User Profiles](#).
4. Create database users and assign profiles, space and accounts. See [Working with Database Users](#).
5. Create roles to define privileges for users groups and grant role membership to users. See [Using Roles to Manage User Privileges](#).
6. Grant additional privileges that are not suitable for role assignment directly to users. See [Granting Privileges Directly To Users](#).

User Access Methods

This document describes permanent database users who are created with the CREATE USER request, have PERM space on disk, and are authenticated by Vantage. For more information on granting privileges to users not managed in the database, for example directory users, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

Types of Users

The following list identifies five typical user types and job functions.

Functional Category	Description
General users	Database end-users who only need to read the data or to run pre-existing queries or macros to generate reports.
Update users	Privileged users who perform some general user functions and who also may need to insert, update or delete data, and create new database objects.
Batch users	High-level users who typically perform batch-level functions, for example: <ul style="list-style-type: none"> • Load, update, and export operations, including creation and deletion of staging tables. • Data backup, archive, and restore operations.
Database programmers	Users who design and create queries, macros, and stored procedures, as well as database objects, for use by the user community. Programmers may require administrator privileges within a development database, while needing only limited privileges in the main production database.
Assistant administrators	Administrative users who assist the principal administrator, user DBADMIN. Assistant administrative users may have most or all of the same privileges granted to user DBADMIN, but have a much smaller permanent space allocation because they have a limited ownership of objects.

Note:

You can create additional user types where necessary to identify functional differences, however, the examples provided in this document are based on these user types.

Assessing Database User Needs

Assess database user needs and develop a user management strategy before you begin executing user management setup tasks.

1. Create a list of all users who require access to Vantage and identify each one according to user type. Minimize the number of user types to simplify user management.
2. Examine user space requirements:
 - Users who create or own databases, tables and other space-consuming objects require permanent storage space (*perm space*).
 - Users who execute SQL queries, macros, or stored procedures require spool space to contain the required temporary database structures. The spool specification in a profile limits the amount of available space a profile member can use.
3. Define requirements for one or more account strings. Each profile can include one or more accounts. Each account can specify:
 - Request priority. This classification is in effect only when TASM or TIWM classification processes cannot match the request to a user-defined workload.
 - A four-character account identifier based on department, group, or function.

- A date and time stamp
 - Define the user default database, that is, the database where the user most often works.
 - Define password control parameters. Refer to the sections on password controls and managing passwords in *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.
Consider your site security policy and decide whether or not all users can share global password parameters, or if you need to set them by user group, in profiles.
4. Review the database objects (such as views, tables, macros, functions, and procedures) that users or user groups must access to do their job. Always define database privileges at the highest level that is appropriate for a particular user. For example, if a user requires the same privileges on all views in a database, assign privileges at the database level.
 5. Identify groups of users with common database privilege requirements and create roles to define the privileges for each group. Consolidate minor differences in access requirement where possible to minimize the number of roles.
 6. You may need to assign specialized privileges directly to individual users.

Best Practices for Creating Users

While there is no single set of procedures that would best fit all the varieties of system configurations possible and meet all site requirements, consider the following suggestions for best practices in creating users.

- Create *separate* users for the security administrator and database administrator.

Establish a security administrator user to perform security-related tasks. The biggest threat to security is usually the misuse of information or privileges by authorized users. No one single user should have all the privileges for everything. Neither should an administrative user have access to something for which he does not need access.

- Ensure that all users are uniquely identified. This means not allowing several users to log in to Vantage using the same username.

Setting up users to be unique enables you to effectively monitor user activities and helps you identify the source of a security breach if there is one. By disallowing users to use a generic or shared username, each user is held accountable for his specific actions. In addition, unique users can be allowed to view or not view certain information that is protected by row-level security constraints. For more information, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

- Consider the function of the user. Create administrative users under separate users/databases so that privileges can be granted from the owning user/database. For example, the HR database and Marketing database can have separate administrative users to manage privileges for their respective users.
- For non-administrative users, if possible, assign the user to a role with the required privileges rather than granting privileges to the user directly. It is easier to use roles to manage privileges. Profiles should also be created for non-administrative users (see [Creating User Profiles](#)).
- Limit the permanent and spool space of users and grant additional space if it becomes necessary. Limit spool space using a profile allows you to protect the system from runaway queries.

Creating User Accounts

Each user logon to the database should specify an account string.

Accounts are used by the database to:

- Prioritize requests. This classification is in effect only when TASM or TIWM classification processes cannot match a request to a user-defined workload.
- Monitor resource usage
- Charge for space or resources used

Teradata recommends that you specify at least one account for each user. Users with similar responsibilities and resource requirements should be assigned to the same account(s) to reduce the overall number of accounts you must maintain.

You can specify accounts in the following:

- A CREATE PROFILE or MODIFY PROFILE statement for the profile specified in the user definition (preferred).
- A CREATE USER or MODIFY USER statement.

Note:

If a user is a member of a profile that specifies one or more accounts, the profile account assignments supersede and invalidate any accounts specified in the user definition.

Related Information

Topic	Resources for Further Information
Specifying the account string at logon	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Creating Account Strings

Account strings are an optional method of controlling the granularity of resource accumulations reported in the DBC.Acctg table. Account strings allow the collection of CPU and I/O that is reported in DBC.Acctg to be grouped by application, time of day, and priority. Variable substitution parameters included in the account string will be resolved at execution time.

A secondary use of account strings is to direct the classification of a request to a default Timeshare workload in TASM or TIWM. This classification will be effective only in cases where normal classification processes do not match the request to a user-defined workload.

If you use account strings, Teradata recommends that you devise an account system and then identify the accounts required for various groups in the user community before attempting to specify account information in user profiles. An account string is limited to 128 characters.

Account strings should follow this format:

```
'$W00MSIR&D&H'
```

Note:

If a CREATE or MODIFY statement specifies more than one account string, the individual strings must be separated by commas and the entire list of strings enclosed by parentheses.

where:

Element	Description
Account String Variables	
\$W00	<p>Optional. For compatibility with previous releases of Analytics Database, you can classify a request to a Timeshare workload. If a request can be classified into a TASM workload and also has \$W00 in the account string, the TASM workload classification takes precedence. If there are no workloads defined or if the request does not fit into an existing workload, TASM uses these characters to determine which Timeshare tier priority level applies to the workload.</p> <p>Specify one of the following instead of \$W00:</p> <ul style="list-style-type: none"> • \$R00 = Timeshare Top • \$H00 = Timeshare High • \$M00 = Timeshare Medium (default) • \$L00 = Timeshare Low <p>The default is \$M00.</p> <p>Note:</p> <p>This default classification based on the starting characters in the account string will occur only if the request fails to classify to any other defined workload. This default classification exists to be backward-compatible with older versions of software and to help users transition easily to SLES priority management. Teradata recommends that users of current software rely upon normal TASM/TIWM workload classification rather than this default approach.</p>
MSI	<p>The application/workload name.</p> <p>Recommendation: You can use the 3 characters of the application/workload name to differentiate accounts and to provide an abbreviated description of the account function. For example, a department, such as finance (FIN) or marketing (MKT).</p>
R	<p>A one-character workload identifier. For example:</p> <ul style="list-style-type: none"> • Reporting • Batch • Tactical • Online
<p>Account String Expansion (ASE) Variables</p> <p>Analytics Database tags the session with information specified by the ASE variables for later use in management of database sessions. Each ASE variable <i>expands</i> to the number of characters needed to express the related information. The string limit is enforced against the expanded form.</p> <p>Specification of the the following ASE variables is optional, but recommended. Other variables are available. For more information, see Logging Resource Usage Data with Account String Variables.</p>	

Element	Description
<i>&D</i>	The ampersand (&) designates an account string expansion (ASE) variable. Analytics Database automatically replaces these variables with actual values upon opening the session. You can use either or both of the following: <ul style="list-style-type: none"> • <i>&D</i> = the date • <i>&H</i> = the time Teradata no longer recommends using the <i>&S</i> (session) variable.
<i>&H</i>	

Determining the Account for a User Session

The system determines the account for a user when the user logs on as follows:

- If the user is assigned only one account, the logon uses that account by default.
- If the user is assigned multiple accounts and the default account (the first listed in the operant user or profile definition) is appropriate for the session, the logon need not contain the account string.
- If the user has multiple accounts and does not want to use the default, the logon string can specify another assigned account.
- During a session, the user can submit a SET SESSION ACCOUNT statement to change the session to another assigned account.
- If the user has no account assignment, the session defaults to the first account listed for the owning user or database. Such a use of default owner accounts is not recommended.

Creating User Profiles

Profiles

You can create profiles to define resource parameters for groups of users with similar needs instead of defining them for each individual user. Then list the profile in a CREATE or MODIFY USER statement to establish membership in a profile for each user.

Profiles can be based on the primary user types such as those shown in [Types of Users](#), or other factors that group users according to database resource requirements.

Use profiles to specify parameters such as:

- Temporary and spool space allocations
- Accounts
- Password control parameters
- Default database assignment
- Query band

Note:

The value for a profile attribute supersedes and replaces any corresponding values specified in global defaults, or in a CREATE USER request.

Creating Profiles

1. Open your favorite client software to construct your SQL request. For example:

```
CREATE PROFILE profile_name AS
  ACCOUNT = ('account_str1','account_str2')
  DEFAULT DATABASE = database_name,
  SPOOL = spool_space,
  PASSWORD =
    (EXPIRE = n_days,
     MAXLOGONATTEMPTS = attempts,
     LOCKEDUSEREXPIRE = n_minutes),
  QUERY_BAND = 'Pair_name=pair_value;' [NOT] DEFAULT';
```

2. Include the following parameters:

Parameter	Description
Name	The unique name for the profile. Include such things as the user type or user department to identify its function. For example, GenUser_Profile or Finance_Profile. Consolidate user requirements where possible to avoid the unnecessary proliferation of profiles.
Account	<p>The database uses accounts for user job tracking and to prioritize requests.</p> <p>Recommendation: Enter one or more account strings in accordance with the format shown in Creating User Accounts.</p> <p>If you assign more than one account string to a profile, enclose each string in apostrophes, separate the strings with commas, and enclose the complete specification in parentheses, for example:</p> <pre>('\$M00FINB&D&H', '\$L00accO&D&H', '\$R00MKTR&D&H')</pre> <p>Note:</p> <p>The content of an account string is limited to 128 characters.</p>
Default Database	<p>Optional. The database the user is most likely to access.</p> <p>If no name is specified, the system uses the name specified in the user definition (CREATE USER statement).</p> <p>Recommendation: Specify the default database at the user level, rather than in the profile, to allow for differences in user default database requirements.</p>
Spool	This value limits the amount of space available for intermediate query results or formatted answer sets to queries and volatile tables. Spool space for a user is shared among the queries that user is executing concurrently. The system borrows spool space for a user from unused permanent space in the system.

Parameter	Description
	<p>Recommendation: Spool space requirements in a profile depend on the activities of member users. Begin by specifying spool space in bytes according to the following percentages of the total perm space allocation for the Spool_Reserve database:</p> <ul style="list-style-type: none"> • General users: 5% • Update users: 10% • Batch users: 10% • Assistant administrative users: 10% • Database programmers: 10% <p>To specify spool space, you can also use an expression that returns a numeric value, for example:</p> <pre>CREATE PROFILE Bennie AS SPOOL = '2e6' BYTES;</pre> <p>Periodically monitor usage patterns and modify spool space assignments when required using the MODIFY PROFILE statement.</p> <p>Spool space can also be specified as part of a CREATE USER statement or modified as part of a MODIFY USER statement.</p> <p>Note:</p> <p>Profile members that do not require permanent space must still be allocated a small amount of spool space in the profile, for example, 1 MB.</p>
Temporary	<p>This is required only when using global temporary tables.</p> <p>Recommendation: Initially omit a temporary space specification unless you have a predetermined temporary space requirement for this group of users. If necessary, you can add a temporary space specification later with a MODIFY PROFILE statement.</p>
Password	<p>Optional. The password control parameters allow you to apply different settings in the profile than are in the global default settings created in DBC.SysSecDefaults, as part of step 2 in Setting Up the Database Administrator User.</p> <p>Recommendation: Do not reset any password control parameter values in a profile unless the profile members cannot use the system defaults.</p>
Cost Profile	<p>Optional. An Optimizer cost profile associated with the profile.</p> <p>Note:</p> <p>Optimizer cost profiles are not intended for use on production systems. The Cost Profile parameter is for use only under the direction of Teradata Support Center personnel.</p>
Constraint	<p>Optional. Name of one or more row-level security constraints, each followed by a list of the hierarchical levels or non-hierarchical categories, valid for the constraint, which are being assigned to the profile_name.</p>
Query_Band [NOT] DEFAULT	<p>Optional. Creates a query band that is automatically set for the session at logon. The DEFAULT option allows users to set different values for a session or transaction query band than the values in the profile query band. The NOT DEFAULT option prevents a user from changing the query band values for a session or transaction. For more details, see Defining Name-Value Pairs for Profile Query Bands.</p>

Parameter	Description
Ignore Query_Band Values	Optional. Defines the values that are discarded if they appear in a SET QUERY_BAND request. This option prevents a user with this profile from setting a query band with certain values. For more details, see Defining Name-Value Pairs for Profile Query Bands .

- Specify profile membership by doing one of the following:
 - Assign a profile to each user as part of [Working with Database Users](#).
 - After creating users, use a MODIFY PROFILE statement to include member users.
- Close the client software or go on to [Working with Database Users](#).

Related Information

Topic	Resources for Further Information
Syntax and options for the CREATE PROFILE statement	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
Information on monitoring and reallocating space	Managing Space: Operational DBAs
Global temporary tables	<i>Teradata Vantage™ - Database Design</i> , B035-1094
Password control options	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Dropping Profiles

DROP statements are recorded in the Data Dictionary. However, the result of a DROP PROFILE, DROP DATABASE, or DROP ROLE statement is not cascaded to the user rows in DBC.Dbase, so the corresponding default setting for each affected user is not reset to NULL. When an affected user next logs on, the system does not return an error or warning.

If you drop a default profile, the system uses the following by default:

- ACCOUNT, SPOOL, TEMP, and DEFAULT DATABASE specifications in the CREATE USER or latest MODIFY USER statement
- Password attributes defined at the system level in DBC.SecurityDefaultsV (see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100)

Note:

If you re-create a profile with the same name after it was dropped, users defined for that profilename are assigned the profile parameters at the next logon. The effect of a profile re-creation is not immediate.

Working with Database Users

Types of Users

This task creates individual database users, including lower level administrators, database programmers and other privileged users, and database end-users.

Creating Users

1. From the client program of your choice, log on to Vantage as user DBADMIN.

Note:

This procedure creates users individually. To automate the creation of users by using BTEQ scripts, see [Using BTEQ Scripts to Create Database Objects](#).

2. Determine values for the following objects:

Field or Control	Explanation
User Name	<p>Required. The name of an individual Vantage user.</p> <p>Each database user should be uniquely identified for accurate logging. Do not allow users to share a username for accessing the database.</p> <p>Note:</p> <p>The username must conform to the rules for SQL object names.</p>
Owner	<p>Required. The name of the database or user that owns the space in which the user is created.</p> <p>Recommendation: DBADMIN. All users should be created in space owned by the database administrator.</p>
Password	<p>Required. A <i>temporary</i> password. The user is prompted to change the temporary password to a permanent, private password at first logon.</p>
Permanent Space	<p>May be required. Space in bytes that can be used to contain objects the user creates or owns. Permanent space is required for those users that create and store space-consuming objects.</p> <p>Recommendation: Base the permanent space specification on user needs.</p> <ul style="list-style-type: none"> • General users: Not required. • Update users: Required if users create and store objects. • Batch users: Required if users create and store objects. • Assistant administrators: Required if users create and store objects. • Database programmers: Required to create and store objects.
Spool Space	<p>Optional. Specifies the <i>limit</i> to the amount of space in bytes available for temporary process constructs such as intermediate query results or formatted answer sets to queries and volatile tables. Spool space for a user is shared among the queries that user is executing concurrently.</p>

Field or Control	Explanation
	<p>The system borrows user spool space from any unused permanent space found on the system.</p> <p>The spool space specification in a profile takes precedence over spool space specified in a CREATE USER statement.</p> <p>Recommendation: Specify spool space as part of a profile unless the user requires special spool considerations.</p> <p>If you do enter a value for an individual user, specify approximately 20% of the perm space available to the largest database the user will query.</p>
Temporary Space	<p>Not applicable. Required only when using global temporary tables.</p> <p>Specifying temporary space requires that you develop a strategy for use of the advanced features that need such space. If necessary, you can add a temporary space specification later with a MODIFY USER or MODIFY PROFILE statement.</p>
Account	<p>Optional. You can set up account information to form the basis of various user management options.</p> <p>Recommendation: Do not specify accounts at the user level, unless the user requires unique system priorities. Instead name a profile in the Profile parameter that contains the needed account(s). For information, see Creating User Profiles.</p> <p>Note:</p> <p>If a user is a member of a profile that specifies one or more accounts, the profile account assignment supersedes and invalidates all account assignments specified in the user definition.</p>
Default Database	<p>Optional. Identifies the database that owns the space in which Analytics Database stores or searches for new or target objects in an SQL query, unless a different database is specified in the query.</p> <p>Note:</p> <p>Users can use the DATABASE statement to establish a new default database for the current session.</p> <p>Recommendation: Specify the database most often accessed by the user.</p> <ul style="list-style-type: none"> • General users: Specify a read-only Views database. If restrictions are necessary, use role privileges to allow access to particular views. • Update users: Specify either a read-only Views or an updatable Views database, depending on where the user normally works. • Batch users: Specify the <i>Tables_Database</i>, where batch operations take place. • Assistant administrators: Specify DBADMIN. • Database programmers: Specify the name of the development database.
Profile Name	<p>Optional. Gives the user membership in the named profile.</p> <p>For syntax elements that appear in both the CREATE USER statement for a user, and in a profile in which the user is a member, the profile values take precedence.</p> <p>A user can be a member of only one profile.</p> <p>Recommendation: Specify profile membership for all users. Select from the drop-down list and highlight the needed profile.</p>

Field or Control	Explanation
Default Role	<p>Identifies the default role used to determine user privileges in the database for the current session. This is usually the role associated with the activities performed most frequently by the user.</p> <p>Note:</p> <p>The user can employ the SET ROLE statement at the beginning of a session or transaction to change from the default role to another role. The SET ROLE ALL statement allows access to all roles of which the user is a member.</p> <p>Recommendation: You may specify a default role later after you have created the user and the roles. Note that you must specify a default role even if the user is a member of only one role.</p>
Default Journal	<p>Optional.</p> <p>Recommendation: Do not use for initial database implementation.</p>
Startup String	
Before Journal	
After Journal	
FALLBACK	<p>Optional. Specification of FALLBACK automatically creates a duplicate of each table stored in the user space, in addition to the duplicates already created by disk mirroring. The system reverts to the FALLBACK tables in the event of a failure.</p> <p>Note:</p> <p>You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.</p>
Collation	<p>Optional. Database default values that apply to all users are already set as part of system installation and configuration.</p> <p>Recommendation: Do not change the default values for a user unless the default value is not applicable.</p> <p>Note:</p> <p>All users must use the same collation value.</p>
Time Zone	
Default Date Form	
Default Character Set	

Run a CREATE USER statement, similar to the following:

```
CREATE USER "GenUser1" FROM "DBADMIN"
AS PERM = 0
PASSWORD = "temp123"
STARTUP = ''
DEFAULT DATABASE = "Personnel_Views"
NO BEFORE JOURNAL
NO AFTER JOURNAL
PROFILE="GenUser_Profile";
```


Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
3	Syntax and options for the CREATE USER statement	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
	The creation and use of accounts	Creating User Accounts

Creating Temporary Passwords for First Login

Vantage requires you to associate a username with a password when you first create a user. You must specify a password, even if temporary. Otherwise the Parser rejects the statement as incomplete.

The security administrator can prompt users for a new password upon first login if:

- At the time of installing a new system, the administrator has created generic passwords for all users and wants them to choose their own passwords.
- New password requirements must be enforced. For example, if the security administrator wants to require mixed case and wants all users to change their passwords.

Note:

Teradata password rules do not apply to external authentication. For more information, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

To use SQL, see “Passwords” under “CREATE USER” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Using Roles to Manage User Privileges

Do the following to set up roles to manage user privileges:

1. Create roles, as shown in [Creating User Roles](#).
2. Grant privileges to each role, as shown in [Granting Privileges to a Role](#).
3. Grant role membership to users, as shown in [Granting User Membership in a Role](#).

Note:

Some privileges cannot be granted to a role. You must grant those privileges directly to a user. For information, see [Granting Privileges Directly To Users](#).

User Types and Minimum Required Privileges

The following table describes the minimal privileges required by the three basic user types.

User Type	Privilege Requirements
General	<p>Database end-users who typically read data and execute macros in a read-only Views database.</p> <p>Assign the following privileges on a read-only Views database to all General users:</p> <ul style="list-style-type: none"> • EXECUTE • SELECT <p>The read-only Views database must have privileges on the <i>Tables_Database</i>, as shown in Working with Table Access Privileges for Views.</p>
Update	<p>Privileged database users who update data generally require the following privileges:</p> <p>Privileges required on a read-only Views database:</p> <ul style="list-style-type: none"> • EXECUTE • SELECT • CREATE VIEW and DROP VIEW • CREATE MACRO and DROP MACRO <p>Privileges required on an updatable Views database:</p> <ul style="list-style-type: none"> • EXECUTE • SELECT • INSERT, UPDATE, and DELETE • EXECUTE PROCEDURE • EXECUTE FUNCTION • CREATE VIEW and DROP VIEW • CREATE MACRO and DROP MACRO <p>Additional recommended privileges on an updatable Views database for database programmers:</p> <ul style="list-style-type: none"> • SHOW • ALTER, CREATE, and DROP PROCEDURE • ALTER, CREATE, and DROP FUNCTION • ALTER and CREATE EXTERNAL PROCEDURE • CREATE TRIGGER and DROP TRIGGER <p>To provide update capability on views, you must grant the updatable Views database SELECT, INSERT, UPDATE, DELETE WITH GRANT OPTION privileges on the referenced tables in the <i>Tables_Database</i>, as shown in Working with Table Access Privileges for Views.</p>
Batch	<p>Data movers, who typically perform batch-level functions, such as:</p> <ul style="list-style-type: none"> • Batch data load, update, and export operations. • Data backup, archive, and restore operations, including creating and deleting staging tables. <p>Batch users must have the following privileges on the <i>Tables_Database</i>:</p> <ul style="list-style-type: none"> • SELECT • INSERT, UPDATE, DELETE • DUMP and RESTORE • CHECKPOINT • CREATE TABLE and DROP TABLE
Assistant Administrators	<p>If you need one or more assistant administrators to help share administrative duties, they may require the same privileges as those granted to DBADMIN, or a subset of those</p>

User Type	Privilege Requirements
	privileges. Use the procedure shown in Setting Up the Database Administrator User to create assistant administrators and grant the privileges they need.
Database Programmers	Database programmers may require administrator-level privileges within the development database to allow them to create and test database objects. Administrators can then deploy the objects to the production database.

Types of Privileges

You can explicitly grant database privileges to users, roles, or databases. Users also gain other privileges without a formal grant. Before you decide which privileges to explicitly grant, make sure you understand the privileges users receive by other means.

Privilege	Description
Implicit Privileges	
Ownership	<p>Users who own perm space have certain implicit (ownership) privileges on any object contained in the space they own, even if they did not create the object.</p> <p>Note: You can transfer ownership using the GIVE statement.</p>
Explicit privileges	
Automatic	<p>When a user creates a database object, Analytics Database automatically grants privileges to:</p> <ul style="list-style-type: none"> • The creator of the object • A newly created user or database
GRANT	<p>You can grant privileges:</p> <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 external role to one or more groups of directory users
Inherited	<p>Privileges that a user acquires indirectly.</p> <ul style="list-style-type: none"> • All users inherit the privileges of the system-generated user, PUBLIC, a role-like collection of privileges available by default, whether or not they have any other privileges. You can grant additional privileges to PUBLIC. • A user inherits all the privileges granted to each role of which the user is a member. • Directory users inherit the privileges of the database users and external roles to which they are mapped.

Note:

The system logs automatic and explicit privileges in exactly the same way, and are indistinguishable as to how they were acquired. All privileges except implicit privileges are stored in the data dictionary in the DBC.AccessRights table, by each user.

For a detailed discussion of privileges, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

Privileges That Must Be Explicitly Granted

Some privileges cannot be acquired by indirect means and *must* be explicitly granted to a user, database, or role. For information, see “Privileges that Must Be Explicitly Granted” in *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

Creating User Roles

1. From the client of your choice, log on as user DBADMIN.
2. Create a user role, for example:

```
CREATE ROLE  role_name;
```

Granting Privileges to a Role

Use the GRANT statement to grant privileges to the roles necessary to serve the basic user types. A GRANT statement takes the following basic form:

```
GRANT  privilege
ON  database_object_name
TO  role_name;
```

1. From the client program of your choice, log on to Vantage as user DBADMIN.
2. Grant privileges to each role as follows:

Fields and Controls	Explanation
Database Name	<p>Required.</p> <ul style="list-style-type: none"> • To grant privileges on an entire database, specify the name of the database. • To grant privileges on an object, specify the parent database that contains the object. <p>Recommendation: When specifying privileges for:</p> <ul style="list-style-type: none"> • General users, select a read-only Views database. • Update users, select a read-only Views database or an updatable Views database. • Batch users, select the <i>Tables_Database</i>. • Administrative users, select DBADMIN. • Database programmers, select a Development database.

Fields and Controls	Explanation
Object Type	<p>Required.</p> <p>Specify the type of database object on which privileges are being granted.</p> <p>For example:</p> <ul style="list-style-type: none"> To grant privileges on the Tables_Database, specify Database. To grant privileges on the tables located in the Tables_Database, specify Table. <p>Recommendation: Always grant privileges at the highest level that is appropriate, while observing security concerns and user “need to know.”</p>
Objects	<p>Required when objects are displayed.</p> <p>If the Object Type is Database, this field is blank and not required. Otherwise, the Objects field displays all of the database objects of the object type contained in the named database.</p> <p>For example, if Database Name is Tables_Database and Object Type is Table, this field displays all of the tables contained in the Tables_Database.</p> <p>Select the specific database object on which privileges are being granted.</p>
To/From User	Not applicable for granting privileges to roles.
Role	<p>Required. The role to which the specified privileges are granted.</p> <p>Select the name of a previously created role from the displayed list.</p>
Public	<p>Optional.</p> <p>Checking the Public box grants the specified privileges to PUBLIC, a group of rights accessible by any user with logon privileges, even if no other privileges have been granted.</p> <p>Recommendation: Do not grant additional privileges to PUBLIC at this time.</p>
Normal	<p>Required. Specify the privileges by checking the boxes, based on the guidelines in User Types and Minimum Required Privileges.</p> <p>Recommendation: Consider the privileges that the user may get without formal granting, as shown in Types of Privileges, before deciding on what privileges to grant.</p>
Create	
Drop	
All	<p>Optional. Do not use for roles.</p> <p>Grants all privileges that apply on the database object. This box is usually used only for the master administrator, DBADMIN.</p>
Dictionary	Optional. Grants administrative privileges on data dictionary tables.
Access	<p>Optional.</p> <p>Provides basic SELECT and EXECUTE privileges on the specified object.</p>
All But	<p>Optional.</p> <p>When the All But box is checked, role members receive all privileges except those checked in the Normal, Create, and Drop sections.</p>

Not all privileges need to be granted using roles. It may be more efficient to grant some specialized privileges directly to a user, rather than creating a role with a membership of one.

Note:

Any user can grant privileges *on* any database object it owns *to* any other user or role, excluding the CREATE and DROP privileges, which are not ownership privileges.

For example:

```
GRANT EXECUTE, SELECT, INSERT, UPDATE, DELETE, DUMP, RESTORE, CHECKPOINT,
CREATE TABLE, DROP TABLE ON "Tables_Database" TO "Role_BatchUser_2";
```

3. Repeat this procedure to grant privileges to all roles.

Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
3	Syntax and options for the GRANT (SQL Form) statement	<i>Teradata Vantage™ - SQL Data Control Language</i> , B035-1149
	Using roles to manage user access	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Granting User Membership in a Role

After creating database users, grant them membership in one or more roles to provide the required database access privileges.

1. From the client program of your choice, log on to Vantage as user DBADMIN.
2. Run a GRANT request similar to the following:

```
Grant "Role_1" To "GenUser1";
```

3. Repeat this procedure for each role to which a user requires membership.

Related Information

Reference topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
1	Syntax and options for the CREATE ROLE statement	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
	Using roles to manage user access	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Step	Topic	Resources for Further Information
2	Syntax and options for the GRANT (SQL form) statement	<i>Teradata Vantage™ - SQL Data Control Language</i> , B035-1149
	Database privileges, including those conferred by object ownership	
3	Syntax and options for GRANT (Role form)	

Defining the Default Role for a User

To ensure that role privileges are available to users at log on, you must define a default role for each user that is a member of one or more roles. If you do not specify a default role, the user must employ the SET ROLE statement to select the operant role. For example:

```
SET ROLE  role_name;
```

or

```
SET ROLE ALL;
```

You can also use the SET ROLE statement to change from one role to another role within a session.

To ensure that the user is operating with the appropriate role, specify the default role for a user as follows

1. From the client program of your choice, log on to Vantage as user DBADMIN.
2. Modify the user, for example:

```
MODIFY USER "GenUser1" AS
STARTUP = ''
DEFAULT DATABASE = "Personnel_Views"
NO BEFORE JOURNAL
NO AFTER JOURNAL
COLLATION = HOST
DEFAULT CHARACTER SET LATIN
DATEFORM=INTEGERDATE
TIME ZONE=NULL
DEFAULT ROLE="Role_GenUser_2";
```

Related Information

Related topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
1	Syntax and options for the SET ROLE statement	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
7	Syntax and options for the MODIFY USER statement, including the default role	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
	Using roles to manage user access	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Granting Privileges Directly To Users

Some privileges cannot or should not be granted to roles, but should instead be granted directly to users. Examples include these privileges:

- System-level privileges. These are for administrators only and should not be granted to other users.
- Object-level privileges that apply to too few users to require creation of a role.

Note:

A user automatically has the privilege to grant most privileges *on* any database object the user owns to any other user or role. This excludes CREATE and DROP privileges, which are not automatic ownership privileges. The CREATE and DROP privileges must be granted directly to a user, and must include the WITH GRANT OPTION clause, before that user can grant these privileges to others. For information on ownership privileges, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

Prerequisites

You can only grant privileges to users that already exist in the database. For information on creating users, see [Working with Database Users](#).

The Sys_Calendar database and Calendar view must exist. They are normally created as part of system installation by running the DIPALL script.

Example: Granting Privileges on Sys_Calendar to All Users

Sys_Calendar is a system database used to contain the Sys_Calendar.CalDates table and Sys_Calendar.Calendar view. The Calendar view is a user-accessible tool for date arithmetic. It permits easy specification of arithmetic expressions and aggregation, and is particularly useful when requesting values aggregated by weeks, months, year-to-date, years, and so on. You can use the Calendar view to get attributes for any date between the years 1900 and 2100, such as which day of the week or which week of the month a date occurs.

Use the following procedure to grant SELECT privilege on Sys_Calendar to all Vantage users.

1. Start the client program of your choice and log on to Vantage as DBADMIN.
2. Execute the following request:

```
GRANT SELECT ON Sys_Calendar TO PUBLIC;
```

Related Information

Related topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
All	Sys_Calendar and the Calendar view	<i>Teradata Vantage™ - Data Dictionary</i> , B035-1092

Granting Privileges to a User

1. From the client program of your choice, log on to Vantage as DBADMIN.
2. Run a GRANT request similar to the following:

```
GRANT privilege
ON database_object_name
TO username;
```

privilege

Privilege to grant to the user.

database_object_name

Database object on which to grant privilege to user.

Based on the values of the Database Name, Object Type, and Objects fields.

username

User to whom to grant privilege on database object.

Based on the user selected from the To/From User tab.

Automatic Privileges for Creators

When you create a user or database, the system automatically grants you privileges on the created database or user:

- ANY
- CHECKPOINT
- CREATE AUTHORIZATION
- CREATE DATABASE

- CREATE MACRO
- CREATE TABLE
- CREATE TRIGGER
- CREATE USER
- CREATE VIEW
- DELETE
- DROP AUTHORIZATION
- DROP DATABASE
- DROP FUNCTION
- DROP MACRO
- DROP PROCEDURE
- DROP TABLE
- DROP TRIGGER
- DROP USER
- DROP VIEW
- DUMP
- EXECUTE
- INSERT
- SELECT
- STATISTICS (also for creators of tables)
- RESTORE
- UPDATE

Note:

These automatic privileges are not granted to the creators of zones.

Automatic Privileges for Created Users and Databases

When you create a user or database, it automatically gets the following privileges on itself:

- ANY
- CHECKPOINT
- CREATE AUTHORIZATION
- CREATE MACRO
- CREATE TABLE
- CREATE TRIGGER
- CREATE VIEW
- DELETE
- DROP AUTHORIZATION

- DROP FUNCTION
- DROP MACRO
- DROP PROCEDURE
- DROP TABLE
- DROP TRIGGER
- DROP VIEW
- DUMP
- EXECUTE
- INSERT
- SELECT
- STATISTICS
- RESTORE
- UPDATE

Note:

These same privileges are automatically given to creators, with the exception of CREATE USER, CREATE DATABASE, DROP USER, and DROP DATABASE.

Related Information

Related topics are arranged according to the first step in which the topic appears.

Step	Topic	Resources for Further Information
All	Syntax and options for the GRANT (SQL form) statement	<i>Teradata Vantage™ - SQL Data Control Language</i> , B035-1149
	Database privileges, including those conferred by object ownership	<i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100

Viewpoint Monitoring Privileges

All database users can benefit from using Teradata Viewpoint to monitor the database, but not all users need all Teradata Viewpoint capabilities.

You can use the examples in the following table to help determine the database monitoring functions needed by various user types, and in creating and assigning privileges to Teradata Viewpoint roles.

Role	Privileges
Default Teradata Viewpoint roles	
Administrator	Assign this role only to the Teradata Viewpoint administrator(s).

Role	Privileges
	Has all privileges (default). This role can access the functions in the Admin tool menu to configure Teradata Viewpoint alerts, users, roles, and systems.
User	Includes minimal privileges
TD_SE (Teradata Solution Engineer)	This role is intended for the Teradata Solution Engineer when they need access to system health and metrics. You can enable or disable this role, but do not reconfigure it.
TD_TSS (Teradata Technical Support Specialist)	This role is intended for Teradata Technical Support Specialists when they need access to your system SQL. You enable or disable this role, but do not reconfigure it.
Example custom roles Note: Roles are based on the user types/duties shown in Types of Users .	
Database Administrator	Database administrators may require all Teradata Viewpoint monitoring functions to perform administrative duties in the database. Suggested role privileges: Create a Database Administrator role that has privileges to use all portlets and portlet functions, except System Administration, which should only be accessible to the Teradata Viewpoint administrator(s). Note: You may find it useful to create an additional role that limits access to some portlets and functions for lower level administrators.
Database Programmer/ Power User	Database programmers perform functions similar to administrators, creating database objects, running ad hoc queries, and monitoring performance. Suggested role privileges: Similar to Database Administrator, but without analytical functions. Note: Programmer privileges in the database are often restricted to a development database, but Teradata Viewpoint monitors all databases within the system.
Batch User	Batch users run load and export utilities or perform backup, archive, and restore operations. Suggested role privileges include: <ul style="list-style-type: none"> • For job planning: Capacity Heat Map, Space Usage, and Calendar. • During job execution and troubleshooting: System Health, My Queries or Query Monitor, Workload Monitor, SQL Scratch Pad, and Console Utilities. • Privilege to set preferences and share portlets.
Update User	Users who perform miscellaneous insert/update/delete operations. Suggested role privileges: <ul style="list-style-type: none"> • Calendar (no create or edit privileges) • System Health (View Summary only) • My Queries • Workload Monitor

Role	Privileges
	<ul style="list-style-type: none"> SQL Scratchpad
General User	<p>Low level users that run macros and stored procedures to access data, which select data and generate reports.</p> <p>Suggested role privileges:</p> <ul style="list-style-type: none"> Calendar (no create or edit privileges) System Health (View Summary only) My Queries

Setting Up Teradata Viewpoint Users

For users that are not already set up to access Teradata Viewpoint, as part of initial system installation, you must perform Teradata Viewpoint set up operations.

1. Log on as the Teradata Viewpoint administrator and select **Admin**, which includes a drop down list of administrative functions.
2. Use the **Roles Manager** to create Teradata Viewpoint roles and define sets of user privileges for each role.

When creating roles:

- Define the portlets available to the role.
- Define whether user members can set their own preferences and share portlets.
- Give or remove user permissions for portlet tools and functions.

Teradata Viewpoint provides two default roles, but you may find it useful to configure custom roles, based on the suggestions in [Viewpoint Monitoring Privileges](#).

3. Use the **Users Manager** to create Teradata Viewpoint users and assign membership in one or more roles according to user job function. Include an email address for each user, so that Teradata Viewpoint can send them alert emails, if necessary.
4. Instruct users to setup their Teradata Viewpoint profile by specifying their username, password, and account, so the system can verify that each user exists in the specified database system and that the account is valid for the user. Profile information is used by some portlets, for example:
 - My Queries portlet, to track query activity for the user
 - SQL Scratchpad portlet, to determine database access privileges for the user

Related Information

Topic	Resources for Further Information
The duties typically performed by the user types for which customer roles are recommended	Types of Users
Creating Teradata Viewpoint users, roles, and profiles	<ul style="list-style-type: none"> Viewpoint application Help screens

Topic	Resources for Further Information
	<ul style="list-style-type: none"> • <i>Teradata® Viewpoint User Guide</i>, B035-2206

Configuring User Dashboards

You can create custom dashboard pages, and then use the Add Content function to populate each page with a set of related portlets from among those available to the user.

The primary dashboard should contain the most important portlets for live monitoring of database activity, for example:

- System Health (all)
- My Queries (users) or Query Monitor (administrators and programmers)
- 1 or 2 other optional portlets, depending on the type of user

You can create additional dashboards to contain other portlets.

Note:

You may need to limit the number of portlets per dashboard page. Displaying too many Teradata Viewpoint portlets on one dashboard may make it hard to find needed data, and for some browsers, may slow portlet loading and response times.

For details about configuring dashboards and portlets, see *Teradata® Viewpoint User Guide*, B035-2206.

Creating a Dashboard

From the Teradata Viewpoint portal:

1. Use the **Add Page** option to name the dashboard (for example, MainDashboard).
2. Use the **Add Content** function to select portlets to include in the dashboard.

Loading and Exporting Data: Application DBAs

This section provides an overview of the different methods used to load data into or export data from Vantage. The section also provides guidelines on choosing the best utility for your purpose and examples of utility job performance analysis and capacity planning.

Teradata Parallel Transporter

Teradata Parallel Transporter (PT) is an object-oriented client application that provides scalable, high-speed, and parallel data:

- Extraction
- Loading
- Updating

These capabilities can be extended with customizations or third-party products.

Teradata PT uses but expands on the functionality of the traditional Teradata extract and load utilities: FastLoad, MultiLoad, FastExport, and TPump, also known as *standalone utilities*.

Teradata PT can load data into and export data from any accessible database object in Analytics Database or other data store using Teradata PT operators or access modules.

Load Operator

The Load operator uses the Teradata FastLoad protocol to load a large volume of data at high speed into an empty table in Analytics Database.

The Load operator is typically used for the initial loading of Teradata tables. It inserts data from data streams into individual rows of a target table.

Update Operator

The Update operator emulates the Teradata MultiLoad utility to perform highly scalable and parallel inserts, updates, deletes, and upserts into new or pre-existing Vantage tables in a single pass.

The Update operator can also be used to insert new rows into the database, but it *cannot* perform select operations. The Update operator is not supported for column-partitioned tables.

Export Operator

The Export operator uses the Teradata FastExport protocol to extract large amounts of data at high speed from Vantage using block transfers over multiple sessions. The Export operator reads data from one or more tables in Vantage.

Basic Teradata Query for Loading Tables

Differences Between BTEQ and Teradata PT

BTEQ and Teradata PT can both load data into Teradata and can update tables, among other functions. Other similarities include:

- **Capacity** – Both can send up to 64 KB per request and up to 1 MB per request with array support.
- **Capabilities** – Both can support multiple sessions and can load, export, process DDL, and act as a command line interface.

The following table explains the major differences between BTEQ and Teradata PT:

BTEQ	Teradata PT
Consumes less client CPU.	Consumes more client CPU.
Loads or updates either all records or stops after x records that you specify with the REPEAT command.	Can also load or update only selected rows.
Limited error handling.	Performs error handling and creates error log tables.
No checkpoint restart.	Checkpoint restart.
No SERIALIZE option.	SERIALIZE option ensures the correct sequencing of transactions.
Does not process multiple DML statements in a single pass of the input data.	Can perform conditional apply or multiple apply.

Recommendation: If your job has millions of rows, use Teradata PT. If not, especially for a load job that will be used again in the future, consider both. See which performs better and causes less system overhead. BTEQ can be a good choice for smaller amounts of data, exploratory work, and DDL.

Updating Tables

Frequently, when updating the database, you make the same change to many rows. It is helpful to eliminate the repetitive work required to update them individually. The REPEAT command serves this purpose.

Updating Rows Using a Single Request

If you can describe all of the rows by some formula, then you can use a single request to update them all. The following request, for example, updates the salaries of all employees in department 600 of the database *Personnel*:

```
update personnel.employee set salary=salary*1.07
  where deptno=600;
```


Updating Rows Using Multiple Requests

If you cannot describe all of the rows by a formula, you must describe them individually. The following requests, for example, update the salaries of two employees:

```
update personnel.employee set salary=salary*1.07
  where empno = 10006;
```

```
update personnel.employee set salary=salary*1.07
  where empno = 10013;
```

This approach is convenient for a few updates, but if there were hundreds, the script would become very long. It would also be very inflexible; you would have to edit each entry to use the script again for another list of employees.

Updating Rows by Importing a File

A better approach would be to create a separate file (for example, RAISEEMP) containing two records in data format representing the employee numbers 10006 and 10013. Then use the BTEQ IMPORT command with a REPEAT command to update the rows in the database.

Using the REPEAT Command When Importing a File

The REPEAT command appears before the request and specifies the total number of requests to be submitted.

The following BTEQ script using the BTEQ REPEAT command opens the character-format file RAISEEMP and repeats the update sequence twice:

```
.IMPORT data file=raiseemp
.REPEAT 2
using enumb (char(5))
update personnel.employee set salary=salary*1.07
  where empno = :enumb;
```

For each employee, BTEQ reads a value from the RAISEEMP file for the variable called *enumb* and carries out the UPDATE for the row whose *EmpNo* equals the value of *enumb*.

Note:

When using a file originally exported with the BTEQ EXPORT command as the source for an IMPORT command across a different platform type, ensure that the endianness type of both platforms is the same. You can verify this from the “Client Platform Byte Order” tab in the output of the SHOW CONTROLS command.

Loading Data into Teradata Vantage

BTEQ Input Stream

Use the following BTEQ input stream to load (import) data into Vantage:

```
.IMPORT DATA FILE=DeptData.imp
.REPEAT *
USING      DeptNo (SMALLINT),
           DeptName (VARCHAR(14)),
           Loc (CHAR(3)),
           MgrNo (SMALLINT)
INSERT INTO Personnel.newtable (DeptNo, DeptName,
                                Loc,MgrNo)
VALUES      (:DeptNo, :DeptName, :Loc, :MgrNo);
```

where:

- The REPEAT * command repeats the next request until the data in the import file is exhausted.
- The term *newtable* refers to an empty table to receive the data being loaded.

Related Information

Topic	Resources for Further Information
BTEQ commands, including PACK	<i>Basic Teradata® Query Reference</i> , B035-2414
Running batch jobs to submit BTEQ commands and Teradata SQL	<i>Basic Teradata® Query Reference</i> , B035-2414

Restarts and Aborts on BTEQ Jobs with Identity Column

If a restart occurs during a BTEQ import, BTEQ will resubmit the last uncompleted row insert after the system recovers from the restart. Identity column numbering will continue from there. It is possible, however, for a restart to cause duplicates because BTEQ may do a reinsert even if the previous insert has completed. Duplicates will not be detected if the target table is MULTiset and not defined with a UPI.

If a session abort occurs during a mainframe-attached BTEQ import, the last uncompleted row insert will not be resubmitted and associated data may be lost. Associated data may also be lost if a workstation-attached BTEQ import session is aborted and there is no other session through which to resubmit the insert.

In both cases, manually restarting the import can result in duplicate rows if rows newly inserted before the session abort are not deleted unless the target table is defined with a UPI.

Reading Committed Data While Loading to the Same Table

Users can read committed data in a table while the table is being loaded when all of these conditions are met:

- The table must be a load-isolated table.

You identify a table as load isolated using the ISOLATED LOADING option with CREATE TABLE or ALTER TABLE requests. For example:

```
CREATE TABLE ldi_table01, WITH CONCURRENT ISOLATED LOADING FOR ALL (col_a
INT, col_b VARCHAR(32));
```

- The load is performed using isolated loading.

As described later in this section, isolated loading can occur even when you don't explicitly specify ISOLATED LOADING. If you want to ensure isolated loading, do one of the following things:

- Use the ISOLATED LOADING option with your SQL request (INSERT, UPDATE, MERGE, and DELETE). For example:

```
INSERT WITH CONCURRENT ISOLATED LOADING INTO ldi_table01 SELECT *
FROM src_Tab;
```

This method works if the group of tables being loaded are all within one database transaction.

- Use the BEGIN ISOLATED LOADING statement. For example:

```
BEGIN ISOLATED LOADING ON ldi1, ldi2 USING QUERY_BAND
'LDILoadGroup=Grp1' IN MULTIPLE SESSION;
```

This method works well if the related group of tables are being loaded in different scripts, programs, or SQL transactions in one or more sessions. Before starting load operations, use a BEGIN ISOLATED LOADING request, and after the loads complete, use an END ISOLATED LOADING request.

- The SELECT request must use the LOAD COMMITTED locking option to concurrently read committed data from the table. For example:

```
LOCKING TABLE ldi_table01 FOR LOAD COMMITTED SELECT * FROM ldi_table01;
```

You can use Teradata PT with the SQL Inserter and SQL Selector operators for load isolation because these operators execute SQL requests. (The Load and Update operators execute SQL requests when the MLOADX protocol is used; however, Analytics Database determines the protocol that is used for each job. If the Load or Update operator uses the MULTILoad protocol, then concurrent reads will not be allowed.)

You can determine whether the ISOLATED LOADING option is being used for a load by using the DBQL step level logging facility. See DMLLoadID in QryLogStepsV.

If a load is performed on a load-isolated table using an SQL request without the ISOLATED LOADING option or without the BEGIN ISOLATED LOADING request, then the default is to:

- Allow concurrent reads on committed rows while the table is being all-AMP updated
- Disallow concurrent reads while the table is modified with less than an all-AMP operation

Note:

All SQL requests in a transaction must use the same type of concurrency as the first request in the transaction or the transaction will fail.

For details on CREATE TABLE and ALTER TABLE syntax or the isolated loading statements BEGIN/END ISOLATED LOADING and CHECKPOINT ISOLATED LOADING, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144. For details on the SELECT, INSERT, UPDATE, MERGE, DELETE, and LOCKING request modifier syntax, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.

For information on locking and load isolation, see *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142.

Cleaning up Load Isolated Tables

Load Isolation uses row versioning to ensure that only the most recently committed data is retrieved. Logically deleted rows are created as an artifact of row versioning and, as a result, a load-isolated table typically grows faster than a table without load isolation. A DBA must physically delete the logically deleted rows at regular intervals to reclaim disk space and improve query performance.

Larger tables with more concurrent reads during updates will require more frequent cleanup. When you collect summary statistics on your load-isolated tables, monitor the logically deleted rows (DelRowCount) as a percentage of physical rows (PhyRowCount). Perform cleanup when the table contains a significant percentage of logically deleted rows or you notice degraded query performance on the table while modifications occur.

To clean up logically deleted rows, you can use either the RELEASE DELETED ROWS option of an ALTER TABLE request or issue a stored procedure to perform cleanup on cascaded, logically deleted rows in load-isolated tables and underlying join indexes and secondary indexes.

Alter Table Cleanup Method

Use the RELEASE DELETED ROWS option in an ALTER TABLE request to interactively clean up logically deleted rows from a load-isolated table. For example:

```
ALTER TABLE Employee RELEASE DELETED ROWS;
```

For details on the syntax, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Stored Procedure Cleanup Method

Use a stored procedure to clean up logically deleted rows from multiple load-isolated tables and their underlying join indexes and secondary indexes. Use `LDI_Clean_ANSI` in ANSI mode and `LDI_Clean` in Teradata mode. The input and output is the same for both ANSI and Teradata mode. (These stored procedures are created typically during Vantage installation but can be added later using the `DIPDEM` or `DIPPOST` scripts.)

The following example uses `LDI_Clean` in Teradata mode:

```
call ldi_clean('database_name', 'table_name', 'Reset_flag_value');
```

where *Reset_flag_value* is either 'Y' or 'N' to specify whether you want to reset the incremental value of the current load to 0. For example:

```
call ldi_clean('finance', 'acct_payable', 'y');
```

Teradata responds with the following output:

```
acct_payable altered, finance altered
```

Keep in mind the following points:

- An exclusive lock is required while performing cleanup on a load-isolated table.
- The `DROP TABLE` privilege is required to perform the cleanup.

FastLoad Utility

FastLoad allows you to load data from the client to your Vantage system. It allows fast loading of a single empty table that has *no* SIs, JIs, HIs, RI (FKs), or column partitioning.

FastLoad loads large amounts of data into an empty table, with or without a primary index, on Analytics Database. However, row size and the number of columns, more than any other factors, do affect performance.

When you load large tables, consider the following things.

IF you are loading a large table...	THEN...
with fallback	<ul style="list-style-type: none"> • Observe the suggestions for loading the large table without fallback. • Create the table without fallback protection. • Load the table. • Use BTEQ to alter the table to have fallback protection.

IF you are loading a large table...	THEN...
without fallback	<ul style="list-style-type: none"> Initially, set session to the number of AMPs in your system. Then, experiment by reducing the number of sessions. Avoid checkpointing too frequently. Avoid NULLIF clauses, VAR fields, or indicator bits whenever possible. Run concurrent FastLoad jobs. The maximum number of FastLoad jobs you can run is 30. The default is 5. (See <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 for more information.) You can also change the maximum number of jobs allowed by defining throttle rules through the Teradata Viewpoint Workload Designer portlet. For more information on FastLoad, see Maximum Number of Concurrent FastLoad, MultiLoad, and FastExport Jobs. <p>Note: You cannot use the NO FALLBACK option and the NO FALLBACK default on platforms optimized for fallback.</p>

FastLoad does not support the following:

- HIs
- JIs
- FKs
- SIs (Both USIs and NUSIs are not supported)
- LOBs
- Column partitioning

For more information on this utility, see *Teradata® FastLoad Reference*, B035-2411

FastExport Utility

FastExport quickly exports data from a Vantage system to the client platform. This utility can format and export very large amounts of data very quickly. The maximum number of FastExport jobs you can run is 60. The default is 5. You can also change the maximum number of jobs allowed by defining throttle rules through the Teradata Viewpoint Workload Designer portlet. (See [Maximum Number of Concurrent FastLoad, MultiLoad, and FastExport Jobs](#) for more information.)

To improve performance:

- Do not use too many sessions.
- Avoid using any option that causes the evaluation of individual fields within a layout, including NULLIF clauses and APPLY WHERE conditions.

The following restrictions and limitations apply:

- Exponential operators
- Concatenated fields

- Hexadecimal forms

For more information, see *Teradata® FastExport Reference*, B035-2410

MultiLoad Utility

MultiLoad allows you to upload data from the client to your Vantage system. It operates on multiple tables simultaneously and can also insert, update, and delete data.

The maximum number of MultiLoad jobs you can run is 30. The default is 5. You can also change the maximum number of jobs allowed by defining throttle rules through the Teradata Viewpoint Workload Designer portlet. (See [Maximum Number of Concurrent FastLoad, MultiLoad, and FastExport Jobs](#) for more information.)

To improve performance:

- Minimize concatenation and redefinition of input data.
- Restrict the number of NUSIs.
- Make the PI of each data table unique or nearly unique.
- Minimize the use of error tables.
- Do not checkpoint too often.
- Avoid using too many sessions.

MultiLoad does not support the following:

- Aggregate operators
- Exponential operators
- Arithmetic functions
- Concatenation of data files
- Hexadecimal forms
- NoPI tables
- Foreign keys
- Loading LOBs
- Column-partitioned tables

For more information, see *Teradata® MultiLoad Reference*, B035-2409

You can perform the following steps as an alternative to using MultiLoad:

1. Create a staging table that is otherwise identical with the target table, but has no constraints, triggers, or indexes defined on it, such as a NoPI.
2. Import into the staging table from the external source (for example, using FastLoad or Teradata Parallel Data Pump Array INSERT operations).
3. (Optional) create an error table for the target table if error logging is desired.
4. Execute a MERGE into the target table using the staging table as source.
5. Drop the staging table and error table.

For information on NoPI tables or Teradata Parallel Data Pump Array INSERT operations, see *Teradata Vantage™ - Database Design*, B035-1094.

Related Information

For more information on.	See...
MERGE	<ul style="list-style-type: none"> the Orange Book "Exploring the Benefits of Teradata 12.0 – ANSI MERGE Enhancements." <i>Teradata Vantage™ - SQL Data Manipulation Language</i>, B035-1146
MERGE INTO	<i>Teradata Vantage™ - SQL Data Manipulation Language</i> , B035-1146

Maximum Number of Concurrent FastLoad, MultiLoad, and FastExport Jobs

If you do not use the Teradata Viewpoint Workload Designer portlet to throttle concurrent load and unload jobs, the MaxLoadTasks and the MaxLoadAWT fields of the DBSControl determine the combined number of FastLoad, MultiLoad, and FastExport jobs that the system allows to run concurrently. The default is 5 concurrent jobs.

If you have the System Throttle (Category 2) rule enabled, even if there is no Utility Throttle defined, the maximum number of jobs is controlled by the Teradata dynamic workload management software and the value in MaxLoadTasks field is ignored.

For more information, see "MaxLoadTasks" and "MaxLoadAWT" in the section on DBS Control in *Teradata Vantage™ - Database Utilities*, B035-1102.

Utility Management

To manage load utilities such as FastLoad, MultiLoad, and FastExport, you can classify them into a workload definition based on a combination of the following criteria:

- Utility Type
- "Who" criteria (such as, account name, user name, query band, and so on)
- Database and table or view name (except for archive and restore jobs)

Use the following utility-related Teradata Active System Management (TASM) rules to define how Vantage manages its workloads:

- Utility Throttle** (also called Utility Limit in the Teradata Viewpoint Workload Designer portlet) – This rule enforces system wide limits on each type of the Teradata utilities, such as FastLoad, MultiLoad, FastExport, Teradata Parallel Transporter Load operator, JDBC FastLoad, and so on.
- Utility Workload Throttle** – This rule enforces the limits based on the following criteria:
 - Utility Type
 - "Who" criteria (such as account name, user name, query band, and so on)

- Database and table or view name (not available for archive or restore jobs)

To define a Utility Workload Throttle, create a workload definition with a throttle limit and the qualification criteria (or a combination) listed here for this rule.

- **AWT Resource Limit** – This rule specifies an AMP Worker Task (AWT) limit for utilities. The rule enforces the limit based on the utility type, request source (for example, username and account), and query band.
- **Utility Session** – This rule specifies how many utility sessions can be used for the various kinds of utilities. The supported qualification criteria are:
 - Utility Type
 - Account name, user name, query band, and so on

When the Teradata dynamic workload management software is enabled and there is no applicable user-defined Utility Session rule, the Teradata dynamic workload management software automatically uses the default session rules to select the number of sessions based on the utility type, system configuration, and optional data size.

You cannot delete default session rules. However, you can modify the utility session information (that is, the default numbers assigned to various utility kinds).

You can manage your load utilities and DSA by using Teradata Viewpoint.

Teradata Parallel Data Pump

The Teradata Parallel Data Pump (TPump) utility allows real time updates from transactional systems into the data warehouse. TPump executes INSERT, UPDATE and DELETE requests, or a combination, to more than 60 tables at a time from the same source feed.

TPump is an alternative to MultiLoad. The benefits of TPump include:

- Real time INSERTs and UPDATEs to more than 60 tables simultaneously
- Low volume batch maintenance
- Can provide continuous feed to the warehouse

The data handling functionality of TPump is enhanced by the TPump Support Environment. In addition to coordinating activities involved in TPump tasks, it provides facilities for managing file acquisition, conditional processing, and certain Data Manipulation Language (DML) and Data Definition Language (DDL) activities of Vantage.

TPump has the following restrictions:

- It does not support aggregate operators or concatenation of data files
- TPump performance is severely affected by access or DBQL logging.

The TPump Support Environment enables use of variable substitution, conditional execution based on the value of return codes and variables, expression evaluation, character set selection options and more. For more information, see *Teradata® Parallel Data Pump Reference*, B035-3021.

Restarts on TPump Jobs with Identity Column

TPump works on multistatement SQL requests. Each request has a specific number of statements depending on the PACK specification in the BEGIN LOAD command.

In ROBUST mode, each request is written into a restart log table. Since Analytics Database only rolls back statements in a packed request that fail rather than rolling back the entire request, the restart log will always accurately reflect the completion status of a TPump import.

If a restart occurs, TPump will query the restart log table and re-execute requests that are not logged. This means it may be possible for a restart to generate duplicates if an insert request is repeated. Duplicates will not be detected if the target table is not defined with a UPI.

TPump will flag an error if it is run in simple mode and the target table has an identity column PI. This is because no restart log is used for restart recovery and duplicate rows could result if some requests are reprocessed.

For more information on this utility, see *Teradata® Parallel Data Pump Reference*, B035-3021.

Loading Geospatial Data

The TDGeolImport data load program can convert geospatial data into a format that can be used by the Teradata geospatial UDTs.

TDGeolImport converts layers from ESRI, MapInfo, and TIGER/Line data sources and loads them directly into the database. This program can be used with geometries that have web representations of any size up to a maximum of 16MB.

Loading Unicode Data with Unicode Pass Through

To eliminate the need to cleanse Unicode data before loading and to reduce translation errors, use Unicode Pass Through. This option allows pass through characters to be imported into and exported from Teradata. Pass through characters include:

- BMP from Unicode versions 6.1.0 to 9.0.0, which Teradata does not support, including emoji
- SMP from all Unicode versions, which Teradata does not support
- Unassigned characters
- Private use characters

UTF-8 or UTF-16 sessions with Unicode Pass Through enabled allow pass through characters to be used in requests and to be stored in or retrieved from Unicode columns. Noncharacters and characters with an invalid encoding form are changed to the REPLACEMENT CHARACTER (U+FFFD), which can pass through and be stored in Teradata with this option.

Unicode Pass Through is not supported with FastLoad, MultiLoad, Teradata Parallel Data Pump (TPump), and FastExport. Use Teradata Parallel Transporter (TPT), BTEQ, or the Teradata JDBC Driver to load or unload data containing pass through characters.

Note:

If you do not specify Unicode Pass Through for a session, the default is OFF.

BTEQ Example with Unicode Pass Through

```
.SESSIONS 5
.LOGON mysystem/uptuser,emoji
.REPEAT 5
SET SESSION CHARACTER SET UNICODE PASS THROUGH ON;
```

Teradata Parallel Transporter Example with Unicode Pass Through

```
VARCHAR UnicodePassThrough = 'ON',
```

The syntax applies only to these TPT operators:

- Load
- Update
- Export
- Stream
- DDL
- SQL Inserter
- SQL Selector

JDBC Example with User Startup String

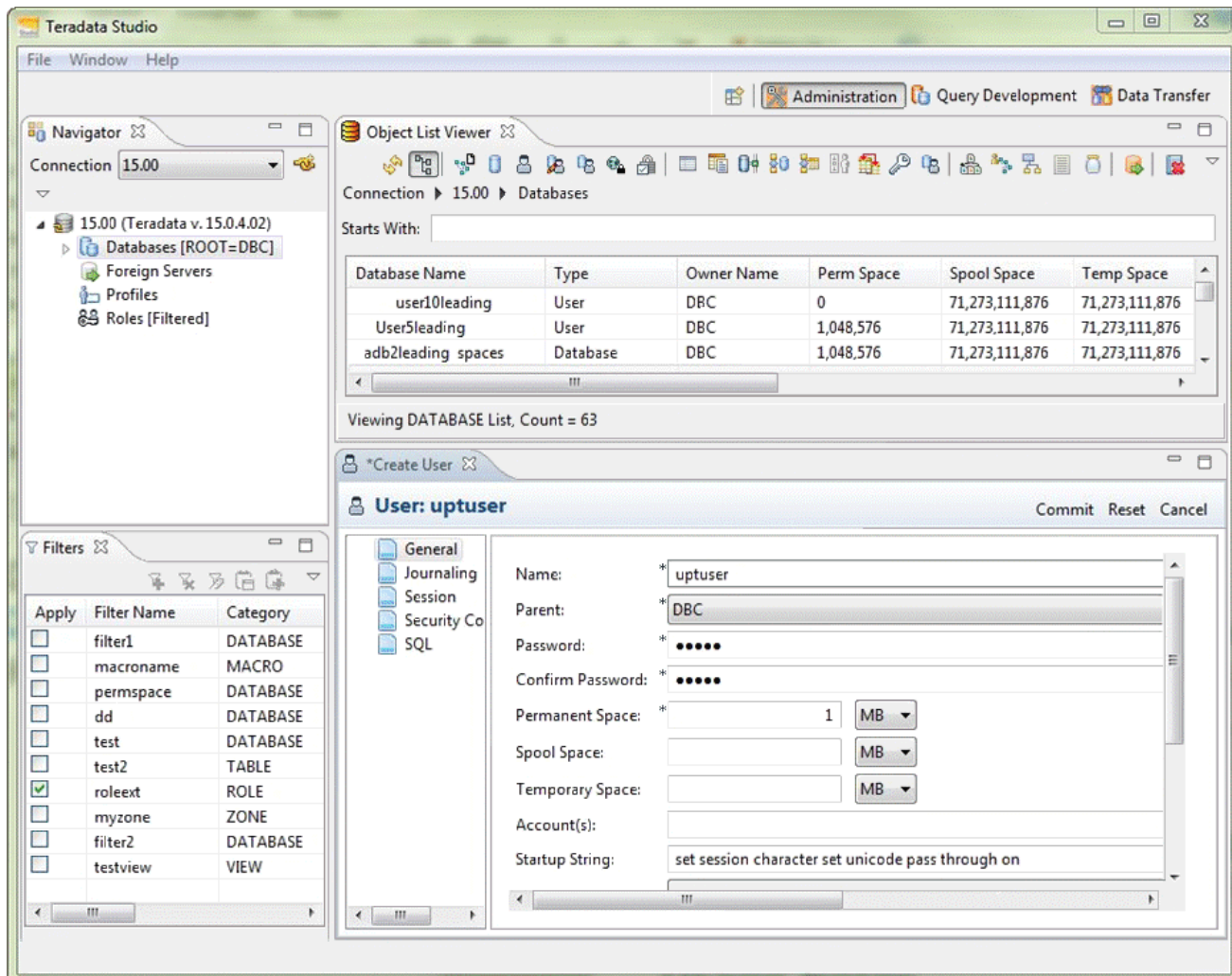
```
create user uptuser as password = emoji, startup = 'set session character set
unicode pass through on';
Connection con = DriverManager.getConnection("jdbc:teradata://dbs/
RUNSTARTUP=ON", "uptuser", "emoji");
```

JDBC Example with SQL Request

```
stmt.executeUpdate("set session character set unicode pass through on");
```

Teradata Studio Example

In the Studio **Administration Create User** form, provide a startup string, such as “set session character set unicode pass through on”.



When creating the connection profile for that user, add the JDBC property RUNSTARTUP=ON.

New Teradata Connection Profile

Specify a Driver and Connection Details
Select a driver from the drop-down and provide login details for the connection.

Select a driver from the drop-down: **Teradata**

Database Server Name: * test

User Name[Domain]: * uptuser

Password: *

Authentication Mechanism: **PASSWORD_PROTECTED**

Database:

☐ Save Password

JDBC Connection Properties

CHARSET=UTF8
RUNSTARTUP=ON
TMODE=ANSI
QueryBand = 'ApplicationName = [APPID]; Version = [VERSION]; ClientUser = [LOGONID];'

Add...
Remove
Clear All
Reset to Defaults
Save to Preference

Cache Properties Delete Cache Files

☒ Connect when the wizard completes
☐ Connect every time the workbench is started

Test Connection

? < Back Next > Finish Cancel

Usage Notes

- Customers should not use Unicode Pass Through if they rely on Teradata to screen out Teradata unsupported characters.
- You may have existing processes in place to cleanse Unicode data before loading it into Vantage, such as access modules and UDFs. To take full advantage of Unicode Pass Through, you must change or eliminate prior methods of cleansing Unicode data before loading. See the TPT documentation for details on configuring how TPT uses access modules. For details about configuring access modules to allow pass through characters, see *Teradata® Tools and Utilities Access Module Reference*, B035-2425, in particular the sections regarding automatic character conversions.

Related Information

For more information on...	See...
SET SESSION CHARACTER SET UNICODE	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
Unicode Pass Through	<i>Teradata Vantage™ - Analytics Database International Character Set Support</i> , B035-1125

Interpreting LOAD Utility Status

When observing the progress of load, archive, export, and other jobs, remember the following:

- The system uses LSN to calculate the impact of all sessions.
- All work the system performs during a session is charged to that session.

Because not all sessions are used during an operation, some sessions with the same LSN and user name may not report any activity.

The Query Session utility (QRYSESSN) can provide detailed status of each session involved in the load task.

Choosing the Best Utility for Your Purpose

Some of the client utilities have similar functions. However, to optimize performance on resource utilization, it is important to choose the best utility for your job. Note that the following section does *not* include Teradata Parallel Transporter.

Definitions

To better understand the guidelines in this section, refer to the definitions in the following table.

Term	Meaning
Small table	<1 million rows
Large table	>100 million rows
Small number of rows	<5% of total rows
Moderate number of rows	20 - 30% of total rows
Large number of rows	40 - 50% of total rows

Guidelines for Inserting Rows

The following table provides some guidelines you can use to make the best choice of utilities when inserting rows. Keep in mind that you may find a better way through experimenting.

Task	First Choice	Second Choice
Insert a small number of rows into an empty table	FastLoad	TPump/BTEQ
Insert a large number of rows into an empty table	FastLoad	TPump
Insert a small number of rows into a small populated table	BTEQ, MultiLoad, TPump or MERGE when the transaction density is too low for satisfactory performance	
Insert a small number of rows into a large populated table	TPump, BTEQ, or MERGE	
Insert a moderate number of rows into a large populated table	MultiLoad or MERGE	TPump when the transaction density is too low for satisfactory performance
Insert a large number of rows into a large populated table	MultiLoad or MERGE	1. FastLoad rows into an empty table. 2. INSERT ... SELECT rows into an empty new table.
Insert a large number of rows into multiple populated tables	MultiLoad	TPump
Insert a large number of rows when logged on session are near their full capacity	INSERT ... SELECT or MERGE	TPump
Insert a large number of rows when the specific order of updates is important	Tpump	INSERT ... SELECT or MERGE
Insert a large number of rows into multiple empty tables	Multiple FastLoads	MultiLoad

Guidelines for Deleting Rows

The following table provides some guidelines you can use to make the best choice of utilities when deleting rows. Keep in mind that you may find a better way through experimenting.

Task	First Choice	Second Choice
Delete a small number of rows from any table	TPump	BTEQ
Delete a large number of rows from a large table	MultiLoad	BTEQ
Perform DELETE operations on a large number of rows on multiple tables	MultiLoad	TPump
Delete all rows from a table	BTEQ In this case, Teradata marks the table header with an indicator to say that the table is empty. Assuming there are no locks on the table when this delete is requested, the delete will complete in a second or two.	
Delete some rows from a table	BTEQ Space permitting, it can also be an option to create an empty table, insert into this table the rows that you wish to retain and drop the original table and then rename the new table to the name of the original table.	MultiLoad deletes are good for deleting large volumes of data, but not all data, from a table. MultiLoad reads the data block by block. It will “touch” each block consumed by the table and remove rows as appropriate, and replace the block. This can be very efficient and is faster than an SQL delete in many cases because there is no rollback logging. However, MultiLoad must complete or else the table will remain in an unstable state. For MultiLoad to work best, drop SIs, run the MultiLoad delete, and then replace the indexes. This is quicker and ensures that the work table is small.

Note:

You cannot delete rows from a table using the SQL MERGE statement.

Guidelines for Other Batch Operations

When updating, exporting, or doing other batch operations, use the guidelines in the following table to consider the best choice for your job.

Task	First Choice	Second Choice
Merging a large number of rows when logged on session are near their full capacity	INSERT ... SELECT or MERGE	Tpump

Task	First Choice	Second Choice
Merging a large number of rows when the specific order of updates is important	TPump	INSERT ... SELECT or MERGE
Export a small number of rows from any table	BTEQ	FastExport
Export a moderate or large number of rows from a large table	FastExport	BTEQ
Update a small number of rows in any table	TPump	BTEQ
Update a large number of rows in a large table	MultiLoad	BTEQ insert table into a new one.
Perform multiple DML (INSERT, UPDATE, DELETE) operations on a small number of rows on multiple tables	MultiLoad or TPump when the transaction density is too low for satisfactory performance	BTEQ
Perform multiple DML (INSERT, UPDATE, DELETE) operations on a large number of rows on multiple tables	MultiLoad	TPump
Copy one or several tables to a different Vantage system	DSA	FastExport and FastLoad
Copy all tables in the Vantage system to a different Vantage system	DSA	
Load an identity column table	TPump (in ROBUST mode)	BTEQ .IMPORT
Load the data into staging table	Teradata PT Load Operator	Teradata PT Update Operator

Utility Job Performance Analysis and Capacity Planning

You can query the DBQL utility job log table, DBC.DBQLUtilityTbl, to analyze the performance of load or export utility jobs and do capacity planning. This table is populated if you use the WITH UTILITYINFO option with a BEGIN/REPLACE QUERY LOGGING request. For details on the contents of the corresponding view QryLogUtility[V], see *Teradata Vantage™ - Data Dictionary*, B035-1092 and [Maximum Number of Concurrent FastLoad, MultiLoad, and FastExport Jobs](#).

Examples of Using the DBQL Utility Job Log Table

You can use DBQLUtilityLogTbl to diagnose utility performance issues and do capacity planning. The following are just some examples of how to use the table.

Investigating Why a Job Ran Slower Today

In this scenario, a FastLoad job submitted today took significantly longer than yesterday. You want to determine the root cause. Although the example is specific to FastLoad, you can use this general approach for other utilities.

1. Retrieve the rows for the jobs yesterday and today from DBQLUtilityTbl using the following criteria:
 - Job attributes that uniquely identify this FastLoad job. This criteria returns all instances of this FastLoad job. Possible job attributes include:
 - UtilityName = 'FASTLOAD'
 - Username
 - Query band
 - Utility request
 - Job end time: either today or yesterday.
2. Compare various column values between the two rows to determine the possible causes. Suggestions include:
 - a. Compare RowsInserted values. If the job today loaded significantly more rows, this could be the cause. If not, proceed to the next step.
 - b. Compare DelayTime values. An increase indicates the job today was delayed longer by TASM.
 - c. Compare the elapsed time of each phase (PhaseEndTime - PhaseStartTime) to identify the phase with a significant increase in elapsed time.
 - d. If the phase with a significant increase in elapsed time is the Acquisition phase, check for an increase in MaxDataWaitTime, which might be caused by more load on the client machine or slower network response time.
 - e. Compare the resource usages of the phase with a significant increase in elapsed time to find additional clues. For example:
 - An increase in PhaseMaxCPUTime may indicate data skew.
 - An increase in Phase1 messages (Phase1BlockCount) may indicate that a smaller message size was used.
3. Examine related query log data in DBQLLogTbl. To select the desired data, use:
 - LSN to select all rows of a particular job.
 - PhaseStartTime and PhaseEndTime to select rows of a specific phase.

Performance Anomaly Detection

In this scenario, you want to detect job instances whose elapsed time is 50% longer than the last 30-day average for the same job. This approach can be used to detect anomalies for other metrics, such as CPU time or I/Os.

1. Select rows from DBQLUtilityTbl for jobs ending within the last 30 days and store them in a temporary table called Last30dayJobInstances.


```
CREATE VOLATILE TABLE Last30dayJobInstances AS
  (SELECT U.*,
  SUBSTRING(UtilityRequest FROM 1 FOR 100) AS UtilReqShort
  FROM DBC.DBQLUtilityTbl U
  WHERE CAST(JobEndTime AS date) >= DATE-30)
WITH DATA
PRIMARY INDEX (UtilityName, UserName)
ON COMMIT PRESERVE ROWS;
```

2. Use the Last30dayJobInstances table to calculate the last 30-day average elapsed time of each job, grouped by attributes such as utility name, user name, and utility request. Store the results in another temporary table called Last30dayAverage. Each row contains the average elapsed time and job attributes.

```
CREATE VOLATILE TABLE Last30dayAverage AS
  (SELECT AVERAGE((JobEndTime - JobStartTime) HOUR TO SECOND)
    AS AvgJobElapsedTime,
    UtilityName, UserName,
    SUBSTRING(UtilityRequest FROM 1 FOR 100) AS UtilReqShort
  FROM Last30dayJobInstances J
  GROUP BY UtilityName, UserName,
    SUBSTRING(UtilityRequest FROM 1 FOR 100))
WITH DATA
PRIMARY INDEX (UtilityName, UserName)
ON COMMIT PRESERVE ROWS;
```

3. Select job instances whose elapsed time is more than 50% of the average using the Last30dayJobInstances and Last30dayAverage tables.

```
SELECT I.*
FROM Last30dayJobInstances I,
     Last30dayAverage A
WHERE I.UtilityName = A.UtilityName
     AND I.UserName = A.UserName
     AND I.UtilReqShort = A.UtilReqShort
     AND (
       (EXTRACT(HOUR FROM ((I.JobEndTime - I.JobStartTime) HOUR TO SECOND))
       * 3600) +
       (EXTRACT(MINUTE FROM ((I.JobEndTime - I.JobStartTime) HOUR TO
SECOND)) * 60) +
       (EXTRACT(SECOND FROM ((I.JobEndTime - I.JobStartTime) HOUR
TO SECOND)))
       ) > (
       ((EXTRACT(HOUR FROM A.AvgJobElapsedTime) * 3600) +
       (EXTRACT(MINUTE FROM A.AvgJobElapsedTime) * 60) +
```



```
(EXTRACT(SECOND FROM A.AvgJobElapsedTime))
) * 1.5);
```

Capacity Planning

In this scenario, you calculate these monthly metrics for each utility for the previous 12 months:

- Number of job instances
- Average job elapsed time
- Total processing volume (byte counts and row counts)
- Average processing volume
- Average throughput (rows per second and bytes per second)

```
SELECT TRUNC(CAST(JobEndTime AS DATE), 'RM') JobMonth, UtilityName,
       COUNT(*) AS NumJobs,
       AVG(
         (EXTRACT(HOUR FROM ((JobEndTime - JobStartTime) HOUR TO SECOND)) *
3600) +
         (EXTRACT(MINUTE FROM ((JobEndTime - JobStartTime) HOUR TO SECOND)) *
60) +
         (EXTRACT(SECOND FROM ((JobEndTime - JobStartTime) HOUR TO SECOND)))
       ) AS AvgJobTime,
       SUM(RowsInserted), AVG(RowsInserted),
       SUM(RowsUpdated ), AVG(RowsUpdated ),
       SUM(RowsDeleted ), AVG(RowsDeleted ),
       SUM(RowsExported), AVG(RowsExported),
       SUM(Phase1ByteCount), AVG(Phase1ByteCount),
       SUM(Phase2ByteCount), AVG(Phase2ByteCount),
       AVG(RowsInserted) / AvgJobTime AS InsRowsPerSec,
       AVG(RowsUpdated ) / AvgJobTime AS UpdRowsPerSec,
       AVG(RowsDeleted ) / AvgJobTime AS DelRowsPerSec,
       AVG(RowsExported) / AvgJobTime AS ExpRowsPerSec,
       AVG(Phase1ByteCount) / AvgJobTime AS AvgPhase1BytesPerSec,
       AVG(Phase2ByteCount) / AvgJobTime AS AvgPhase2BytesPerSec
FROM   (SELECT *
        FROM DBC.DBQLUtilityTbl U
        WHERE CAST(JobEndTime AS DATE) >= ADD_MONTHS(TRUNC(CURRENT_DATE,
'RM'), -12)
        AND CAST(JobEndTime AS DATE) < TRUNC(CURRENT_DATE, 'RM') )
AS OneYearLog
GROUP BY JobMonth, UtilityName
ORDER BY JobMonth, UtilityName;
```


Working with Sessions and Accounts: Operational DBAs

This section describes the differences between ANSI and Teradata session modes and how to set the system session mode. It also describes Teradata sessions: how to obtain information about them, troubleshoot problems with them, and improve their performance. System account information is also provided: how to create accounts, work with system accounting views, and use account string expansion to measure query and AMP usage.

Session Modes and Transaction Processing

Teradata Vantage supports two session modes:

- ANSI
- Teradata

In ANSI session mode:

- Transaction processing follows the rules defined by the ANSI/ISO SQL:2011 standard.
- For Teradata SQL features that support both the ANSI/ISO SQL standard syntax and the Teradata SQL dialect, the ANSI/ISO SQL standard syntax is used.
- The default case specification for character data is CASESPECIFIC.
- When a character string requires truncation, the excess pad characters are truncated without reporting an error. Truncation of characters other than trailing nonblank characters results in a truncation exception.
- Cursors are always positioned. Updating or deleting the most current fetched cursor row using the UPDATE/DELETE ... WHERE CURRENT OF statement is valid.
- The default table type is MULTiset, which allows duplicate rows.
- The TRIM function trims both leading and trailing pad characters by default.

In Teradata session mode:

- Transaction processing follows the rules defined by Teradata before the emergence of the ANSI/ISO SQL standard. These rules largely conform with ANSI/ISO semantics, but in some cases differ in important ways from the ANSI/ISO semantics.
- For Teradata SQL features that support both the ANSI/ISO SQL standard syntax and the Teradata SQL dialect, the Teradata SQL syntax is used.
- By default, character data is NOT CASESPECIFIC, except when the character set is GRAPHIC. GRAPHIC character data by default is CASESPECIFIC.
- The default case specification for character data is NOT CASESPECIFIC. The exception is data of type CHARACTER (n) CHARACTER SET GRAPHIC, which is always CASESPECIFIC.
- When a character string requires truncation, the string is truncated silently with no truncation notification given. This is true for truncation of pad or non-pad characters.

- Cursors are never positioned. Updating or deleting the most current fetched cursor row using the UPDATE/DELETE ... WHERE CURRENT OF statement is not valid.
- The default table type depends on whether the table has a primary index. Tables with a PI are SET (do not allow duplicate rows). Tables without a primary index (including column-partitioned tables) are MULTiset (allow duplicate rows).
- The TRIM function trims only trailing pad characters by default.

Note:

Case specificity does not apply to CLOBs.

A transaction is a logical unit of work. The statements nested within the transaction either execute successfully as a group or do not execute. A Teradata SQL transaction can be a single Teradata SQL statement, or a sequence of Teradata SQL statements.

Transaction Processing in ANSI Session Mode

In ANSI mode, transactions are always implicitly started and explicitly closed.

A transaction initiates when one of the following happens:

- The first SQL request in a session executes
- The first request following the close of a previous transaction executes

The COMMIT or ROLLBACK/ABORT statements close a transaction.

If a transaction includes a DDL statement (including DATABASE and SET SESSION, which are considered DDL statements in this context), it must be the last statement in the transaction other than the transaction closing statement.

If an error is found in a request, then that request is aborted. Normally, the entire transaction is not aborted; however, some failures will abort the entire transaction. Locks placed by erroneous statements are not released.

Sessions in ANSI mode do not support BEGIN TRANSACTION/END TRANSACTION statements.

Transaction Processing in Teradata Session Mode

In Teradata mode, transactions can be either implicit or explicit.

A request that is not enclosed by the BEGIN TRANSACTION and END TRANSACTION statements is treated as an implicit transaction.

An implicit transaction can be one of the following:

- A single DML statement
- A macro or trigger
- A multistatement request

DDL statements are not valid in a multistatement request and are therefore not valid in an implicit transaction composed of a multistatement request.

Explicit transactions contain one or more statements enclosed by BEGIN TRANSACTION and END TRANSACTION statements. The first BEGIN TRANSACTION initiates a transaction and the last END TRANSACTION terminates the transaction. Explicit transactions can contain multistatement requests.

When multiple statements are included in an explicit transaction, you can specify only a DDL statement if it is the last statement, and is immediately followed by END TRANSACTION.

If an error occurs, the request fails, the entire transaction is aborted and all locks are released.

Sessions in Teradata mode do not support the COMMIT statement. The END TRANSACTION statement terminates a transaction with commit, while the ABORT and ROLLBACK statements terminate a transaction with a rollback.

Setting the Session Mode

A system parameter specifies the default transaction mode for a site. You can override the default transaction processing mode for a session using one of the following:

- SessionMode field of the DBS Control Record
- BTEQ command .SET SESSION TRANSACTION
- Preprocessor2 TRANSACT() option
- Session Mode field of the Teradata ODBC Driver Options
- JDBC TeraDataSource.setTransactMode() method

Related Information

Topic	Resources for Further Information
<ul style="list-style-type: none"> • Transaction, request, and statement processing • Transaction processing in ANSI and Teradata session modes 	<i>Teradata Vantage™ - SQL Request and Transaction Processing</i> , B035-1142
SQL statements for processing transactions: <ul style="list-style-type: none"> • BEGIN TRANSACTION/END TRANSACTION • COMMIT • ABORT • ROLLBACK 	<i>Teradata Vantage™ - SQL Data Manipulation Language</i> , B035-1146
Setting the Session Mode field of the DBS Control Record	<i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Using the SESSION TRANSACTION BTEQ command	<i>Basic Teradata® Query Reference</i> , B035-2414
Setting the Session Mode Field in the Teradata ODBC Driver Options dialog box	<i>ODBC Driver for Teradata® User Guide</i> , B035-2526

Checking for Logged On Sessions

You may need to check if sessions are running on the system to warn users when you want to bring Teradata Vantage down or disable logons. There are several ways to check if there are established sessions on the system.

- Run the Query Session utility (see “Query Session (qrysessn)” in *Teradata Vantage™ - Database Utilities*, B035-1102). If there are sessions, it will report as follows:

```
Session state query results : 08/07/16 14:28:59
```

Host	Session	PE	DBC User ID
-----	-----	-----	-----
1	1015	16383	JohnSmith

```
State details : IDLE
```

- Use BTEQ or Teradata Studio
- Run a select on the DBC.SessionInfoV view.

Obtaining Session Information

To find information about a session, for example CPU or I/O data for a specific session, try one of the following tools.

- Teradata Viewpoint** – Using Teradata Viewpoint, you can view information about queries running in a Vantage system so you can spot problem queries. You can analyze and decide whether a query is important, useful, and well-written.
- PM/APIs** – If you write your own application, use the MONITOR SESSION request or the MonitorSession function to collect current status and activity information for sessions that are logged on to Vantage. PM/APIs allow you to immediately access session information but you must have proper privileges to submit the requests or use the functions. For more information, see *Teradata Vantage™ - Application Programming Reference*, B035-1090.
- DBC.AmpUsage view and Account String Expansion (ASE)** – Use ASE to generate 1 row into DBC.AmpUsage per AMP per session. Note that the AmpUsage rows are not written until the query completes. For more information on ASE, see [Logging Resource Usage Data with Account String Variables](#).
- DBC.SessionInfo view** – Use this view to see the client interface that established the session, the logon date and time, and current role among other information. There is one row for each time a user is logged on.
- DBQL information** – You must have first already enabled logging by submitting the BEGIN QUERY LOGGING statement for the users/accounts/applications you wish to monitor.

For more information on this topic, see [Determining the Account for a User Session](#).

Troubleshooting Problems with Sessions

When a session appears idle, it is also referred to as a *hung* session. Hung sessions can be a result of problems with lock contention, a transaction rollback on a job with an error, the TDP losing communication with Vantage, or some other problem. The tools listed in the following table can help troubleshoot hung sessions.

Utility	What It Determines
Query Session (qrysessn)	The status of each session. For more information on what the different statuses mean, see "Query Session (qrysessn)" in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102.
Recovery Manager (rcvmanager)	If the system is rolling back any transaction. See "Recovery Manager (lokdisp)" in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102. Note: Some rollbacks can take a long time (several hours) to process because every TJ on every AMP must be accessed and updated.
Lock Display (lokdisp)	Transaction locks blocking the system. See "Lock Display (lokdisp)" in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102.
Show Locks (showlocks)	If there are host utility locks or HUT locks. You may have determine if there is a lock on a certain table cleared by release lock override and system running normally. See "Show Locks (showlocks)" in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102.

If there is an error during an archive or the archive job is killed prematurely, the outstanding locks from the job may block other sessions. To resolve this problem:

1. First run Query Session (qrysessn) to determine which session are blocked.
2. Then run Recovery Manager to see if anything is in recovery.
3. Next, check to see if there are any locks present and what kind of locks they are.
 - If you notice DBC has a WRITE lock on anything, run the CheckTable utility to determine the state of the tables. If any tables are corrupted, try dropping and restoring the table.
 - If there are DSA jobs that are hung, release the locks by using the RELEASE LOCK (databasename) ALL, OVERRIDE statement.
4. Use QuerySession to check whether formerly blocked sessions are now active.

Sometimes, you may have to abort the session and restart the system (tpareset) to clear a hung session. However, first contact Teradata Support and determine if you should take a crashdump before restarting the system so that you can try isolating the problem causing the block.

You may run across problems with FastLoad or MultiLoad sessions. This is usually indicated by either a 2633, 2574, or 2738 error. Use the DBC.SessionInfoV view to determine the host number, username, and partition associated with the session number.

If a session experiences an SQL error and the system attempts to roll back the transaction, and other sessions become blocked during this time, you may need to issue aborts using Teradata Viewpoint. See Help in Teradata Viewpoint for information on how to issue an abort.

For information on how to handle a slow or hung job, see the sections beginning with [Investigating Query Blocks and Delays](#).

Working with Accounts

Purpose of System Accounting

System accounting serves three important administrative functions:

- Charge-back billing (for equitable cost allocation)
- Capacity planning (to anticipate your needs)
- Resource management (to identify CPU and I/O usage)

For more information on how to identify and analyze session behavior and apply resource control, see [Managing Database Resources: Operational DBAs](#).

Charge-back Billing

You may need to charge users for their use of resources. Charge-back billing permits objective cost allocation of system resources across all users.

The account string enables you to summarize resource usage by account ID. The system table DBC.Acctg tracks CPU use in seconds and I/O resources expended by a session. The I/O resource tracks the number of AMP to disk read and write operations generated by a given user or account, and charges them to the current account associated with a session.

Use the DBC.AMPUsage view to access aggregations of DBC.Acctg contents.

Resetting the Resource Usage Counters

A systems administrator can use the CPU usage information in DBC.Acctg to bill users every month. At the beginning of each month, the systems administrator runs ClearAccounting to reset the resource usage counters back to zero:

```
EXEC DBC.ClearAccounting;
```

Capacity Planning

To plan for the resources needed to accommodate growth, you must know how the current workload is affecting the system. To assess the effect of the current workload, you can collect and analyze information about resource utilization.

Collecting and analyzing current resource usage information is one component of data analysis but another valuable component is the collection of historical usage statistics. The accounting feature can

be used to determine the activity on current workloads, which assists you in anticipating future needs. For information on capacity planning for a new system or database, see *Teradata Vantage™ - Database Design*, B035-1094.

Workload Management

You may need to control who gets specific system resources using the Workload Designer portlet in Viewpoint. For additional information, see [Managing the Database Workload](#).

Creating Accounts

For information on how to create an account, see [Creating User Accounts](#).

Working with System Accounting Views

Teradata Vantage provides the following views to support administration of accounts and accounting functions:

- DBC.AccountInfoV
- DBC.AMPUsage

By auditing the information in these views, you can see which users heavily or rarely use the system. Or, you can make adjustments to the priority assigned to specific accounts.

DBC.AccountInfoV

The DBC.AccountInfoV view accesses the DBC.Dbase, DBC.Profiles, and DBC.Accounts dictionary tables to return information about all valid accounts for all databases, users, and profiles.

Use DBC.AccountInfoV to find out:

- What accounts are valid for which user
- The assignment of account string expansion variables. For more information, see [Logging Resource Usage Data with Account String Variables](#)

Dictionary Table	What It Stores
DBC.Accounts	All accounts for all databases, users, and profiles. If an account is not specified at creation time, the default is used. DBC.Accounts is used to verify any account entered by a user at logon time or with a SET SESSION ACCOUNT statement.
DBC.DBase	The default account for each database and user. If a database or user is defined with multiple accounts, the first is used as the default. If a user is assigned a profile that is defined with one or more accounts, the first profile account is used as the default. (Profile accounts take precedence over user accounts.)
DBC.Profiles	The default account for each profile. If multiple accounts are defined, the first is used.

Dictionary Table	What It Stores
	Note: If an account is not specified for a profile, the value is NULL in the DefaultAccounts field for that profile.

Example SELECT on DBC.AccountInfoV

The following query selects all account strings with a Timeshare Top priority code, and whether the name associated with an account is an individual user or a profile:

```
SELECT AccountName,UserName,UserOrProfile
FROM DBC.AccountInfoV
WHERE AccountName LIKE '$R%'
ORDER BY AccountName ;
```

In this example, the view returns:

AccountName	UserName	UserOrProfile
-----	-----	-----
\$R1\$ABCD&D&H	AcctsRecv	Profile
\$R1\$EFGH&D&H	DBADMIN	User
\$R2\$IJKL&D&H	DBC	User
\$R2\$MNOP&D&H	SysAdmin	User
\$R2&QRST&D&H	SystemFe	User

DBC.AMPUsage View

The DBC.AMPUsage view provides information about the usage of each AMP for each user and account. It also tracks the activities of any console utilities. By user, account, or console utility session, DBC.AMPUsage stores information about:

- CPU seconds consumed
- Number of read/write (I/O) operations generated

Note:

AMPUsage reports logical I/Os explicitly requested by the database software, even if the requested segment is in cache and no physical I/O is performed.

DBC.AMPUsage uses the DBC.Acctg table to provide aggregated information by username, account ID, and AMP. Updates to the table are made periodically during each AMP step on each processor affected by the step. (This means if there are long-running steps, AMPUsage numbers show large

increases periodically, instead of continuous incremental additions.) The data is collected and continually added to what is already in the table until you reset the counters to zero (see [Resetting the Resource Usage Counters](#)).

You can use the information provided by DBC.AMPUsage to do the following:

- Bill an account for system resource use.
- Determine what resources were used, by user and account string, after hours as well as during the day.
- Summarize and archive the information, then zero it out on a per shift, per day, or per week basis.
- Determine which session caused reduced performance (in-depth analysis can be performed with DBQL data).
- Derive capacity needs to plan for expansion.

DBC.AmpUsage does not record the activity of parsing the query, or of processing on a query basis.

You can use query logging to capture query text, step information, and elapsed processing time, and to differentiate queries submitted by SQL-generating products that do not provide a variety of user IDs and account ids in the logon string. For instructions and a description of the data capture options, see [Tracking Query Behavior with Database Query Logging: Operational DBAs](#).

Use Teradata Viewpoint for a look at up-to-the-moment activity in real-time.

Example: Totaling CPU Time and I/O by User

This SQL statement requests totals for CPU time and I/O for user DBA01.

The totals are aggregates of all resources.

```
SELECT UserName (FORMAT 'X (16)')
,AccountName (FORMAT 'X (12)')
,SUM (CpuTime)
,SUM (DiskIO)
FROM DBC.AMPUsage
WHERE UserName = 'DBA01'
GROUP BY 1, 2
ORDER BY 3 DESC ;
```

For this example, AMPUsage returns the following rows:

UserName	AccountName	SUM (CpuTime)	SUM (DiskIO)
-----	-----	-----	-----
DBA01	\$M2\$ABCD&D&H	6,336.76	505,636
DBA01	\$M2\$EFGH&D&H	4,387.14	303,733
DBA01	\$M2\$IJKL&D&H	1.28	166

For detailed information on these and all the system views, see *Teradata Vantage™ - Data Dictionary*, B035-1092. For more information on how to use DBC.AMPUsage and other views to find problems and improve performance, see [ASE Impact on PE and AMP Performance](#).

Logging Resource Usage Data with Account String Variables

You can log resource usage data in the DBC.Acctg table by creating substitution variables in the account ID part of the user logon string. These Account String Expansion (ASE) variables let you to measure query and AMP usage more precisely. You can gather statistics for capacity planning, gather information for use in charge back and accounting, and track resource consumption.

The variables allow you to include date and time information in the string. Using ASE is a best practice. If ASE is activated, Teradata Vantage collects statistics based on the variables used during the active session, and stores the information into the underlying table of the DBC.AmpUsage view.

Enabling Account String Expansion

Before a user can activate any of the ASE functions, there must be a matching account string in the DBC.UsersV view for that user and ASE function. This means that the user cannot simply add the ASE variables to any logon account string unless first authorized by the DBA. This permits a level of control over users and programmers who may or may not use ASE properly.

For example, to create a user that can use ASE, define the variables in his account string or the profiles that he will use.

The DBA or any user with CREATE USER privileges can define the following:

```
CREATE USER john FROM finance AS PERM = 100000, ACCOUNT =
'$H00MSIR&D&H', '$M00MSIR&D&H';
```

While the recommended ASE variable combination is &D&H, others include time (&T) and the time of log on (&L), or a combination thereof. Teradata no longer recommends use of the session variable (&S). For more information on the different variables, see [ASE Substitution Variables](#).

Setting the Default Account String to Use ASE

To enable ASE, define one or more of the ASE variables in an account string either directly in the user definition (with the CREATE USER statement), or in the definition of the profile assigned to the user (with the CREATE PROFILE statement). You should set up the account to default to automatically use ASE.

Examples

You assign two account strings to user TAW. The first entry is the default.

```
MODIFY USER TAW AS ACCOUNT=(' $M00ACTB&D&H', '$H00FINR&D&H');
```


When TAW logs on without specifying an account, TAW will default to using the first account. At query execution time, Vantage replaces &D with the current date and &H with the current hour.

In another case, you assign two account strings to user DTG. However, the first (default) account string does not specify date or hour expansion:

```
MODIFY USER DTG AS ACCOUNT = ('$M00acc0', '$L00accB&D&H');
```

When DTG logs in without specifying an account, then the ASE logging *does not* occur. To initiate them, DTG must type &D&H in the account string either when he logs in or during the session:

- Log in using *.logon DTG, mypassword, '\$M00accO&D&H'*
- Change the account during a session with the following statement: *SET SESSION ACCOUNT='\$M00accO&D&H'*

Because assigning the first account to DTG without ASE variables means ASE logging does not occur by default, Teradata does not recommend listing the account without &D&H in the account string first.

In another case, assume you omit the ASE variables in the two account strings for user AM1:

```
MODIFY USER AM1 AS ACCOUNT=('$M00accR', '$H00accT');
```

AM1 cannot invoke the ASE feature at any time. Teradata does not recommend assigning the first account string without the &D&H variables.

ASE Substitution Variables

The system resolves these variables at logon or at SQL statement execution time. For example, \$M00MARB&D becomes \$M00MARB170704 for a logon at July 4, 2017, under the MAR batch (B) account.

The ASE variables may be used in any combination and in any order, subject to the constraints on length and position:

- The maximum length of an account string is 128 characters.
- The increasing levels of granularity that result when additional rows are written to the DBC.Acctg table.

There is a minor increase in overhead associated with managing the frequency with which the DBC.Acctg table is cleared.

Teradata recommends that if you use DBQL, you should only use &D and &H jointly (&D&H).

Variable	Format	Description	Length
&D	YYMMDD	Date. The system substitutes the date it received the SQL request into the account string.	6
&H	HH (24 hour clock)	Hour. The system substitutes the hour of day it received the SQL request. This is useful for identifying the large resource users during peak periods. In charge back systems, use &H to	2

Variable	Format	Description	Length
		give users preferential billing rates for running queries during off-peak time periods. If you use the &H variable without the &D variable, the system sums statistics collected for a specified hour on one day with existing statistics for the same hour on other days.	
&I	LLLLSSSSSSSSSSRRRRRRRRR	The system substitutes the logon host ID, current session number, and sequential request number. Note: All the queries within one stored procedure CALL command are reported under one request number. The request number is the client request number, which is the request of the CALL. If you need more detail about queries within a stored procedure, the information will be available in DBC.DBQLogTbl.RequestNum.	22
&L	YYMMDDHHMMSS.hh	Logon date-time substitution. The system substitutes the logon timestamp. This value does not change until the user logs off and then logs on again. Because there is only one logon string for each session pool, the &L option generates only one row per session, regardless of the number of users connected to the pool. If a group of users share user IDs and passwords, the system accumulates all DBC.AMPUsage statistics under the same user ID. Use the &L option to generate separate statistics and to monitor the LogonSource field of DBC.LogOnOffV.	15
&T	HHMMSS (24 hour clock)	Time substitution. The system substitutes the time of day it received the SQL request. Note: Using &T can be very resource intensive. If you notice an impact on system performance, delete rows from DBC.Acctg and discontinue using &T. This variable allows for one second granularity, causing the system to write a row for virtually every individual SQL request. If the system receives two or more SQL requests for the same user/account ID pair in the same second, the system sums AMP usage statistics. This summation can be any combination of subsecond requests, or a combination of subsecond requests with a longer request.	6

Variable	Format	Description	Length
		<p>If the system receives a multistatement request, each individual SQL statement in the request has the same timestamp; therefore, the row written to DBC.AMPUsage contains the summation of the statistics of the individual statements.</p> <p>Using the &T variable without the &D variable causes the system to sum statistics collected for a specified time on one day with existing statistics for the same time on other days.</p>	

Restrictions and Usage Rules

The following requirements can influence the standard formats of ASE:

- Account strings cannot exceed 128 characters.
- The character count includes separation characters, such as colons (:) in time fields and slashes (/) in dates. If a string consists of all the ASE variables, the result is 32 characters long. If an account string exceeds the character limit, characters to the right of the limit are truncated.
- You can intersperse ASE variables with literals, subject to the constraints of length.
- To the greatest extent that is appropriate, capture request-level usage detail. If DBQL is enabled, however, ASE becomes less important as a mechanism for enabling request-level usage detail because DBQL can capture usage for queries.

Recommendation for account string setup: `$W00MS/R&D&H`, where:

`$W00` = Workload performance category. This optional classification is in effect only when TASM or TIWM classification processes cannot match the request to a user-defined workload.

Note:

This default classification capability exists to be backward-compatible with older versions of software and to help users transition easily to SLES 11 Priority Scheduler. For users of current software it is recommended that normal TASM/TIWM workload classification be relied upon, rather than this default approach.

Possible values include:

- `$R00`: Timeshare Top
- `$H00`: Timeshare High
- `$M00`: Timeshare Medium
- `$L00`: Timeshare Low

`MSI` = Application identifier, such as `FIN` for Finance

`R` = Workload identifier, such as `R` (Reporting), `B` (Batch), `T` (Tactical), or `O` (Online)

&D&H = ASE Variables for date and hour

- For requests requiring short response time, the account string setup should not materially impact the performance of the request.
- Provide the detailed information necessary to effectively manage the system. When consistently adhered to, the proper information will be captured to facilitate a wide variety of analyses and produce a set of standard metrics by which to measure the system.
- When combining ASE variables, keep the following in mind:
 - You can use multiple ASE variables in any order in the account string following the \$W00 variable.
 - The account string limit is 128 characters.
 - If you specify &H or &T without &D, statistics collected on one day at one time are combined with statistics collected on other days at that same time.

Using ASE With Client Utilities

Except for the utilities and variables noted in the table that follows, you can use ASE with any utility that uses a standard Teradata Vantage interface to log on, including:

- BTEQ
- FastLoad
- MultiLoad
- Teradata Parallel Transporter
- TPump (except for &T)
- FastExport
- Teradata Studio

The exceptions are described in the following table.

Do not use ...	With ...	Because ...
TPump	&T	&T generates a row in AMPUsage for nearly every SQL statement.
PMPC statement	any ASE code	requests generated are not parsed and substituting variables cannot be expanded.

Managing Space: Operational DBAs

This section describes space management:

- Considerations for setting up disk space limits
- How to find and resolve system, database, and user space issues
- How to find and fix skewed tables
- Creation of a macro for space usage reporting
- Methods of viewing space utilization
- How to manage peak values, data blocks, and cylinders

Types of Space

Space (whether permanent, spool, or temporary) is measured in bytes. The following table lists the three types of database space.

Type of Space	Description
Permanent (PERM)	<p>Permanent space allocated to a user or database is a repository for database objects, for example, tables, indexes, and journals. Perm space can be defined as a fixed quantity or as a constant expression.</p> <p>When an object is created or data rows are inserted, the system allocates space as needed from the PERM space of the immediate owner. The space is returned automatically when no longer needed.</p> <p>Note:</p> <p>Perm space allocated to a user or database that is currently unused is available for use as spool or temporary space.</p> <p>A database or user with no PERM space can still own views, macros, and triggers but cannot have objects that require space such as tables, UDFs, stored procedures, HIs, JIs, or journals.</p>
Spool	<p>Spool space holds intermediate query results or formatted answer sets to queries and volatile tables. The system can use unassigned PERM space for spool space.</p> <p>When a user creates new users or databases, the amount of spool space for those new objects must not exceed the spool space limit of their immediate owner. If you do not specify a spool space limit, a new user automatically inherits the spool limit of its parent.</p> <p>To more easily manage spool space, define the value for spool in a profile. You can then assign a profile to users and all of the users will inherit the spool space definition.</p>
Temporary (TEMP)	<p>Temp space defines the number of bytes the system will use to store data for global temporary tables. The value you specify must not exceed the value of the immediate parent at the time of creation. If you do not specify a value, the maximum value defaults to that of the parent.</p>

Type of Space	Description
	<p>Note:</p> <p>Global temporary tables require both PERM and TEMP space. The database/user must have adequate perm space to accommodate the global temporary table header on each AMP. Table header size varies by table definition and the maximum size for a table header is 1 MB. Temp space is required for storing the actual rows.</p> <p>To more easily manage TEMP space, define the value for it in a profile. You can then assign a group of users to this profile and all of the users will inherit the TEMP space definition.</p>

You can manage space using Teradata SQL, or with Teradata Viewpoint administrative tools.

Capacity Planning

This section covers space issues for an already running system. For information on capacity planning for a new system or database, see *Teradata Vantage™ - Database Design*, B035-1094.

Global Space Accounting

DBAs can adopt one of two strategies for managing permanent, spool, and temporary space:

- AMP level only. The per-AMP space quota is the maximum permissible AMP-level space. This is the default.
- Global level, which maintains both an overall system limit and AMP limits. This strategy can provide extra space to AMPs when needed by temporarily reducing space for AMPs that do not need the space now.

When DBAs manage space at the AMP level, transactions or long-running load jobs abort when space use exceeds hard limits set for the database or user. If a DBA manages space at the global level, two allowances can be made to increase space to AMPs when needed:

Space Allowance	Creation Method	Description
Skew factor	<p>The SKEW option in a CREATE USER/DATABASE or MODIFY USER/DATABASE request.</p> <p>If the SKEW option is not specified or is specified as DEFAULT, the skew factor is controlled by three DBS Control utility fields:</p> <ul style="list-style-type: none"> • DefaultPermSkewLimitPercent: Controls the skew factor on permanent space • DefaultSpoolSkewLimitPercent: Controls the skew factor on spool space • DefaultTempSkewLimitPercent: Controls the skew factor on temporary space 	<p>Gives extra space to one or more AMPs when needed, up to the value of the SKEW option. The skew option can be defined as a percentage or a constant expression. You can apply this option to PERM, SPOOL, and TEMP space.</p> <p>A skew limit value of 0 means that the AMP space quota is the space limit, and Vantage does not provide more space to AMPs, even when other AMPs have space available. A non-zero soft limit percent permits this quota to increase by the soft limit percent. Users receive an error if actual space use exceeds</p>

Space Allowance	Creation Method	Description
		the soft limit. The default value for the default skew limit percent fields is 0. The DBC.GlobalDBSpace table records the cumulative allocations of the AMPs and maintains database and user global-level space limits.
Global soft limit	Set a non-zero value for the GlobalSpaceSoftLimitPercent field in the DBS Control utility	The percentage by which a database or user is allowed to exceed the maximum space limit. This setting applies to PERM, SPOOL, or TEMP space. The default value is 0. The system sends alerts when processing exceeds soft limits, so the DBA can take corrective action. Processes that cannot be aborted (such as transaction recovery) may be in progress when the soft limit is exceeded or when physical storage limits are almost reached. In this case, the system continues processing but reports alerts with higher severity levels. The DBA is expected to increase affected space as soon as possible in this situation.

Note:

Teradata recommends that you change the values of the default skew limit percent fields and the GlobalSpaceSoftLimitPercent field only under the direction of Teradata Support Center personnel.

Create a User with a Skew Limit

The following example creates a user with a 10 percent skew limit for permanent space and a 20 percent skew limit for spool space:

```
CREATE USER Caspian AS PERM = 1e9 SKEW = 10 PERCENT, SPOOL = 2e9 SKEW = 20 PERCENT;
```

In the preceding example, if the system has 4 AMPs, each AMP has a limit of 250 MB of permanent space. With the permanent skew limit of 10 percent, any AMP can exceed the permanent space limit by 25 MB. An AMP may use up to 275 (250 + 25) MB of permanent space, as long as the total permanent space used by all AMPs does not go beyond the 1 GB global limit.

Similarly, each AMP has a limit of 500 MB of spool space. With the spool skew limit of 20 percent, any AMP can exceed the spool space limit by 100 MB. An AMP may use up to 600 (500 + 100) MB of spool space, as long as the total spool space used across all AMPs does not exceed the 2 GB global limit.

Considerations for Global Space Accounting

Consider the following factors about global space accounting:

- Teradata recommends global space accounting for a database or user when actual space use is expected to be non-uniform, for example, when large, unknown data sets that may be skewed are often loaded into the underlying tables or when the database or user has stored procedures or UDFs that record very few rows with object-specific data.
- AMP-level space accounting does not have the overhead of dynamic, need-based space allocation and is recommended when the database is uniformly distributed across all AMPs in the system or when the global limit is high enough to allow for variability in data distribution due to skew .
- An unlimited skew limit for spool space may not be a good option, since design issues or a lack of statistics may cause a request to use excessive spool on some AMPs. A lower spool limit causes requests using excessive spool space to abort.
- To reduce the chance of space shortages causing aborted transactions, consider setting a higher value for the global soft limit if the system has long-running transactions that consume a lot of space.
- Skew limits cannot solve space problems, especially when actual use is close to the limits. If there is not enough space, it is best to increase the space limits instead of allowing more skew.

Interaction between Skew Factor and a Profile

- Profile values supersede user values. For example, if a spool limit is defined in both the CREATE USER request and the profile a user is a member of, Vantage uses the profile value.
- If a user is a member of a profile, the skew factor is controlled by the user definition.

For example, if a CREATE USER request gives user Joe 10 MB spool with 20% skew, but the profile Joe belongs to provides 5 MB spool, then Vantage applies the 5 MB spool limit with 20% skew. The same is true for temporary space specifications.

Interaction between the Skew Factor and Soft Limits

Skew factor and soft limits can be cumulative. For example, consider the following numbers:

- Global limit: 800 MB
- Soft limit: 10 percent
- Per-AMP limit on a 4-AMP system: 200 MB
- Skew factor: 25 percent

With these numbers, the global limit can exceed the 800 MB limit by 10 percent (to 880 MB). In addition, an AMP can exceed the 200 MB limit by 25% (to 250 MB) if the total across all AMPs does not exceed 880 MB. Each of the AMPs can also receive an additional 20 MB because of the global soft limit. The cumulative effects of the skew factor and global soft limit can increase the maximum possible space allotment to an AMP to $250 + 20 = 270$ MB, as long as the total space used across all AMPs does not exceed 880. The user will receive an error message if either of the following limits are exceeded:

- The global limit of 880
- The per-AMP limit of 270

Related Information

For more information on...	See...
the SKEW option for DDL in CREATE DATABASE and CREATE USER or MODIFY DATABASE and MODIFY USER	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144.
DBS Control fields	<i>Teradata Vantage™ - Database Utilities</i> , B035-1102.

Fixing Issues with Space Accounting

Two SQL stored procedures, `FixAllocatedSpace` and `FixCurrentSpace`, can resolve issues that occur with space accounting. These procedures correct the space usage and allocations for a database when numbers in the `DBC.DatabaseSpace` table do not reflect the actual space and allocations.

FixAllocatedSpace Procedure

Deallocates unused permanent, temporary, and spool space allocated for a database that uses global space allocation. It can update space for one or more databases.

Fixes the inconsistencies between the AMP-level space allocations in the `DBC.DatabaseSpace` table and the global-level space allocations in the `DBC.GlobalDBSpace` table.

Syntax

```
CALL [SYSLIB.] FixAllocatedSpace (
  'FixSpaceOption',
  'AllDatabases',
  'AllProxyUsers',
  'FixDatabaseName',
  FixedDatabaseCount
  Errinfo
) [;]
```

Syntax Elements

Syntax Element	Description
SYSLIB	The name of the database where the function is located.
<i>FixSpaceOption</i>	The type of allocated space to be updated. Specify one of the following options: <ul style="list-style-type: none"> P (permanent) S (spool) A (all options) It cannot be NULL.

Syntax Element	Description
<i>AllDatabases</i>	Update allocated space for all databases. Specify one of the following values: <ul style="list-style-type: none"> • Y • N • NULL
<i>AllProxyUsers</i>	Update allocated space for all proxy users. Specify one of the following values: <ul style="list-style-type: none"> • Y • N • NULL Only the proxy users with an assigned profile qualify for the fix space operation.
<i>FixDatabaseName</i>	Target database name. It is required if both <i>AllDatabases</i> and <i>AllProxyUsers</i> are not specified (have a value of N or NULL). <i>FixDatabaseName</i> should be NULL if either <i>AllDatabases</i> or <i>AllProxyUsers</i> has a value of Y.
<i>FixedDatabaseCount</i>	Output count of databases whose allocated space was fixed.
ErrInfo	This is NULL if the fix space operation was successful. Otherwise, it provides information about the error that caused the procedure to stop before the operation completed.

Authorization

You must have EXECUTE privileges on the stored procedure or on the SYSLIB database.

Argument Types

Expressions passed to this procedure must have the following data types:

Data Type	Value
<i>FixSpaceOption</i>	CHAR(2) CHARACTER SET LATIN
<i>AllDatabases</i>	CHAR(1) CHARACTER SET LATIN
<i>AllProxyUsers</i>	CHAR(1) CHARACTER SET LATIN
<i>FixDatabaseName</i>	VARCHAR(128) CHARACTER SET UNICODE

Usage Notes

Extra space may be allocated to AMPs in expectation of future work, which may not arrive. This extra space may be needed by other AMPs for the same database or by another database in the same AMP. This unused space may increase over time, and the automatic deallocation that is supposed to occur may not have happened for some reason. Use this procedure to return unused space to the global pool.

When the current space usage is 0, the allocated space is not deallocated, even after running this procedure.

When To Use This Procedure

This procedure applies only to databases that are using global space accounting. Use this procedure after executing the SPAAGGRTRACKINFOGLOBAL UDF if the results of that UDF show any inconsistency between allocated space values across vprocs and the total allocated value in DBC.GlobalDBSpace. For example, in the following, the sum of AllocatedPerm across all AMPs is not consistent with the allocated space of DBC.GlobalDBSpace:

```
select base.databasename databasename,
       sum(dbSPACE.allocatedpermSPACE) dbSPACEallocperm,
       syslib.SPAAGGRTRACKINFOGLOBAL(cast(dbglocal.Trackinfo as
byte(3880)), 'P') delta,
       dbglocal.allocatedpermSPACE globalallocperm
from dbc.databasespace dbSPACE, dbc.globaldbSPACE dbglocal, dbc.dbase base
where dbSPACE.databaseid=dbglocal.databaseid and
dbSPACE.databaseid=base.databaseid and base.databasename='db1'
and dbSPACE.tableid='00000000'XB
group by 1,3,4;
```

Result:

```
*** Query completed. One row found. 4 columns returned.
*** Total elapsed time was 1 second.
```

databasename	db1	
dbSPACEallocperm		1,999,900
delta		100
globalallocperm		2,000,000

Example

The following example shows how to execute FixAllocatedSpace to update all the allocated space fields for the database Travel.

```
call SYSLIB.FixAllocatedSpace('A', 'N', 'N', 'Travel', cnt, err);
```

FixCurrentSpace Procedure

Counts the permanent, temporary, spool, and persistent spool space used by each object in a database. For databases that use global space allocation, the allocations across the AMPs are fixed. If any AMP has excessive space, this procedure deallocates space appropriately.

Use this procedure to fix phantom spool problems or correct inconsistencies in the DBC.DATABASESPACE table, which might occur because of rare types of system failures.

Syntax

```
CALL [SYSLIB.] FixCurrentSpace (
  'FixSpaceOption',
  'AllDatabases',
  'AllProxyUsers',
  'FixDatabaseName',
  FixedDatabaseCount
  Errinfo
) [;]
```

Syntax Elements

Use the following syntax elements for FixCurrentSpace:

Syntax Element	Description
SYSLIB	The name of the database where the function is located.
<i>FixSpaceOption</i>	The type of current space to be updated. Specify one of the following options: <ul style="list-style-type: none"> • P (permanent) • T (temporary) • S (spool) • PS (persistent spool) • A (all options) It cannot be NULL.
<i>AllDatabases</i>	Update current space for all databases. Specify one of the following values: <ul style="list-style-type: none"> • Y • N • NULL
<i>AllProxyUsers</i>	Update current space for all proxy users. Specify one of the following values: <ul style="list-style-type: none"> • Y • N • NULL Only proxy users with an assigned profile qualify for the fix space operation.
<i>FixDatabaseName</i>	Target database name. It is required if both AllDatabases and AllProxyUsers are not specified (have a value of N or NULL). FixDatabaseName should be NULL if either AllDatabases or AllProxyUsers has a value of Y.
<i>FixedDatabaseCount</i>	Output count of databases whose current space was fixed.
ErrInfo	This is NULL if the fix space operation was successful. Otherwise, it provides information about the error that caused the procedure to stop before the operation completed.

Authorization

You must have EXECUTE privileges on the stored procedure or on the SYSLIB database.

Argument Types

Expressions passed to this procedure must have the following data types:

- *FixSpaceOption*= CHAR(2) CHARACTER SET LATIN
- *AllDatabases*= CHAR(1) CHARACTER SET LATIN
- *AllProxyUsers*= CHAR(1) CHARACTER SET LATIN
- *FixDatabaseName*= VARCHAR(128) CHARACTER SET UNICODE

Usage Notes

FixCurrentSpace performs the same functions for space fields as the UpdateSpace utility. Because this procedure both calculates current space use and deallocates unused space, it may take some time. FixCurrentSpace uses AMP worker tasks, unlike the UpdateSpace utility, which has separate, dedicated tasks executing the work. For this reason, consider using the UpdateSpace utility instead of the FixCurrentSpace procedure if there is heavy workload in the system.

Example

The following example shows how to execute the procedure to update all the current space fields for all databases.

```
call SYSLIB.FixCurrentSpace('A', 'Y', 'N', null, cnt, err);
```

Identifying and Correcting System-level Space Problems

Several Teradata Viewpoint portlets as well as some Teradata views provide space usage information, which you can use to identify space problems.

Using the System Health Portlet to Find System Space Issues

Use the System Health portlet to track current system-level space usage. This portlet displays segmented bar graphs that show normal, degraded and critical space usage ranges.

Parameter	Description	Actions
Total disk space	Percentage of total disk space in use across the system	If system disk space usage moves beyond the normal range (80%), add disk capacity.
DBC disk space	Percent of disk space assigned to DBC currently in use	The DBC database contains logs, system tables and views, and other functions necessary for system operation.

Parameter	Description	Actions
		If DBC space usage moves beyond the normal range, reallocate perm space from DBADMIN to DBC.

Identifying Space Problems Using Viewpoint Space Usage Portlet

Use the Space Usage portlet to track detailed space usage parameters, including available space, current usage, and peak usage for each space type, by database or user.

1. Select the system you want to investigate and click **Next**.
2. In the selection menu, select **Database: By most space** and click **Next**.
3. Use the **Database Total** view to display perm space usage for all databases that have assigned perm space. The following columns may indicate perm space problems.

Parameter	Description	Actions
Current Perm%	Percentage of available perm space currently being used by each user and database.	Consider allocating additional perm space to any user or database when it reaches 90% perm space usage. See Reallocating Perm Space to Resolve Space Issues .
Peak Perm%	The highest percentage of total available perm used by each user and database since the last reset (default 24 hours).	
Current Perm Skew%	A measure of how the perm space for a database distributes across all AMPs. Higher percentages indicate uneven distribution.	A high perm skew percentage indicates that one or more tables in the database is skewed, that is, it has a poorly designed primary index.

Note:

If you think there is a space problem, you can click on a database name to display the Details view, which shows the current perm, peak perm, and skew% for each table in the database.

4. Use the **Users Over 85% Spool** view to display any users that have exceeded the safe range of spool usage since the last reset.

Note:

In some cases excess spool usage may be the result of poorly constructed queries rather than an actual shortage of spool.

Parameter	Description	Actions
Current Spool%	Percentage of available spool space currently being used by each user and database (list)	Monitor regularly to determine whether certain users regularly exceed 85% spool. <ul style="list-style-type: none"> • Use Query Monitor to track query activity for users exceeding the limit, to determine whether the excessive spool usage is caused by poorly constructed queries. • Increase the spool space limit for users who need spool by modifying the user or applicable profile definition. See Increasing the Spool Space Limit for a User.
Peak Spool%	Largest percentage of available spool space recently used by each listed user and database since the last reset (default 24 hours).	

Identifying Space Issues by Querying System Views

Querying the DiskSpaceV View to Find System Space Issues

The DBC.DiskSpaceV view returns AMP information about disk space usage at the database or user level. It also can report spool space usage. DiskSpaceV figures are calculated only for the space owned by the user submitting the query. To find information about the full system, log on as the topmost user in the hierarchy (usually your site administrative user).

You can determine the amount of space allocated to a profile by querying the MaxProfileSpool and MaxProfileTemp columns of DBC.DiskSpaceV. By querying MaxSpool and MaxTemp of the restricted view (DBC.DiskSpaceVX), you get the space limits for your individual user space. The following queries show the kinds of information you can obtain from DBC.DiskSpaceV.

To find the...	Submit the following query...
total disk space in the entire system	<pre>SELECT SUM(MaxPerm) FROM DBC.DiskSpaceV;</pre>
disk space currently in use	<pre>SELECT SUM(CurrentPerm), SUM(MaxPerm), (((Cast(Sum(Currentperm) as float) / NullifZero(Cast(Sum(maxperm) as float)) *100)) (Title '%MaxPerm', Format 'zz9.99') FROM DBC.DiskSpaceV;</pre>
disk space for a given database	<pre>SELECT sum(maxperm) FROM DBC.DiskSpaceV WHERE databasename='xxxx';</pre>
percent of disk space available for spool	<pre>SELECT (((cast(SUM(MaxPerm) as float) - cast(SUM(CurrentPerm) as float))) / NULLIFZERO(cast(SUM(MaxPerm) as float))) * 100)</pre>

To find the...	Submit the following query...
	<pre>(TITLE '% Avail for Spool', format 'zz9.99') FROM DBC.DiskSpaceV;</pre>
percent of space used by each database in the system	<pre>SELECT Databasename (format 'X(12)') ,SUM(maxperm) ,SUM(currentperm) ,((cast(SUM(currentperm) as float))/ NULLIFZERO (cast(SUM(maxperm) as float)) * 100) (FORMAT 'zz9.99%', TITLE 'Percent // Used') FROM DBC.DiskSpaceV GROUP BY 1 ORDER BY 4 DESC WITH SUM (currentperm), SUM(maxperm);</pre>
users who are running out of PERM space	<pre>SELECT Databasename (format 'X(12)') ,SUM(maxperm) ,SUM(currentperm) ,((cast(SUM(currentperm) as float))/ NULLIFZERO (cast(SUM(maxperm) as float)) * 100) (format 'zz9.99%', TITLE 'Percent // Used') FROM DBC.DiskSpaceV GROUP BY 1 HAVING (cast(SUM(currentPerm) as float) / NULLIFZERO(cast(SUM(maxperm) as float))) > 0.9 ORDER BY 4 DESC;</pre>
users who are using a lot of spool	<pre>SELECT databasename ,SUM(peaks pool) FROM DBC.DiskSpaceV GROUP BY 1 HAVING SUM(peaks pool) > 5000000000 ORDER BY 2 DESC;</pre> <p>Note: You can change the value 5000000000 to whatever value is appropriate for your site. Some sites with more space may have higher tolerance for higher spool usage and spool limits.</p>

Querying the TableSizeV View

The DBC.TableSizeV view provides AMP information about disk space usage at the table level. Optionally use *viewnameVX* for information on only those tables that the requesting user owns or has SELECT privileges on.

To find...	Use the following query...
the table distribution	<pre>SELECT tablename (TITLE 'Table') ,currentperm (TITLE 'CurPerm')</pre>

To find...	Use the following query...
	<pre>,vproc (TITLE 'Amp') FROM DBC.tablesizeV WHERE databasename='xxx' AND tablename = 'xxxx' ORDER BY 2 DESC;</pre>
disk space for a given table	<pre>SELECT SUM(currentperm) FROM DBC.tablesizeV WHERE databasename='xxx' AND tablename = 'xxxx';</pre>

If you submit a SELECT statement against DBC.TableSizeV and notice it takes a long time to process, replace the view definition by submitting the following query:

```
REPLACE VIEW DBC.TableSizeV
AS SELECT DataBaseSpace.Vproc,
    Dbase.DatabaseName (NAMED DatabaseName),
    Dbase.AccountName,
    TVM.TVMName (NAMED TableName),
    DataBaseSpace.CurrentPermSpace(NAMED CurrentPerm,
FORMAT '---,---,---,---,--9'),
    DataBaseSpace.PeakPermSpace(NAMED PeakPerm,
FORMAT '---,---,---,---,--9')
FROM DBC.Dbase, DBC.DataBaseSpace, DBC.TVM
WHERE DataBaseSpace.TableID <> '000000000000'XB
AND DataBaseSpace.TableID = TVM.tvmid
AND TVM.DatabaseId = Dbase.DatabaseId
WITH CHECK OPTION;
```

This improved definition helps the Optimizer choose a better plan to process SELECT requests. (For more information, see Knowledge Article SD1000B999E.)

You may also want to look at the individual tables within the database to see if any are skewed.

Use of the AllSpaceV View

The DBC.AllSpaceV view provides space usage information at the object level (table, join index, permanent journal, or stored procedures) *and* the database/user level. However, Teradata recommends using the DBC.DiskSpaceV and DBC.TableSizeV views instead because DBC.AllSpaceV can return misleading results.

For example, the following query returns the current permanent space from the detail rows *and* summary rows for each user or database. Note that the view sums data from both types of rows:


```

SELECT DatabaseName
       ,Sum(CurrentPerm)
FROM DBC.AllSpaceV
GROUP BY 1 having sum(currentperm) > 0
ORDER BY 2 desc;

```

Result:

DatabaseName	Sum(CurrentPerm)
-----	-----
DBC	47,183,872
Sys_Calendar	2,648,064
SysAdmin	1,380,352
SYSLIB	319,488
SystemFe	148,480

However, a similar query using DBC.DiskSpaceV will return the desired result and, on most systems, will run faster:

```

SELECT DatabaseName
       ,Sum(CurrentPerm)
FROM DBC.DiskSpaceV
GROUP BY 1 having sum(currentperm) > 0
ORDER BY 2 desc;

```

Result:

DatabaseName	Sum(CurrentPerm)
-----	-----
DBC	23,591,936
Sys_Calendar	1,324,032
SysAdmin	690,176
SYSLIB	159,744
SystemFe	74,240

The view DBC.DiskSpaceV includes only summary rows and the view DBC.TableSizeV includes only detail rows. The view DBC.AllSpaceV, however, includes both summary and detail rows and when you sum currentperm from AllSpaceV, you get the sum from both the summary and detail rows added together.

If you want space usage information at the database/user level, use the DBC.DiskSpaceV view which selects only summary (user/database) rows. If you want space usage information at the object level, use the DBC.TableSizeV view.

Example Requests for Space Information

The following table provides examples of querying Data Dictionary tables for space information:

Information Required	Example Request
Total disk space in the system	<pre>SELECT SUM(MaxPerm) FROM DBC.DiskSpaceV;</pre> <p>You could also use the following request. The information is in a single row for a given database and the request is more efficient. DBC.DiskSpaceV has one row per database per AMP, so it scans more rows.</p> <pre>SELECT SUM(MaxPerm) FROM DBC.GlobalDBSpaceV;</pre>
Disk space currently in use	<pre>SELECT SUM(CurrentPerm), SUM(MaxPerm), (cast(SUM(currentperm) as float) / cast((NULLIFZERO(SUM(maxperm)) as float) * 100) (TITLE '%DiskSpace_in_use', FORMAT 'zz9.99') FROM DBC.DiskSpaceV;</pre>
Disk space for a given database	<pre>SELECT sum(MaxPerm) FROM DBC.DiskSpaceV WHERE databasename='xxxx';</pre> <p>The preceding request uses a partition scan to obtain the result. This processing is expected to be very efficient. Alternatively, use the following request; notice aggregation is not required, and this request is even more efficient because it is a row-hashed look up.</p> <pre>SELECT MaxPerm FROM DBC.GlobalDBSpaceV WHERE databasename='xxxx';</pre>
Percentage of disk space that is available for spool	<pre>SELECT ((cast((SUM(MaxPerm) - SUM(CurrentPerm)) as float)/ cast((NULLIFZERO(SUM(MaxPerm))) as float))*100) (TITLE'% Avail for Spool', format'zz9.99') FROM DBC.DiskSpaceV;</pre>
Percentage of space used by each database in the system	<pre>SELECT Databasename (format 'X(12)') ,SUM(maxperm) ,SUM(currentperm) ,(((cast(SUM(currentperm) as float))/ cast(NULLIFZERO(SUM(maxperm)) as float)) * 100) (FORMAT 'zz9.99%', TITLE 'Percent // Used') FROM DBC.DiskSpaceV GROUP BY 1 ORDER BY 4 DESC WITH SUM (currentperm), SUM(maxperm);</pre>

Information Required	Example Request
Users who are running out of permanent space	<pre>SELECT Databasename (format 'X(12)') ,SUM(maxperm) ,SUM(currentperm) ,(((cast(SUM(currentperm) as float))/ cast(NULLIFZERO(SUM(maxperm)) as float)) * 100) (format 'zz9.99%', TITLE 'Percent // Used') FROM DBC.DiskSpaceV GROUP BY 1 HAVING ((cast(SUM(currentperm) as float)) /cast(NULLIFZERO(SUM(m axperm)) as float)) > 0.9 ORDER BY 4 DESC;</pre> <p>You can change the value 0.9 to whatever threshold ratio is appropriate for your site.</p>
Users who are using a lot of spool	<pre>SELECT databasename ,SUM(peaks pool) FROM DBC.DiskSpaceV GROUP BY 1 HAVING SUM(peaks pool) > 5000000000 ORDER BY 2 DESC;</pre> <p>You can change the value 5000000000 to whatever value is appropriate for your site. Some sites with more space may have greater tolerance for higher spool usage and spool limits.</p>
Users and databases with a non-zero skew limit on permanent space (global PERM space accounting)	<pre>SELECT databasename ,permskew FROM DBC.GlobalDBSpaceV WHERE permskew > 0 ORDER BY 2 DESC;</pre> <p>The request provides the users and databases for which the system allocates space dynamically to AMPs as needed. The skew limit allows some AMPs to exceed the per-AMP quota.</p>
Users and databases having permanent space skewed but defined with 0 skewlimit	<pre>SELECT COALESCE((cast(MAX(currentPerm) as float)/ cast(NULLIFZERO(AVG(currentPerm)) as float) - 1), 0) * 100 (FORMAT '9999.99') AS spaceskew, databasename FROM DBC.diskspacev GROUP BY 2 HAVING spaceskew > 125.0 WHERE permskew = 0;</pre> <p>The request checks databases that have space usage skewed beyond a normal 25%. Change 125.0 to any value that suits your needs. For resulting databases, you may want to consider defining a smaller permskewlimit to better manage space. SkewLimit > 0 implies need-based space allocations for AMPs.</p>
Current permanent usage and AMP level space	<pre>SELECT vproc, currentperm, maxperm, AllocatedPerm FROM DBC.DiskSpaceV WHERE databasename='xxxx' order by vproc;</pre>

Information Required	Example Request
allocations for a given database	
Allocation for databases that have skewed permanent usage defined	<pre>SELECT databasename, allocatedperm, maxperm FROM DBC.GlobalDBSpaceV WHERE permskew > 0 or permskew IS NULL;</pre>
Unallocated space remaining in the database	<pre>SELECT databasename, allocatedperm, maxperm,maxperm- allocatedperm (TITLE 'UnallocatedSpace'), permskew FROM DBC.GlobalDBSpaceV;</pre>
AMPs that are almost running out physical space	<pre>SELECT SUM(maxperm) m, SUM(currentperm+currentspool+currenttemp) c , cast(((m-c)*100.00) as float)/cast(m as float) (format 'zz9. 99%') p, vproc FROM DBC.diskspacev group by vproc having p < 5.0;</pre> <p>The preceding request returns all AMPs whose actual usage leaves less than 5% of their total space remaining. Change 5.00 to another value as required.</p>
The permanent space use for databases and users associated with a given default map	<pre>SELECT dbv.databasename, sum(currentperm), sum(maxperm) FROM DBC.DiskSpaceV dbspace, DBC.DatabasesV dbv WHERE dbspace.databasename = dbv.databasename AND defaultmapname='td_map5' GROUP BY 1;</pre>
The size of the tables defined in a given map in a given database	<pre>SELECT tabsz.databasename, tabsz.tablename, vproc, sum(currentperm) FROM DBC.TableSizeV tabsz, DBC.TablesV tabv WHERE tabsz.databasename = tabv.databasename AND tabsz.tablename = tabv.tablename AND tabv.mapname='td_map1' and tabsz.databasename='SysAdmin' GROUP BY 1,2,3 ORDER BY 1,2,3;</pre>
Permanent space use by map for each database in each AMP	<pre>SELECT tabv.mapname, tabv.databasename, vproc, sum(currentperm) FROM DBC.TableSizeV tabsz, DBC.TablesV tabv WHERE tabsz.databasename = tabv.databasename AND tabsz.tablename = tabv.tablename GROUP BY 1,2,3 ORDER BY 1,2,3;</pre>
Permanent space use by map for each database in the system	<pre>SELECT tabv.mapname, tabv.databasename, sum(currentperm) FROM DBC.TableSizeV tabsz, DBC.TablesV tabv WHERE tabsz.databasename = tabv.databasename AND tabsz.tablename = tabv.tablename</pre>

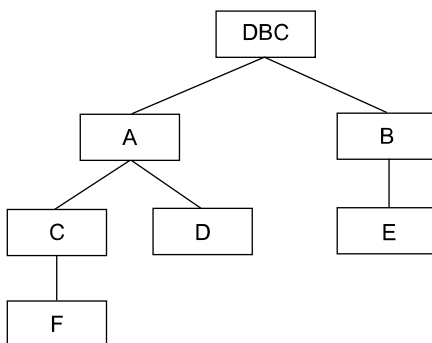
Information Required	Example Request
	GROUP BY 1,2 ORDER BY 1,2,3;
The maximum allocated peak spool for a given user	SELECT databasename, PeakAllocatedSpool from DBC.GlobalDBSpaceV;

Transferring Ownership and Permanent Space

Transferring Permanent Space

With CREATE, GIVE, and DROP, you can transfer the permanent space of one database or user to another. This is particularly useful if you want to transfer permanent space from a child of a child back to user DBC when user DBC is not the immediate owner. For information on dropping databases or users, see [Dropping a Database or User](#). For information on transferring ownership of a database or user with GIVE, see [Transferring Ownership with GIVE](#).

For example, assume the following hierarchy:



Also, assume that:

- F has a MAXPERM of 10, a MAXSPOOL of 50, and a MAXTEMP of 25.
- E has a MAXPERM of 10, a MAXSPOOL of 20, and a MAXTEMP of 15.

To increase permanent space for E:

1. From space owned by F, create temporary database X with a MAXPERM of 5:

```
CREATE DATABASE X FROM F AS PERM = 5 ;
```

The default is to allocate to a new database the same spool and temporary space as its owning database, so MAXSPOOL for X defaults to 50 and MAXTEMP for X defaults to 25. The PERM

allocation for X is taken from the space of its owning database; thus, the MAXPERM of F is reduced to 5.

2. Give X to E by using the GIVE statement to transfer ownership:

```
GIVE X TO E;
```

3. Drop X with the following statement:

```
DROP DATABASE X;
```

This increases the MAXPERM of E to 15.

The MAXSPOOL and MAXTEMP of E are unchanged at 20 and 15, respectively.

Reallocating Perm Space to Resolve Space Issues

1. In the Space Usage portlet, highlight the database or user that requires more space.

Note:

You can also define the value for spool in a profile, and then assign profile membership to a user. See [Creating Profiles](#).

2. Click the **Add Space** button and:

- Enter your username and password (you must have DROP privileges on the database or user that requires more perm space).
- Enter the space in megabytes and the name of the user or database that directly owns the perm space being transferred.

The system generates a MODIFY DATABASE or MODIFY USER statement that reallocates the space.

Increasing the Spool Space Limit for a User

Specify a new value for SPOOL in a MODIFY PROFILE or MODIFY USER statement, depending on where the user spool is defined, for example:

```
MODIFY [PROFILE profile_name | USER user_name ] AS
SPOOL = bytes ;
```


Finding and Fixing Skewed Tables by Querying the TableSizeV View

The DBC.TableSizeV contains information that can help identify skewed tables. Table skew results from an inappropriate primary index. A query that references a skewed table may try to process more rows on some AMPs than others, and may run out of space:

- SELECT statements may run out of spool space if the accessed object has a defined spool space limit.
- INSERT and UPDATE statements may run out of perm space.

1. Use the following SQL statement to find skewed tables in the DBC.TableSizeV:

```
SELECT vproc AS
"AMP", TableName (FORMAT 'X(20)'), CurrentPerm
FROM DBC.TableSizeV
WHERE DatabaseName = 'database'
ORDER BY TableName, "AMP" ;
```

Result:

AMP	TableName	CurrentPerm
---	-----	-----
0	employee_upi_onempid	18,944
1	employee_upi_onempid	18,944
2	employee_upi_onempid	18,944
3	employee_upi_onempid	19,968
0	employee_nupi_onddept	4,096
1	employee_nupi_onddept	30,208
2	employee_nupi_onddept	15,360
3	employee_nupi_onddept	12,228

In this example, the answer set displays space allocations by AMP for two tables. The results show that:

- CurrentPerm is similar across all vprocs for employee_upi_onempid. Permanent space distribution is relatively even across all AMPs in the system.
- The table Employee_nupi_onddept is poorly distributed. The CurrentPerm figures range from 4,096 bytes to 30,208 bytes on different AMPs.

2. Redefine the primary index for any skewed tables that you find. See [Choosing a Primary Index](#) .

Example of Finding Skewed Tables by Querying the TableSizeV View

In this example, the SELECT request looks for poorly distributed tables by displaying the average, minimum, and maximum of the CurrentPerm figures allocated on each AMP to every table in the USER database. Each table is reported separately, ordered by name.

```
SELECT
  TableName (FORMAT 'X(20)'),
  MIN(CurrentPerm) AS "AMP Minimum",
  AVG(CurrentPerm) AS "AMP Average",
  MAX(CurrentPerm) AS "AMP Maximum"
FROM DBC.TableSizeV
WHERE DatabaseName = 'USER'
GROUP BY TableName
ORDER BY TableName;
```

Result:

TableName	AMP Minimum	AMP Average	AMP Maximum
-----	-----	-----	-----
employee_nupi_onddept	4,096	15,488	30,208
employee_upi_onempid	18,944	19,200	19,968

The result displays two tables. Notice the results show that:

- The table Employee_upi_onempid is evenly distributed. CurrentPerm is similar across all vprocs (the minimum and maximum are close to the average). Permanent space is relatively evenly distributed across all AMPs in the system.
- The table Employee_nupi_onddept is poorly distributed. The CurrentPerm figures range from a minimum of 4,096 bytes to a maximum of 30,208 bytes on different AMPs, indicating a wide variance from the average.

Defining Temporary Space Limits

While some amount of permanent space is used to store the definition of a global temporary table, temporary space is used to hold rows of materialized global temporary tables. Temporary space is allocated at the database or user level, but not the table level.

Define a temporary space limit with the TEMPORARY parameter of a CREATE/ MODIFY PROFILE or CREATE/MODIFY USER/DATABASE statement.

Note:

A profile definition overrides any user definition, it does not append settings to the definition.

The maximum and default limits for temporary allocation are determined as described in the following table.

IF you ...	AND a profile...	THEN the limit is inherited from the...
specify TEMPORARY in a CREATE/MODIFY USER/DATABASE statement	does not apply	immediate owner and may not exceed the limit of the immediate owner.
	applies	user who submitted the CREATE/MODIFY PROFILE statement. The limit may not exceed the limit of that user. The limit is determined as follows: <ul style="list-style-type: none"> • If that user has a profile, the system uses the limit in the profile. • If a profile does not apply to that user, the system uses the limit in the CREATE/MODIFY USER statement for that user. • If no TEMPORARY is defined for that user, the system uses the limit of the immediate owner.
do not specify a TEMPORARY limit in a CREATE/MODIFY USER/DATABASE statement	does not apply	immediate owner of the user.
	applies	profile specification.
	applies but the SPOOL parameter is NULL or NONE	specification for the user who submitted the CREATE /MODIFY PROFILE statement, determined as follows: <ul style="list-style-type: none"> • If that user has a profile, the profile specification. • If a profile does not apply to that user, the specification in the CREATE/MODIFY USER statement for that user. • If no TEMPORARY is defined for that user, the limit of the immediate owning database or user.

When using the CREATE USER, CREATE DATABASE, or CREATE PROFILE statements to assign temporary space limits, keep your space limits in mind. Query the DBC.DiskSpaceV view to find the system levels for temporary space.

The following table describes the different types of temporary space.

Temporary Space Level	Description
CURRENTTEMP	This is the amount of space currently in use by Global Temporary Tables. Note: This does not include the amount of permanent space required for the table header stored on each AMP for the base global temporary table definition.
PEAKTEMP	This is the maximum temporary space used since the last session. Temporary space is released when the session terminates.
MAXTEMP	MaxTemp specifies the limit of space available for global temporary table rows.

Temporary Space Level	Description
	<p>Note:</p> <p>This does not include the amount of permanent space required for the table header stored on each AMP for the base global temporary table definition.</p> <p>The value may not exceed the limit of the:</p> <ul style="list-style-type: none"> • Creator or modifier of the profile, when setting TEMPORARY in a profile • Immediate owner of the user being created or modified, if a profile does not apply <p>If you do not specify a value and the user is associated with a profile, MaxTemp defaults to the value of the profile, if defined. If the profile TEMPORARY is set to NULL or NONE, or the user is not associated with a profile, MaxTemp defaults to the value of the parent of the user.</p>

Protecting Transactions by Reserving Cylinders for Perm Space

A transaction is aborted when it requires more space than is currently available for the requesting user. When you create a user or database, the system requests space on cylinders from the pool of free cylinders. To protect transactions, you can specify that a number of cylinders be reserved for transactions needing permanent space.

The number of free cylinders to keep in reserve for requests needing permanent space is determined by the File System field named “Cylinders Saved for PERM” in DBS Control (see “Cylinders Saved for PERM” in *Teradata Vantage™ - Database Utilities*, B035-1102). When a statement requires space, this field determines allocation as described in the following table.

IF the statement needs ...	AND there are...	THEN the statement ...
permanent space	one or more free cylinders	succeeds.
	less than one free cylinder	fails with a disk full error.
spool space	more free cylinders than the amount specified in Cylinders Saved for PERM	succeeds.
	fewer free cylinders than the amount specified in Cylinders Saved for PERM	fails with a disk full error.

This means that requests needing SPOOL space might fail more often than requests needing PERM space. This can be advantageous because failure of a request for spool space rarely involves a rollback.

To monitor and reallocate permanent disk space, you can use Teradata Viewpoint.

Creating a Macro for Space Usage Reporting

You can create a macro like the one shown here.

Example of Creating a Macro for Space Usage Reporting

The following macro shows space used.

```
CREATE MACRO superspace as (
SELECT databasename, SUM(maxperm), SUM(currentperm)
  FROM DBC.DiskSpaceV
 GROUP BY databasename
 ORDER BY databasename
 WITH SUM(maxperm), sum(currentperm);
);
```

When executed, the macro returns the following results.

```
execute superspace;
```

Result:

```
*** Query completed. 28 rows found. 3 columns returned.
*** Total elapsed time was 1 second.
```

DatabaseName	Sum(MaxPerm)	Sum(CurrentPerm)
-----	-----	-----
macro		
All	0	0
cliuser	10,000,000	7,168
CONSOLE	100,000	0
Crashdumps	1,024,000,000	0
Custo	5,120,000,000	100,082,688
CUSTOMER_SERVICE	10,000,000	5,632
dash	20,000,000	9,216
DBC	24,316,695,812	89,080,832
Default	0	0
Test_User	20,000,000	4,096
explainsample	1,000,000	12,288
EXTUSER	0	0
mkt_bsktTest	1,280,000,000	1,108,992
PDTBASE	10,000,000	33,792
PUBLIC	0	0

qttest	500,000,000	10,240
SQLJ	600,000,000	0
SysAdmin	40,000,000	1,693,696
SYSLIB	10,000,000	12,288
SYSSPATIAL	110,000,000	2,931,712
SystemFe	60,000,000	207,872
SYSUDTLIB	100,000,000	0
Sys_Calendar	15,000,000	2,654,208
TDPUSER	0	0
testdb	800,000	133,120
testuser	20,000,000	9,216
udtuser	10,000,000	0

Sum(MaxPerm)	33,277,595,812	197,997,056

Viewing Space Usage Trends for Databases and Users

The Metric Heatmap portlet displays many of the same parameters as the Space Usage portlet, but allows you to see space usage trends over several months. A review of space usage trends in Metric Heatmap can be useful when planning and scheduling batch jobs or other large space-using tasks, or when estimating growth in system disk requirements.

Other Ways to View Space Utilization

Other ways to view space utilization include Teradata Studio, the Ferret utility, and SQL macros.

Teradata Studio

In Teradata Studio, click on the name of the database to view space usage. Select **Child Space** to see the space usage for all child databases of the selected database. Select **Tables** to see the space usage for all tables of the selected database.

The Ferret Utility and Equivalent Macros

You can also use the system utility, Ferret, which is started via the Database Console, to see space usage. Use the ShowSpace command to view space usage (perm, spool, and temporary) at the system level. Use the ShowBlocks command to view the allocation of permanent space at the table and subtable level.

The ShowBlocks command can provide a level of detail that is not available with DBC.TableSize. For example, ShowBlocks can answer “How much perm space is used for a secondary index?” ShowBlocks also provides subtable space information. You can multiply the typical block size times the number of blocks to determine subtable space usage.

Vantage includes a set of File System Information macros that retrieve and display information similar Ferret SHOW commands. Because this information is presented in regular database tables, it can be

easier to share and use for further processing. Permissions to create and view this information are also easier to manage using standard SQL GRANT and REVOKE privilege statements. For more information about these macros, see "File System Information Macros and Functions" in *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Data Space Tool Macros

Teradata Vantage includes a set of Customer Data Space (CDS) tools that show data space consumption at the object, database, and system level. The tools report the logical size of your data in its uncompressed form, and eliminate the need for laborious manual space accounting calculations to find object sizes and estimated compression ratios for block-level compressed data.

Customer Data Space tools consist of a set of macros in the SystemFE database, and a new Data Dictionary view, CDSTableSizeV. For more information, see *Teradata Vantage™ - SystemFE Macros*, B035-1103 and *Teradata Vantage™ - Data Dictionary*, B035-1092.

Managing Data Blocks

A data block is the unit of physical I/O when Teradata Vantage writes data to disk. A data block contains one or more data rows from the same table. The system stores data blocks in segments, which are grouped in cylinders. You can set data block sizes for tables using SQL or set them globally using the DBS Control utility. If there is a conflict between the data block size value set for the system and for the table, the table value overrides the global value.

Setting Data Block Sizes Using SQL

Set data block sizes for tables using the CREATE TABLE or ALTER TABLE statement. The DATABLOCKSIZE = n [BYTES/KBYTES/ KILOBYTES] specification allows you to define n as the maximum multi-row data block size used for storing rows of this table.

Teradata recommends the following:

IF...	THEN...
most queries to a table access a single row of data	define smaller data blocks for the table to enhance row retrieval efficiency.
most queries to the table access data across the entire table	define larger data blocks for the table to reduce the number of I/Os required to read the table.

Setting Data Block Sizes Using the DBS Control Utility

You can set data block sizes globally using these DBSControl fields: PermDBAllocUnit, PermDBSize, and JournalDBSize. These fields determine the maximum size of permanent data blocks that hold multiple rows. (A row that exceeds the size of a multirow data block is put into a block of its own.)

Example of Changing PermDBSize

To change the PermDBSize, you need to use the modify and write commands of the DBS Control utility. The following example changes the data block size to 254 sectors:

1. From the command-line prompt (or the Supervisor window of the Database Window):

```
dbcontrol
```

2. From the DBS Control window:

```
display FILESYS
```

3. Use the modify command to change flag 3 (PermDBSize):

```
modify FILESYS 3 = 254
```

4. Write the changes to the GDO:

```
write
```

Setting Data Block Merging Limits

As part of full-table modify operations, where the system sequentially scans the table as part of the operation, Vantagecan merge small DBs into larger ones as the modifications are performed. While this can make some of the initial modifications to the table more expensive, it makes future full table modifications and queries cheaper by reducing the number of DBs in the table, which reduces disk I/Os.

You can control the frequency of merges and the size of resulting data blocks by:

- Setting the system-level MergeBlockRatio field in DBS Control.
- Specifying the table-level MergeBlockRatio attribute in the CREATE TABLE or ALTER TABLE statement.

Note:

The table-level attribute overrides the system-level setting for any table that has the attribute. However, if the Disable Merge Blocks field in DBS Control is enabled, both the table-level attribute and the system-level setting will be ignored.

MergeBlockRatio settings allow you to make adjustments so that merged data blocks end up at a reasonable size (that is, the block does not split again quickly after it undergoes a merge).

To consider the right MergeBlockRatio setting, you must take into account the following:

- How often you expect to access or modify certain tables
- How much of the data is being modified
- The desired general size of blocks across certain tables

Managing Cylinders

Setting Free Space Percent Limits

You can control the Free Space Percent (FSP) at the global and table level using the following parameters.

Parameter	Description
FreeSpacePercent field of DBS Control	<p>Global parameter that the system uses to determine the percentage of space to leave unused on each cylinder during bulk loading operations such as MultiLoad and FastLoad. This default can be overridden at the table level with the FREESPACE clause.</p> <p>Choose a percentage that reflects the net growth rate of your data tables (INSERTs minus DELETEs). For example, most sites choose 5 to 15%.</p>
FREESPACE = <i>n</i> of a CREATE/ALTER TABLE statement	<p><i>n</i> percent of cylinder space to remain free on each cylinder when bulk loading this table (where <i>n</i> is an integer constant).</p> <p>Note:</p> <p>Do not use the Ferret utility PACKDISK command to change this value once you have created a table or have modified its FREESPACE PERCENT with an ALTER TABLE request.</p> <p>Instead, submit an ALTER TABLE request to change the free space percent value for the table and then immediately afterward submit a PACKDISK command which will pick up the new free space percent value (see the documentation for the Ferret utility in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 for more information and instructions for running PACKDISK).</p>
DEFAULT FREESPACE of an ALTER TABLE statement	<p>Reset the current free space percent for a table at the global or table level. DEFAULT FREESPACE resets Free Space to the value defined in the FreeSpacePercent field in DBS Control.</p>
FREESPACEPERCENT option in PACKDISK	<p>Optionally specifies the percentage of cylinder space to leave free. This allows for future table growth without requiring the allocation of new free cylinders.</p> <p>FSP can be specified for individual tables by using the FREESPACE option to CREATE TABLE and ALTER TABLE.</p> <p>The FREESPACE value defined for a table with CREATE TABLE takes precedence over the FREESPACEPERCENT value with PACKDISK, unless PACKDISK includes the FORCE option. In that case, PACKDISK will pack to the specified FREESPACEPERCENT, but after that, the AutoCylPack job running in the background will eventually repack the table cylinders to the FREESPACE value that was specified in CREATE TABLE. For information on the FORCE and PACKDISK commands, see the Ferret Utility in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102.</p> <p>Use the FREESPACE option of ALTER TABLE to add or modify an FSP value for an existing table. For more information on the FREESPACE option, see <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144.</p>

Parameter	Description
	<p>Note:</p> <p>Because the AutoCylPack (automatic background cylinder packing) feature monitors and maintains proper levels of cylinder free space using AutoCylPackFSP, you only need to use the FREESPACEPERCENT option in PACKDISK in exceptional situations. For a list of these exceptions, see “Ferret Utility (ferret)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102.</p>

For more information on the DBS Control utility, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For information on the CREATE TABLE or ALTER TABLE statements, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Finding Loosely Packed Tables with the SHOWFSP Command

Use the SHOWFSP command of the Ferret utility before running PACKDISK to estimate the effectiveness of PACKDISK commands. SHOWFSP helps you find loosely packed tables.

The SHOWFSP command gives an approximation of the number of cylinders that can be freed by PACKDISK. It may or may not match with the exact number of cylinders that will be freed. If you feel that PACKDISK reports a very different number of cylinders freed than SHOWFSP, then execute a defrag with the DEFRAG command.

```
=====
Ferret ==>
showfsp
Fri Jun 06, 2003 12:06:02 : showfsp will be started on All AMP vprocs
Fri Jun 06, 2003 12:06:02 : to find all the tables specified greater than
0 cylinders
Fri Jun 06, 2003 12:06:02 : which could free up more than 0 cylinders
Fri Jun 06, 2003 12:06:02 : only tables with a current FSP greater than 0%
Fri Jun 06, 2003 12:06:02 : will be considered
Do you wish to continue based upon this scope?? (Y/N)
y
Fri Jun 06, 2003 12:06:04 : ShowFsp has been started
On All AMP vprocs
vproc 0 (0000) response
There are 6 tables larger than 0 cylinders on amp 0
```

Database Name	Table Name	fsp %	Recoverable Cylinders	Current Cylinders
CUSTOMER	ppitable3	25	1	5


```
vproc 1 (0001) response
```

```
There are 6 tables larger than 0 cylinders on amp 1
```

Database Name	Table Name	fsp %	Recoverable Cylinders	Current Cylinders
-----	-----	-----	-----	-----
CUSTOMER	ppitable4	34	1	5

```
2 of 2 vprocs responded with the above tables fitting the criteria
ShowFsp has completed
```

For information on the settings and options for SHOWFSP, see “Ferret Utility (ferret)” in *Teradata Vantage™ - Database Utilities*, B035-1102.

Freeing Cylinders Using PACKDISK

The PACKDISK command of the Ferret utility reconfigures the contents of an entire disk or an individual table, leaving a percentage of free space for cylinders within a scope defined by the SCOPE command. PACKDISK uses the default Free Space Percent or a new percentage defined as part of the command to pack the entire disk or a table.

The allowable scope for PACKDISK is vprocs or tables but not both.

The system automatically moves data blocks from one cylinder to another, stopping when the required number of free cylinders is available. This operation runs when the number of free cylinders is critically low. You can use the PACKDISK command to force the packing to occur earlier.

Specifying Free Space Percent

PACKDISK packs either the entire disk or a single table, leaving a specified percentage of the cylinders occupied by the object empty, to account for subsequent insert and update operations. This is the FSP you specify using the tools described in [Setting Free Space Percent Limits](#).

Setting the PACKDISK Scope

The PACKDISK command of the Ferret utility reconfigures the contents of an entire disk or an individual table, leaving a percentage of free space for cylinders within a scope defined by the SCOPE command.

Running PACKDISK

To run PACKDISK, enter in the command window of the Ferret partition, for example:

```
PACKDISK FREESPACEPERCENT = 25
```

(where 25 equals the percentage of cylinder free space). Key the command in uppercase, lowercase, or a combination of both.

Stopping PACKDISK

To stop PACKDISK, enter ABORT.

Packing applies only to entire logical cylinders, not to the space inside individual data blocks within those cylinders. Data block sizes are the same before and after the PACKDISK operation. Also see [Managing Free Space](#).

Defragmenting Cylinders Using Defragment

Cylinders can become defragmented over time. Use the Ferret utility DEFAGMENT command to defragment the cylinders on either an AMP or the system, depending on the SCOPE options.

AutoCylPack and MiniCylPack

AutoCylPack

AutoCylPack (automatic background cylinder packing) manages space on cylinders, finding cylinders with low or high utilization and returning them to their user-defined FSP (Free Space Percent), that is, the percentage of storage space within a cylinder that AutoCylPack leaves free of data to allow for future table growth.

There is a system wide default for AutoCylPack. It can be overridden on a table-by-table basis by specifying the FSP clause in a CREATE TABLE or ALTER TABLE statement.

Although AutoCylPack runs as a background task issuing I/Os, you can adjust its level of impact. For the DBS Control fields that support AutoCylPack, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Performance

AutoCylPack helps reduce:

- The chances that MiniCylPacks run. MiniCylPacks are strictly concerned with reclaiming whole cylinders.
- The need for you to perform regular Ferret PACKDISK operations.

PACKDISK can be performed on a table, a range of tables or an entire system, but it affects performance of the foreground tasks and thus system performance.

If cylinders have higher utilization:

- System performance improves because there is a higher data block to cylinder index ratio and more effective use of Cylinder Read. There will be less unoccupied sectors read in.
- A table occupies less cylinders. This leaves more cylinders available for other uses.

MiniCylPack

A MiniCylPack moves data blocks in logical sequence from cylinder to cylinder, stopping when the required number of free cylinders is available.

The process continues until one cylinder is completely emptied. The master index begins the next required MiniCylPack at the location that the last MiniCylPack completed.

The Vantage file system will start to MiniCylPack when the number of free cylinders drops to the value set by MiniCylPackLowCylProd. The default is 10.

The File Information Block (FIB) keeps a history of the last five cylinders allocated to avoid MiniCylPacks on them.

Note:

Spool files are never cylinder packed.

Use the DBS Control (see “MiniCylPackLowCylProd” in *Teradata Vantage™ - Database Utilities*, B035-1102) to specify the free cylinder threshold that causes a MiniCylPack. If the system needs a free cylinder and none are available, a MiniCylPack occurs spontaneously.

Migrating Data Blocks

1. If space can be made available either by migrating blocks forward to the next cylinder or backwards to the previous cylinder, choose the direction that would require moving the fewest blocks.

If the number of blocks is the same, choose the direction of the cylinder with the most number of free sectors.

2. If Step 1 fails to free the desired sectors, try migrating blocks in the other direction.
3. If space can be made available only by allocating a new cylinder, allocate a new cylinder. The preference is to add a new cylinder:
 - Before the current cylinder for permanent tables.
 - After the current cylinder for spool tables and while performing FastLoads.

When migrating either forward or backward, the number of blocks may vary because the system considers different blocks for migration.

Because of the restriction on key ranges within a cylinder, the system, when migrating backward, must move tables and rows with the lowest keys. When migrating forward, the system must move tables and rows with the largest keys.

The system follows special rules for migrating blocks between cylinders to cover special uses, such as sort and restore. There are minor variations of these special rules, such as migrating more data blocks than required in anticipation of additional needs, and looking for subtable breaks on a cylinder to decide how many data blocks to attempt to migrate.

Performance

Although cylinder packing itself has a small impact on performance, it often coincides with other performance impacting conditions or events. When the Vantage file system performs a MiniCylPack, the operation frees exactly one cylinder.

The cylinder packing operation itself runs at the priority of the user whose job needed the free cylinder. The cylinder packing operation is the last step the system can take to recover space in order to perform a write operation, and it is a signal that the system is out of space.

Needing to pack cylinders may be a temporary condition in that a query, or group of queries, with very high spool usage consumes all available free space. This is not a desirable condition.

If space is a problem,

- Enable AutoCylPack if you have not done so and specify an FSP of 0% for read-only tables using the CREATE TABLE or ALTER TABLE statement
- Run the Ferret PACKDISK command.

Error Codes

MiniCylPacks are a natural occurrence and serve as a warning that the system may be running short on space. Tightly packed data can encourage future cylinder allocation, which in turn triggers more MiniCylPacks.

The system logs MiniCylPacks in the Software_Event_LogV with the following error codes.

Code	Description
340514100	Summary of MiniCylPacks done at threshold set via the DBS Control.
340514200	A MiniCylPack occurred during processing and a task was waiting for it to complete.
340514300	The system could not free cylinders using MiniCylPack. The MiniCylPack failed. This means that the system is either getting too full or that the free cylinder threshold is set unreasonably high. Investigate this error code immediately.

Frequent 340514200 or 340514300 messages indicate that the configuration is under stress, often from large spool file requirements on all AMPs. MiniCylPacks tend to occur across all AMPs until spool requirements subside. This impacts all running requests.

If table data is skewed, you might see MiniCylPacks even if Vantage has not used up most of the disk space.

Giving One User to Another

When you give one object to another, all space that belongs to it goes with it. If you drop the object, the space goes to the immediate owner. When you give databases or users, all descendants of the object remain descendants of the object. When you give the object to new parents, the ownership of the space is transferred, but the limits remain the same.

Adjusting Perm Space Limits Using SQL

Perm space limits are easy to adjust, for example:

```
CREATE DATABASE Temp FROM HR AS PERM = 2000000;
GIVE Temp TO Finance;
DROP DATABASE Temp;
```

Adjusting Perm Space Limits Using Viewpoint

1. In the Space Usage portlet, highlight the database or user that requires more space.

Note:

You can also define the value for spool in a profile, and then assign profile membership to a user.

2. Click the **Add Space** button and:

- Enter the username and password of a database user that has DROP privileges (required for use of the MODIFY statement) on the database or user that requires more perm space.
- Enter the space in megabytes and the name of the user or database that directly owns the perm space being transferred.

The system generates a MODIFY DATABASE or MODIFY USER statement that reallocates the space.

Maintaining the Database: Operational DBAs

This section describes system maintenance tasks that should be done on a regular basis, including archiving Data Dictionary log tables and using Vproc Manager to manage vproc status and initialize AMP disks. It also describes more occasional maintenance tasks, including applying software patches and resetting peak values in the DBC.DataBaseSpace table.

Database Maintenance Tasks

The maintenance topics that follow are of two types:

- Periodic maintenance tasks that should be done at regular intervals
- Occasional maintenance tasks

Managing Accumulated Log Data

Database Logs

Teradata Vantage contains many log tables in the Data Dictionary that accumulate data, either automatically or after certain features are enabled. Because log data grows over time, you must purge older information to avoid using up permanent space. You can also use the data to generate your own performance analysis reports or use a reporting solution from Teradata Support.

Logs	Description
Logs that can be purged or archived using Teradata Viewpoint.	
Access Log (DBC.AccLogTbl)	Data collected as defined in a BEGIN LOGGING statement, for each privilege check of an attempted user access of a database object.
DBQ Logs, such as: <ul style="list-style-type: none"> • DBC.DBQLogTbl • DBC.DBQLSqlTbl • DBC.DBQLObjTbl • DBC.DBQLExplainTbl • DBC.DBQLStepTbl • DBC.DBQLSummaryTbl • DBC.DBQLXMLTbl • DBC.DBQLXMLLockTbl • DBC.DBQLParamTbl • DBC.DBQLUtilityTbl 	Data collected during database query logging (DBQL).

Logs	Description
Disk space log (DBC.DatabaseSpace)	Data collected automatically on space allocation for databases and tables.
Event Log (DBC.EventLog)	Data collected automatically by the system for each user logon event.
ResUsage Logs, such as: <ul style="list-style-type: none"> • DBC.ResUsageSawt • DBC.ResUsageScpu • DBC.ResUsageShst • DBC.ResUsageSldv • DBC.ResUsageSpdsk • DBC.ResUsageSpma • DBC.ResUsageSps • DBC.ResUsageSvds • DBC.ResUsageSvpr 	<p>ResUsage tables collect data on system resource usage. You can enable ResUsage data collection and set the collection rate using either:</p> <ul style="list-style-type: none"> • The ctl utility SCREEN RSS command • Teradata Viewpoint data collectors <p>The collection rate determines how quickly data accumulates and when the logs should be purged.</p>
SWEvent Log (DBC.SW_Event_Log)	The system automatically inserts rows in response to software errors and system events, for use by Teradata Customer Service.
TDWM Logs: <ul style="list-style-type: none"> • DBC.TDWMSummaryLog • DBC.TDWMLog • DBC.TDWMLExceptionLog • DBC.TDWMLEventHistory 	Logs for Teradata Viewpoint workload management functions.
Logs that must be purged manually purged	
In Doubt transaction Log (DBC.InDoubtResLog)	Contains a row for each transaction where completion was in doubt.

Methods for Maintaining Database Logs

There are several methods for maintaining database logs:

Method	Use	Information
Teradata Viewpoint Monitored Systems portlet, Log Table Cleanup and Cleanup Schedule menus	Configure specific logs for clean up and specify the cleanup interval. This method does not archive log data. Choose either this or the following method, since they both automatically clean up the Data Dictionary.	See Deleting Old Log Data with Teradata Viewpoint
Teradata Viewpoint, Performance Data Collection portlet	Archive log data to PDCR history tables and clean up log files. Choose either this or the previous method, since they both automatically clean up the Data Dictionary.	See Setting Up Automated PDCR Log Maintenance

Method	Use	Information
Manually, using SQL and backup to tape	Copy files to tape and delete objects in the log older than a certain date.	See Deleting Old Log Data Manually from System Tables

Setting Up Automated PDCR Log Maintenance

You can automate the maintenance of Data Dictionary log tables using Performance Data Collection and Reporting (PDCR) history tables. To set up log maintenance, do the following things:

1. Run the DIP script DIPPCR.
This creates PDCR history tables and the data extraction macros that populate the tables. It is not part of DIPALL.
2. Use the Teradata Viewpoint portlet **Performance Data Collection** to set up the necessary jobs and schedule them to run them every day.
These jobs move data from the Data Dictionary log tables you specify to the history tables created by DIPPCR. You can also use the portlet to specify how long the history tables are retained. Depending on the space allocated, you can keep these history tables for months or years. Large systems typically retain data in Data Dictionary log tables for a week or less.

Performance Data Collection Portlet Jobs

The following are the **Performance Data Collection** jobs that archive log data frequently to history tables created by DIPPCR.

Job	Description
Session	Runs every 10 minutes to archive data about user session duration and session counts
Accounting	Runs every hour to archive data about AMP usage
Maintenance	Runs once a week to check retention values and clean up Data Dictionary logs if needed

The following are the **Performance Data Collection** jobs that archive Data Dictionary log tables once a day to history tables created by DIPPCR.

Job	What It Archives
Info Tables	User information, including profiles, groups, and space owners
Logon/Logoff	Data collected automatically by the system for each user logon or logoff
TDWM	Logs for workload management functions: <ul style="list-style-type: none"> • DBC.TDWMSummaryLog • DBC.TDWMEventLog • DBC.TDWMExceptionLog

Job	What It Archives
	<ul style="list-style-type: none"> • DBC.TDWMEventHistory
DBQL	Database query logs, depending on the type of logging you enable
Disk Space	Data collected automatically on space allocation for databases and tables
Resource Usage	System-level performance data from the DBC.ResUsage tables
Access Log	Security data
Statistics	No data; this job refreshes statistics on PDCR tables

Related Information

Topic	Resources for Further Information
Configuration and use of DBQL	Tracking Query Behavior with Database Query Logging: Operational DBAs.
Using the ctl utility SCREEN RSS command to set up ResUsage logging and log intervals	The SCREENRSS command in the section "Control GDO Editor (ctl)" in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102.
Setting up the Teradata Viewpoint Performance Data Collection portlet	<i>Teradata® Viewpoint User Guide</i> , B035-2206.

Deleting Old Log Data with Teradata Viewpoint

Configure log data cleanup as follows:

1. From the Teradata Viewpoint main screen, select **Admin > Teradata System**.
2. To configure specific logs for clean up, and specify the cleanup interval:
 - a. In the **SETUP** column, select **Log Table Clean Up**.
 - b. In the **TABLES** column, select a log table category for cleanup, for example, **DBQ Log**.
 - c. Use default cleanup interval shown unless you have reason to change it.

Note:

Coordinate the interval with your site requirements for storing log history.

- d. Check the **Enabled** box.
 - e. Click the **Apply** button to activate the cleanup
3. To specify the time of day that all configured cleanup takes place:
 - a. From the **SETUP** column, select **Clean Up Schedule**.
 - b. Specify a time when the log tables are not active.

Note:

The clean up operation locks the log table and delays logging in the table until log data deletion is complete.

- c. Check the **Enabled** box.
- d. Click the **Apply** button to set the cleanup schedule.

Deleting Old Log Data Manually from System Tables

Before deleting old log data, determine whether you need to retain records of the data set for deletion. For example, while Teradata recommends that you purge log data at least every 90 to 180 days, security policy for some sites may require retention of access log data for a much longer time period.

Note:

You cannot delete data that is less than 30 days old.

Use the following procedure when deleting old log data from system tables:

1. If your site requires long term retention of log data, do the following:
 - a. Create a duplicate log table in another database using the Copy Table syntax for the CREATE TABLE statement.

```
CT DBC.tablename AS databasename.tablename
```

For further information on using the Copy Table syntax, see “CREATE TABLE” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

- b. Back up the table to tape storage in accordance with your site back up policy.

Tip:

Match the log backup/deletion schedule to the amount of log data you want to maintain locally in system tables. For example, to maintain 90 days of log data in a system table do the first back up at 120 days, then delete the last 30 days of data in the log. Back up the log data every 30 days after that.

- c. Drop the duplicate table using a DROP TABLE statement.
2. Log on to SQL Administrator as DBADMIN or another administrative user with DELETE privileges on database DBC.
3. In the **Query** window, enter an SQL statement to purge old log entries.

For example:

```
DELETE FROM DBC.object_name WHERE (Date - LogDate) > number_of_days ;
```


object_name

The name of the DBC object that allows log deletions, as shown in [Database Logs](#). For example, DeleteAccessLogV is the delete object for the Access Log.

Date

SQL syntax element for the current date.

LogDate

The date contained in the LogDate column of the log view based on the specified *number_of_days* from the current date.

number_of_days

The number of days of data that you want to retain in the system table; the span of days from the current date to the oldest LOGDATE to be retained.

Managing Vproc Status and Initializing AMP Disks with Vproc Manager

The Vproc Manager utility allows you to:

- View or change vproc states
- Initialize and boot a specific vproc
- Initialize the storage associated with a specific vproc
- Force a manual Analytics Database restart

You can start Vproc Manager from the DBW Supervisor window (start vprocmanager) or from the Linux command line (vprocmanager). Valid commands are STATUS, INITVDISK, RESTART, BOOT, SET, HELP, and QUIT.

Vproc Manager Command	Description
STATUS	Provides STATUS information about vprocs.
INITVDISK <vproc ID>	Initializes the DBS File System on a vdisk. It applies only to vprocs with a status of NEWPROC or FATAL AMP. Valid vproc IDs are decimals ranging from 0 to 16383. Hex numbers can be used with a trailing "x."
RESTART [TPA] [NODUMP DUMP = {YES, NO}] [<RestartKind] [<comment>]	Forces different types of DBS restarts: <ul style="list-style-type: none"> • NODUMP (the default) indicates no system dump. • DUMP = YES indicates a system dump. • DUMP = NO is equivalent to NODUMP. • ValidRestartKind is either COLD or COLDWAIT. • Comment specifies the reason for the restart.

Vproc Manager Command	Description
BOOT <vproc ID>	Reinitialize the AMP disk before an all-table rebuild and starts the DBS partitions on the specified AMP. It applies only to vprocs with a VprocState of FATAL and a ConfigStatus of Down.
SET	Sets the state of a vproc to either ONLINE, OFFLINE, or FATAL.

Maintaining Data Dictionary Tables

Managing the size of Data Dictionary tables (sometimes called logs) can help improve performance. Analytics Database does not delete system data on its own so you must manage the logs and tables yourself. Determine what you should archive and what you should delete as needed for your site. Use one of the methods mentioned in [Methods for Maintaining Database Logs](#).

Purging the System Logs

The system does not automatically purge logs or the tables underlying views such as the AccessLogV, LogOnOffV, and Software_Event_LogV views. Teradata recommends that you use either the Teradata Viewpoint Performance Data Collection portlet or copy data off the DBC tables into a separate database, archive the data (as desired), and then delete information from the DBC tables. For information about the Viewpoint Performance Data Collection portlet, see [Setting Up Automated PDCR Log Maintenance](#). Some DBC objects to consider purging include:

- Resource usage (ResUsage) tables that have logging enabled.
- DBQL logs, which include all DBQL tables that have logging enabled.

(The tables DBC.DBQLRuleTbl and DBC.DBQLRuleCountTbl are not part of the log maintenance list. These tables are automatically maintained by the Teradata SQL BEGIN/END QUERY LOGGING statements; an error is returned if you attempt to delete their contents.)

- DBC.SW_Event_Log
- QCD.DataDemographics. (If you use Query Capture Facility [QCF] with the SQL COLLECT DEMOGRAPHICS statement, you need to explicitly delete rows from this table, DataDemographics, in your user-defined QCD databases.)

Note:

- Entries in DataDemographics are deleted automatically when you use the INSERT EXPLAIN WITH STATISTICS AND DEMOGRAPHICS statement. For more information, see “Query Capture Facility” in *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142 and “COLLECT DEMOGRAPHICS” in *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.
 - The Teradata Viewpoint Performance Data Collection portlet does not archive this table.
-

Also, the security administrator should purge the logs associated with access logging as well as any other security-related views as needed. In addition to maintaining the size of system tables and logs, you can reduce log file sizes as follows:

- Use only DBC.ResUsageSpma instead of activating logging on multiple ResUsage tables. ResUsageSpma may be able to provide all the information you need.
- Use roles to manage privileges which help reduce the size of DBC.AccessRights. For other tips on reducing the size of the DBC.AccessRights table, see [Housekeeping on an Ad-Hoc Basis](#).
- Track only the information you need for DBQL. The DBQLogTbl may be able to provide all the information you need.
- Track only the information you need for accounting.
- Use active row filter mode where possible for ResUsage tables.

For example, if you want to keep the DBC.Acctg table from growing too large, use only the ASE variables that report the level of information you need. Using &T or &I for ASE accounting will potentially make the DBC.Acctg table very large; but using &D or &L usually has less impact on disk space and may still provide enough level of detail. For more information, see [Enabling Account String Expansion](#).

Clearing Out Values Manually in the DBC.Acctg Table

Note:

You can either use the following manual procedure to clear out values in the DBC.Acctg table or use an automated method, such as the Teradata Viewpoint Performance Data Collection portlet. See [Setting Up Automated PDCR Log Maintenance](#).

The DBC.Acctg table logs CPU utilization and I/O statistics for each account a user owns on each AMP. Updates to the table are made periodically during each AMP step on each processor affected by the step. (If there are long-running steps, AMPUsage numbers show large increases periodically, instead of continuous incremental additions.) Data is collected and added to what is already in the table until you reset the counters to zero.

If you use account string expansion, ASE codes can cause the table to grow even more quickly. You may need to update DBC.Acctg on a regular basis to clear out values and reset the accumulators (For details, see [Enabling Account String Expansion](#)). However, use caution when purging rows containing accounts with ASE codes.

NOTICE

Purge obsolete rows from DBC.Acctg carefully when there is an ASE string embedded in the account string by constructing the appropriate WHERE clause. Since ASE is embedded in the AccountName text string instead of a separate field, careful SQL coding is required.

Use the following process to first save the data, then clear out DBC.Acctg and reset the counters.

1. To control growth, archive the entire predefined set of system tables using DSA with an all-AMPs dump of database DBC.

Note:

If you cannot use the tool that created the archive to reload it, try the BTEQ IMPORT command.

2. After the accumulated AMP data is successfully summarized, and archived if necessary, run the ClearAccounting macro to reset the values of the CPU and I/O columns:

```
EXEC DBC.ClearAccounting;
```

If you use ASE but are not sure which rows to purge from the DBC.Acctg:

- a. Create a table similar to DBC.Acctg with a non-unique primary index, date, and time columns.
- b. Populate the table with rows that are created for each respective account by capturing the sum total of the relevant accounting fields at regular intervals.
- c. Run queries against the table. Entries that show no change are candidate rows to purge.

Archiving and Resetting Accumulators and Peak Values

Tracking peak and count values can help you monitor the health of your system. When you no longer need old data, you should periodically archive and reset accumulated values.

The following table lists some recommended peak values you should consider purging when no longer needed or when archived.

IF you want to reset ...	THEN use ...	For instructions and examples, see ...
peak disk space values in DBC.DatabaseSpace	DBC.ClearPeakDisk macro	Housekeeping on an Ad-Hoc Basis.
accumulators in DBC.Acctg	the ClearAccounting macro	Clearing Out Values Manually in the DBC. Acctg Table.
count values, AccessCount and LastAccessTimeStamp fields, in: <ul style="list-style-type: none"> • DBC.Dbase • DBC.TVM • DBC.TVFields • DBC.Indexes 	one of these macros: <ul style="list-style-type: none"> • ClearAllDatabaseUseCount • ClearDatabaseUseCount • ClearTVMUseCount 	Resetting Use Counts.

Correcting DBC.DataBaseSpace and DBC.DBase Values

As a result of very rare types of system failures, you might need to correct inconsistencies in the system tables DBC.DataBaseSpace and DBC.DBase. Use the Update DBC and Update Space utilities only when you need to perform these tasks.

Utility	Description
Update DBC	Recalculates the maximum allowed values for permanent, temporary, and spool space and uses those values to update the DBC.DBase system table and the DBC.DataBaseSpace system table.
Update Space	Examines the storage descriptors and recalculates the current usage of space used by: <ul style="list-style-type: none"> • A single database and its individual tables • All databases in a system and their individual tables <p>Note: Values in DBC.DBase are global values. Values in DBC.DataBaseSpace are local AMP values, that is, the global value divided by the number of AMPs in the system.</p>

For instructions on each utility, see *Teradata Vantage™ - Database Utilities*, B035-1102. For descriptions of the system tables and the views that access them, see *Teradata Vantage™ - Data Dictionary*, B035-1092.

Collecting Statistics on Data Dictionary Tables

If you are using custom applications or business intelligence tools, you may find it useful to collect statistics on Data Dictionary tables because these applications and tools often query the Data Dictionary for validation or information checks in the background.

Components within Vantage use internal express requests that are optimized single-AMP queries. Statistics on Data Dictionary tables will not benefit these internal activities but will instead benefit SQL submitted directly by a user or an application.

Any potential benefit from collecting statistics will depend on the amount and complexity of the dictionary access activity. Using DBQL can help determine the benefit although it requires the DBC logon to collect against Data Dictionary tables. Before you decide whether to collect statistics on Data Dictionary tables, examine the DBQL output to determine if your applications use a lot of resources to access the Data Dictionary.

Note:

Not all companies find value in collecting statistics on dictionary tables. The value depends on the frequency and complexity of the queries that access the dictionary tables, the amount of resource they use, and how long they run on your platform.

The following tables are good candidates for statistics collection:

- DBC.AccessRights
- DBC.DBase
- DBC.Indexes
- DBC.Owners
- DBC.Profiles
- DBC.Roles
- DBC.RoleGrants
- DBC.TVFields
- DBC.TVM
- DBC.UDFInfo

Only hashed dictionary tables allow statistics collections. Do not collect statistics on these non-hashed Data Dictionary tables:

- DBC.Acctg
- DBC.ChangedRowJournal
- DBC.DatabaseSpace
- DBC.LocalSessionStatusTable
- DBC.LocalTransactionStatusTable
- DBC.OrdSysChngTable
- DBC.RecoveryLockTable
- DBC.RecoveryPJTable
- DBC.SavedTransactionStatus
- DBC.SysRcvStatJournal
- DBC.TransientJournal
- DBC.UtilityLockJournalTable

For more information, see <https://downloads.teradata.com/blog/carrie/2010/05/if-you-re-not-collecting-statistics-on-your-dictionary-tables-do-it-now>.

Data Dictionary Tables That Require Maintenance

Dictionary Tables To Maintain

Reset accumulators and peak values using the DBC.AMPUsage view and the ClearPeakDisk macro provided with the software.

DBC.Acctg Resource usage by account/user	DBC.DataBaseSpace Database and table space accounting
--	---

Teradata automatically maintains these tables, but good practices can reduce their size.

DBC.AccessRights User rights on objects	DBC.RoleGrants Role rights on objects	DBC.Roles Defined roles	DBC.Accounts Account codes by user
---	---	-----------------------------------	--

Archive these logging tables (if desired) and purge information 60-90 days old. Retention depends on customer requirements.

DBC.SW_Event_Log Database console log	DBC.ResUsage Resource monitoring tables	DBC.EventLog Session logon/logoff history
DBC.AccLogTbl Logged user/object events	DBC.DBQL tables Logged user/SQL activity	NetSecPolicyLogTbl Logs dynamic security policy audit trails
NetSecPolicyLogRuleTbl Controls when and how dynamic security policy is logged		

Purge these tables when the associated removable media is expired and overwritten.

DBC.RCEvent Archive/recovery events	DBC.RCConfiguration Archive/recovery config	DBC.RCMedia VolSerial for Archive/recovery
---	---	--

Housekeeping on an Ad-Hoc Basis

You should occasionally clean out rows or reset the fields in the following tables as needed:

- **DBC.AccessRights table**

When a user or database is created, there is a default number of privileges added to DBC.AccessRights. These entries should be reviewed and obsolete or redundant entries deleted as necessary. Otherwise, this table can grow very large and can greatly impact space limits negatively. This table may also potentially become skewed.

In addition, when an object is dropped, the system may do a full table scan against DBC.AccessRights. If DBC.AccessRights is large, this can adversely affect the performance of the drop operation. The smaller this table can be made, the better.

To reduce the size of the DBC.AccessRights table:

- Determine which users have the most entries. The following query checks for the userid/database ID that has the most rows that hash to the same value:

```
.set retlimit 10
.set ret cancel
LOCKING ROW FOR ACCESS
SELECT COUNT(*) (TITLE '# of//Entries', FORMAT 'ZZZZZZ9'),
  a.userid|| a.databaseid (TITLE 'UserID|DatabaseID'),
  dbase.databasenamei (TITLE 'UserName')
FROM accessrights a , dbase
WHERE a.userid = dbase.databaseid
GROUP BY 2,3 ORDER BY 1 DESC,2 ;
```

- REVOKE rights at the table level and GRANT rights at the database level to reduce the number of entries.
- Convert individual privileges to role privileges where possible.

- **DBC.TSETQueryText table**

If you receive a 3610 error, the query text may be saved to the DBC.TSETQueryText table for use by Teradata Support. After the problem is resolved, you may need to manually delete the rows where the “Handled” column is set to 1 or has a timestamp that indicates the information is no longer needed.

- **Archive and recovery tables**

When associated removable media is expired and over-written, clean out the following:

Table	Purpose
DBC.RCConfiguration	archive or restore configuration
DBC.RCMedia	VolSerial for archive or restore
DBC.RCEvent	archive or restore events

- **Accumulated values in Data Dictionary tables**

Resetting values can help you do better resource usage analysis, and keep cleaner historical records. The following fields should be reset when necessary:

Fields	Purpose
PeakPermSpace, PeakSpoolSpace, and PeakTempSpace in the DBC.DataBaseSpace table.	<p>Contains database and table space accounting information. To reset the values, execute the macro:</p> <pre>EXEC DBC.ClearPeakDisk;</pre> <p>Due to the importance of tracking space usage in your system, you should try to run this macro regularly. Or, you can submit an UPDATE statement yourself.</p>

Fields	Purpose
CPUTime and DiskIO of the DBC. AMPUsage view (which corresponds to the DBC.Acctg table).	Holds resource usage data by Acct/User. To reset the values, submit the following: <pre>UPDATE DBC.AMPUsageV SET CPUTime = 0, DiskIO = 0 ALL;</pre>
AccessCount and LastAccessTimeStamp of the following views: <ul style="list-style-type: none"> ◦ DBC.ColumnsV ◦ DBC.DatabasesV ◦ DBC.IndicesV ◦ DBC.TablesV ◦ DBC.UsersV 	Tracks the number of times and the last time a particular object was accessed. To reset these fields, use one of these macros: <ul style="list-style-type: none"> ◦ ClearAllDatabaseUseCount ◦ ClearDatabaseUseCount ◦ ClearTVMUseCount

Updating Software (Applying Proper Patches)

Updating your software to the most current patch levels can sometimes help eliminate potential problems. This includes software patches for both the database and the client. While it is not possible or practical to continually update software every time there is a new version, it is a good practice to keep Vantage, as well as other software on your system, as current as reasonably possible. It is a good idea to monitor Tech Alerts and known fixes through Teradata Access:

1. Go to <https://support.teradata.com>.
2. Log in.

If you determine from a Tech Alert that a fix should be applied to your system, either you should contact your Teradata Support personnel or your customer support representative will help initiate the process for the patch or upgrade. When preparing for patching or upgrading software, keep the following in mind:

- Determine when system down time would least impact your operations.
- Factor in time and work required to archive data.
- Check if the gateway version, the PDE version, and DBS version are compatible.

For more information, contact Teradata Support personnel. Or, if you have a service account, check the service website.

Archiving, Restoring, and Recovering Data: Operational DBAs

Archive/Restore Utility Support

If MAPS is enabled, use Teradata Data Stream Architecture (DSA) for archive/recovery operations. See *Teradata® DSA User Guide*, B035-3150.

Managing Database Resources: Operational DBAs

This section describes strategies for managing system I/O, database workload, and tracking system object use and table changes. It also describes query banding and provides an overview of resource usage trend analysis.

Managing I/O with Cylinder Read

A data block is a disk-resident structure that contains one or more rows from the same table and is the smallest I/O unit for the file system. Data blocks are stored in physical disk sectors or segments, which are grouped in cylinders.

Vantage employs Cylinder Read to enable full-file scan operations to run more efficiently by reading a list of cylinder-resident data blocks with a single I/O operation. This reduces I/O overhead from once per data block to once per cylinder. This can significantly reduce the time it takes to do a full-table scan of large tables. During installation, Cylinder Read is enabled by default.

Note:

If you notice a performance hit from Cylinder Read, try decreasing the value of the Cylinder Read Aging Threshold control field in the Ctl DBS screen. For information on this field, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Cylinder Read may improve performance during operations such as:

- Full-table scan operations under conditions such as:
 - Large select
 - Merge INSERT ... SELECT
 - Merge delete
 - Sum, average, minimum, maximum, and count aggregates
- Joins operations that involve many data blocks, such as merge or product joins.

Cylinder reads are not invoked when there is row partitioning elimination or column partitioning elimination.

For more information, see [Managing I/O with Cylinder Read](#).

Tracking Cylinder Read Resource Usage

To track Cylinder Read behavior if you enable resource usage logging, see the Cylinder Read Columns in the “ResUsageSvpr Table” in *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

Managing the Database Workload with Teradata Active System Management

The system provides a number of optional Teradata Active System Management (TASM) features that provide workload management capabilities. You can use the Teradata Viewpoint Workload Designer portlet to create rules that define how the work submitted to Vantage should behave and under what circumstances.

Do the following to implement TASM:

1. Determine workload management needs.
2. Create a ruleset and define the general parameters.
3. Define session rules
4. Define filter rules.
5. Define throttle rules.
6. Define workload classifications.

Determining Workload Management Needs

Before defining TASM workload management parameters, you should evaluate how your database is used. The objective is to identify job handling and query limitation needs that cannot be met by performance group assignments and scheduling.

- Monitor database activity using the Teradata Viewpoint Metric Heatmap portlet to understand overall workload concurrency and possible resource contention. For information on the Metric Heatmap portlet, see *Teradata® Viewpoint User Guide*, B035-2206.
- Use the Teradata Viewpoint Query Monitor and Lock Viewer portlets to study resource contention at various times of day, and to understand blocking and wait time patterns.
- Optionally, use the database query logging (DBQL) logging feature to determine query performance details. For information on tracking processing behavior with DBQL, see [BEGIN/REPLACE/END QUERY LOGGING Statements](#).
- Optionally, monitor the ResUsageSPS table to determine resource usage by performance group. For information on the ResUsageSPS table, see *Teradata Vantage™ - Data Dictionary*, B035-1092.

Defining a Ruleset

You must create a TASM ruleset to contain any workload management rules that you create. You can create multiple rule sets, but only one ruleset can active at a time. For instructions on using the Workload Designer portlet, see *Teradata® Viewpoint User Guide*, B035-2206.

In the Ruleset General View:

1. Name the ruleset.
2. Use the default Interval settings.
3. Use the default Blocker settings.
4. Defining Bypass rules is optional, but not recommended for initial implementation.

Note:

Database user DBC bypasses all rules by default.

5. Create the needed classification session, filter, and throttle rules within the ruleset to manage your database workload, as shown in the topics that follow.

Classification Settings

You can set the following classification parameters for the session, filter or throttle rules you create, to narrow the effects of the rules, unless you want a rule to apply globally.

- Source: account name, account string, username, or client IP address
- Target: The name of a database object that the query accesses, for example, a database, table, macro, view, function name, or method name.
- Query characteristics, for example, statement type, join type, or full table scan
- Query band
- Utility (source of query)

Session Rules

You can use the sessions tab in a ruleset to set limits on the:

- Number of concurrent query sessions.
- Number of concurrent utility sessions
- Number of concurrent sessions for a specific utility
- The order of precedence of utility session rules

Filter Rules

A filter rejects a query *before* the query starts running. You can apply filters globally or according to classification rules.

For example, you can create a filter that prohibits a specific user (source classification) from running a query with an estimated processing time of longer than 15 minutes (query characteristic classification).

For information on Ruleset Filters, see *Teradata® Viewpoint User Guide*, B035-2206.

Throttle Rules

A throttle limits the number of concurrent queries in a filter classification. When the throttle limit is reached, additional affected queries go into the delay queue.

You can create throttles to limit concurrency according to throttle classification, for example:

- Limit a specific user (source classification) to running no more than 10 queries at a time (query session limit)
- Limit the number of utilities (utility classification) simultaneously executing against a table (target classification)

- Limit the number of memory-intensive functions, such as XMLQUERY, and memory-intensive SQL queries executing concurrently

Suggested Custom Throttles

The following example throttles can be useful for most implementations.

Throttle all at 52:

- Limits the database to execution of 52 concurrent queries
- Places additional queries into the delay queue
- Prevents unusually large query demand from putting the system into flow control

Throttle High, Medium, and Low PG queries with estimated processing time > 1 second:

- Sets a limit of 30 concurrently executing H, M, or L queries that have an estimated processing time > 1 second. There is no built-in limit on shorter queries.
- Places additional long queries into the delay queue.
- This throttle does not apply to short queries (<1 sec), or to an R query of any size.

Ensuring a Consistent Response Time for a Workload

On SLES systems, database administrators can specify a minimum response time for a workload, causing Teradata to hold request responses until that time has passed. A minimum response time provides consistent results regardless of changing conditions and helps to set reasonable expectations for users. For example, responses may naturally be very fast in a new system, but once more users and workloads have moved to the new system, responses may take longer. Specifying a minimum response time allows a database administrator to set reasonable expectations from the beginning.

The response wait time can be set from 1 to 3,600 seconds in the Hold Query Responses tab of the Workload Details screen in the Teradata Viewpoint Workload Designer portlet. This option does not apply to tactical workloads. For more information, see *Teradata® Viewpoint User Guide*, B035-2206.

Managing Sessions and Transactions with Query Banding

How can you identify a unit of work such as a report that consists of multiple SQL requests that span multiple tiers in the enterprise, such as a web service? Use query banding.

A query band is a set of name-value pairs assigned to a session, transaction, or profile that you can use to identify the originating source of a query. This is particularly useful for identifying specific users submitting queries from a middle-tier tool or application. Query banding information helps you answer the following types of questions:

- **Which external users are using the most resources?**

Access the information stored in the QueryBand field in the DBC.DBQLogTbl table, unless query logging is disabled. See [Tracking Query Behavior with Database Query Logging: Operational DBAs](#).

For example:


```

Sel Username (Format 'x(10)'), queryband(Format 'x(40)'), AMPCPUTime
from qrylogV
where ampcputime > .154;

```

Result:

```

*** Query completed. 9 rows found. 3 columns returned.
*** Total elapsed time was 1 second.

```

UserName	QueryBand	AMPCPUTime
-----	-----	-----
TWMUSER1	=S> CLUser=KShort;Dept=Sales;Job=z995;	0.188
TWMUSER18	=S> CLUser=TJuli;Dept=DMgr;Job=x1235;	0.170
TWMUSER	=S> CLUser=TJuli;Dept=DMgr;Job=x1234;	0.171
TWMUSER13	=S> CLUser=BPut;Dept=Mgr;Job=q2120;	0.173
TWMUSER1	=S> CLUser=KShort;Dept=Sales;Job=z995;	0.157
TWMUSER27	=S> CLUser=KShort;Dept=Sales;Job=z996;	0.186
TWMUSER2	=S> CLUser=DThom;Dept=Manuf;Job=a100;	0.156
TWMUSER28	?	0.158
TWMUSER	=S> CLUser=DGale;Dept=Mktg;Job=m125;	0.158

```

BTEQ -- Enter your DBC/SQL request or BTEQ command:

```

You can even use query bands to determine what functional area of an application is using the most resources.

- **How can I control how a workload or type of query from a specific session uses system resources?**

Set up rules using Teradata Viewpoint Workload Designer portlet and associate query bands with filter rules or define them as workload definition attributes.

Defining Name-Value Pairs for Session and Transaction Query Bands

To assign values to a query band for a session or transaction, submit a SET QUERY_BAND statement. You can submit a SET QUERY_BAND = NONE FOR SESSION or NONE FOR TRANSACTION at any time to remove the query band. For the syntax and more extensive examples, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

To set a query band for a profile, use a CREATE/MODIFY PROFILE request. For more information, see [Defining Name-Value Pairs for Profile Query Bands](#).

Example: Setting a Query Band For a Session

The following example sets a query band with two name:value pairs for the session.

```
SET QUERY_BAND = 'org=Finance;report=Fin123;' FOR SESSION;
```

The pairs in the query band string are *org=Finance* and *report=Fin123*.

Example: Removing a Query Band From a Session

The following example removes the query band for the session.

```
SET QUERY_BAND = NONE FOR SESSION;
```

The following request is equivalent.

```
SET QUERY_BAND = '' FOR SESSION;
```

Example: Setting a Query Band for the Current Transaction

The following example sets a query band with two name:value pairs for the current transaction.

```
SET QUERY_BAND = 'Document=XY1234;Universe=East;' FOR TRANSACTION;
```

The pairs in the query band string are *Document=XY1234* and *Universe=East*.

Defining Name-Value Pairs for Profile Query Bands

To assign values to a query band in a profile, submit a CREATE/MODIFY PROFILE request. The profile query band is automatically set for the session at logon. You can submit a MODIFY PROFILE QUERY_BAND = NULL request at any time to remove the query band from the profile. For the syntax and more extensive examples, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

If the transaction, session, and profile query bands have the same name but with different values, the name/value pair used for workload management classification is the first one found in a query band searched in the following order:

- Transaction query band
- Session query band
- Profile query band

Example: Creating a Query Band in a Profile

```
CREATE PROFILE salesprofile AS
  SPOOL = 2e6*(HASHAMP()+1),
  QUERY_BAND = 'GROUP=sales;AREA=retail;';
```

Defining Profile Query Bands with Ignore Query Band Values

If you choose the IGNORE QUERY_BAND VALUES option, which prevents a user with that profile from setting certain query band values, an empty string means that the user is not allowed to set any values for that name. For example:

```
CREATE PROFILE testprofile AS
  IGNORE QUERY_BAND VALUES = 'IMPORTANCE=';
```

Trailing spaces are always removed from names and values, so if a value is specified as several spaces, the spaces will be removed and the value will be an empty string.

To specify multiple values for a name, specify each name-value pair separately, as in the following example. When a user with *salesprofile* issues a SET QUERY_BAND FOR SESSION/TRANSACTION request, the following example causes Vantage to ignore the values WARM and HOT for TVSTemperature and issue an error message.

Example: Creating a Query Band in a Profile with Ignore Query Band Values

```
CREATE PROFILE salesprofile AS,
  QUERY_BAND = 'TVSTemperature=COLD;' DEFAULT,
  IGNORE QUERY_BAND VALUES = 'TVSTemperature=HOT;TVSTemperature=WARM;'
```

Query Bands and Proxy Users

When using a middle-tier application to perform requests to Vantage on behalf of an application end user, use the SET QUERY_BAND statement. The SET QUERY_BAND statement adjusts the session active user and allows privilege checking and auditing based upon the PROXYUSER and PROXYROLE name-value pairs. The system validates the privileges of the current user or role to connect as a specified proxy user.

The proxy connection lasts for the life of the query band. The session query band remains set for the session until the session ends or the query band is set to NONE.

The session query band is stored in the session table and recovered after a system reset. The transaction query band is discarded when the transactions ends (commit, rollback, or abort) or the transaction query band is set to NONE. The transaction query band is not restored after a system reset.

Finding the Origin of a Query Using Query Bands

A query band is a set of name-value pairs assigned to a session or transaction that identifies the originating source of a query. You define these identifiers and they get stored along with other session data in DBC.SessionTbl.

This is particularly useful for identifying specific users submitting queries from a middle-tier tool or application, since these often use pooling mechanisms that hide the identity of the users when each connection in the pool logs in to the database using the same user account.

For more information on query bands and how to assign them to a session or transaction, see SET QUERY_BAND in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

After you create a query band set, retrieve the query band and the name-value pairs by using:

- HELP SESSION or query the QueryBand field of the DBC.SessionInfo view.
- System UDFs and external stored procedures. For example, to find the query bands of active sessions, use the MonitorQueryBand function:

```
BTEQ -- Enter your DBC/SQL request or BTEQ command:
SELECT t1.sessionno, MonitorQueryBand(Hostid, sessionNo, runvprocno)
FROM TABLE (MonitorSession(1,'*',0)) AS t1;
```

Result:

```
*** Query completed. 2 rows found. 2 columns returned.
*** Total elapsed time was 1 second.

SessionNo  MonitorQueryBand(HostId,SessionNo,RunVprocNo)
-----
1299
1298 =S> REPJOBID=495;REPORT=Sales;CLIENTMACHINE=XYZ12;REPTYPE=10;
```

For more information on the MonitorQueryBand function and a full list of available functions and procedures, see *Teradata Vantage™ - Application Programming Reference*, B035-1090.

- Query logging (DBQL). Query logging stores query band information in the DBC.DBQLLogTbl table. Use the DBC.QryLogV view to access this table. You can use the information to analyze queries after they execute.

```
BTEQ -- Enter your DBC/SQL request or BTEQ command:
```



```
sel queryband(FORMAT 'X(50)'), querytext(FORMAT 'X(30)') from qrylogv
order by
StartTime, queryid;
```

Result:

```
*** Query completed. 10 rows found. 2 columns returned.
*** Total elapsed time was 1 second.

QueryBand                                QueryText
-----
?                                         Begin query logging on a
=S> REPJOBID=495;REPORT=Sales;CLIENTMACHINE=XYZ12; sel count(*) from dbc.da
=S> REPJOBID=495;REPORT=Sales;CLIENTMACHINE=XYZ12; sel count(*) from dbc.ta
=S> REPORT=Costs;CLIENTMACHINE=XYZ12;    SET QUERYChapter 11,
Managing C_BAND='REPORT=C
=S> REPORT=Costs;CLIENTMACHINE=XYZ12;    sel * from tdwm.rules;
=S> REPORT=Costs;CLIENTMACHINE=XYZ12;    sel * from tdwm.ruledefs
=S> REPORT=Costs;CLIENTMACHINE=XYZ12;    sel * from configuration
=S> REPJOBID=99;REPORT=Acct;CLIENTMACHINE=BBB1; SET QUERY_BAND='REPJOBID
=S> REPJOBID=99;REPORT=Acct;CLIENTMACHINE=BBB1; sel database from dbc.da
?                                         End query logging on all
```

For more information on the database query log, see [Tracking Query Behavior with Database Query Logging: Operational DBAs](#).

- Teradata Viewpoint to track requests that meet a certain CPU or duration threshold.

Teradata also supplies library functions to enable UDFs, methods, and external procedures to retrieve query band information. For example:

TO return this value from the current query band...	USE this library function...
name	FNC_GetQueryBand
value	FNC_GetQueryBandValue
all name:value pairs	FNC_GetQueryBandPairs

For more information, see *Teradata Vantage™ - SQL External Routine Programming*, B035-1147.

Other Uses for Query Bands

Other uses for query bands include:

- Preventing development work from running on the production platform at critical times. For example, queries with a query band of 'Application=Test;' can be prevented from running, based on a TASM filter rule.
- Recording the parameterized values that the query uses for each execution of tactical Java applications. These values are available in DBQL after the queries complete.
- Choosing between dynamic plan execution or static query plan execution.
- Setting the temperature of data being loaded into the system.

Related Information

For the syntax for setting a query band, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144. For the available query band values, see *Teradata Vantage™ - SQL Data Definition Language Detailed Topics*, B035-1184.

Tracking Object Use and Table Changes

You can track both object use and table changes to understand database use and provide an accurate foundation for automated statistics management.

Object use counts and last access timestamps provide a simple way to identify how frequently user queries access or use specific database objects. You can analyze system performance by noting objects that are heavily used and adjust accordingly. Or you can improve resource availability by deleting objects the system rarely accesses to reclaim disk space or tune frequently used objects.

Objects That Can Be Tracked

You can determine the number of times user queries access or use the following objects:

<ul style="list-style-type: none"> • Columns • Databases • Indexes • Macros 	<ul style="list-style-type: none"> • Stored Procedures • Table statistics • Tables • Triggers 	<ul style="list-style-type: none"> • UDFs • UDMs • UDTs • Users • Views
---	---	--

Note:

Vantage does not collect object use counts and last access timestamps for Data Dictionary tables or views or for the EXPLAIN, INSERT EXPLAIN, or DUMP EXPLAIN request modifiers.

Table Changes That Can Be Tracked

In addition to collecting access counts (which are done once per request), object use counts collect the number of updates, deletes, and inserts performed against database tables involved in the request. This

information is useful in calculating table row count estimations, statistics extrapolations, and thresholds for statistics collection.

Vantage supports these requests and logs them to the Data Dictionary table `DBC.ObjectUsage`:

- Selects
- Updates
- Deletes
- Inserts
- Merges (including merge, merge-delete, and merge-update)
- Upserts
- Statistics access

Note:

To see how many rows are updated, inserted, and deleted in a request (not categorized by table), enable database query logging and see the `StmtDMLRowCount` field in either `DBC.DBQLogTbl` or the `DBC.QryLogV` and `DBC.QryLogV_sz` views.

Enabling Object Use Counts

To enable object use counts, use the `WITH USECOUNT` option of the `BEGIN/REPLACE QUERY LOGGING` request. This request starts collection of object use counts in the Data Dictionary table `DBC.ObjectUsage`. Object use counts are disabled by default, but it is a best practice to enable them for production databases to ensure the accuracy of automated statistics.

Vantage tracks two versions of these counts, which collect identical information:

- User counts (a user can reset).
- System counts (only the system can reset).

Controlling How Often Counts Are Recorded

The `ObjectUseCountCollectRate` field in `DBS Control` controls when the system writes the counts to the Data Dictionary. The field sets the number of minutes you want the system to wait before it writes the cached counts to the Data Dictionary table `DBC.ObjectUsage`. The recommended minimum value is 10 minutes. The default value is also 10 minutes. Any rate below 10 minutes may impact performance of systems with heavy workloads. The counts also write to disk when the cache is full. You can also write the counts to `DBC.ObjectUsage` whenever applications require it by using a `FLUSH QUERY LOGGING WITH USECOUNT` request.

Views That Report Count Data

These Data Dictionary views report the count data.

This view...	Provides this information...
ColumnsV[X]	AccessCount and LastAccessTimeStamp for selected columns.
DatabasesV[X]	AccessCount and LastAccessTimeStamp for selected databases.
IndicesV[X]	AccessCount and LastAccessTimeStamp for hash indexes or join indexes.
TablesV[X]	AccessCount and LastAccessTimeStamp for selected tables, views, macros, stored procedures, triggers, and functions.
Users	AccessCount and LastAccessTimeStamp for users that the requesting user owns or has MODIFY or DROP privileges on.
DatabaseUseCountV[X]	database use counts, by usage type.
ObjectUseCountV[X]	object use counts, by object usage type.
ColumnUseCountV[X]	column use counts, by usage type.
IndexUseCountV[X]	index use counts, by usage type.
StatUseCountV[X]	statistics use counts, by table and statistics.
QueryStatUseCountV[X]	query statistics use counts, by table and statistics.
UpdateUseCountV[X]	update count per table and column, by usage type.
DeleteUseCountV[X]	delete count per table, by usage type.
InsertUseCountV[X]	insert count per table, by usage type.

Resetting Use Counts

When you enable query logging for a database, system and user use counts and timestamps are reset. However, when you disable query logging, only system counts and timestamps are reset. User use counts and timestamps are not reset. To reset them, use these macros:

This macro...	Resets...
ClearAllDatabaseUseCount	AccessCount and LastTimeStamp for all objects in the system.
ClearDatabaseUseCount	AccessCount and LastTimeStamp of objects in a specified database.
ClearTVMUseCount	the AccessCount and LastAccessTimeStamp for all of the objects in a specified table, view, macro, trigger, stored procedure, join index, user-defined type, user-defined method, or user-defined function.
ClearUserUseCount	all access counts (excluding statistics) for a particular user.
ClearUserStatCount	all statistics counts for a particular user.
ClearUserUDICount	all update, delete, and insert counts for a particular user.

Note:

You must have the EXECUTE privilege on the macros to use them.

To create these macros, check if the DIP utility script DIPVIEWSV has run. If not, run DIPVIEWSV.

Note:

All SQL statements used by the ClearDatabaseUseCount macros place write locks on the database objects involved.

Examples of Resetting Use Counts

TO reset the AccessCount and LastAccessTimeStamp fields for...	USE the following syntax...
the table tab1	Exec DBC.ClearTVMUseCount('db1', 'tab1'); Specify the databasename and the tablename in apostrophes, separated by a comma.
the database db2	Exec DBC.ClearDatabaseUseCount('db2'); Specify databasename in apostrophes.
all the objects in the system	Exec DBC.ClearAllDatabaseUseCount;

Usage Recommendations

When collecting object use counts, remember the following:

- Do not enable collection when a Data Dictionary ARCHIVE or RESTORE is in progress.

Note:

If the Data Dictionary is being archived and the archive gets blocked by the ObjectUseCount collection, the system will turn off the ObjectUseCount collection. You must enable it again after the archive is complete.

- The recommended value of ObjectUseCountCollectRate is 10 minutes or more. Setting ObjectUseCountCollectRate to less than 10 minutes impacts system performance.

Copying the Use Counts of Table

CREATE TABLE ... AS ... WITH DATA AND STATISTICS copies the statistics and data of an original table to a target table. If the USECOUNT option is enabled on the target database, the AND STATISTICS option also copies the use count information. For more information, see [Copying Statistics From a Base to a Target Table](#) or *Teradata Vantage™ - SQL Data Definition Language Detailed Topics*, B035-1184, CREATE TABLE (AS Clause).

Renaming Objects Has No Effect on Use Counts

Renaming an object does not reset its use count. That is, neither the access count nor the last access timestamp changes when you rename an object. For example, the following statement:

```
RENAME TABLE Employee TO Emp;
```

does not reset the use count for the table “Emp” to 0. The count remains what it was for “Employee” with the timestamp it was last used. When “Emp” is next used, the system continues the tally and updates the timestamp. Similarly, for columns of a table, if you rename a column using the following request:

```
ALTER TABLE t1 RENAME oldcolumnname TO newcolumnname;
```

this does not reset the use count information for the column.

Because using the RENAME statement does not affect use count information, if you need to reset the count, manually reset it using the ClearTVMUseCount macro.

Example: Object Use Counts for Tables

Assume two databases, db1 and db2, each contain some tables and other objects. Assume also that count collection is enabled.

Now specify some SQL requests to access the tables in the two databases.

```
sel *from db1.tab1;
sel *from db1.tab1 where col1=7;
sel *from db1.tab1 where col2=7;
sel *from db1.tab2;
sel *from db2.tab3;
sel *from db2.tab3 where col1=7;
sel *from db2.tab4;
```

Then use the Data Dictionary view TablesVX to find the use counts for the tables.

```
sel DatabaseName, TableName, AccessCount, LastAccessTimeStamp
from DBC.TablesVX where TableName IN ('tab1', 'tab2', 'tab3', 'tab4');
```

The output is:

```
*** Query completed. 4 rows found. 4 columns returned.
*** Total elapsed time was 1 second.
DatabaseName  db1
              TableName  tab2
```



```

      AccessCount 1
LastAccessTimeStamp 2006-12-17 14:51:53
      DatabaseName db2
      TableName tab4
      AccessCount 1
      DatabaseName db1
LastAccessTimeStamp 2006-12-17 14:51:53
      TableName tab1
      AccessCount 3
LastAccessTimeStamp 2006-12-17 14:51:53
      DatabaseName db2
      TableName tab3
      AccessCount 2
LastAccessTimeStamp 2006-12-17 14:51:53
=====

```

Note:

If any of the named objects have not been accessed since the last use count reset, the value in the AccessCount column for that object appears as zero, and the LastAccessTimeStamp will appear as a question mark.

Analyzing Trends with Account String Expansion Variables

You can use account string expansion variables to analyze resource usage trends. See [Logging Resource Usage Data with Account String Variables](#).

Resource Usage Trend Analysis

ResUsage Macros and Tables

You can use resource usage macros to report resource usage data. You can enable resource usage tables to:

- Observe the balance of disk and CPU usage
- Obtain pdisk usage information
- Check for bottlenecks
- Provide an overall history of the system operation
- Monitor the utilization of AMP Worker Tasks (AWTs)

Resource usage data is useful for the following purposes:

- Measuring system performance

- Measuring component performance
- Helping with on-site job scheduling
- Identifying potential performance impacts
- Planning installation, upgrade, and migration
- Analyzing performance degradation and improvement
- Identifying problems such as bottlenecks, parallel inefficiencies, down components, and congestion

Enabling RSS Logging

You can enable the resource usage tables using the Control GDO Editor (ctl) utility or the database commands in Database Window (DBW).

Before you enable the resource usage tables, determine which tables apply to the resource usage macros you want to run. For more information, see *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

ctl

You can set various configuration settings using ctl. The RSS screen in ctl allows you to specify the rates of resource usage data logging. For detailed information on starting ctl and modifying the settings, see "ctl" in *Teradata Vantage™ - Database Utilities*, B035-1102.

DBW

Note:

For instructions on starting the Database Window, see "Database Window (xdbw)" in *Teradata Vantage™ - Database Utilities*, B035-1102.

To enable RSS logging from DBW

1. Open the **Supvr** window.
2. Set the Node Logging Rate using the following database command.

```
SET RESOURCE { LOGGING | LOG } number
```

where *number* is the number of seconds.

Note:

A rate of zero disables the logging function.

3. Specify the table you want to enable logging to using the following database command.

```
SET LOGTABLE { tablename | ALL } { ON | OFF }
```


where *tablename* is the suffix part of ResUsage Xxxx. For example, for the DBC.ResUsageSpma table, the *tablename* would be “Spma.”

After the table is enabled for logging, you can log rows in Summary Mode.

Note:

To log rows in Summary Mode, you must enable the table specified in both the RSS Table Logging Enable group and in the RSS Summary Mode Enable group.

4. (Optional) Enable Summary Mode on the table specified using the following command.

```
SET SUMLOGTABLE tablename { ON | OFF }
```

Example of Enabling Table Logging and Setting the Logging Rate

The following example shows how to enable table logging and set the Logging rate using the database commands in DBW. Suppose you want to enable the ResUsageShst table and set the logging rate for 10 minutes (600 seconds). You would enter the following:

```
set logtable shst on  
set resource log 600
```


Managing Queries: Operational DBAs

This section provides:

- Instructions for enabling and disabling Redrive protection for queries
- Techniques for finding and resolving common problems with queries

Redrive Protection for Queries

Redrive protection makes database restarts transparent to applications and users. After a Analytics Database restart, incomplete SQL requests are automatically resubmitted for sessions that have Redrive enabled.

Enabling Redrive Protection

To enable Redrive protection...	Do...
at the system level	Set the RedriveProtection DBS Control field to 1.
for all sessions by default	<ul style="list-style-type: none"> • Ensure that the RedriveProtection DBS Control field is set to 1. • Set the RedriveDefaultParticipation DBS Control field to 1.
at the session level	Set the Redrive field in the Session Options parcel to 'Y'.
at the user, account, or profile level	Insert or modify a row in the TDWM.Redrive table and set the value of the RedriveMethod column to 2 for the user, account, or profile. For details, see Setting Redrive Protection for Users, Accounts, and Profiles .

Note:

If any of the following is true, the session will not participate in Redrive:

- The RedriveProtection DBS Control field is set to 0
- The Redrive field in the Session Options parcel is set to 'N'
- There is a row in the TDWM.Redrive table indicating that Redrive is disabled for this session

In addition, Recoverable Network Protocol must be enabled in order for you to use Redrive. For details about Recoverable Network Protocol, see the DisableRecoverableNetProtocol DBS Control field in *Teradata Vantage™ - Database Utilities*, B035-1102.

The Redrive capability is assigned to a session at logon time. Once a session is enabled with Redrive, you can subsequently use the Redrive query band to toggle Redrive protection off and on during the life of the session using this syntax:


```
SET QUERY_BAND = 'redrive=on;' FOR SESSION;
SET QUERY_BAND = 'redrive=off;' FOR SESSION;
```

For details on using the Redrive query band, see *Teradata Vantage™ - SQL Data Definition Language Detailed Topics*, B035-1184.

Disabling Redrive Protection

To disable Redrive protection...	Do...
at the system level	<ul style="list-style-type: none"> Disable logons from the Supervisor window of Database Window (DBW) or similar console subsystem interface, such as cnstern. Ensure that no sessions are logged on. Set the RedriveProtection DBS Control field to 0.
for all sessions by default	<ul style="list-style-type: none"> Ensure that the RedriveProtection DBS Control field is set to 1. Set the RedriveDefaultParticipation DBS Control field to 0.
at the session level	Set the Redrive field in the Session Options parcel to 'N'.
at the user, account, or profile level	Insert or modify a row in the TDWM.Redrive table and set the value of the RedriveMethod column to 1 for the user, account, or profile. For details, see Setting Redrive Protection for Users, Accounts, and Profiles .

Setting Redrive Protection for Users, Accounts, and Profiles

You can enable or disable Redrive protection for individual users, accounts, and profiles using the TDWM.Redrive table:

1. Ensure that TASM is running.
2. Insert, modify, or delete rows in TDWM.Redrive to specify whether particular users, accounts, or profiles will participate in Redrive.
3. From the Teradata Viewpoint Workload Designer portlet, issue the TDWM Activate command after changing the TDWM.Redrive table in order for the new changes to take effect.

The table definition for TDWM.Redrive is:

```
CREATE SET TABLE TDWM.Redrive ,FALLBACK ,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL,
  CHECKSUM = DEFAULT,
  DEFAULT MERGEBLOCKRATIO
(
  ObjectType VARCHAR(10) CHARACTER SET UNICODE
    NOT CASESPECIFIC NOT NULL,
```



```

ObjectName VARCHAR(128) CHARACTER SET UNICODE
            NOT CASESPECIFIC NOT NULL,
RedriveMethod INTEGER NOT NULL)
UNIQUE PRIMARY INDEX ( ObjectType ,ObjectName );

```

Insert a row into the table or modify an existing row in the table to specify whether Redrive protection is enabled for a particular user, account, or profile:

Column Name	Valid Values	Description
ObjectType	In order of precedence, where USER is most specific and PROFIL is least specific: <ul style="list-style-type: none"> • USER • ACCT (account) • ACCTSTR (account string) • PROFIL (profile) 	Specifies whether you are setting Redrive protection for a user, an account, or a profile.
ObjectName	The name of the object	The name of the user (or application), account, or profile for which you are setting Redrive protection. For example, if ObjectType is USER, then ObjectName is the user name.
RedriveMethod	<ul style="list-style-type: none"> • 1 (Disable Redrive) • 2 (Enable Redrive) 	Determines whether Redrive is enabled or disabled for the user, account, or profile specified by ObjectName.

When there is a conflict, the most specific ObjectType is used. For example, suppose these rows are in TDWM.Redrive:

ObjectType	ObjectName	RedriveMethod
USER	Mark	1
PROFIL	Manager	2

If a session logs on as USER Mark and PROFIL Manager, since both rows in the table qualify, the most specific ObjectType is chosen. USER is more specific than PROFIL; therefore, RedriveMethod 1 (Do not use Redrive) is chosen and the session will not participate in Redrive.

If there is no row in TDWM.Redrive for the session, the RedriveDefaultParticipation DBS Control field determines whether the session participates in Redrive.

Redrive attributes are assigned to users, accounts, and profiles at logon time, and these attributes remain for the entire session. The attributes are not changed for that session even if new entries are made to TDWM.Redrive that would apply to the session. However, once Redrive is enabled, you can use the Redrive query band to turn Redrive off and on during the life of the session.

Redrive Restrictions

- Redrive only hides database restarts from requests running in the SQL partition.
- If a UDF or external stored procedure modifies data that is external to Vantage, then the request involving that function or procedure cannot be redriven.
- If a database restart occurs while SELECT AND CONSUME requests are waiting for rows to consume, the restart is hidden from the application, but the queued up requests will not be in the same order as they were at the time of the restart.
- If a PE node fails, then the restart cannot be hidden from applications that were logged onto the failing node.
- Database restarts cannot be hidden if the restart occurs:
 - During execution of stored procedures.
 - After the first request of an explicit Teradata transaction or an ANSI/ISO transaction and the transaction placed a write lock or exclusive lock on a table in database DBC.
 - After the request is committed on the AMPs but before the PE is notified of the commit.

Example: Redrive Settings and Session Behavior

When the session logs on and you have these settings...	Then the session...
RedriveProtection DBS Control field is 0	will not participate in Redrive.
<ul style="list-style-type: none"> • RedriveProtection DBS Control field is 1 • Redrive field in Session Options parcel is 'Y' • TDWM.Redrive contains a row matching your user name, account, account string, or profile with RedriveMethod = 1 (Do not use Redrive) 	will not participate in Redrive because the row in TDWM.Redrive specifies not to use Redrive.
<ul style="list-style-type: none"> • RedriveProtection DBS Control field is 1 • Redrive field in Session Options parcel is 'N' • TDWM.Redrive contains a row matching your user name, account, account string, or profile with RedriveMethod = 2 (Use Redrive) 	will not participate in Redrive because the Redrive field in the Session Options parcel is 'N'.
<ul style="list-style-type: none"> • RedriveProtection DBS Control field is 1 • Redrive field in Session Options parcel is 'Y' • TDWM.Redrive does not contain any matching row or it contains a row matching your user name, account, account string, or profile with RedriveMethod = 2 (Use Redrive) 	will participate in Redrive.

Example: Redrive Recommendations

If you want to use Redrive, but do not want to change any application settings:

- If you only want a single account to use Redrive:
 1. Set the RedriveProtection DBS Control field to 1.

2. Set the RedriveDefaultParticipation DBS Control field to 0.
 3. Add a row in TDWM.Redrive for the account and enter a value of 2 in the RedriveMethod column for the account.
- If you want all sessions to use Redrive except those from a particular account string:
 1. Set the RedriveProtection DBS Control field to 1.
 2. Set the RedriveDefaultParticipation DBS Control field to 1.
 3. Add a row in TDWM.Redrive for the account string and enter a value of 1 in the RedriveMethod column for the account string.

Related Information

Topic	Resources for Further Information
The RedriveProtection and RedriveDefaultParticipation DBS Control fields	<i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Using the REDRIVE query band to enable or disable Redrive protection for a session	<i>Teradata Vantage™ - SQL Data Definition Language Detailed Topics</i> , B035-1184

Recommendations for Common Query Problems

You can monitor queries using the Teradata Viewpoint Query Monitor portlet and other tools and utilities to identify poorly performing queries. Poor query performance is usually due to:

Query Problem	Result
Stale statistics	<p>The optimizer uses statistics to plan query execution. If the statistics are out of date, the optimizer may create a poor plan.</p> <p>Recommendation: Review tables to see whether statistics are out of date, and recollect statistics if required, as shown in Improving Query Performance Using COLLECT STATISTICS: Application DBAs.</p>
Badly skewed data	<p>Causes table rows to be distributed unevenly among AMPs and queries to execute unevenly, which slows processing time.</p> <p>Recommendation: Monitor table skew. Redefine the primary index if required. See Finding and Fixing Skewed Tables.</p>
Poor query construction	<p>Poor SQL syntax or missed punctuation can result in bad queries.</p> <p>Recommendations:</p> <ul style="list-style-type: none"> • Identify underperforming queries and the users responsible for them. See Identifying Poorly Performing Queries. • Abort underperforming queries when required. See Controlling Poorly Performing Queries.
Concurrency issues	<p>Some queries are delayed or prevented from executing because of lock contentions, or the relative priorities of concurrent queries.</p> <p>Recommendations:</p>

Query Problem	Result
	<ul style="list-style-type: none">• Identify the cause of a delayed query. See Investigating Query Blocks and Delays.• Release a query from the throttle delay cue. See Controlling Poorly Performing Queries.• Order the release of requests from the throttle delay queue based on workload priority, instead of first in, first out. See the General tab in the Teradata Viewpoint Workload Designer portlet.• Change the priority for a database operation or user.• Establish more detailed concurrency controls using TASM.

Identifying Poorly Performing Queries

When notified of delays by a user or an alert, you can use the Teradata Viewpoint Query Monitor portlet to find out if the delay is the result of a poorly performing query.

Note:

The Query Monitor portlet has many functions beyond what is shown in this example procedure. See *Teradata® Viewpoint User Guide*, B035-2206 for detailed setup and options.

1. Monitor query activity using the **By Session** tab.
2. Look for poorly performing queries using the following Query Monitor parameters:

- High CPU Use% combined with low REQ I/O accesses.
If the CPU divided by the I/O is > 100, the query is probably bad.
- High CPU Skew% with significant Delta CPU.
- High spool usage.

Note:

You can click on and sort the display by a parameter instead of by session number.

3. Select a session row that shows CPU characteristics indicating a bad query to display the **Details View**, which provides information about the query, including the user and account that submitted the query.
4. Select the **SQL** tab to display the SQL for the query. Examine the SQL for errors and poor construction.
5. Select the **Explain** tab to display an abbreviated version of the step statistics and explain text (generated from a full EXPLAIN request), along with a listing of the active steps, completed steps, and steps yet to run.
6. Check for the following in the Explain:
 - The steps required to execute the query
 - The type of operation being performed, and whether it includes a product join
Often the query is stuck on the step where the product join occurs.
7. You can copy the query from the **SQL** tab, run it from Teradata Studio, and request a full EXPLAIN to get more details and validate the source of the problem.

Controlling Poorly Performing Queries

To control a query:

1. In the Query Monitor portlet, click on the session with the identified bad query.
The **Details** view appears.
2. You can use the **Tools** menu to:
 - Change the query priority
 - Abort the query

- Abort and log off the session

Note:

Use the following guidelines when considering whether to abort a session:

- Do not abort the session if it is still running other queries.
- If the session is not running other queries:
 - If the query is a SELECT statement, you can safely abort the query.
 - If the query includes an INSERT, UPDATE, DELETE, or other statement that changes data, aborting a partially completed query may cause data problems.

Finding and Fixing Skewed Tables

The DBC.TableSizeV contains information that can help identify skewed tables. Table skew results from an inappropriate primary index. A query that references a skewed table may try to process more rows on some AMPs than others, and may run out of space:

- SELECT statements may run out of spool space if the accessed object has a defined spool space limit.
- INSERT and UPDATE statements may run out of perm space.

1. Use the following SQL statement to find skewed tables in the DBC.TableSizeV:

```
SELECT vproc AS
"AMP", TableName (FORMAT 'X(20)'), CurrentPerm
FROM DBC.TableSizeV
WHERE DatabaseName = 'database'
ORDER BY TableName, "AMP" ;
```

Result:

AMP	TableName	CurrentPerm
---	-----	-----
0	employee_upi_onempid	18,944
1	employee_upi_onempid	18,944
2	employee_upi_onempid	18,944
3	employee_upi_onempid	19,968
0	employee_nupi_onddept	4,096
1	employee_nupi_onddept	30,208
2	employee_nupi_onddept	15,360
3	employee_nupi_onddept	12,288

In this example, the answer set displays space allocations by AMP for two tables. The results show that:

- CurrentPerm is similar across all vprocs for employee_upi_onempid. Permanent space distribution is relatively even across all AMPs in the system.
- The table Employee_nupi_ondept is poorly distributed. The CurrentPerm figures range from 4,096 bytes to 30,208 bytes on different AMPs.

2. Redefine the primary index for any skewed tables that you find.

See [Choosing a Primary Index](#).

Finding and Resolving Lock Contentions

Note:

The command-line Lock Logger utility (dumplocklog) and related DBS Control settings are deprecated, but remain available for compatibility with prior releases. Database locks should be logged using the Database Query Log (DBQL). This lock information can be accessed by means of the Teradata Viewpoint Lock Viewer portlet, or by querying the DBC.QrylockLogXMLV view.

- For more information on DBQL, see *Teradata Vantage™ - Database Administration*, B035-1093 and the BEGIN QUERY LOGGING statement in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.
- For more information on the Lock Viewer portlet, see *Teradata® Viewpoint User Guide*, B035-2206.

Lock Contentions

Analytics Database automatically locks database objects accessed by each transaction, to prevent or limit simultaneous access by other transactions. Analytics Database requests a lock on the data before a transaction begins and releases the lock when the transaction is complete.

Lock Level	Description
Read	Prevents write and exclusive locks. Generally caused by a SELECT. Ensures consistency during read operations. Several users may hold concurrent read locks on the same data, during which no modification of the data is permitted.
Write	Prevents other read, write, and exclusive locks. Generally caused by an INSERT, UPDATE, or DELETE statement. Enables users to modify data while locking out all other users except those using access locks. Until a write lock releases, no new read or write locks are allowed.
Exclusive	Prevents any other type of concurrent access. Generally caused by statements that make structural changes, for example, using an ALTER TABLE statement to add, drop, or change a column, when the statement uses the LOCKING modifier. Exclusive locks are the most restrictive type of lock. When you hold an exclusive lock, other users cannot read or write to the data.

Lock Level	Description
Access	<p>Prevents exclusive locks only. Caused by the LOCKING ... FOR ACCESS or LOCKING ... FOR LOAD COMMITTED locking modifier.</p> <p>When you use the LOCKING FOR ACCESS locking modifier you can read data while changes are in process. However, if you use this modifier, concurrent requests may access data that has not been committed, causing inconsistent results. Use this modifier for decision support on large tables that are updated only by small, single-row changes.</p> <p>When you use the LOCKING ... FOR LOAD COMMITTED locking modifier on load-isolated tables, you can read committed rows while concurrent changes are in process. Unlike LOCKING FOR ACCESS, this modifier returns only committed data. For more information on Load Isolation, see Reading Committed Data While Loading to the Same Table.</p>

Investigating Query Blocks and Delays

Most blocks are momentary and do not require attention. However, if a block persists, you can investigate it to identify the cause and determine whether further action is required.

1. Use the Teradata Viewpoint Query Monitor portlet to monitor the parameters related to query blocks and delays.

State	Description
Blocked	Indicates that a session query is held up by a lock on an object that the query is attempting to access.
Blocked Time	<p>How long the query has been blocked.</p> <p>Momentary locks are normal and not harmful. Persistent locks may indicate:</p> <ul style="list-style-type: none"> • A runaway query or hung query that cannot complete and release its locks, and which should be aborted • A long-running query, such as large-scale update or backup operation, which should be rescheduled to avoid resource contention
Delayed	Indicates that the query is in a delay queue caused by a workload rule.

2. Click the session ID for the query to access the **Details View**.
 - If a query is blocked, see the **Blocked By** tab for details
 - If the query is delayed, the **Delay** tab provides details about the cause of the delay
3. You can also use the Lock Viewer portlet to find out more about the block, including:
 - Block Time
 - Database
 - Table
 - Delay
 - Blocked User
 - Blocking Level

- Blocking User

Improving Query Performance Using COLLECT STATISTICS: Application DBAs

This section describes how to generate the most efficient query plans by using automated or manual statistics to ensure accurate data demographics.

Statistics Collection

Collecting statistics provides the Optimizer with the data demographics it needs to generate good query plans. The more accurate and up-to-date the statistics, the better the Optimizer can decide on plans and choose the fastest way to answer a query. The computed results are stored in the Data Dictionary DBC.StatsTbl for use during the optimizing phase of statement parsing. The Optimizer does not accept pragmas or hints, and bases query plans only on statistics.

Required Privileges

You must have STATISTICS privileges on a table (except a volatile table) to run a COLLECT STATISTICS request.

Resource Considerations

Collecting statistics normally involves a full-table scan, so you may need to collect statistics for large tables during off-hours.

For very large tables, you can also collect statistics on a sampled subset of table rows instead of all the rows for a table, using the SAMPLE clause. For more information, see [Sampling Statistics with the USING SAMPLE Option](#).

Automated Statistics Management

You have the choice of managing statistics collection manually or allowing Vantage to do it for you. To configure the Teradata system to automatically manage the statistics for one or more databases, the following steps are recommended:

1. Use the Teradata Viewpoint Stats Manager portlet to identify the databases and tables whose statistics should be automated. Alternatively, directly call the Automate-related APIs available in the TDStats database.
2. Use the Teradata Viewpoint Stats Manager portlet to define jobs that collect the statistics identified in step 1. Consider defining Analyze jobs to identify additional statistics beneficial to query optimization. Alternatively, directly call Collect and Analyze-related APIs in the TDSTATS database.
3. If you defined Analyze jobs in step 2, enable the STATSUSAGE logging option via the BEGIN QUERY LOGGING statement for users or applications that access the objects identified in step 1.

4. Schedule the Collect and Analyze jobs defined in step 2 using either the Teradata Viewpoint Stats Manager portlet or a different scheduling tool (for example, Cron).

For more information, see *Teradata Vantage™ - Application Programming Reference*, B035-1090, Viewpoint documentation, and the Teradata Orange Book for *Automated Statistics Management*, 541-0009628.

When To Collect Statistics

Collect statistics at the following phases of table development:

- Newly created, unpopulated tables.
Collect statistics on unpopulated tables to set up the interval histogram used in internal processing. This initial collection makes subsequent statistics collections faster. Make sure to recollect statistics after data is added.
- Prototype phase, newly populated tables.
- Production phase, after a significant percentage of change to the table or partition (~10% rows). For high volumes of very nonunique values, such as dates or timestamps, it may be advantageous to recollect at 7%.
Recommendation: Collect production phase statistics after you have created users and applied real world query loads to the database (up to about 3 months of querying).
- Collect statistics in the first few weeks after an upgrade or migration during periods of low CPU utilization.

Collecting Statistics

Use the COLLECT STATISTICS statement to define the table columns on which you want to collect statistics, for example:

```
COLLECT STATISTICS ON database_name.table_name COLUMN (column_1, column_2 ...);
```

At minimum, you should collect statistics on the primary index (PI) of each table. For small tables (<100 rows per AMP) PI statistics may be sufficient.

Recommendations for Large Tables

For best results on large tables, collect SUMMARY statistics only on non-partitioned tables. Other recommendations for large tables include collecting statistics on:

- All indexed columns
- All frequently joined columns
- Columns frequently referenced in WHERE clauses, particularly if the column contains very skewed, nonunique data

- The system-derived column PARTITION (for all row and column-partitioned tables)

Related Information

For more information on syntax and options for the COLLECT STATISTICS statement, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Stale Statistics

Although the system has some ability to extrapolate reasonable estimates for rows added since statistics were last collected, statistics that are far out of date (stale) may be worse for Optimizer query planning than having no statistics at all.

Recommendation: Use the THRESHOLD option for COLLECT STATISTICS so the system automatically determines if statistics are stale and need to be recollected or if recollection can be skipped. See [Skipping Unneeded Statistics Recollection with the THRESHOLD Option](#).

Note:

If the current statistics are stale, but you do not want to recollect them because of resource contention, you can temporarily drop the statistics using the DROP STATISTICS command, and use the default statistics automatically collected by the system. However, you should recollect new statistics at the earliest opportunity.

SUMMARY Statistics

Teradata recommends collecting SUMMARY statistics for both partitioned and non-partitioned tables. This is a fast way to collect the current total number of rows in a table. Collecting SUMMARY statistics is useful if collecting full statistics is too expensive. Collecting SUMMARY statistics also allows more accurate extrapolation from previously collected full statistics, allowing resource-intensive full statistics collection to be delayed longer.

Skipping Unneeded Statistics Recollection with the THRESHOLD Option

To skip unneeded statistics recollections, specify the THRESHOLD option of the COLLECT STATISTICS request. With this option, if the amount of data that changed since the last statistics collection is below a specified threshold or the statistics are newer than a specified age, Teradata does not recollect statistics. You have two options for creating a threshold:

- Allow Vantage to determine the appropriate threshold
- Specify a change percentage following the THRESHOLD keyword and/or the number of days

You can issue COLLECT STATISTICS requests at regular intervals without being concerned that you might be wasting I/O and CPU by collecting statistics too soon. Teradata skips the recollections for the statistics

whose thresholds are not met and recollects the statistics for those columns that are over the threshold. For example, if the threshold is 10% data change and you request a recollection, Teradata skips recollection if less than 10% of the data changed since the last collection.

If you specify the THRESHOLD option the first time you collect statistics, Teradata collects the statistics and remembers the THRESHOLD option. When you recollect statistics, Teradata uses the saved threshold to determine whether statistics recollection is needed. To tell Teradata to ignore the THRESHOLD option (without resetting it) the next time it collects statistics, use the FOR CURRENT option after the percent specification.

To allow Vantage to determine the appropriate threshold, in the DBS Control utility disable both the DefaultTimeThreshold and DefaultUserChangeThreshold fields and set the SysChangeThresholdOption field to 0, 1, or 2. By default, DefaultTimeThreshold and DefaultUserChangeThreshold are disabled and SysChangeThresholdOption is 0. See *Teradata Vantage™ - Database Utilities*, B035-1102 for information on the different options for these fields.

You can query the LastCollectTimeStamp column of the Data Dictionary table DBC.StatsTbl to see whether data is collected or not.

The user-specified change threshold percent is not supported for statistics on views and queries.

For the syntax of the COLLECT STATISTICS statement, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Sampling Statistics with the USING SAMPLE Option

If you find that recollection on columns with evenly distributed data (that is, data with little or no data skew) is too costly in terms of performance, try using sampled statistics instead.

To use sample statistics, specify the USING SAMPLE option of the COLLECT STATISTICS request. You can collect sample statistics with a system-determined sample or you can specify a sample percentage. If you specify the sample percentage, Teradata reads the specified percent of rows to collect statistics.

If Teradata determines the sample percent, the system determines when to switch to sampling and the appropriate degree of sampling that can be used without sacrificing quality. The downgrade approach works like this:

1. The first time you request a sample statistics collection, Teradata collects full statistics. This gives Teradata information it needs to determine when to downgrade to a smaller percentage.
2. On subsequent requests to recollect statistics, Teradata collects full statistics until it has collected enough statistics history.
3. Using the statistics history, Teradata evaluates the nature of the column (for example, skewed, rolling, or static). Teradata considers the column usage (detailed or summary data), which it maintains in DBC.StatsTbl.UsageType field, and the user-specified number of intervals to decide when to downgrade to sample statistics. The system is more aggressive in downgrading to sample statistics for the histograms whose summary data is used.
4. Teradata determines a sample percentage (2% to 100%) for recollecting statistics and decides which formula to use for scaling based on the history and nature of the column.

5. To ensure quality, Teradata compares the recollected statistics to the history. The system lowers the sampling percentage only if a smaller sample provides quality results.

For example, Teradata never switches to sampled statistics for small tables, skewed columns, a column that is member of the partitioning expression, or columns that require details.

After few sample statistics collections, Teradata may collect full statistics to verify and adjust the sample percentage and the extrapolation method.

Sample statistics are not supported on views and queries.

To determine the sampling percentage in use, you can query the Data Dictionary table column DBC.StatsTbl.SampleSizePct.

For more information, also see “Sampled Statistics” in *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142.

Example: User-Specified Sample Percentage

The following requests sample statistics on the o_orderkey with user-specified sample percentage. Statistics are collected by reading 10% of the table rows:

```
COLLECT STATISTICS
    USING SAMPLE 10 PERCENT
    COLUMN o_orderkey
    ON orders;
```

Example: System-Determined Sample Statistics

The following requests system-determined sample statistics on the o_orderdatetime column. The default sample percentage in DBS Control determines the number of rows sampled. Teradata collects full statistics and remembers the sample option. Teradata considers downgrading the sample percentage after the system captures enough history and the column is identified as eligible for sample statistics:

```
COLLECT STATISTICS
    USING SYSTEM SAMPLE
    COLUMN o_orderdatetime
    ON orders;
```

Related Information

For guidelines and suggested tools for collecting statistics, see the following:

- “COLLECT STATISTICS (Optimizer Form)” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144
- *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142:

- “Query Capture Facility”
- Submit the SHOW STATISTICS statement with either the SUMMARY option to see summary statistics or the VALUES option to see detailed statistics. For more information, see “SHOW STATISTICS (Optimizer Form)” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.
- Teradata Viewpoint Stats Manager portlet described in Viewpoint documentation.

Working with Special Cases

Collecting Statistics on Tables Protected with Row-Level Security

If a table is protected by row-level security (at least one column is defined with a constraint-column-definition clause), you must have additional privileges, as follows:

To use COLLECT STATISTICS, HELP STATISTICS, and SHOW STATISTICS with the VALUES clause on...	You must have the OVERRIDE SELECT CONSTRAINT privilege on...
a base table	the base table or database.
a join index	the underlying tables or database.
a hash index	the underlying tables or database.
a view	the view or database.
a query	the underlying tables or database.

For copying statistics, in addition to the SELECT privilege, you must have the OVERRIDE SELECT CONSTRAINT privilege if the source table is protected by row-level security.

Collecting Statistics on Null

When you collect statistics, Vantage records three different null counts: NumNulls, PNullUniqueValueCount, and AllNulls.

- NumNulls is the number of rows that have a null in at least one field. This includes rows with *all* fields null.
- PNullUniqueValueCount is the number of unique values in rows with partial nulls.
- All-Nulls is the number of rows that have *all* fields null. For example, in the following table, NumNulls count is 4, All-Nulls count is 2, and PNullUniqueValueCount is 2:

x	y
-----	-----
1	a

?	b
?	?
4	?
?	?

Use the SHOW STATISTICS statement to report the number of nulls after collecting statistics. By considering NumNulls, All-Nulls, and PNullUniqueValueCount counts, Teradata can better estimate the number of unique values in the presence of nulls.

Collecting Statistics on the Data Dictionary

For information on collecting statistics on the Data Dictionary, see [Collecting Statistics on Data Dictionary Tables](#).

Tracking Query Behavior with Database Query Logging: Operational DBAs

This section describes how to track query behavior by using Database Query Logging (DBQL). It also describes how to maintain DBQL logs, archive DBQL data, and shred the lock plan information in the XML Lock Log table, DBQLXMLLockTbl. For information about DBQL macros see [DBQL Macros](#). For a list of DBQL tables, along with some brief information about them, see [DBQL Tables](#). For a list of DBQL views, along with some brief information about them, see [DBQL Views](#). For greater details about the DBQL views, see "Views Reference" in *Teradata Vantage™ - Data Dictionary*, B035-1092.

DBQL Overview

The predefined DBQL logs are created as relational tables in DBC during normal Vantage installation.

For SQL Engine 17.00 and later, the default mode for DBQL logging is set to Algorithm 3. For upgrades to Release 17.00, the previous setting is retained, although Teradata recommends switching to Algorithm 3 if that was not the previous setting for your environment. To switch to Algorithm 3, change DBS Control General field 64 DBQL CPU/IO Collection, see [Setting DBQL Logging Algorithm](#).

To keep DBQL logging performing optimally, the best practice is to set up a recurring job to offload data to PDCR. See: [Maintaining the Logs](#).

AMPs use a DBQL cache called DBQL Performance Stats (DPS) cache. The DPS cache size is controlled by the DBQL_AWTDPS_CacheMaximum field 37 in the DBS Control Performance section. The default cache size is 8192. If the size needs to be adjusted, see [Changing the DBQL Performance Stats Cache Size](#).

If you choose not to use DBQL logging, the tables remain empty.

DBQL Uses

DBQL can be used to:

- Capture query/statement counts and response times. You can look in the DBQL logs to see if this is the first time this query has been broken and what the historical pattern of its execution has been.
- Validate that nonunique secondary indexes (NUSIs) are actually used. You can free up space and improve table update performance by deleting NUSIs.
- Determine which aggregate join indexes (AJIs) are used the most. This allows you to make a cost-benefit analysis on their AJIs, comparing usage against size and time to build.
- Analyze SQL text and process steps. View the SQL text that inadvertently wiped out the system using DBQL historical data and then attempt to rebuild the lost data.
- Log optimizer queries as XML documents.
- Make further refinements to workload groups and scheduling.
- Discover potential application improvements.
- Understand basic workload trends, such as growth patterns and workload mix, and identify outliers.

- Determine whether the query has redrive protection and if the request was redriven after a database failure .

DBQL provides a series of predefined tables and views that store historical records of queries and their duration, performance, and target activity based on rules you specify.

DBQL is flexible enough to log information on the variety of SQL requests, from short transactions to longer-running analysis and mining queries that run on DBQL can log information for an application name, a user or group of users, or a list of accounts.

How to Use DBQL

To use DBQL:

1. Grant the proper privileges to your administrative user for query logging. See [Granting DBQL Administrative Privileges to Other Users](#).
2. Determine what type of information you want to collect. This section lists the descriptions of each field in each DBQL log. Use the descriptions to get an idea of what type of information you want to collect and then look at the options you can employ to control the type of details and the amount of data logged. The options are discussed in [WITH Logging Options](#) and [LIMIT Logging Options](#).
3. Determine what you want to collect data for. Create DBQL rules to log queries of a user, group of users, an account, a list of accounts, or an application.

Teradata recommends creating rules for accounts so that queries will be logged for any user who logs in under that account.

4. Run the Database Initialization Program (DIP) script DIPPCR to create archival tables for longer-term storage of DBQL and other performance data. DIPPCR also creates the data extraction macros that populate these tables.
5. Run the Viewpoint portlet Performance Data Collection to set up a recurring daily DBQL job. This job moves data from the DBQL log tables in DBC to the archival tables in the PDCRDATA database.
6. Use the BEGIN QUERY LOGGING statement to create logging rules or use the REPLACE QUERY LOGGING statement to replace existing logging rules with new logging rules. Teradata recommends using scripts to enable and disable logging to avoid typographical errors.

Note:

You must have the EXECUTE privilege on the DBC.DBQLAccessMacro macro to enable or disable DBQL logging. That is, you must have this privilege to submit the BEGIN QUERY LOGGING, REPLACE QUERY LOGGING, and END QUERY LOGGING statements.

As users establish sessions and submit queries, DBQL will automatically enable rules according to the Best Fit Hierarchy. (See [Best Fit Rule Hierarchy](#).)

DBQL collects query information into a data cache based on rules you specify, flushes the cache periodically (see [Options for Flushing the DBQL Cache](#)), and writes the information to the DBQL dictionary tables. This information includes historical records of queries and their duration, performance data, and target activity.

7. Upon examining the query log data, you may decide you need to adjust rules. You can use the `REPLACE QUERY LOGGING` statement to replace or delete existing rules using the `END QUERY LOGGING` statement.

Overview of Collection Options

DBQL is flexible enough to log information on the variety of SQL requests that run on Vantage, from short transactions to longer-running analysis and mining queries.

Options enable you to control the volume and detail of the logged data. You can define rules, for example, that log the first 5,000 characters of any query that runs during a session invoked by a specific user under a specific account. Or you could define a rule to log queries that take more time to complete than the specified time threshold. You could even log queries that exceed a specific CPU time specified in hundredths of a second. For example, specify `"THRESHOLD=500 CPUTime"` for queries that take longer than five seconds of AMP CPU time to complete.

The [WITH Logging Options](#) controls what details the system logs. The [LIMIT Logging Options](#) controls how much of information the system logs. Collection options include:

- Default logging reports for each query with at least the leading SQL characters, the time of receipt, the number of processing steps completed, the time the first step was dispatched, the times the packets were returned to the host, and the number of rows updated, inserted, and deleted.
- Summary logging reports the count of all queries that completed processing time within specified time intervals, I/O criteria, or CPU usage criteria.
- Threshold logging can log a combination of default and summary data:
 - Default data for each query that ran beyond the threshold limit
 - Summary counts of all queries that ran within the threshold time

Use elapsed time, I/O counts, or CPU time as a criterion for threshold logging of details or summary
- Detail logging, which includes:
 - Default data
 - Step level activity, including parallel steps
 - Object usage per query
 - Full SQL text
 - Explain text
 - XML query plan
- Preventing logging for a specific user, user/account pair, user/account list, or application name.

Logging Rules for Applications

Enable or disable logging for a utility by specifying a rule for the application name. The system logs information for any user that logs on under the application name. For example, to log all queries that result from a FastLoad job, submit:


```
BEGIN QUERY LOGGING ON APPLNAME= 'FASTLOAD';
```

DBQL first searches for a rule for an application name before searching for rules for the user and account.

Creating logging rules by application name allows control over certain utilities, such as load and unload utilities, regardless of what user/account that the application runs under. For example, if you run utilities under the same users and accounts that are used for interactive queries, creating logging rules for utilities allows you to specify minimal logging for utilities and more extensive logging for interactive users.

The application names you specify with the APPLNAME option are the names the system passes in the reserved query band UtilityName.

The following table lists the UtilityName strings.

Client/DBS Protocol	Utility Type	Utility Name Value
FastExport	Standalone FastExport	FASTEXP
	Teradata Parallel Transporter EXPORT operator	TPTEXP
	JDBC FastExport	JDBCE
	.NET FastExport	DOTNETE
FastLoad	Standalone FastLoad	FASTLOAD
	Teradata Parallel Transporter LOAD operator	TPTLOAD
	JDBC FastLoad	JDBCL
	.NET FastLoad	DOTNETL
	Crashdumps Save Program (CSP) Save Dump	CSPLOAD
MultiLoad	Standalone MultiLoad	MULTILOAD
	Teradata Parallel Transporter UPDATE operator	TPTUPD
	JDBC MultiLoad	JDBCM
	.NET MultiLoad	DOTNETM

Best Fit Rule Hierarchy

You may enable many different rules for various users, user accounts, or applications. DBQL checks for rules to apply in the following order:

1. All users under a specific application name
2. Specific user, specific account
3. Specific user, all accounts
4. All users, specific accounts
5. All users, all accounts

To create exceptions to the rules, use the WITH NONE option. You can create a rule to exclude certain sessions from a broader rule by using the WITH NONE option. For example, if you want to create a rule that enables logging for all users logging on under a specific application except for user1, you could create a rule to log all users for that application and create another rule using the WITH NONE option for user1 to disable logging just for user1. (See the example under [Scenarios of Logging Accounts](#).)

As each user logs on, DBQL first searches the rule cache for a rule, and if no rule is found, it searches the DBQLRuleTbl table. The searches are based on the order of this best fit rule hierarchy.

Rules Validation

The following table lists the rules DBQL validates when DBQL logging begins. DBQL checks against the application name, user name, and account string to validate rules.

IF a...	THEN DBQL ...
user logs on to start a session	checks the username, account string, application name, and zone ID (in secure-zoned systems) in the rules cache and also looks for this in the dictionary table.
match is found in the rules cache	logs according to any options in the rules table.
match is not found in the rules cache	searches for a matching user or account in the Data Dictionary.
match is found in the Data Dictionary	creates an entry in the rules cache and logs according to any options in the rules table.
match is not found in the Data Dictionary	creates a “do not log” rule in the rules cache but does not perform logging.
“WITH NONE” rule is found	creates a “do not log” rule in the rules cache but does not perform logging.

Note:

If the account string of a session is changed via the SET SESSION ACCOUNT statement, the system ensures that any rules associated with the user and account string applies when the next query is issued from that session.

SQL Statements That Should Be Captured

The following requirements should determine which SQL statements ought to be captured:

- To the greatest extent possible, SQL text and the associated tables and views referenced should be captured and logged to facilitate performance tuning, access modeling, physical modeling modification considerations, and so on.

When enabled, DBQL captures all SQL statement types and stores them into an existing 20-character field named StatementType in DBQLLogTbl.

In addition, usage of databases, tables, columns, indices, macros, views, triggers, stored procedure, and User-Defined Functions (UDFs) are logged in DBQLObjTbl. By querying DBQLObjTbl information, DBAs are able to see which views and macros users access. This enables DBAs to delete unused objects.

- It is assumed that user IDs and their associated account strings will adhere to the conventions defined in [Logging Resource Usage Data with Account String Variables](#).
- Step, explain, and XML plan logging can also be logged when needed for more detailed analysis, like performance tuning or troubleshooting.

SQL Logging Statements

When creating the necessary rules to enable SQL logging using DBQL, a rule correlates exactly to a specific SQL statement. For example, the following SQL statements show a BEGIN QUERY LOGGING statement along with its corresponding END QUERY LOGGING statement.

```
BEGIN QUERY LOGGING WITH SQL ON ALL ACCOUNT='$M00MSIR&D&H' ;
END QUERY LOGGING WITH SQL ON ALL ACCOUNT='$M00MSIR&D&H' ;
```

The implication is that the SQL used to create the various rules should be retained so as to easily facilitate removing a rule.

The preceding account string format is considered a best practice. For more information, see [Logging Resource Usage Data with Account String Variables](#).

All rules can be easily removed by the following command:

```
END QUERY LOGGING ON ALL RULES;
```

Note:

Support will be removed for the WITH option of the END QUERY LOGGING statement. Use of the option results in the following message: “Deprecated option WITH is used and will soon be removed. Do not continue to use WITH option, refer to END QUERY LOGGING statement syntax.”

SQL Logging Considerations

The BEGIN QUERY LOGGING and REPLACE QUERY LOGGING statements give the administrator flexibility in determining what SQL requests are to be logged based on the user ID and/or account string. It also determines what level of detail is captured for a selected request. DBQL does not, however, allow for rules to be selective based on request type or object accessed. In other words, DBQL logs all request types and target database objects. Therefore, the user ID and account string serve as the primary selection criteria.

Note:

If a Teradata dynamic workload management software exception occurs, the Teradata dynamic workload management software causes rows to be written to the DBQLSqlTbl.

For information on this type of an exception, see *Teradata Vantage™ - Application Programming Reference*, B035-1090 or use Teradata Viewpoint.

XML Query Plan Logging

The WITH XMLPLAN option of the BEGIN QUERY LOGGING statement provides more detailed Optimizer query plan information than the INSERT EXPLAIN statement modifier. That plan information, structured and stored as an XML document, can be easily parsed and analyzed by those reading the log.

The XMLPLAN option does not log query plans for EXPLAIN request modifiers, DDL requests, or for the INSERT EXPLAIN and DUMP EXPLAIN statements.

Query plan logging has benefits:

- It does not require users to create a separate database to capture query plans.
- It allows users to capture the plans for a set of queries easily via a single BEGIN QUERY LOGGING or REPLACE QUERY LOGGING statement rather than having to issue a separate INSERT EXPLAIN or DUMP EXPLAIN statement for each one.
- It provides more plan details than the INSERT EXPLAIN modifier.

Capturing DBQL Through the Account String

Teradata recommends that you specify which requests will be captured through DBQL via the account string. The recommendation is to have 10 to 40 different account strings based on various workloads on the system. The recommendation is to characterize the difference between work groups using 4 characters (for example, ABCD). The 4 characters should identify each work group based on workload characteristics, such as business priority and by reporting. Differentiating work groups in the account string allows you to report usage, such as CPU consumption by account string. The recommended account string format for all usage is '\$M00ABCD&D&H'.

The advantage to logging DBQL by account string is that it keeps the logging rules to a minimum and alleviates the need to log a DBQL rule every time a user ID is created.

Note:

You cannot log use count information for account strings.

SQL Logging by Workload Type

SQL logging is based on the type of work being performed. In general, all Teradata Active EDW workloads can be broadly grouped into three categories as follows:

- Short subsecond known work (Tactical Work)

This workload can be identified by work typically done by TPUMP and single-AMP, subsecond tactical requests only.

- Mostly long running work

This workload can be identified by work typically done by Teradata Studio, Report Applications, MultiLoads, FastLoads, and BTEQ. It is typical mixed workload usage.

- Mostly short subsecond requests (with occasional long running or unknown work)

This workload can be identified by usage that is typically tactical (millions of short, subsecond queries), but there is an unknown factor, like unauthorized usage of the user ID, or a longer running component, that should be logged in a more detailed manner as to allow tuning or troubleshooting.

SQL Statements to Control Logging

Start and stop logging with the Teradata SQL statements `BEGIN QUERY LOGGING`, `REPLACE QUERY LOGGING`, and `END QUERY LOGGING`. Only a user with `EXECUTE` privilege on `DBC.DBQLAccessMacro` can execute the statements.

Note:

You cannot issue a `BEGIN/REPLACE/END QUERY LOGGING` statement while running in ANSI session mode. In Teradata session mode, you cannot issue a `BEGIN/REPLACE/END QUERY LOGGING` statement within a BT/ET transaction. Instead, use these statements outside a BT/ET transaction or, for ANSI session mode, log off and set the transaction mode to Teradata session mode.

BEGIN/REPLACE/END QUERY LOGGING Statements

The purpose of each of the Teradata SQL DDL statements used to enable and disable DBQL logging is described in the following table.

Statement	Purpose
<code>BEGIN QUERY LOGGING</code>	<p>When submitted by a user with <code>EXECUTE</code> privilege on <code>DBQLAccessMacro</code>, this statement enables logging for the named users, accounts, and applications triplet.</p> <p>Log only a maximum of:</p> <ul style="list-style-type: none"> • 100 users • One user with 100 accounts • 100 applications <p>per statement. If logging is to be enabled for more than 100 users, use blocks of <code>BEGIN QUERY LOGGING</code> statements with 100 users each. For active sessions, logging begins when Vantage receives the next query.</p> <p>You can also enable a rule to make sure no logging takes place for a user, account, or application name.</p> <p>For more information, see Logging Scenarios.</p>

Statement	Purpose
REPLACE QUERY LOGGING	When submitted by a user with EXECUTE privilege on DBQLAccessMacro, this statement enables logging and deletes the existing rule in the DBC.DBQLRuleTbl table if one matches the users, accounts, applications triplet, and then creates a new rule. If no rule exists, the new rule is added to the DBC.DBQLRuleTbl table.
END QUERY LOGGING	<p>When submitted by a user with EXECUTE privilege on DBQLAccessMacro, this statement stops logging for the named users, accounts, or applications. Each END QUERY LOGGING statement should list fewer than 100 users/ 100 accounts for one user/ 100 applications. DBQL calls a routine that commits the data and flushes all the DBQL caches. Each time you issue an END QUERY LOGGING statement, the system will flush all the rules cache and will flush the DBQL data tables.</p> <p>You can also flush the DBQL cache using other methods. See Options for Flushing the DBQL Cache.</p> <p>For more information on END QUERY LOGGING, see Effects of Dynamically Enabling /Replacing/Disabling Logging on Current Rules.</p>

Every BEGIN QUERY LOGGING statement you enable must be disabled using the proper END QUERY LOGGING statement syntax. For example, if you submit:

```
BEGIN QUERY LOGGING WITH SQL ON user1 ACCOUNT='Marketing';
```

you must use the following syntax to end logging for that user:

```
END QUERY LOGGING ON user1 ACCOUNT='Marketing';
```

Note:

If you list more than one account, use parentheses. For example, ACCOUNT=('Marketing', 'Finance').

For statement syntax, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144. To enable other users to submit these statements, see [Granting DBQL Administrative Privileges to Other Users](#).

WITH Logging Options

The WITH logging option that can be used with the BEGIN/REPLACE QUERY LOGGING statements enables logging into the following appropriate tables.

The DBQL option...	Logs to...
WITH ALL	<p>the following tables:</p> <ul style="list-style-type: none"> • DBC.DBQLLogTbl • DBC.DBQLExplainTbl • DBC.DBQLObjTbl • DBC.DBQLStepTbl • DBC.DBQLSQLTbl

The DBQL option...	Logs to...
WITH EXPLAIN	<ul style="list-style-type: none"> • DBC.DBQLLogTbl • DBC.DBQLExplainTbl
WITH FEATUREINFO	DBC.DBQLLogTbl
WITH LOCK= <i>n</i>	DBC.DBQLXMLLockTbl
WITH [NO COLUMNS] OBJECTS	<ul style="list-style-type: none"> • DBC.DBQLLogTbl • DBC.DBQLObjTbl
WITH NONE	nothing. This option disables logging on the specified user or account, list of users or accounts, or application.
WITH PARAMINFO	DBC.DBQLParamTbl
WITH SQL	<ul style="list-style-type: none"> • DBC.DBQLLogTbl • DBC.DBQLSQLTbl
WITH [DETAILED] STATSUSAGE	DBC.DBQLXMLTbl
WITH STEPINFO	<ul style="list-style-type: none"> • DBC.DBQLLogTbl • DBC.DBQLStepTbl
WITH USECOUNT	DBC.ObjectUsage
WITH UTILITYINFO	DBC.DBQLUtilityTbl
WITH [VERBOSE] XMLPLAN	<ul style="list-style-type: none"> • DBC.DBQLLogTbl • DBC.DBQLXMLTbl

Note:

The WITH ALL option enables logging into all of the DBQL tables with the exception of DBQLXMLTbl and DBQLXMLLOCKTbl. The WITH NONE disables logging on the specified user/account, list of users, account list, or application.

The following table describes the different WITH options available in more detail.

Parameter	Logging Behavior
WITH ALL	<p>Use the WITH ALL option sparingly and only for selected users because it can consume excessive CPU resources and grow the logs (which consume DBC PERM space) very quickly.</p> <p>WITH ALL logs the information generated by these WITH rules: EXPLAIN, OBJECTS, SQL, and STEPINFO. It does not log the information generated by XMLPLAN, LOCK, FEATUREINFO, STATSUSAGE, PARAMINFO, and UTILITYINFO. The WITH ALL option generates:</p> <ul style="list-style-type: none"> • One default row per query in DBQLLogTbl that includes the first 200 characters of the SQL statement, unless you define LIMIT SQLTEXT=0. • One row per target object per query in DBQLObjTbl.

Parameter	Logging Behavior
	<ul style="list-style-type: none"> • One row per step per query in DBQLStepTbl. • One or more rows per SQL statement in the DBQLSQLTbl. • One or more rows per query in DBQLExplainTbl.
WITH EXPLAIN	<p>This option generates the EXPLAIN for the query and logs it into the DBQLExplainTbl. If the additional EXPLAIN text is greater than 31,000 characters, multiple rows are generated and the ExpRowNo field indicates the row number.</p> <p>This option inserts a default row into DBQLLogTbl.</p> <p>WITH EXPLAIN generates and logs the unformatted EXPLAIN text for each request. It does not generate EXPLAIN text for requests preceded by the EXPLAIN request modifier, nor does it log rows for cached requests.</p> <p>Note: You cannot specify the WITH EXPLAIN option with the SUMMARY or THRESHOLD options.</p>
WITH FEATUREINFO	<p>This option captures which features an SQL request has used, and logs that information into the FeatureUsage column of DBQLLogTbl. The option captures only use of features visible on the parsing engine (PE). If a feature is visible on the AMPs only, this option does not capture information about that feature. This option does not track third-party application use of database features.</p>
WITH LOCK= <i>n</i>	<p>Logs in XML format in DBQLXMLLOCKTbl any lock contention longer than <i>n</i> centiseconds. The minimum acceptable value for <i>n</i> is 5.</p> <p>You can access the DBQL lock log through the Viewpoint lock viewer portlet or query the system table DBC.DBQLXMLLOCKTbl or the view DBC.QrylockLogXML[V].</p> <p>Note: The DBQL lock logging rule active at the time a session logs on is applied for the duration of the session, even if lock logging is disabled or enabled after the session logs on.</p>
WITH NONE	<p>This option causes no logging to take place for the specified user, user/account pair, user/account list, application name, or list of application names.</p> <p>For example, if there is a rule for default logging on ALL users, you can exclude logging for a specific user <i>user1</i>, by submitting the following statement:</p> <pre>BEGIN QUERY LOGGING WITH NONE ON user1;</pre>
WITH [NO COLUMNS] OBJECTS	<p>This option logs database, table, column, and index information in DBQLObjTbl. This option can be used with SUMMARY or THRESHOLD.</p> <p>This option inserts:</p> <ul style="list-style-type: none"> • A default row in DBQLLogTbl • One row per target object per query in DBQLObjTbl <p>Use this option selectively. Object data is useful for analyzing queries that make heavy use of join indexes and indexed access, but it can generate many rows.</p> <p>By default, the WITH OBJECTS option logs a separate row for each column referenced in a query. The NO COLUMNS suboption turns off column logging. This suboption is especially useful for tables with many columns. Use of NO COLUMNS reduces overhead and controls the size of the DBQLObjTbl table.</p>

Parameter	Logging Behavior
	<p>Note:</p> <p>Any DBC database tables and columns used by the system while processing a query are not reflected in the DBQL object rows for that query. If a statement accesses a DBC table, the DBC table will not appear. This means, for example, that statements like CREATE TABLE or SELECT FROM DBC.xxx do not have objects logged through DBQL because all objects referenced by the plan generated for the statement are DBC tables and columns. However, other objects that are accessed by the statement will be logged in the DBC.DBQLObjTbl.</p> <p>Also note that certain optimized requests may bypass the Dispatcher so that DBQL will not log them.</p> <p>The names of macros, views and triggers will be displayed. The tables and columns mentioned in the macro or view will be those of the base table, not the field names in the macro or view. Objects in DBQLObjTbl are those that the Optimizer actually accesses, not necessarily the objects that appear in an SQL statement.</p>
WITH PARAMINFO	<p>This option logs parameterized request variable names, types, positions, and values into DBQLParamTbl.</p>
WITH SQL	<p>This option logs the full text of all SQL statements in DBQLSqlTbl. It inserts:</p> <ul style="list-style-type: none"> • A default row in DBQLLogTbl. • The entire SQL statement for each request for each user being logged. Large statements can cause multiple rows to be written to log the full query text. <p>Note:</p> <p>This option can be used with THRESHOLD.</p> <p>DBQL is limited to logging information about base tables and logging direct SQL statements. Macros, views and triggers do not result in complete logging information. For example, SQL statements within a trigger are not logged by DBQL. However, the SQL statement that triggers the action will be logged.</p> <p>Note:</p> <p>If you set LIMIT SQLTEXT=0 when you specify the WITH SQL option, you avoid duplicate SQL in the DBQLLogTbl.</p>
WITH [DETAILED] STATSUSAGE	<p>This option logs query optimizer statistics and usage recommendations for DML requests that undergo query optimization. Logged data is stored as an XML document in DBC.DBQLXMLTbl and is accessible from DBC.QryLogXML[V] view. WITH [DETAILED] STATSUSAGE does not log these requests: EXPLAIN, INSERT EXPLAIN, and DUMP EXPLAIN.</p> <p>Enable this option to provide data to automated statistics features, in particular the AnalyzeStatsUsage open API. Teradata recommends that you access the open APIs via the Teradata Viewpoint Stats Manager portlet. For more information, see Automated Statistics Management.</p> <p>If logging option XMLPLAN is also enabled, the data from both options is logged in one document. If only STATSUSAGE is enabled and the optimized statement has no relevant statistics usage data, no document is logged to reduce overhead.</p> <p>Note:</p> <p>This option requires the GRANT EXECUTE privilege on DBC.dbqlaccessmacro. This option is not enabled when you specify the ALL option.</p>

Parameter	Logging Behavior
	<p>The DETAILED suboption logs in XML format summary statistics details from the existing statistics for all objects referenced in the query plan.</p> <p>The DETAILED suboption logs the following statistics details:</p> <ul style="list-style-type: none"> • StatTimeStamp • Version • OrigVersion • NumColumns • NumBValues • NumEHIntervals • NumHistoryRecords • NumNulls • NumAllNulls • NumAMPs • NumPNullDistinctVals • AvgAmpRPV • PNullHighModeFreq • AvgAmpRPV • HighModeFreq • NumDistinctVals • NumRows • CPUUsage • IOUsage <p>DETAILED does not support geospatial statistics.</p>
WITH STEPINFO	<p>This option logs AMP step-level information in DBQLStepTbl. When the query completes, it logs one row for each query step, including parallel steps.</p> <p>The WITH STEPINFO option inserts a default row into DBQLLogTbl.</p> <p>Note: This option can be used with THRESHOLD.</p>
WITH UTILITYINFO	<p>This option logs utility information to DBC.DBQLUtilityTbl for the following utilities:</p> <ul style="list-style-type: none"> • FastLoadProtocol: FastLoad, Teradata Parallel Transport (TPT) Load operator, and JDBC FastLoad. • MLOAD Protocol: MultiLoad and TPT Update operator. • MLOADX Protocol: TPT Update operator. • FastExport Protocol: FastExport, TPT Export operator, and JDBC FastExport. • Data Stream Architecture.
WITH USECOUNT	<p>This option logs use count information for a database or user.</p> <p>If you specify USECOUNT for a user, you can specify any of the other logging options. If you specify USECOUNT for a database, you cannot specify any other logging options.</p> <p>Note: You cannot enable the WITH USECOUNT option on account strings.</p>

Parameter	Logging Behavior
WITH [VERBOSE] XMLPLAN	<p>This option logs the query plan in XML format in DBQLXMLTbl and inserts a default row into DBQLLogTbl.</p> <p>For example, the following statement logs the XML plan for all DML statements performed by User1.</p> <pre>BEGIN QUERY LOGGING WITH XMLPLAN LIMIT SQLTEXT=0 ON User1;</pre> <p>This option does not log query plans for EXPLAIN request modifiers, DDL requests, or for the INSERT EXPLAIN and DUMP EXPLAIN statements.</p> <p>Because the XML plan includes the query and EXPLAIN text, Teradata recommends that when you use the XMLPLAN option, do not also use the WITH EXPLAIN and WITH SQL options. Set SQLTEXT to 0 (SQLTEXT=0) to avoid redundant logging if query and EXPLAIN text is not needed for DDL statements.</p> <p>The XML schema for the log this option produces is maintained at http://schemas.teradata.com/queryplan/queryplan.xsd.</p> <p>Note:</p> <ul style="list-style-type: none"> • WITH XMLPLAN cannot be used with the THRESHOLD and SUMMARY limit options. • The VERBOSE suboption logs the VERBOSE EXPLAIN text of a query in XML format and contains details on SpoolAsgnList and HashFields that are not available with just EXPLAIN.

For more information on the SQL syntax, see “BEGIN QUERY LOGGING” or “REPLACE QUERY LOGGING” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

LIMIT Logging Options

The following options control how much and when the data gets logged into DBQLLogTbl and DBQLSummaryTbl.

LIMIT SQLTEXT=n

This option determines how much SQL text to capture for queries logged to DBQLLogTbl.

Use this option to capture less than or more than the automatic first 200 characters in the default row. To turn off text capture in DBQLLogTbl completely, specify 0. Specify LIMIT SQLTEXT=*n* where *n* is some value greater than 0 to control the amount of SQL text to be recorded in DBC.DBQLLogTbl. For example, to log 500 characters of the SQL, submit the following:

```
BEGIN QUERY LOGGING LIMIT SQLTEXT=500 ON USER2;
```

The maximum is 10,000 characters. If you specify LIMIT SQLTEXT but do not specify a value, up to 10,000 characters are logged in DBQLLogTbl.

To store the complete statement regardless of length, specify the WITH SQL option; the system logs as many rows as needed to contain the full text in DBQLSQLTbl.

If you specify the WITH SQL option and log to the DBQLSQLTbl, you may define LIMIT SQLTEXT=0 to avoid redundant logging in both the default row and DBQLSQLTbl.

LIMIT SUMMARY=*n1*, *n2*, *n3* (units)

Use this option to group queries by duration intervals or number of I/Os, count each group, and store the count results in DBQLSummaryTbl. You can specify up to 3 intervals. A fourth interval is created by default as anything longer than *n3*.

If you specify all three intervals, the intervals are:

- 0-*n1*
- *n1*-*n2*
- *n2*-*n3*
- >*n3*

SUMMARY is useful for tracking many short queries, such as for OLTP applications, because it does not log to DBQLogTbl.

The LIMIT SUMMARY option:

- Flushes summary information at system-controlled intervals of 10 minutes.
- Does not write rows if no data has been collected for a summary logging user/account in a 10-minute interval.
- Cannot be used with LIMIT THRESHOLD.
- Cannot be used with logging options.

You can define the intervals for LIMIT SUMMARY with one of the following:

Unit	Summary Data Count Result
CPUTIME	AMP CPU time, where <i>n1</i> , <i>n2</i> , <i>n3</i> are specified in hundredths of a second.
CPUTIMENORM	Normalized AMP CPU time, where <i>n1</i> , <i>n2</i> , <i>n3</i> are specified in hundredths of a second. Note: Normalized columns, such as CPUTIMENORM, are for co-existence systems only.
ELAPSEDSEC	Elapsed time in seconds. If you do not specify units, ELAPSEDSEC is the default.
ELAPSEDTIME	Elapsed time, where <i>n1</i> , <i>n2</i> , <i>n3</i> are specified in hundredths of a second.
IOCOUNT	Count of total I/Os, where <i>n1</i> , <i>n2</i> , <i>n3</i> are specified in number of I/Os.

If you do not specify units, the summary is based on ELAPSEDSEC.

For example, to group queries based on elapsed time such as 0-1 seconds, 1-2 seconds, 2-3 seconds, and log queries running greater than 3 seconds, submit:

```
BEGIN QUERY LOGGING LIMIT SUMMARY=1,2,3 ON ALL;
```

To summarize queries for 0.1, 0.5 and 1.0 seconds of CPU, (that is, four groups: 0-0.1, 0.1-0.5, 0.5-1.0, and greater than 1.0 CPU seconds) submit the following:

```
BEGIN QUERY LOGGING LIMIT SUMMARY=10,50,100 CPUTIME ON ALL;
```


To count queries using normalized CPU time rather than “raw” CPU time, use the CPUTIMENORM modifier.

Note:

Normalized columns, such as CPUTIMENORM, detect skew more accurately and are for co-existence systems only.

To group queries based on I/O, submit:

```
BEGIN QUERY LOGGING LIMIT SUMMARY=1000,5000,10000 IOCOUNT ON ALL;
```

LIMIT THRESHOLD = n [units]

Queries that run in n “units” or less are counted in the DBQLSummaryTbl. Queries that run more than n units are logged in the DBQLLogTbl. If you do not specify n , the default is 5 *units*. Units can be one of the following:

Unit	Summary Data Count Result
CPUTIME	AMP CPU time, where n is specified in hundredths of a second.
CPUTIMENORM	Normalized AMP CPU time, where n is specified in hundredths of a second. Note: Normalized columns, such as CPUTIMENORM, are for co-existence systems only.
ELAPSEDSEC	Elapsed time in seconds. If you do not specify units, ELAPSEDSEC is the default.
ELAPSEDTIME	Elapsed time, where n is specified in hundredths of a second.
IOCOUNT	Count of total I/Os, where n is specified in number of I/Os.

LIMIT THRESHOLD can be used with these logging options:

- OBJECTS
- STEPINFO
- SQL

The option ELAPSEDTIME is useful for specifying partial seconds.

For example, to log queries that use more than 8 CPU seconds in detail and tally queries that require less than 8 CPU seconds submit the following:

```
BEGIN QUERY LOGGING LIMIT THRESHOLD=800 CPUTIME ON ALL;
```

THRESHOLD is specified in hundredths of a second. If you set THRESHOLD=8, this would set the threshold at 0.08 CPU seconds.

The default is 5 hundredths CPU seconds. For example, if you submitted the following statement:

```
BEGIN QUERY LOGGING LIMIT THRESHOLD CPUTIME ON ALL;
```


without assigning a numeric value for THRESHOLD, the system defaults to summarizing queries that use less than 0.05 CPU seconds and gives details for all queries using more than 0.05 CPU seconds.

To count queries using normalized CPU time rather than “raw” CPU time, use the CPUTIMENORM modifier.

Note:

Normalized columns, like CPUTIMENORM, detect skew more accurately and are for co-existence systems only.

To base the threshold on the number of I/Os and log all queries with more than 5,000 I/Os, use the following:
`BEGIN QUERY LOGGING LIMIT THRESHOLD=5000 IOCOUNT ON ALL;`

You can even combine THRESHOLD with SQLTEXT to capture more than just the first 200 characters of a query that runs longer than THRESHOLD seconds (because the SQL text of short queries is not logged in DBQLogTbl).

Defining a threshold will determine whether to log a query or just count it, as follows:

Query Duration	DBQL Action
Completes at or under the threshold elapsed time, CPU seconds, or I/O count.	<ul style="list-style-type: none"> • Increments the query count and adds the elapsed time of the query, CPU time, and I/O counts to the summary row for this session in the current collection period. • Stores the final count for the session as a row in DBQLSummaryTbl.
Runs beyond the threshold value.	Logs a default row for the query in DBQLSqlTbl, DBQLStepTbl, and DBQLogTbl so you can examine query execution time through time stamps and the number and level of processing steps.

Mode Logging Option

The following option controls which CPU/IO collection algorithm Teradata uses for this query logging request.

Parameter	Logging Behavior
MODE = <i>m</i>	<p>The CPU/IO collection algorithm to use for this query logging request, as specified by <i>m</i>. Possible values are:</p> <ul style="list-style-type: none"> • 0 = Use the default value for this release (algorithm 1). • 1 = Use the classic algorithm 1 with step adjustments. • 2 = Use AMP algorithm 2 (diagnostic only). • 3 = Use AMP algorithm 3 (which includes data on aborted and parallel steps). <p>This is the default setting for fresh installations for Release 17.00 and later.</p> <p>If you do not specify MODE, the algorithm is controlled by the DBS Control General field 64 DBQL CPU/IO Collection. For more information, see <i>Teradata Vantage™ - Database Utilities</i>, B035-1102.</p> <p>To set the algorithm in DBS Control, see Setting DBQL Logging Algorithm.</p>

For SQL Engine 17.00 and later, the default mode for DBQL logging is set to Algorithm 3. Algorithm 3 collects information more accurately than the previous default of Algorithm 1. Algorithm 3 captures data for aborted and parallel steps, step instances for iterative steps, and Algorithm 3 includes additional information and status about step and request execution over Algorithms 1 and 2.

You must have the EXECUTE privilege on DBC.DBQLModeMacro to use this option.

Setting DBQL Logging Algorithm

The default mode for DBQL logging is set to Algorithm 3. If your environment uses a different mode, Teradata recommends switching to Algorithm 3. To switch to Algorithm 3, change DBS Control General field 64 DBQL CPU/IO Collection.

For details about DBQL CPU/IO Collection, see *Teradata Vantage™ - Database Utilities*, B035-1102.

1. Switch the mode to Algorithm 3. Modify General (g) field 64:

```
dbcontrol m g 64 = 3
```

2. Write the DBS control record for the change to take effect:

```
write
```

DBQL Macros

These macros are created in database DBC by the DIP utility script DIPVIEWSV during installation. For details, see “Database Initialization Program (DIP)” in *Teradata Vantage™ - Database Utilities*, B035-1102.

Dictionary Object	Purpose
DBQLAccessMacro	Controls the authority of users to execute the Teradata SQL BEGIN/REPLACE /FLUSH/END QUERY LOGGING statements.
DBQLModeMacro	Controls the authority of users to execute the MODE option of the BEGIN/REPLACE QUERY LOGGING statements.

DBQL Tables

The DBQL tables are created in database DBC by the DIP utility during installation. For details, see “Database Initialization Program (DIP)” in *Teradata Vantage™ - Database Utilities*, B035-1102. The tables are empty until you run a BEGIN QUERY LOGGING or REPLACE QUERY LOGGING request with the required option, as shown in the following table.

Dictionary Table	Purpose	Method Used To Populate
DBQLExplainTbl	Contains the explain information in an unformatted string without line breaks.	The EXPLAIN option.

Dictionary Table	Purpose	Method Used To Populate
DBQLLogTbl	Is the main table containing information about queries being logged.	A BEGIN QUERY LOGGING or REPLACE QUERY LOGGING request. No option is required.
DBQLObjTbl	Stores information on the target objects of the query being logged. One row is logged for each object referenced by the query.	The OBJECT option.
DBQLParamTbl	Logs the parameter variable name, type, position, and value for each parameter in a parameterized request. This table contains confidential user information and should only be accessed by personnel trusted with the DBC password.	The PARAMINFO option.
DBQLRuleCountTbl	Stores the cardinality of DBQLRuleTbl (for internal use only).	A BEGIN QUERY LOGGING request.
DBQLRuleTbl	Stores the rules resulting from each BEGIN QUERY LOGGING or REPLACE QUERY statement. One row exists for each set of username, account string, and application name specified in the BEGIN QUERY LOGGING and REPLACE QUERY LOGGING statements. The END QUERY LOGGING statement removes rows from the rule table. This table is for internal use only.	A BEGIN QUERY LOGGING or REPLACE QUERY LOGGING request.
DBQLSQLTbl	Stores the full SQL text of the query. One query string may require more than one row.	The WITH SQL option. Note: If a Teradata Dynamic Workload Management software exception occurs, the Teradata Dynamic Workload Management software causes rows to be written to the DBQLSQLTbl.
DBQLStepTbl	Stores information about each processing step used to satisfy the query. One row is logged for each step, including parallel steps.	The WITH STEPINFO option.
DBQLSummaryTbl	Stores information about queries that meet the criteria for a rule specifying the LIMIT SUMMARY or LIMIT THRESHOLD option.	The LIMIT SUMMARY or LIMIT THRESHOLD option.
DBQLUtilityTbl	Each row stores information about one completed load/export or Data Stream Architecture job.	The WITH UTILITYINFO option.
DBQLXMLLockTbl	Logs lock delays from the Lock Manager in the AMPs in XML format.	The WITH LOCK option.

Dictionary Table	Purpose	Method Used To Populate
DBQLXMLTbl	Stores the query plan for all DML statements and statistics usage information in an XML document.	The WITH XMLPLAN option or the WITH [DETAILED] STATSUSAGE option. You can also specify these options together.

Querying DBQL Tables

You can query the main table (DBQLLogTbl) alone, but to query other DBQL tables you need to create a join with DBQLLogTbl. Use the QueryID field to create the join.

DBQL Views

There is at least one view for each DBQL table. These views are created in database DBC during installation by the DIP utility scripts DIPVIEWSV and DIPJSON (for JSON views). For details, see “Database Initialization Program (DIP)” in *Teradata Vantage™ - Database Utilities*, B035-1102.

Dictionary Object	Purpose
DBQLRules[V]	Displays the current rules in DBC.DBQLRuleTbl (to a user with DBC or SystemFE privileges).
QryLockLogXML[V]	Accesses the DBQLXMLLockTbl.
QryLogExplainDoc[V]	Accesses the DBQLExplainTbl and provides a single, readable document per query.
QryLogExplain[V]	Accesses the DBQLExplainTbl.
QryLogFeatureListV	Accesses the FeatureNames table function, SYSLIB.FeatureNames_TBF().
QryLogFeatureUseCountV	Accesses the FeatureUsage column of DBQLLogTbl and the QryLogFeatureList view.
QryLogFeatureUseJSON	Accesses the FeatureUsage column of DBQLLogTbl and, in a JSON document, lists the features used by a particular request.
QryLogObjects[V]	Accesses DBQLObjTbl, and logs one row for each time an object is referenced by a query.
QryLogParamV	Accesses the parameterized query log table, DBQLParamTbl.
QryLogParamJSON	Accesses the parameterized query log table, DBQLParamTbl, in JSON format.
QryLogSQLDoc[V]	Accesses the DBQLSQLTbl and combines multiple rows of SQL text information with the same queryid into one document.
QryLogSQL[V]	Accesses the DBQLSQLTbl.
QryLogSteps[V]	Accesses the DBQLStepTbl.

Dictionary Object	Purpose
QryLogSummary[V]	Accesses the DBQLSummaryTbl.
QryLogTDWM[V]	The Teradata Dynamic Workload Management view of the default log table, DBQLLogTbl.
QryLog[V]	Accesses the DBQLLogTbl and contains the default DBQL information for a query
QryLogUtilityV	Accesses the DBQLUtilityTbl.
QryLogXMLDocV	Accesses the DBQLXMLTbl. Teradata recommends using this view instead of QryLogXML[V] to make sure your applications remain compatible with future enhancements.
QryLogXML[V]	Accesses the DBQLXMLTbl.

The DBQL views are predefined for ease of use. However, database administrators can create their own customized views by taking data from one or more DBQL tables and displaying the information in the format they need.

Note:

For greater details about the DBQL views, see *Views Reference* in *Teradata Vantage™ - Data Dictionary*, B035-1092.

Shredding the Lock Plan Information in the XML Lock Log Table, DBQLXMLLockTbl

You can shred (format as a table) the lock plan information in the XMLTextInfo column of DBQLXMLLockTbl to be able to query it and read it. Without shredding, output from that column typically looks like this:

```
sel xmltextinfo from dbc.dbqlxmllocktbl;
```

Result:

```
*** Query completed. 7 rows found. One column returned.
*** Total elapsed time was 1 second.
XMLTextInfo
<?xml version="1.0" encoding="UTF-8"?> <!--XML row for
DBQLLockXML--> <DBQLLockXML xsi:schemaLocation="http://schemas.teradata.com/
dbqllockplan dbqllockplan.xsd" xmlns="http://schemas.teradata.com/dbqllockplan"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <LockContention
QueryID="307191340665578443" CollectTimeStamp="2014-09-07T23:27:07.16"
DelayStartTime="2014-09-07T23:24:01.70" AbortFlag="FALSE" TransactionCount="0">
<ContentionData LockDelay="1438" LocalDeadLock="false" GlobalDeadLock="false"
```



```

MultipleBlocker="false" vproc="1"> <BlockerRef> <RequestRef>
<RequestID RequestID_1="0" RequestID_2="9"/> <Step StepLevel="0"
StepNum_1="1" StepNum_2="0" StatementNo="1"/> <Transaction
unique_1="1" unique_2="26210" vproc="30719" TransactionState="Active"
TransactionStartTime="2014-09-07T23:23:07.2" /> </RequestRef> <SessionRef
LogicalHostId="1"> <Session SessionID_1="0" SessionID_2="1139"
SessionID_Combined="1139"/> <User UserName="DBC" AccountName="DBC"/> </
SessionRef> <LockData LockTypeRequested="Read" LockObjectRequested="R"
LockKind="RowHash on All Partitions" > </LockData> <LockObject
DatabaseName="MAM" TableName="t1"/> <Job> <Message MessageClass="21"
MessageKind="8" MessageClassName="SYSMGS2SCLASS" MessageKindName="TNT"/>
<Operation WorkLevel="0" OperationType="TerminAted Transaction (Spawned
StpEDT)" JobType="AmpStep"/> </Job> </BlockerRef> <BlockedRef> <RequestRef>
</RequestRef> <SessionRef LogicalHostId="1"> <Session SessionID_1="0"
SessionID_2="1140" SessionID_Combined="1140"/> </SessionRef> <LockData
LockTypeRequested="Write" LockObjectRequested="R" LockKind="RowHash on All
Partitions" StartPartition="0" StartRowHashInt="4294967295" EndPartition="0"
EndRowHashInt="4294967295" > </LockData> </BlockedRef> </ContentionData> </
LockContention> </DBQLLockXML>

```

Note:

You cannot shred the contents of the XMLTextInfo column in QryLockLogXMLV. To shred this column, you must use the underlying table, DBQLXMLLockTbl.

Shredding the Data in the DBQLXMLLockTbl

To shred the lock plan data in the XMLTextInfo field and see details of a lock contention, follow this procedure:

1. Make sure that the DIPPOST script ran on the system. At installation, DIPPOST is the final script executed as part of DIPALL. The script executes quickly if you need to run it. From the Linux command line, start DIP and then run the DIPPOST script:

```
dip
```

Select the name of the post commands script:

```

DIPPOST
Executing DIPPOST
Please wait...
DIPPOST is complete

```

```
Would you like to execute another DIP script (Y/N)?
```



```
n
Exiting DIP...
```

- Using BTEQ, log on, delete any objects created by previous use of this procedure, and create a user-defined target database:

```
.logon unzen,dbc;
drop database TargetDBName
create database TargetDBName as perm=10000000
```

- Grant privileges to the new user locklogshredder and log on as locklogshredder.

Note:

The DIPPOST script created user locklogshredder.

```
grant all on TargetDBName to TargetDBName with grant option;
grant all on TargetDBName to dbc with grant option;
grant all on TargetDBName to locklogshredder with grant option;
grant all on locklogshredder to TargetDBName with grant option;
grant all on td_sysxml to TargetDBName with grant option;
grant all on TargetDBName to td_sysxml with grant option;

.logon unzen/locklogshredder;
```

- Drop the target tables in case they remain from a previous shredding operation.

Note:

The shredding operation creates the target tables to store the shredded data. The user does not create the SQL DDL. However, the user must prepare space for and accommodate the table. After each shredding operation, clean up the data from the previous shredding operation to avoid accumulating old data.

```
drop table TargetDBName.Shredded_LockTbl;
drop table TargetDBName.Shredded_TransTbl;
```

- Prepare the output to be formatted as needed using BTEQ.
- Call the stored procedure that creates the target tables with the shredded data. If the tables already exist, a non-zero result is returned.

```
CALL
LockLogShredder.SP_CreateTable('TargetDBName','Shredded_LockTbl',
'Shredded_TransTbl', resultCode);
```

- Call the stored procedure that populates the target tables.


```
CALL LockLogShredder.SP_LockLog_Shredder('select random(0,21437899),
CREATEXML(xmltextinfo) from dbc.dbqlxmllocktbl',
'TargetDBName','Shredded_Locktbl', 'Shredded_TransTbl', resultCode);
```

This stored procedure shreds the XML data on DBQLXMLLockTbl, converting each XML from a long text string into multiple fields and storing them in two target tables. Target table Shredded_LockTbl holds the shredded output of most of the lock information in the TextInfo column of DBQLXMLLockTbl. The lock contention information contains only the blocking transaction and the first transaction that is blocked by it. Target table Shredded_TransTbl expands this to include all blocked transactions waiting behind the blocking transaction.

8. Display the count of the shredded rows in the target tables, so you know in advance the size of the result:

```
sel count(*) from TargetDBName.Shredded_LockTbl;
sel count(*) from TargetDBName.Shredded_TransTbl;
```

9. Query the shredded lock table and transaction table to see the list of blocked transactions:

```
sel queryid from TargetDBName.Shredded_LockTbl order by 1;
sel queryid from TargetDBName.Shredded_TransTbl order by 1;
```

10. Query the data to see the primary lock table rows.

The following example retrieves information from Shredded_LockTbl to show the queries that are blocked from getting requested locks. Here table t3 already has a Write lock on it, and two other requests are blocked when they try to get Read and Exclusive locks on the same table. The Shredded_LockTbl.BlockedLockTypeRequested field gives information about the lock request that blocked the queries.

```
SELECT
QueryId,
DatabaseName,
TableName,
SessionID_Combined ,
BlockedUserName,
BlkedDatabaseName ,
BlkedTableName ,
BlkedPEVproc,
BlkedTransactionStartTime,
BlkedStepNum_1,
BlkedStepNum_2,
BlockedSessionID_Combined,
BlockedLockTypeRequested,
```



```
BlkedTransactionState
FROM TargetDBName.Shredded_LockTbl;
```

Result:

```
*** Query completed. 2 rows found. 14 columns returned.
*** Total elapsed time was 1 second.
```

```

        queryid  307195142968284064.
      DatabaseName  USER1
        TableName  t3
  SessionID_Combined      1646
    BlockedUserName  ?
    BlkedDatabaseName  ?
      BlkedTableName  ?
      BlkedPEVproc      ?
BlkedTransactionStartTime      ?
      BlkedStepNum_1      ?
      BlkedStepNum_2      ?
BlockedSessionID_Combined      1650
BlockedLockTypeRequested  Read
  BlkedTransactionState  ?
        queryid  307195142968284063.
      DatabaseName  USER1
        TableName  t3
  SessionID_Combined      1650
    BlockedUserName  ?
    BlkedDatabaseName  ?
      BlkedTableName  ?
      BlkedPEVproc      ?
BlkedTransactionStartTime      ?
      BlkedStepNum_1      ?
      BlkedStepNum_2      ?
BlockedSessionID_Combined      1651
BlockedLockTypeRequested  Exclusive
  BlkedTransactionState  ?
```

11. Query the data to see the transaction table rows.

The following example shows the transaction state of a request during a lock contention. A query was blocked infinitely for requesting an Exclusive lock on t3 and was aborted after a while. In the following output, a second entry for the same queryid, without any object information and transaction state, shows that the query was aborted due to infinite waiting.


```

SELECT
QueryID,
DatabaseName,
TableName,
StepNum_1,
StepNum_2,
TransactionState,
TransactionStartTime,
unique_1 ,
unique_2 ,
PEVproc ,
LockObjectRequested,
LockTypeRequested,
LockStatus
FROM TargetDBName.Shredded_TransTbl;

```

Result:

```

*** Query completed. 3 rows found. 13 columns returned.
*** Total elapsed time was 1 second.

```

```

        queryid  307195142968284072.
DatabaseName  USER1
  TableName  t3
  StepNum_1      1
  StepNum_2      0
TransactionState  Inactive
TransactionStartTime  2017-11-29 06:09:01.930000
        unique_1      15
        unique_2      10014
        PEVproc      30719
LockObjectRequested  R
  LockTypeRequested  Exclusive
    LockStatus  Waiting
      queryid  307195142968284071.
DatabaseName  ?
  TableName  ?
  StepNum_1      ?
  StepNum_2      ?
TransactionState  ?
TransactionStartTime      ?
        unique_1      ?
        unique_2      ?

```



```

        PEVproc          ?
LockObjectRequested ?
    LockTypeRequested ?
        LockStatus ?
            queryid 307195142968284072.
        DatabaseName ?
            TableName ?
                StepNum_1 ?
                StepNum_2 ?
        TransactionState ?
TransactionStartTime ?
    unique_1 ?
    unique_2 ?
        PEVproc ?
LockObjectRequested R
    LockTypeRequested Exclusive
        LockStatus Waiting

```

12. Join the shredded target tables Shredded_LockTbl and Shredded_TransTbl using the QueryId field, which is in both tables.

```

SELECT
a.queryid (TITLE 'QUERYID'),
a.DatabaseName (TITLE 'Lock_DatabaseName'),
a.TableName (TITLE 'Lock_TableName'),
a.UserName (TITLE 'Lock_UserName'),
a.AccountName (TITLE 'Lock_AccountName'),
a.MessageClass (TITLE 'Lock_MessageClass'),
a.MessageKind (TITLE 'Lock_MessageKind'),
a.MessageClassName (TITLE 'Lock_MessageClassName'),
a.MessageKindName (TITLE 'Lock_MessageKindName'),
a.WorkLevel (TITLE 'Lock_WorkLevel'),
a.OperationType (TITLE 'Lock_OperationType'),
a.JobType (TITLE 'Lock_JobType'),
a.LockTypeRequested (TITLE 'Lock_LockTypeRequested'),
a.LockObjectRequested (TITLE 'Lock_LockObjectRequested'),
a.BlockedLockTypeRequested (TITLE 'Lock_BlockedLockTypeRequested'),
a.BlockedLockObjectRequested (TITLE 'Lock_BlockedLockObjectRequested'),
a.LockDelay (TITLE 'Lock_LockDelay'),
a.LocalDeadLock (TITLE 'Lock_LocalDeadLock'),
a.GlobalDeadLock (TITLE 'Lock_GlobalDeadLock'),
a.MultipleBlocker (TITLE 'Lock_MultipleBlocker'),
a.ErrorText (TITLE 'Lock_ErrorText'),
a.AbortFlag (TITLE 'Lock_AbortFlag'),

```



```

a.ErrorCode (TITLE 'Lock_ErrorCode'),
b.queryid (TITLE 'COMMON QUERYID'),
b.DatabaseName (TITLE 'Trans_DatabaseName'),
b.TableName (TITLE 'Trans_TableName'),
b.unique_1 (TITLE 'Trans_unique_1'),
b.unique_2 (TITLE 'Trans_unique_2'),
b.PEVproc (TITLE 'Trans_PEVproc'),
b.LockTypeRequested (TITLE 'Trans_LockTypeRequested'),
b.LockObjectRequested (TITLE 'Trans_LockObjectRequested')
FROM TargetDBName.Shredded_LockTbl a, TargetDBName.Shredded_TransTbl b
WHERE a.QueryId = b.QueryId and a.GlobalDeadLock='true';

```

Output of Shredding

After shredding the output of the DBQLXMLLockTbl XMLTextInfo column in Shredded_LockTbl and joining it with the shredded output of Shredded_TransTbl, the result looks like this.

The GlobalDeadLock field has a TRUE value, showing a global deadlock condition.

```
*** Query completed. 6 rows found. 31 columns returned.
```

```
*** Total elapsed time was 1 second.
```

```

                QUERYID  307184741735817865.
      Lock_DatabaseName DB1
        Lock_TableName TVFields
          Lock_UserName USER2
        Lock_AccountName ?
      Lock_MessageClass      21
        Lock_MessageKind    11
Lock_MessageClassName SYSMGS2SCLASS
      Lock_MessageKindName AAR
        Lock_WorkLevel      128
      Lock_OperationType Asynch Abort Release lock message
        Lock_JobType AmpStep
Lock_LockTypeRequested Read4W
Lock_LockObjectRequested R
Lock_BlockedLockTypeRequested Write
Lock_BlockedLockObjectRequested T
        Lock_LockDelay      26400.
        Lock_LocalDeadLock false
        Lock_GlobalDeadLock true
        Lock_MultipleBlocker true
        Lock_ErrorText Amp deadlocking
        Lock_AbortFlag TRUE

```



```

        Lock_ErrorCode          2631
        COMMON QUERYID 307184741735817865.
Trans_DatabaseName ?
Trans_TableName ?
Trans_unique_1      ?
Trans_unique_2      ?
Trans_PEVproc       ?
Trans_LockTypeRequested Write
Trans_LockObjectRequested R
        QUERYID 307184741735817865.
Lock_DatabaseName DB1
Lock_TableName TVFields
Lock_UserName USER2
Lock_AccountName ?
Lock_MessageClass    21
Lock_MessageKind     11
Lock_MessageClassName SYMSGSS2SCLASS
Lock_MessageKindName AAR
Lock_WorkLevel       128
Lock_OperationType Asynch Abort Release lock message
Lock_JobType AmpStep
Lock_LockTypeRequested Write
Lock_LockObjectRequested R
Lock_BlockedLockTypeRequested Write
Lock_BlockedLockObjectRequested T
        Lock_LockDelay          26400.
Lock_LocalDeadLock false
Lock_GlobalDeadLock true
Lock_MultipleBlocker true
Lock_ErrorText Amp deadlocking
Lock_AbortFlag TRUE
Lock_ErrorCode          2631
COMMON QUERYID 307184741735817865.
Trans_DatabaseName ?
Trans_TableName ?
Trans_unique_1      ?
Trans_unique_2      ?
Trans_PEVproc       ?
Trans_LockTypeRequested Write

Trans_LockObjectRequested R
        QUERYID 307184741735817865.
Lock_DatabaseName DB1
Lock_TableName TVFields

```



```

        Lock_UserName USER2
        Lock_AccountName ?
        Lock_MessageClass          21
        Lock_MessageKind           11
        Lock_MessageClassName SYMSG2SCLASS
        Lock_MessageKindName AAR
        Lock_WorkLevel             128
        Lock_OperationType Asynch Abort Release lock message
        Lock_JobType AmpStep
        Lock_LockTypeRequested Read4W
        Lock_LockObjectRequested R
        Lock_BlockedLockTypeRequested Write
        Lock_BlockedLockObjectRequested T
        Lock_LockDelay              26400.
        Lock_LocalDeadLock false
        Lock_GlobalDeadLock true
        Lock_MultipleBlocker true
        Lock_ErrorText Amp deadlocking
        Lock_AbortFlag TRUE
        Lock_ErrorCode              2631
        COMMON QUERYID 307184741735817865.
        Trans_DatabaseName DB1
        Trans_TableName TVFields
        Trans_unique_1             14
        Trans_unique_2             61610
        Trans_PEVproc              30719
        Trans_LockTypeRequested Write
        Trans_LockObjectRequested R
        QUERYID 307184741735817865.
        Lock_DatabaseName DB1
        Lock_TableName TVFields
        Lock_UserName USER2
        Lock_AccountName ?
        Lock_MessageClass          21
        Lock_MessageKind           11
        Lock_MessageClassName SYMSG2SCLASS
        Lock_MessageKindName AAR
        Lock_WorkLevel             128
        Lock_OperationType Asynch Abort Release lock message
        Lock_JobType AmpStep
        Lock_LockTypeRequested Write
        Lock_LockObjectRequested R
        Lock_BlockedLockTypeRequested Write
        Lock_BlockedLockObjectRequested T

```



```

        Lock_LockDelay                26400.
    Lock_LocalDeadLock false
    Lock_GlobalDeadLock true
    Lock_MultipleBlocker true
        Lock_ErrorText Amp deadlocking
        Lock_AbortFlag TRUE
        Lock_ErrorCode        2631
        COMMON QUERYID 307184741735817865.
    Trans_DatabaseName DB1
        Trans_TableName TVFields
        Trans_unique_1    14
        Trans_unique_2    61610
        Trans_PEVproc     30719
    Trans_LockTypeRequested Write
    Trans_LockObjectRequested R
        QUERYID 307184741735817865.
    Lock_DatabaseName DB1
        Lock_TableName TVFields
        Lock_UserName USER2
        Lock_AccountName ?
        Lock_MessageClass    21
        Lock_MessageKind     11
    Lock_MessageClassName SYSMGS2SCLASS
    Lock_MessageKindName AAR
        Lock_WorkLevel      128
    Lock_OperationType Asynch Abort Release lock message
        Lock_JobType AmpStep
    Lock_LockTypeRequested Read4W
    Lock_LockObjectRequested R
    Lock_BlockedLockTypeRequested Write
    Lock_BlockedLockObjectRequested T
        Lock_LockDelay                26400.
    Lock_LocalDeadLock false
    Lock_GlobalDeadLock true
    Lock_MultipleBlocker true
        Lock_ErrorText Amp deadlocking
        Lock_AbortFlag TRUE
        Lock_ErrorCode        2631
        COMMON QUERYID 307184741735817865.
    Trans_DatabaseName DB1
        Trans_TableName TVFields
        Trans_unique_1    14
        Trans_unique_2    61610
        Trans_PEVproc     30719

```



```

Trans_LockTypeRequested Write
Trans_LockObjectRequested R
      QUERYID  307184741735817865.
    Lock_DatabaseName DB1
      Lock_TableName TVFields
      Lock_UserName  USER2
      Lock_AccountName ?
      Lock_MessageClass      21
      Lock_MessageKind      11
    Lock_MessageClassName SYMSG2SCLASS
    Lock_MessageKindName  AAR
      Lock_WorkLevel      128
      Lock_OperationType Asynch Abort Release lock message
      Lock_JobType AmpStep
    Lock_LockTypeRequested Write
    Lock_LockObjectRequested R
    Lock_BlockedLockTypeRequested Write
    Lock_BlockedLockObjectRequested T
      Lock_LockDelay      26400.
      Lock_LocalDeadLock false
      Lock_GlobalDeadLock true
      Lock_MultipleBlocker true
      Lock_ErrorText Amp deadlocking
      Lock_AbortFlag TRUE
      Lock_ErrorCode      2631
      COMMON QUERYID  307184741735817865.
    Trans_DatabaseName DB1
      Trans_TableName TVFields
      Trans_unique_1      14
      Trans_unique_2      61610
      Trans_PEVproc      30719
    Trans_LockTypeRequested Write
    Trans_LockObjectRequested R

```

***Shredded_LockTbl*: Blocking Transaction Lock Shredding Target Table**

The *TargetDBName.Shredded_LockTbl* stores shredded lock contention data for the blocking transaction and the first transaction that is blocked by it. The exact name of the target table is specified by the call to the stored procedure LockLogShredder.SP_CreateTable.

<i>Shredded_LockTbl</i> field	Description
DatabaseName	The database containing the object locked by the blocker

<i>Shredded_LockTbl</i> field	Description
TableName	The table locked by the blocker
RequestID_1	Byte 1 of the request ID of the blocker
RequestID_2	Byte 2 of the request ID of the blocker
unique_1	Part 1 of the transaction ID of the blocker
unique_2	Part 2 of the transaction ID of the blocker
PEVproc	PE vproc of the blocker
TransactionState	Transaction state of the blocker Possible values include the following: <ul style="list-style-type: none"> • Inactive • Active • DontAbort • Aborting • PKAborted
TransactionStartTime	Transaction start time of the blocker
StepLevel	The dispatcher recorded step level of the blocker
StepNum_1	Part 1 of the step number of the blocker
StepNum_2	Part 2 of the step number of the blocker
StatementNo	The statement number of the blocker
SessionID_1	Part 1 of the session ID of the blocker
SessionID_2	Part 2 of the session ID of the blocker
SessionID_Combined	Combined session ID of blocker, as recorded in SessionTbl
UserName	User name of the blocker
AccountName	Account name of the blocker
ExpReqProc	Originating vproc of the internal express request lock of the blocker
seq_1	Part 1 of the logon sequence number of a HUT lock blocker
seq_2	Part 2 of the logon sequence number of a HUT lock blocker
LogicalHostId	Logical host ID of the blocker
Zoneld	Secure zone ID of the blocker
MessageClass	Internal AMP message class ID of the blocker
MessageKind	Internal AMP message kind ID of the blocker
MessageClassName	Internal AMP message class name of the blocker

<i>Shredded_LockTbl</i> field	Description
	<p>Possible values include the following:</p> <ul style="list-style-type: none"> • SYMSGSTPCCLASS (messages containing Parser-generated steps sent to the AMPs from the Dispatcher) • SYMSGSG2SCLASS (messages sent between steps during row redistribution or count cascading) • SYMSGGHUTCLASS (messages sent between host utilities and AMPs) • SYMSGEXPCLASS (messages sent as express requests between the Parser and AMPs)
MessageKindName	Internal AMP message kind name of the blocker
WorkLevel	Operation work level number of the blocker
OperationType	Type of SQL operation being performed by the blocker
JobType	<p>Job types</p> <p>Possible values include the following:</p> <ul style="list-style-type: none"> • AmpStep • ExpressRequest • Utilities • Archive/Restore • Undetermined
SQLType	Type of SQL performed in the operation of the blocker
LockTypeRequested	<p>Type of lock requested by the blocker</p> <p>Possible values include the following:</p> <ul style="list-style-type: none"> • Access • Read • Write • Exclusive • IAccess (intentional access) • IRead (intentional read) • IWrite (intentional write) • IExclusive (intentional exclusive) • Read4X (read for exclusive) • IRead4X (intentional read for exclusive) • Read4W (read for write) • ReadHL (read host utility lock) • IReadHL (intentional read host utility lock)
LockObjectRequested	<p>Type of object on which the blocker requested a lock</p> <p>Possible values include the following:</p> <ul style="list-style-type: none"> • D (database) • T (table) • R (row hash) • RK (row hash in one partition)

<i>Shredded_LockTbl</i> field	Description
	<ul style="list-style-type: none"> • RN (row key range) • RP (row hash in partition range) • TP (table partition range)
LockKindRequested	<p>The kind of lock requested by the blocker</p> <p>Possible values include the following:</p> <ul style="list-style-type: none"> • Rowhash range • Rowhash on all partitions • Rowkey • Rowkey range (locked for a range of rows) • Rowhash and partition range • Partition range with complete rowhash range • Rowkey range for internal use
BlkedDatabaseName	The database containing the object on which the blocked request is waiting for a lock
BlkedTableName	The table on which the blocked request is waiting for a lock
BlkedRequestID_1	Byte 1 of the ID of the blocked request
BlkedRequestID_2	Byte 2 of the ID of the blocked request
Blkedunique_1	Part 1 of the transaction ID of the blocked request
Blkedunique_2	Part 2 of the transaction ID of the blocked request
BlkedPEVproc	PE vproc of the blocked request
BlkedTransactionState	<p>Transaction state of the blocked request</p> <p>Possible values include the following:</p> <ul style="list-style-type: none"> • Inactive • Active • DontAbort • Aborting • PKAborted
BlkedTransactionStartTime	Start time of the blocked transaction
BlkedStepNum_1	Part 1 of the step number of the blocked request
BlkedStepNum_2	Part 2 of the step number of the blocked request
BlockedSessionID_1	Part 1 of the Session ID of the blocked request
BlockedSessionID_2	Part 2 of the session ID of the blocked request
BlockedSessionID_Combined	Combined session ID of the blocked request as recorded in SessionTbl
BlockedUserName	User name for the blocked request

<i>Shredded_LockTbl</i> field	Description
BlockedAccountName	Account name for the blocked request
BlockedExpReqProc	Originating vproc of the internal express request lock of the blocked request
Blockedseq_1	Part 1 of the logon sequence number for the blocked request
Blockedseq_2	Part 2 of the logon sequence number for the blocked request
BlockedLogicalHostId	Logical host ID of the blocked request
BlockedStartPartition	Starting partition number for the partition-level lock of the blocked request
BlockedStartRowhashInt	Starting row hash number for the partition-level lock of the blocked request
BlockedEndPartition	Ending partition number for the partition-level lock of the blocked request
BlockedEndRowHashInt	Ending row hash number for the partition-level lock of the blocked request
BlockedLockTypeRequested	Type of lock requested by the blocked request Possible values include the following: <ul style="list-style-type: none"> • Access • Read • Write • Exclusive • IAccess (intentional access) • IRead (intentional read) • IWrite (intentional write) • IExclusive (intentional exclusive)) • Read4X (read for exclusive) • IRead4X (intentional read for exclusive) • Read4W (read for write) • ReadHL (read host utility lock) • IReadHL (intentional read host utility lock)
BlockedLockObjectRequested	The type of object on which the blocked request needs a lock Possible values include the following: <ul style="list-style-type: none"> • D (database) • T (table) • R (row hash) • RK (row hash in one partition) • RN (row key range) • RP (row hash in partition range) • TP (table partition range)
BlockedLockKindRequested	The kind of lock requested by the blocked request Possible values include the following: <ul style="list-style-type: none"> • Rowhash range

<i>Shredded_LockTbl</i> field	Description
	<ul style="list-style-type: none"> • Rowhash on all partitions • Rowkey • Rowkey range (locked for a range of rows) • Rowhash and partition range • Partition range with complete rowhash range • Rowkey range for internal use
LockDelay	Delay time of the blocked request in centiseconds
LocalDeadLock	True if a local deadlock is detected, otherwise false
GlobalDeadLock	True if a global deadlock is detected, otherwise false
MultipleBlocker	True if there are multiple blocked requests, otherwise false
AmpVproc	AMP number where the lock contention occurred
QueryID	Unique query ID for joining with related lock shredded tables or other DBQL tables
CollectTimeStamp	Timestamp of when the lock contention was recorded
DelayStartTime	Timestamp of when the lock delay began
ErrorText	Error text associated with the lock contention, if any
AbortFlag	True if the lock contention has resulted in an abort
ErrorCode	Error code associated with the lock contention, if any
TransactionCount	Number of additional blocked transactions

Investigating Deadlocks: Querying *Shredded_LockTbl*

In the following scenario, the user received an error message indicating an AMP deadlock, which happens if two sessions simultaneously try to lock the same table. The following example provides further information about the aborted request that caused the error message.

```
SELECT
QueryId,
DatabaseName,
TableName,
UserName,
AccountName,
OperationType,
JobType,
LockTypeRequested,
```



```

LockObjectRequested,
ErrorText,
AbortFlag,
ErrorCode
FROM TargetDBName.Shredded_LockTbl
WHERE AbortFlag = 1 and ErrorCode = 2631;

```

The following output provides details about the blocking transaction and the first transaction that is blocked by it.

```

*** Query completed. 3 rows found. 8 columns returned.
*** Total elapsed time was 1 second.

```

```

      queryid  307184741735818131.
DatabaseName  USER1
      TableName  t3
      UserName  USER1
AccountName ?
OperationType  Asynch Abort Release lock message
JobType        AmpStep
LockTypeRequested  Write
LockObjectRequested  R
      ErrorText  Amp deadlocking
      AbortFlag  TRUE
      ErrorCode      2631
      queryid  307184741735818130.
DatabaseName  USER1
      TableName  t3
      UserName  USER1
AccountName ?
OperationType  Asynch Abort Release lock message
JobType        AmpStep
LockTypeRequested  Write
LockObjectRequested  R
      ErrorText  Amp deadlocking
      AbortFlag  TRUE
      ErrorCode      2631

```


***Shredded_TransTbl*: Blocked Transaction Lock Shredding Target Table**

The *TargetDBName.Shredded_TransTbl* stores additional shredded data for all blocked transactions involved in a lock contention. This table joins with the related table *Shredded_LockTbl*. The exact name of the target table is specified by the call to the stored procedure *LockLogShredder.SP_CreateTable*.

<i>Shredded_TransTbl</i> field	Description
QueryID	Unique query ID for joining with <i>Shredded_LockTbl</i> or other DBQL tables
TransactionOrder	Sequence order of the blocked transaction
DatabaseName	The database containing the object on which the blocked request is waiting for a lock
TableName	The table on which the blocked request is waiting for a lock
RequestID_1	Byte 1 of the ID of the blocked request
RequestID_2	Byte 2 of the ID of the blocked request
unique_1	Part 1 of the transaction ID of the blocked request
unique_2	Part 2 of the transaction ID of the blocked request
PEVproc	PE vproc of the blocked request
TransactionState	Transaction state of the blocked request Possible values include the following: <ul style="list-style-type: none"> • Inactive • Active • DontAbort • Aborting • PKAborted
TransactionStartTime	Start time of the blocked transaction
StepNum_1	Part 1 of the step number of the blocked request
StepNum_2	Part 2 of the step number of the blocked request
SessionID_1	Part 1 of the Session ID of the blocked request
SessionID_2	Part 2 of the Session ID of the blocked request
SessionID_Combined	Combined session ID of the blocked request, as recorded in <i>SessionTbl</i>
LockStatus	Lock status of the blocked transaction Possible values include the following: <ul style="list-style-type: none"> • Waiting • Granted

<i>Shredded_TransTbl</i> field	Description
LockTypeRequested	<p>Type of lock requested by the blocked request</p> <p>Possible values include the following:</p> <ul style="list-style-type: none"> • Access • Read • Write • Exclusive • IAccess (intentional access) • IRead (intentional read) • IWrite (intentional write) • IExclusive (intentional exclusive)) • Read4X (read for exclusive) • IRead4X (intentional read for exclusive) • Read4W (read for write) • ReadHL (read host utility lock) • IReadHL (intentional read host utility lock)
LockObjectRequested	<p>The type of object on which the blocked request needs a lock</p> <p>Possible values include the following:</p> <ul style="list-style-type: none"> • D (database) • T (table) • R (row hash) • RK (row hash in one partition) • RN (row key range) • RP (row hash in partition range) • TP (table partition range)

Query Data Storage and Protection

Because of its nature, there is no recovery mechanism for the cache in which DBQL row values are collected. Should a restart occur, any rows in cache that have not been sent to the AMPs are lost. However, query logging is not aborted as a result of a restart; DBQL uses the contents of DBQLRuleTbl to continue logging.

For information on flushing the DBQL cache information to the logs, see [Options for Flushing the DBQL Cache](#) and [Flushing the DBQL Caches Manually](#).

Protection for a Committed Row

DBQL tables occupy permanent space in database DBC. This means that once a row has been sent to an AMP, its insertion into a DBQL log is safeguarded through the transaction and recovery journaling process.

In addition, the data in every DBQL table is protected with FALLBACK. This means that the data is always available unless two or more AMPs in the same clique fail simultaneously.

Things to Consider When Logging DBQL Data

- You must have EXECUTE privilege on the DBC.DBQLAccessMacro macro to log DBQL data. To grant the EXECUTE privilege to an administrative user, see [Granting DBQL Administrative Privileges to Other Users](#).
- Teradata recommends that you use BTEQ script files to enable DBQL. You can define the rules for logging in these scripts. Just be sure to keep the scripts up to date and saved for future use. You can also create files that disable query logging for your convenience.

Logging Overhead

Before you enable logging, first consider how much and what sort of data you need for adequate analysis. The more information you ask for and the more users you track, the higher the cost to performance and the faster the logs will grow. However, you can specify summary and threshold limits to obtain meaningful data with minimal cost.

The following table lists examples that describe the overhead incurred per user and account.

IF the collection type is ...	THEN logging is per ...	AND overhead entails ...	IN the following log(s) ...
default (no options specified)	query	one default row per query	DBQLLogTbl
individual, because the query ran longer than THRESHOLD seconds	query that runs in more time, or used more CPU or I/O limits	one default row per long-running query	DBQLSqlTbl, DBQLStepTbl, DBQLObjTbl, and DBQLLogTbl
counts of queries that complete within SUMMARY intervals	query that runs in less time, or used less CPU or I/O limits	when the query count is >0, one row per logging interval (every 10 minutes). Possible maximum is four rows every 10 minutes for the duration of the session	DBQLSummaryTbl
steps process detail	query	one row for each step generated	DBQLStepTbl
object detail	query	one row for each object used to resolve the query	DBQLObjTbl
explain detail	query	as many rows as it takes to capture the complete explain text. The explain text is not formatted. Collecting explain detail has some performance impact.	DBQLExplainTbl
SQL text detail	query	as many rows as it takes to capture the complete text of the SQL request	DBQLSQLTbl

IF the collection type is ...	THEN logging is per ...	AND overhead entails ...	IN the following log(s) ...
XML query plan detail	query	as many rows as it takes to capture the complete XML plan. The XML plan includes the SQL request and other things such as the EXPLAIN text Note: For more information, see <i>Teradata Vantage™ - SQL Request and Transaction Processing</i> , B035-1142.	DBQLXMLTbl
XML lock logging	lock of qualifying duration	as many rows as it takes to capture the XML lock plan	DBQLXMLLOCKTbl

Options for Flushing the DBQL Cache

Before Vantage writes DBQL data to Data Dictionary tables, it holds the data in cache until either:

- The cache is full.
- You end query logging.
- The number of seconds has elapsed that you define in the DBS Control utility field DBQLFlushRate.

Note:

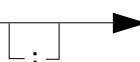
You can select a rate from 1 to 3,600 seconds. However, Teradata recommends a flush rate of at least 10 minutes (600 seconds), which is the default. Less than 10 minutes can impact performance.

- You flush the DBQL cache manually.

Flushing the DBQL Caches Manually

If you need to flush DBQL caches more quickly than the time defined by DBQLFlushRate, take one of the following actions:

- Reduce the cache size. See [Changing DBQL Cache Size](#).
- Issue the FLUSH QUERY LOGGING request:

FLUSH QUERY LOGGING WITH `flush_option` 

where *flush_option* can be:

Option Name	Table Cache Flushed
ALL	DBQLogTbl, DBQLSQLTbl, DBQLStepTbl, DBQLObjTbl, DBQLXMLTbl, DBQLExplainTbl, DBQLXMLLockTbl, DBQLParamTbl, and DBC.ObjectUsage
ALLDBQL	DBQLogTbl, DBQLSQLTbl, DBQLStepTbl, DBQLObjTbl, DBQLXMLTbl, DBQLExplainTbl, DBQLXMLLockTbl, and DBQLParamTbl
DEFAULT	DBQLogTbl
EXPLAIN	DBQLExplainTbl
LOCK	DBQLXMLLockTbl
OBJECTS	DBQLObjTbl
PARAMINFO	DBQLParamTbl
SQL	DBQLSQLTbl
STEPINFO	DBQLStepTbl
SUMMARY	DBQLSummaryTbl
USECOUNT	DBC.ObjectUsage
XMLPLAN	DBQLXMLTbl

This request:

- Requires the EXECUTE privilege on DBC.DBQLAccessMacro
- Is valid in BTET (Teradata) mode only
- Cannot be part of a multistatement request
- Cannot be used in a stored procedure

Note:

This request is especially recommended when a system is not busy and caches are not filling up and written to Data Dictionary tables as frequently. This request can also flush Teradata dynamic workload management software caches. Only the DBQL options are described here. For the Teradata dynamic workload management software options, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

- Reset the value in the DBQLFlushRate to a faster rate.

Note:

Changes to DBQLFlushRate also causes the summary/threshold cache to flush, which causes the summary/threshold collection time interval to change.

Changing DBQL Cache Size

Use the following procedure to change how much data DBQL caches hold before writing to Data Dictionary tables. Cache size also determines whether DBQL uses 64 KB or 1 MB data blocks for disk writes. Increasing the cache size reduces overhead and resource contention and improves performance. Although 2 MB is optimal in most cases, consider increasing the cache size if detailed query logging is enabled and queries experience resource contention.

1. Flush all DBQL caches so that their contents are not lost during the required system restart:

```
FLUSH QUERY LOGGING WITH ALL;
```

2. Determine the current value of the DBS Control field DBQLDefCacheSize, which is in the Performance group.

```
DISPLAY PERFORMANCE
```

3. Change the value of DBQLDefCacheSize:

```
MODIFY PERFORMANCE 35 = value
```

Valid values are from 0 to 16 MB, where 0 sets the cache size to 64 KB and the data block size to 64 KB. Choose a size larger than 0 to set the data block size to 1 MB and increase the cache size of almost all DBQL tables.

The DBQLParamTbl cache size is controlled separately by DBS Control field 30, DBQLLOBCacheSize. DBQLParamTbl is populated when you begin query logging with the PARAMINFO option.

4. Restart.

Changing the DBQL Performance Stats Cache Size

AMPs use a DBQL cache called DBQL Performance Stats (DPS) cache. This cache holds DBQL request and step data for active requests.

The DPS cache size is controlled by the DBQL_AWTDPS_CacheMaximum field 37 in the DBS Control Performance section. DBQL_AWTDPS_CacheMaximum limits the size to which the per-AMP AMP Worker Tasks (AWT) DPS cache can grow. The default size of the DPS cache is 8192 KB.

The cache size range is 1024 KB - 16384 KB. The cache holds approximately 2,500 entries per 1024 KB. When the cache is full, DBQL AMP data is not captured. Space in the cache is dynamically reused, but the memory for the cache is not released until the database is restarted. The recommended range is 4096 KB - 10240 KB.

Generally you do not have to adjust these values because the default value of 8192 is optimal for DBQL data collection to handle all types of workload.

For more information, see DBQL_AWTDPS_CacheMaximum in *Teradata Vantage™ - Database Utilities*, B035-1102.

Perform the following steps to change the DPS cache size:

1. For example, increase the DPS cache size to 10240 KB. Modify Performance (p) field 37:

```
dbcontrol m p 37 = 10240
```

2. Write the DBS control record for the change to take effect:

```
write
```

Logging Scenarios

The following tables offer examples of the type of data that is logged according to the rules you defined and the behavior of the query.

IF you...	THEN ...
try to drop a user or database with DBQL enabled	an error 5780 occurs. You cannot drop a user or database that has DBQL logging enabled. First disable logging for that user or database.
log on as the authorized user DBC (or your DBA user) and attempt to DROP, UPDATE or ALTER any DBQL object	the statement fails with an access error: Failure 3523 (<username> does not have <drop update alter> access to <DBQL table name>)
log on as an unauthorized user (not DBC or an administrative user) and submit a BEGIN QUERY LOGGING or REPLACE QUERY LOGGING statement	the BEGIN QUERY LOGGING or REPLACE QUERY LOGGING statement fails with an access error: Failure 3523 (username does not have statement permission)
disable query logging for a user running a session that is being logged	no more rows are cached for that session or that user after the running query completes.
abort a session that is being logged	the AbortFlag value is set to T in the DBQLLogTbl row for the query.
begin query logging for a user and that user runs the same query multiple times during one session	multiple rows are logged in DBQLLogTbl. If the query is in the steps cache when it is executed after the first execution, the CacheFlag is set to T. The second time a query is run, it goes into the steps cache. The third and subsequent times, the CacheFlag is set to T.
want to view all logged rows for a query	use the QueryID field to join DBQLLogTbl rows with (depending on the rules for the user) other DBQL table rows.
want to forbid logging for any MULTILOAD job	submit the following statement: BEGIN QUERY LOGGING WITH NONE ON APPLNAME= 'MULTLOAD';

IF you...	THEN ...
	Note: Valid application names are drawn from the pool of names returned in the LogonSource string, and for MultiLoad, it returns MULTLOAD (no l). For a list of the commonly seen LogonSource strings, see “Application Names” in <i>Teradata Vantage™ - SQL Data Definition Language Detailed Topics</i> , B035-1184.
want to start logging queries for a user who already has a WITH NONE rule in place	submit REPLACE QUERY LOGGING. REPLACE QUERY LOGGING WITH STEPINFO ON user5 ACCOUNT = 'abc';
want to find out if statement errors occurred in the successful multistatement or iterated request	check the ErrorCode column of the DBQL log table. If the ErrorCode column is populated with error 9259, this means that the query succeeded, but contains statement errors.

Scenarios of Logging Accounts

IF you...	THEN ...
submit one of the following statements: BEGIN QUERY LOGGING ON ALL; Or REPLACE QUERY LOGGING ON ALL;	the system creates one rule for the user named “ALL.” If the DBQL rule for ALL is in the system, each query is logged for every user that logs on.
submit one of the following statements: BEGIN QUERY LOGGING ON ALL ACCOUNT = 'ABC'; Or REPLACE QUERY LOGGING ON ALL ACCOUNT = 'ABC';	any user that logs on with the account string 'ABC' will be logged.
begin query logging for a specific account for a user, but the account does not exist	the BEGIN QUERY LOGGING statement is accepted (accounts do not need to exist). DBQLRules[V] shows a rule for the user, but queries run by that user are not logged because the sessions never match the user/account/application triplet.
begin query logging for a specific account (define a rule) for a user, and that user logs on under an account that does not match the rule	no rows are logged for any queries run during that session.
want to log SQL for all users <i>except</i> specifically when user2 logs on under account 'ABC'	submit the following statements: BEGIN QUERY LOGGING WITH SQL ON ALL; BEGIN QUERY LOGGING WITH NONE ON myuser2 ACCOUNT='abc'; If user2 issues a query under account ABC, the system will not log the query. However, if user2 logs on with another account such as DEF, the system will log the

IF you...	THEN ...
	<p>query and DBQLSQLTbl will show the SQL for user2 /DEF queries.</p> <p>Note: To stop these logging rules, submit the END QUERY statement for each rule. The rule for user2 must be specifically removed with a separate END QUERY LOGGING statement.</p> <pre>END QUERY LOGGING ON ALL; END QUERY LOGGING ON myuser2 ACCOUNT='ABC';</pre>
begin query logging for a specific account for a user, and the account includes account string expansion (ASE) codes	<p>both the input account string and the expanded account string are logged. For details on ASE codes, see Logging Resource Usage Data with Account String Variables.</p> <p>The system automatically strips away leading blanks and converts all letters in an account string to upper case before storing it in DBC.DBQLRuleTbl.</p>
submit the following: BEGIN QUERY LOGGING WITH OBJECTS ON ALL ACCOUNT='WebQry&D&H';	A row in DBQLObjTbl and DBQLLogTbl for each object for each query during each 'WebQry&D&H' session.
submit the following: BEGIN QUERY LOGGING WITH STEPINFO ON ALL ACCOUNT='\$HTactQry&D&H';	A row in DBQLStepTbl and DBQLLogTbl for each step of each query during each '\$HTactQry&D&H' session.
submit the following: BEGIN QUERY LOGGING WITH SQL, STEPINFO, OBJECTS LIMIT THRESHOLD=3 ON ALL ACCOUNT='\$LStratQry&D&H';	<p>For each '\$LStratQry&D&H' session:</p> <ul style="list-style-type: none"> One row of count data in DBQLSummaryTbl for all queries that completed in less than three seconds (within the 10-minute logging intervals) For each query that ran longer than three seconds, one row of data is logged in DBQLSQLTbl, DBQLStepTbl, DBQLObjTbl, and DBQLLogTbl

Note:

If you define more than one account string in your SQL statement, use a comma to separate the delimited list and enclose the account strings in parentheses. See *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144 for more information on syntax.

Scenarios of Detailed Logging

IF you...	THEN ...
Objects	
define a rule for a user specifying WITH OBJECTS	if the user runs a SELECT that joins two tables owned by the same database:

IF you...	THEN ...
	<ul style="list-style-type: none"> One row for the query is logged in DBQLogTbl Rows are logged in DBQLObjTbl as follows: <ul style="list-style-type: none"> A row for the database One row for each table One row for each accessed column, based on the Optimizer plan for the query
define a rule with WITH OBJECTS and the user runs a query that causes the Optimizer to reference the same object twice	<ul style="list-style-type: none"> One row for the object is logged in DBQLObjTbl. The value in the FreqofUse field is incremented to 2.
SQL	
begin query logging with no options for a user, and that user subsequently logs on and runs a query	a default row is logged for that user in DBQLogTbl with 200 characters of the SQL.
begin logging for a user with no options and the user runs a query with more than 200 characters	a row is logged in DBQLogTbl that includes the first 200 characters of the query.
create a rule for a user that specifies LIMIT SQLTEXT= 0 and the user runs a query	<p>logging depends on the following:</p> <ul style="list-style-type: none"> If you also specified the WITH ALL or WITH SQL option, then the SQL text is logged only to DBQLSQLTbl. If you did not also specify WITH ALL or WITH SQL, then no SQL characters are logged.
define a rule specifying LIMIT SQLTEXT=10000 and the user runs a query containing 15000 characters	a row is logged in DBQLogTbl that includes the first 10,000 SQL characters.
create a rule specifying LIMIT SQLTEXT=32000 (or anything larger than 10,000) and the user runs a query comprising >31000 SQL characters	a row is logged in DBQLogTbl that includes the first 10,000 SQL characters.
define a rule with just the WITH SQL option	The first 200 characters of the SQL statement are logged in DBQLogTbl and the entire statement is logged in as many rows as required in DBQLSQLTbl.
define a rule with both the WITH SQL option and LIMIT SQLTEXT=1000	The first 1,000 characters of the SQL statement are logged in DBQLogTbl and the entire statement is logged in as many rows as required in DBQLSQLTbl.
define a rule with the WITH SQL option and LIMIT SQLTEXT=0	None of the SQL characters are saved in DBQLogTbl. The entire statement is logged in as many rows as required in DBQLSQLTbl.
Steps	
define a rule with WITH STEPINFO and the user runs a query that does not generate parallel steps	One row is logged in DBQLStepTbl for each step used to resolve the query. In each row, the value of StepLev2Num is 0.

IF you...	THEN ...
define a rule with WITH STEPINFO and the user runs a query that generates parallel steps	One row is logged in DBQLStepTbl for each step used to resolve the query and each parallel step is differentiated by the step level number in StepLev2Num.

Scenarios of Threshold and Summary Options

IF you...	THEN ...
create rules for a user and specify LIMIT SUMMARY=5,10,15 and during the next session, and every query takes longer than 5 seconds but less than 10 seconds to complete	all queries fall into the second bucket (5 to 10 seconds), so the second group is the only query count logged for the session in DBQLSummaryTbl.
create rules for a user and specify LIMIT SUMMARY = 5,10,15 and during the next session, every query completes in less than 5 seconds	all queries fall into the first bucket (up to 5 seconds), so the first group is the only query count logged for the session in DBQLSummaryTbl.
create rules and specify LIMIT SUMMARY=1, 15,10	the statement is accepted (no checking is performed on SUMMARY input values) but the results are unpredictable.
create rules for UserA and specify SQL, OBJECTS LIMIT THRESHOLD (without a time value), and UserA then processes four queries, where: <ul style="list-style-type: none"> One query takes more than 5 seconds to complete Three queries complete in less than 5 seconds 	the statement is accepted and the default value of 5 seconds is assigned. For the next session of UserA: <ul style="list-style-type: none"> The longer query is logged in DBQLSqlTbl, DBQLObjTbl, and DBQLLogTbl, with values in all valid fields of the rows. For each of the three shorter queries: <ul style="list-style-type: none"> No entries are made in DBQLSqlTbl, DBQLObjTbl, and DBQLLogTbl. DBQLSummaryTbl will have a row with these values: <pre>COUNT = 3 SECONDS = 10 LOWHIST = 5 HIGHHIST = 0</pre>
create a rule and specify LIMIT THRESHOLD=100000	An error is returned; THRESHOLD must be less than 32K.

Example of OBJECT Data for One Query

The following example illustrates the rows in DBQLObjTbl resulting from a query logged with the WITH OBJECT option:

ObjDBName	ObjTblName	ObjColName	ObjID	ObjNum	ObjType	FreqofUse
-----	-----	-----	-----	-----	-----	-----
D_PERSONNEL	?	?	00001604	0	DB	1

D_PERSONNEL	DEPARTMENT	?	00009005	0	Tab	1
D_PERSONNEL	DEPARTMENT	DeptNo	00009005	1,025	Col	2
D_PERSONNEL	DEPARTMENT	DeptName	00009005	1,026	Col	1
D_PERSONNEL	DEPARTMENT	EmpCount	00009005	1,027	Col	1
D_PERSONNEL	DEPARTMENT	Loc	00009005	1,028	Col	1

Example of STEP Data for One Query

The following example illustrates the rows in DBQLStepTbl resulting from a query logged with the WITH STEPINFO option:

StepLev1	StepLev2	StepName	StepStartTime	StepStopTime	RowCount	
1	0	MLK	2004-07-08 20:37:22.770000	2004-07-08 20:37:23.770000	1	1
2	0	MLK	2004-07-08 20:37:23.770000	2004-07-08 20:37:23.780000	0	0
3	0	MLK	2004-07-08 20:37:23.780000	2004-07-08 20:37:23.790000	1	1
4	1	SAT	2004-07-08 20:37:23.790000	2004-07-08 20:37:23.790000	0	0
4	2	SAT	2004-07-08 20:37:23.790000	2004-07-08 20:37:23.790000	0	0
4	3	INS	2004-07-08 20:37:23.800000	2004-07-08 20:37:23.800000	1	1
4	4	INS	2004-07-08 20:37:23.800000	2004-07-08 20:37:23.810000	1	1
4	5	INS	2004-07-08 20:37:23.820000	2004-07-08 20:37:24.830000	1	1
4	6	INS	2004-07-08 20:37:23.830000	2004-07-08 20:37:24.840000	1	1
4	7	CTRts	2004-07-08 20:37:24.110000	2004-07-08 20:37:25.060000	1	1
5	0	Ctb	2004-07-08 20:37:25.080000	2004-07-08 20:37:25.100000	1	1
6	0	Edt	2004-07-08 20:37:25.120000	2004-07-08 20:37:25.130000	1	1

Examining the Logged Data

First-level information is captured in DBQLLogTbl and DBQLSummaryTbl. For short-running, high-volume queries, you can request the THRESHOLD option in combination with the SQL, STEPINFO, or OBJECTS logging option to reduce collection overhead. You can further specify the units of elapsed time, CPU usage, or number of I/Os.

Use the data from DBQLLogTbl and DBQLSummaryTbl to identify issues such as workloads that do not meet response time service-level agreements. If space is a major concern, you can use threshold logging, but having the detail can provide much more insight.

Best practices for DBQL logging dictate that you log at the system level with the SQL and OBJECTS logging options. If there are specific account strings, or user IDs that only do millions of tactical, subsecond requests, Teradata recommends that you use the THRESHOLD logging option with the STEPINFO logging option to log CPU usage.

When used selectively, detailed data can be invaluable. Analyze it to:

- Optimize queries or your query management or priority strategy; for example, compare:
 - Results of different queries that target the same join index
 - Elapsed times of the same query run on different days or hours

- Determine reasons for high consumption or poor performance by correlating DBQL data to other data collections with query characterization, including QCF, ResUsage, and DBC.AMPUsage
- Make efficient use of existing capacity and plan for future expansion by factoring in exponential growth of query demand

Comparing CollectTimeStamp Value Between Tables

For all the DBQL tables, CollectTimeStamp is captured at different points in the query processing cycle. Do not use the CollectTimeStamp field value to compare tables for this reason.

Instead, select one or more queries based on the DBQLLogTbl.StartTime field or DBQLLogTbl.FirstRespTime and then join that row with the other tables based on the QueryID field.

Note:

Depending on when the DBQL caches are written, all the rows for a specific QueryID may not be flushed to the disk at the same time. This is also true when you load DBQL rows off DBC space and onto temporary, history, or archive database. Some rows for a specific QueryID may be moved to your database while some will subsequently be logged for the DBC DBQL tables.

Using QueryID for Joins on DBQL Tables

Because each DBQL table has its own separate cache, the system may fill up the cache for one table and log to that DBQL table before it logs into another DBQL table. This results in slightly different CollectTimeStamp values. Therefore, do not include CollectTimeStamp fields in joins.

If you need to join different DBQL tables, use the QueryID field. If you plan on comparing values between different DBQL tables, you can flush the DBQL caches to the dictionary tables. See [Options for Flushing the DBQL Cache](#).

Note:

Although it is rare, you may see orphan rows in join results involving the DBQLObjTbl, DBQLExplainTbl, or DBQLXMLTbl tables. An orphan row is one that does not appear in the default log table (DBQLLogTbl), but has an entry in one of the three other tables.

Teradata recommends that you ignore the orphan rows and do not copy them over when you copy the DBQL tables out of DBC and into temporary or history tables in another database.

Zone-level Query Logging

Teradata Secure Zones allow you to create one or more exclusive database hierarchies, called zones, within a single database system. Access to the data in each zone and zone administration is handled separately from the database system and from other zones. Each zone can have a DBA for that zone.

Teradata Secure Zones are useful in situations where the access to data must be tightly controlled and restricted. You can also use Teradata Secure Zones to support some regulatory compliance requirements for the separation of data access from database administration duties. Other uses include separating and securing data by corporate division or by cloud storage tenant.

NOTICE

Sites that use Teradata Secure Zones should revoke SELECT privileges granted directly on DBC tables and grant access to DBC tables through views only. This prevents users from accessing information not related to their zones.

Query Logging Privileges in a Zoned System

Zone-level query logging allows zone DBAs to monitor user queries in their zones. Zone members can enable query logging only on objects within their zones. Query logging privileges are as follows:

User Type	BEGIN/REPLACE/SHOW/END Query Logging Allowed
<ul style="list-style-type: none"> User DBC Non-zone user with the ZoneBypass privilege and the EXECUTE privilege on DBC.DBQLAccessMacro 	System-level. These users can see and act on zone and non-zone objects. These users cannot end zone-level logging created by zone users.
Zone user with the EXECUTE privilege on DBC.DBQLAccessMacro	Zone level. This user can see and act only on objects in his zone. If this user issues BEGIN QUERY LOGGING ON ALL, only objects in the zone of this user are logged.
Non-zone user with the EXECUTE privilege on DBC.DBQLAccessMacro	Non-zone-level. This user can see and act only on non-zone objects.

Related Information

For more information on Teradata Secure Zones, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

Maintaining the Logs

DBQL log maintenance is automated. To set up DBQL log maintenance, do the following things:

1. Run the DIP script DIPPCR.
This creates DBQL (and other system performance) history tables and the data extraction macros that populate the tables.
2. Use the Viewpoint portlet **Performance Data Collection** to set up the **DBQL** job and run it every day. This job moves data from the DBQL log tables to the history tables created by DIPPCR. You can also use the portlet to specify how long the history tables are retained.

3. Create and maintain an executable BTEQ script file to keep your final BEGIN QUERY LOGGING or REPLACE QUERY LOGGING statements.

Do this in case a system initialization is required in the future for disaster recovery or to migrate to a new platform. After the database has been restored, you can start a BTEQ session and run the script to easily repopulate DBQLRuleTbl and rebuild your rules cache.

Methods for Minimizing Log Table Size

There are several ways you can minimize the size of your DBQL log tables and therefore avoid running out of disk space in DBC:

- Add a date column to avoid multiple casting of the time columns.
- Use MVC on the columns of the copies of DBQL tables.
- Log only the data you need. For example, if default data in DBQLLogTbl is sufficient, do not log WITH SQL or WITH OBJECTS as well.
- Limit summary logging to just the accounts you want to track.
- Limit SQL text logging to a specific number of characters.
- Remember to end logging whenever logging data is no longer needed for a specific user, group of users, account, list of accounts, or applications.

DBQL Objects You Cannot Change

Most of DBQL is user-accessible, except for the following:

- No user, including DBC and SystemFE, can access or change the DBQLAccessMacro macro, DBQLRuleTbl, or DBQLRuleCountTbl.
- No user can alter, drop, or update DBQL table definitions.

Archiving DBQL Data

If your DBQL table archives are defined as PPI tables, do the following:

1. Archive DBQL data nightly. This is strongly recommended so that the DBQL logs do not take up space in DBC and impact the system.
2. Add a log date to each row. The log date is used as the partitioning column and makes querying the archived data much faster.
3. Change the primary indexes. This is done to avoid table skewing.

You can also separate historical data from junk queries to make your DBQL table archives more concise.

Reviewing or Ending Current Rules

The following sections describe how to review rules and stop DBQL logging.

Reviewing Rules

You can review DBQL rules one of two ways:

- Use the SHOW QUERY LOGGING statement to determine which rule would be used for a specified user/account/application. For example:

```
SHOW QUERY LOGGING ON marketinguser;
```

- Use the DBC.DBQLRules[V] view as a window into the contents of the DBC.RuleTbl which stores all DBQL rules in the system.

The following table describes what kind of rules the system checks for depending on the SHOW QUERY LOGGING statement you submit. If you only list a user name or an account string, the SHOW QUERY LOGGING statement shows the Best Fit Rule.

Note:

Only a user with EXECUTE privilege on DBC.DBQLAccessMacro can execute the SHOW QUERY LOGGING statement.

IF you submit a SHOW QUERY LOGGING statement for...	THEN the system will show...
ALL	the first rule in the hierarchy that applies, looking only at all/all accounts
user1 acct1	the first rule in the hierarchy that applies, looking at: <ul style="list-style-type: none"> • user1/account1 • user1/all accounts • all/account1 • all/all accounts
user2	the first rule in the hierarchy that applies, looking only at: <ul style="list-style-type: none"> • user2/all accounts • all/all accounts
all acct3	the rule hierarchy (in hierarchy order) that would apply for any user that logged on under acct 3. The system will look only at: <ul style="list-style-type: none"> • all/account3 • all/all accounts
APPLNAME='xxx'	APPLNAME='xxx' which is the only APPLNAME rule that applies.

The DBC.DBQLRules[V] view provides a window into the contents of the DBQLRuleTbl table and shows which rules would be applied by the system. Only a user with SELECT privilege on DBC.DBQLRules[V] can access the view. For Teradata systems configured with Teradata Secure Zones, access to some Data Dictionary views is constrained by zone. DBQLRules[V] is constrained by zone. For more information, see *Teradata Vantage™ - Data Dictionary*, B035-1092.

A SELECT on the DBQLRules[V] view displays the rules currently in effect. You can qualify the response by user name, or account string (if it exists), or both. For example:

```
SELECT * FROM DBC.DBQLRULESV WHERE ACCOUNTSTRING='$L00Test&D&H';
```

Ending DBQL Logging for Specific Users and/or Accounts

When you are ready to stop logging, specify any accounts on which you enabled DBQL logging. For example, if you submitted the following statement:

```
BEGIN QUERY LOGGING WITH OBJECTS ON APPLNAME='FASTLOAD';
```

to stop logging, use:

```
END QUERY LOGGING ON APPLNAME='FASTLOAD';
```

Note:

Support will be removed for the WITH option of the END QUERY LOGGING statement. Use of the option results in the following message: "Warning: Deprecated option WITH is used and will soon be removed."

Teradata recommends using scripts to enable and disable logging to avoid typographical errors.

Note:

The application names you specify with the APPLNAME option are the names the system passes in the LogonSource string. (For a list of the commonly seen LogonSource string application names, see "Application Names" in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.)

Ending All DBQL Logging

Use the END QUERY LOGGING ON ALL RULES statement to eliminate all rules currently in the DBQLRuleTbl table using the ON ALL RULES option. For example:

```
END QUERY LOGGING ON ALL RULES;
```

The system will:

- Delete rules from the DBQLRuleTbl.
- Set the DBQLRuleCountTbl.NumberOfRules field to 0.
- Flush all rules caches.

Effects of Dynamically Enabling/Replacing/Disabling Logging on Current Rules

When you enable, replace, or disable query logging, the currently running query is not affected, but all subsequent queries take on new rule changes.

The following table describes the DBQL behavior as a result of a change to an active session.

IF you...	AND a query for that session is already...	THEN ...
enable logging (submit a BEGIN QUERY LOGGING statement) for an active session	in process	<ul style="list-style-type: none"> Data for the current query is not collected. Logging begins with receipt of the next query.
replace an existing rule (submit a REPLACE QUERY LOGGING statement)	in process	logging begins with receipt of the next query.
abort a session that is being logged	cached	if a default row is being logged (logging was not just SUMMARY), the AbortFlag is set to T.
disable logging (submit an END QUERY LOGGING statement) for an active session	cached	<ul style="list-style-type: none"> One or more DBQL rows are written (but may be incomplete). The current query will be logged (perhaps in cache if it is in flight). All DBQL caches are flushed. Subsequent queries during that session are not logged. All rules caches are flushed.
change a logon account of a user while the user is submitting queries that are under DBQL logging	in process	whether or not the DBQL logging rule can be applied on the session is re-evaluated on the next query.

Granting DBQL Administrative Privileges to Other Users

You need the EXECUTE privilege on the special macro DBQLAccessMacro to enable and disable query logging. DBQLAccessMacro is created by DIPVIEWSV of the DIP utility.

The system users DBC and SystemFE have the EXECUTE privilege on DBQLAccessMacro and can grant it to others when needed. If you want other users, such as your administrative user, to be able to execute DBQLAccessMacro, follow this procedure:

1. Log on as user DBC (or SystemFE).

2. List the contents of database DBC to see if the DBQLAccessMacro, DBQLRuleTbl, and DBQLRuleCountTbl have been created:

```
HELP DATABASE DBC ;
```

The DBQL tables, views, and macros should be reported.

3. Grant the following privileges to your database administrator user:

```
GRANT EXECUTE ON DBC.DBQLAccessMacro TO DBADMIN ;  
GRANT SELECT ON DBC.DBQLRULESV TO DBADMIN ;
```

4. Log off the DBC or SystemFE session.
5. Log on again as user DBADMIN.
6. Define query logging rules for one or more users, one or more accounts, or applications using BEGIN QUERY LOGGING or REPLACE QUERY LOGGING statements. (For full syntax, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.)
7. Check the DBQLRules[V] view to see if the rules are correct:

```
SELECT * FROM DBC.DBQLRulesV ;
```

If you find an error, submit an REPLACE QUERY LOGGING statement for that user.

Analyzing Requests and Request Plans: Application DBAs

This section describes how to analyze request plans using the EXPLAIN request modifier and the Query Capture Facility.

Using EXPLAIN to Analyze Request Plans

EXPLAIN Request Modifier

The EXPLAIN request modifier reports an English language summary of the request plan generated by the Optimizer to process any valid SQL request. The summary describes the AMP steps the system would perform to resolve the request, but the request is not processed unless the Optimizer uses a dynamic plan and then the request is partially processed.

Note:

To EXPLAIN a request, you must have the privileges required to execute the request.

EXPLAIN is useful for evaluating and optimizing SQL requests. It provides a summary of the access and join plans generated by the Optimizer for the specified SQL request, including information such as:

- The indexes to be used
- Any intermediate spool files that will be generated
- The join types to be performed
- Whether the requests in a transaction will be dispatched in parallel
- The relative time it will take to complete the request
- Whether a dynamic or static request plan will be used

Using EXPLAIN allows you to compare the relative performance of the different versions of a request and choose the most efficient SQL.

Recommendation: Always use EXPLAIN to analyze any new requests under development. Subtle differences in the way a request is structured can produce significant differences in its resource impact and performance, even though the requests may produce identical end results.

Generating an EXPLAIN with Teradata Studio

1. In the **SQL Editor** window, enter the SQL request to be EXPLAINed.
2. Type the word EXPLAIN before the request and execute the request.

Note:

For a dynamic plan, use DYNAMIC EXPLAIN *request*.

Generating an EXPLAIN with BTEQ

To generate an EXPLAIN for an SQL request, add the word “EXPLAIN” at the beginning of the request.

For example, the following SQL performs an EXPLAIN on a SELECT statement:

```
EXPLAIN SELECT * FROM Employee;
```

If you want a dynamic request plan instead, issue DYNAMIC EXPLAIN *request*. For example,

```
DYNAMIC EXPLAIN SELECT * FROM Employee;
```

Related Information

Topic	Resources for Further Information
Syntax and options for the EXPLAIN request modifier	<i>Teradata Vantage™ - SQL Data Manipulation Language</i> , B035-1146
Interpreting the output of the EXPLAIN request modifier for various request types	<i>Teradata Vantage™ - SQL Request and Transaction Processing</i> , B035-1142

Capturing Request Plan Steps

Query Capture Facility

SQL provides a powerful facility for analyzing the request plans the Optimizer creates for various data manipulation statements. This feature, called the Query Capture Facility, permits a DBA to capture access plan information from the Optimizer using the SQL statements COLLECT DEMOGRAPHICS, DUMP EXPLAIN, INSERT EXPLAIN, and BEGIN QUERY CAPTURE (see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144 for more information about these statements) and then analyze the captured data using standard SQL DML statements.

The captured request plans are stored in a relational database called a Query Capture Database (QCD) that you can create for that purpose. QCD is a user database whose space is drawn from available permanent disk space: it is not a set of system tables found in a fixed location.

Applications of QCF include:

- Store all request plans for customer requests. You can then compare and contrast requests as a function of software release, hardware platform, and hardware configuration.
- Provide data so that you can generate your own detailed analyses of captured request steps using standard SQL DML statements and third-party request management tools.

Capturing Request Plans Using DBQL

An alternative to using the Query Capture Facility is to capture request plans using the request `BEGIN QUERY LOGGING WITH XMLPLAN`. This request causes the request plan for every non-cached `INSERT`, `UPDATE`, `SELECT`, `DELETE`, `MERGE`, and `EXEC` statement to be logged as an XML document in `DBC.DBQLXMLTbl`. For details, see [WITH Logging Options](#).

Relationship Between `BEGIN QUERY CAPTURE` and `BEGIN QUERY LOGGING`

`BEGIN QUERY CAPTURE` requests always take precedence over `BEGIN QUERY LOGGING` requests. If you submit a `BEGIN QUERY LOGGING` request when there is already an active `BEGIN QUERY CAPTURE` request on the same user, the query logging takes place for the user in all the sessions except for the `CAPTURE` session and the query capture session continues capturing queries.

This is true whether the `BEGIN QUERY LOGGING` statement was issued in the capture session or from some other session. The opposite is true when you submit a `BEGIN QUERY CAPTURE` request when there is already an active `BEGIN QUERY LOGGING` request on the same user. The query capture takes place for the capture session and the query logging for the user continues for all the other sessions.

DBS Control Parameters for Tuning the Workload Cache Limits for Index and Partition Analysis

Index processing allocates workload cache for the purpose of caching information retrieved from the QCD and other temporary working structures and needed during Analysis and Validation phases of recommending indexes and partitioning expressions for various SQL workloads.

Two DBS Control performance group parameters control the respective workload cache sizes for the Analysis and Validation phases:

- `IAMaxWorkloadCache`
- `IVMaxWorkloadCache`

The following table provides some of the details for these parameters. See *Teradata Vantage™ - Database Utilities*, B035-1102 for further information.

Parameter	Purpose	Valid Range (megabytes)	Default (megabytes)
<code>IAMaxWorkloadCache</code>	Defines the maximum size of the workload cache for Analysis operations. This parameter is applicable to both the <code>INITIATE INDEX ANALYSIS</code> and <code>INITIATE PARTITION ANALYSIS</code> statements.	32 - 256	8

Parameter	Purpose	Valid Range (megabytes)	Default (megabytes)
IVMaxWorkloadCache	Defines the maximum size of the workload cache for Validation operations. Validation is only performed under the direction of Teradata Support Center personnel.	1 - 32	48

Query Capture Database

A query capture database, or QCD, stores the Optimizer query analysis workload data and the statistical and demographic data needed to support query analysis.

The SystemFE DIP creates a default QCD called *TDQCD* beneath the SystemFE user, but you can also create your own QCD databases, and you can define multiple QCD databases for your systems.

You must be granted the INSERT privilege to use a BEGIN QUERY CAPTURE request to capture query plans in XML format at the session level and write data into *TDQCD* or any other user-defined QCD.

Vantage determines the size of *TDQCD* based on the number of system AMPs using the following equation.

$TDQCD_size = 50MB + (number_of_system_AMPs \times 100 \times 1,024 \text{ bytes})$

You can use the *modify_tdqcd.spl* SQL stored procedure to modify the size of the *TDQCD* database based on the number of AMPs in a test system. The definition of *modify_tdqcd.spl* is as follows.

```

REPLACE PROCEDURE TDQCD.MODIFY_TDQCD(IN Basesize int, OUT Success int)
BEGIN
  DECLARE TotalAmps INTEGER;
  DECLARE CurrentSize INTEGER;
  DECLARE TargetSize INTEGER;

  SELECT COUNT(DISTINCT Vproc) FROM Dbc.DiskSpaceV INTO TotalAmps ;
  select ZEROIFNULL(permspace) into :CurrentSize from Dbc.DatabasesV where
  databasename = 'tdqcd';
  SET TargetSize = Basesize + (TotalAmps * 100 * 1024);

  IF (CurrentSize < TargetSize) then
  CALL dbc.sysexecsql('MODIFY DATABASE TDQCD AS PERM = ' || :TargetSize || ' BYTES');
  END IF;

  SET Success = 1;

END;
```


Use BTEQ using the CreateQCF table (see [Procedure Using BTEQ](#)) to create your query capture databases. QCD tables are populated using various methods, including the following:

- BEGIN QUERY CAPTURE
- COLLECT DEMOGRAPHICS
- COLLECT STATISTICS (QCD Form)
- INSERT EXPLAIN WITH STATISTICS
- BEGIN QUERY CAPTURE

NOTICE

Because the QCD is often a set of user tables, you can change both their individual structure and the structure of the QCD; however, you should *never* change *any* of the QCD structures in any way.

The reason for this constraint is that the SQL statements BEGIN QUERY CAPTURE, DUMP EXPLAIN, INSERT EXPLAIN, INITIATE INDEX ANALYSIS, RESTART INDEX ANALYSIS, COLLECT DEMOGRAPHICS, and COLLECT STATISTICS (QCD Form) all assume the default physical model when they capture the specified query, statistical, and demographic information for insertion into the QCD tables.

The results of changing the structure of the QCD tables are unpredictable.

Creating the Query Capture Database Tables

Creating QCD Tables

Before you can capture request plan information from the Optimizer white tree and statistical and data demographic information from their respective data structures, you must either create and secure the QCD tables or create your own user-defined query capture database. This topic explains two ways to set up your QCDs.

Procedure Using BTEQ

Perform the following procedure to create the tables for the query capture database using BTEQ.

The procedure assumes that you have already created *QCD_database_name*.

1. Start BTEQ.
2. Change your current database or user to the database in which the QCD tables are to be created as follows.

```
DATABASE QCF_database_name;
```

where *QCF_database_name* is the name of the database you created for the QCD tables.

3. Perform the following steps in the indicated order.

```
a. .SET WIDTH 254
```

```
b. .EXPORT FILE = file_name
```


- c. `SELECT TabDefinition`
`FROM systemfe.CreateQCF`
`ORDER BY SeqNumber;`
- d. `.EXPORT FILE = file_name`
- e. `.RUN FILE = file_name`

where *file_name* is the name of the file you create to contain the output of the SELECT request.

4. Secure QCD by granting the appropriate privileges to the users who will be analyzing its data.

To use the BEGIN QUERY CAPTURE request to insert data into the QCD tables in *TDQCD*, you must have the ALL privilege.

See *Teradata Vantage™ - SQL Data Control Language*, B035-1149 for information about how to implement column-level and row-level security for your tables.

You can implement additional secured access by creating views on QCD.

5. Populate the QCD tables using the appropriate tools.
 - Use BEGIN QUERY CAPTURE requests to capture query plans. For the syntax of this statement, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.
 - Use INSERT EXPLAIN requests to capture request plans. For the syntax of this statement, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.
 - Use the following statements to capture statistical and demographic data.
 - `COLLECT DEMOGRAPHICS`
 - `COLLECT STATISTICS (QCD Form)`
 - `INSERT EXPLAIN WITH STATISTICS`
 - Use the appropriate workload macros to create workloads for index analysis.

Dropping Query Capture Database Tables

Dropping QCD Tables

This topic describes how to drop all the tables and other database objects from a QCD database.

Procedure Using BTEQ

Perform the following procedure to drop the tables for the query capture database using BTEQ. The procedure assumes that you have already created *QCF_database_name*. This procedure does *not* drop the views and macros on the specified QCD.

1. Start BTEQ.
2. Change your current database to the database from which the QCF tables are to be dropped as follows.


```
DATABASE QCD_database_name;
```

where *QCD_database_name* is the name of the database you created for the QCD tables.

3. Perform the following steps in order.

- a. `.EXPORT FILE = file_name`
- b. `SELECT DelTable`
`FROM systemfe.CleanupQCF`
`ORDER BY SeqNumber;`
- c. `EXPORT RESET;`
- d. `.RUN FILE = file_name`

where *file_name* is the name of the file you create to contain the output of the SELECT request.

Querying QCD Tables

You can analyze the information in QCD using several different approaches.

The following methods are typical ways to use a QCD to analyze query data.

- To execute ad hoc requests of QCD, use interactive SQL DML statements.
- To perform standardized analyses of QCD, create macros, stored procedures, or embedded SQL applications to execute your standard requests and report the results.

QCD Request Macros and Views

Vantage provides a set of views on the QCD tables to restrict their access. Various levels of access to QCD tables are granted to different user categories.

Vantage also provides a set of macros for maintaining various aspects of the query capture database.

QCD Macro and View Versions

Two versions of all the views and macros are created in a QCD.

Version	Definition
X	Access restricted to information inserted by the current user in the QCD.
NonX	Access permitted to information inserted by any user of the QCD.

User Categories

The following categories of users are defined on a QCD to enhance its security.

User Category	Description
Normal	Loads, views, and deletes own plans or workloads only.
Power	Loads and views plans or workloads inserted by any user. Deletes own plans or workloads only.
Administrator	Loads, view, and deletes any plan created by any user. Drops and deletes QCD tables. QCD creator has Administrator privileges granted by default.

Specific User Category Privileges

The following table indicates the specific privileges granted to each user category.

User Category	Database Object Type	Privileges Granted
Normal	QCD tables	<ul style="list-style-type: none"> • INSERT • UPDATE • SELECT on the following tables. <ul style="list-style-type: none"> ◦ IndexRecommendations ◦ SeqNumber ◦ TableStatistics • DELETE on the AnalysisLog table
	X views	SELECT
	X macros	EXEC
	Non-X views	None
	Non-X macros	None
Power	QCD tables	<ul style="list-style-type: none"> • INSERT • UPDATE • SELECT on the following tables: <ul style="list-style-type: none"> ◦ IndexRecommendations ◦ SeqNumber ◦ TableStatistics • DELETE on the AnalysisLog table
	X views	SELECT
	X macros	EXEC
	Non-X views	SELECT
	Non-X macros	EXEC (excluding DELETE plan and workload macros)
Administrator	QCD tables	<ul style="list-style-type: none"> • DELETE

User Category	Database Object Type	Privileges Granted
		<ul style="list-style-type: none"> • DROP • INSERT • SELECT • UPDATE
	X views X views use the value for <i>UDB_Key</i> in the Query table to restrict access to request plans to the user who captures them, while the Non-X views do not restrict that access.	<ul style="list-style-type: none"> • DELETE • DROP • INSERT • SELECT • UPDATE
	X macros	<ul style="list-style-type: none"> • DROP • EXECUTE
	Non-X views	<ul style="list-style-type: none"> • DELETE • DROP • INSERT • SELECT • UPDATE
	Non-X macros	<ul style="list-style-type: none"> • DROP • EXECUTE

Building Baseline Transaction Profiles

Baseline profiles can provide information on typical resource usage by period and by user, on a daily, weekly, or monthly basis. Building baseline transaction profiles can help you track the effects of software upgrades, introductions of new users or new applications, and help you determine what a “healthy and normal” system should look like.

You can build baseline profiles for:

- Single operations (such as FastLoad, full table scans, primary index INSERT ... SELECTs, select joins, and so on.)
- Multiple, concurrently run transactions

Once defined and stored, baseline profiles can help you:

- Compare current to profiled operations on a real-time basis.
- Collect data instantaneously for a set interval.
- Detect and resolve throughput anomalies.

You can gather performance metrics for profiling from:

- ResUsage reports. See *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.
- DBQL. See [Tracking Query Behavior with Database Query Logging: Operational DBAs](#).
- Viewpoint.

Working with System Information and Global Defaults: Operational DBAs

This section provides an overview of Vantage configuration, the global default controls, the client connection software, and describes tools you can use to view or change system-wide (global) information.

Viewing the Software Release and Version

There are several methods for determining the current software release and version.

- From any console or any client session, you can view the currently running Analytics Database version and release level with the following query:

```
SELECT * FROM DBC.DBCInfoV;
```

Note:

You must have the SELECT privilege on the DBC.DBCInfoV view to use this query. Typically, this view grants SELECT access to PUBLIC by default.

The query returns the version and release level in the following form:

```
*** Query completed. 3 rows found. 2 columns returned.
*** Total elapsed time was 3 seconds.
```

InfoKey	InfoData
RELEASE	NN.nn.nn.nn
LANGUAGE SUPPORT MODE	Standard
VERSION	NN.nn.nn.nn

```
BTEQ -- Enter your DBC/SQL request or BTEQ command:
```

Where *NN.nn.nn.nn* is equal to the number of the current release. For example, 15.10.00.00.

- You can use the GET VERSION command in Supvr window of Database Window to view the current version of the PDE, DBS, GTW, TCHN, RSG, and TDGSS.

For more information on Database Window, see “Database Window (xdbw)” in *Teradata Vantage™ - Database Utilities*, B035-1102.

Reviewing or Changing System-Wide Parameters

The values for system-wide defaults are distributed across the configuration via the following:

- If your system is set for Universal hash code, the:
 - DBC.Hosts system table
 - tdlocaledef.txt file
- Globally Distributed Objects (GDOs), which store global configuration settings that are automatically propagated to all nodes of an MPP system. The system uses several GDOs internally, each storing configuration settings that are specific to different system functions or utilities. For example, there are GDOs to store the settings managed by the DBS Control, Cufconfig, Gtwcontrol, and Control GDO Editor utilities. GDOs are used to store configuration settings for Teradata dynamic workload management software and TASM, and many other Vantage functions. Use the gdomview command-line tool from the system console to view a history of the changes made to the GDO. You can access the gdomview online documentation from a system console using the 'man' and 'pdehelp' commands.

International Character Set Settings and Defaults

Vantage uses internal server character sets to represent character data stored within the database and supports many external client character sets.

For information on how to implement character sets, see *Teradata Vantage™ - Analytics Database International Character Set Support*, B035-1125.

International Language Support Mode

During installation, your database is set up with either Standard or Japanese language support mode enabled. All object names are stored in Unicode so that Data Dictionary field definitions across all Teradata platforms are consistent

To determine what language support mode your system is using, submit the following:

```
sel infodata from dbc.dbcinfoV where infokey = 'LANGUAGE SUPPORT MODE';
```

Default Server Character Sets

Whenever you define CHAR or VARCHAR data as part of defining a table, UDF, macro, or stored procedure and you do not specify a character set, the default is determined by the default character set for the user.

Server character sets include:

- LATIN
- UNICODE
- KANJI1
- KANJISJIS

Note:

In accordance with Teradata internationalization plans, KANJI1 support is deprecated and is to be discontinued in the near future. KANJI1 is not allowed as a default character set; the system changes the KANJI1 default character set to the UNICODE character set. Creation of new KANJI1 objects is highly restricted. Although many KANJI1 queries and applications may continue to operate, sites using KANJI1 should convert to another character set as soon as possible. For more information, see “KANJI1 Character Set” in *Teradata Vantage™ - Analytics Database International Character Set Support*, B035-1125.

Client Character Sets

Each client uses a particular character set during a session to transmit user data. The client character set maps each character in the client to an equivalent character in the server set.

Vantage translates data strings received from a client into the server, or internal, character set for storage and processing, and translates it back to the client character set when exporting response data to the client. This translation allows clients using different character sets to access and modify the same stored data.

Vantage offers many predefined client character sets (along with the codes for the appropriate collation sequences). However, you can also define and install your own character set translations and collations on the server and flag those currently desired as available for use. You can assign an active character set as the default for a client and also activate up to a maximum of 16 client character sets.

Note:

The ASCII, EBCDIC, UTF-8, and UTF-16 character sets are permanently installed and always available.

Viewing the Status of Client Character Sets

Use the following views to determine the current status of installed character sets.

View Name	Description
DBC. CharTranslationsV	<p>Displays the names of installed character sets. Each name identifies the set of translation tables needed to define the external-internal character mapping for one language. They can be predefined, or created by you.</p> <p>Note: Because they are permanently loaded and available, the ASCII, EBCDIC, UTF-8, and UTF-16 character sets are not reported.</p>
DBC.CharSetsV	<p>Displays the names of the character sets you flagged in the DBC.Translations table as the ones to activate during a tpareset. These are currently active and available to users at the session level only if a tpareset was performed after you set the flags.</p>

View Name	Description
	Note: A name in this view and CharTranslationsV does not prove that it is active. The InstallFlag column in CharTranslationsV is an indication, but can be misleading if the table was changed without a tpareset.
DBC.HostsInfoV	Displays the names of the character sets you assigned as the client defaults, by host.

Client character sets permanently enabled include:

- ASCII
- EBCDIC
- UTF-8
- UTF-16

For more information on mapping and collation codes, see *Teradata Vantage™ - Analytics Database International Character Set Support*, B035-1125. For IBM mainframe clients, see [TDP Functionality](#) and *Teradata® Director Program Reference*, B035-2416.

For more information on enabling Unicode pass through characters in Teradata, see *Teradata Vantage™ - Analytics Database International Character Set Support*, B035-1125 or [Loading Unicode Data with Unicode Pass Through](#).

For a complete description of the views described in the previous table, see *Teradata Vantage™ - Data Dictionary*, B035-1092.

Changing Character Set Defaults

You can change the default client character set and the default server character set at the user level, and the user can choose alternatives during a session.

Default Client Character Set

If you do not define a client character set as the default for a client in the DBC.Hosts table, the automatic default is the character set native to that client.

During a session, you can find out which client character set is in effect with the SQL HELP SESSION statement and can specify a different (but active) client character set in various ways, depending on the particular client software. For example, with the BTEQ client software, you can use the SESSION CHARSET command:

```
.SET SESSION CHARSET ASCII
```

For details on when the SESSION CHARSET command can be used, see *Basic Teradata® Query Reference*, B035-2414.

Default Server Character Set

To specify the character set for a user, issue a CREATE USER or MODIFY USER request with the DEFAULT CHARACTER SET clause. If you do not specify this clause when you create or modify a user, the setting of the Default Character Set field in the DBS Control utility determines the default character set.

The default value for the Default Character Set field is 0 (Unicode) for Japanese Language Support mode and 1 (Latin) for Standard Language Support mode.

NOTICE

If you change the character set for user DBC, user-defined functions may not work without being recompiled. You must also:

- Rerun DIPUDT and DIPDEM
- Recompile any site-defined DBC functions that use SQL_TEXT

Changing Collation Defaults

You can define the default collation sequence for a user with the COLLATION clause of the CREATE/ MODIFY USER statement. Vantage provides a number of small but flexible sets of collations to choose from. For example, CHARSET_COLL produces a binary ordering based on the current character set; MULTINATIONAL collation can be tailored as needed.

If you do not define a default collation for the user, the automatic default is HOST (the collation that is compatible with the logon client). HOST collation gives you EBCDIC collation on mainframe-attached clients and ASCII collation for all others.

During a session, the user can override the user or current session default with the SQL statement SET SESSION COLLATION. For example:

```
SET SESSION COLLATION JIS_COLL;
```

For a full explanation of collating conventions and instructions on how to define your own sequences, see *Teradata Vantage™ - Analytics Database International Character Set Support*, B035-1125.

Changing Time Zones, Currencies, and Formats for Date, Time, and Numbers

To define or change your time zone or how Vantage formats numeric, date, time, and currency output, use the command-line utility Teradata Locale Definition (tdlocaledef).

Tdlocaledef converts a specification for data formatting (SDF) text file into a Vantage globally distributed object (GDO), an internal binary format file that stores configuration information. The GDO is made available simultaneously to all nodes of an MPP system. The utility can also convert the text file to a local,

non-distributed, binary file, that can be converted back to text in order to ensure the formatting syntax is valid.

Note:

- Format changes take effect only after Vantage is restarted, and do not affect columns that were created prior to the restart.
- A new Teradata system is typically staged with global defaults in the SDF file that are appropriate for the United States. International customers and those who need customized settings, including TimeZoneString, can change the defaults using tdlcalledef.

For more details on the SDF file, see “SDF File” in the description of the Teradata Locale Definition (tdlcaldef) utility in *Teradata Vantage™ - Database Utilities*, B035-1102.

Changing Time Zone Strings and Rules

Note:

Teradata requests that customers open an incident so that Teradata Support can assist you in changing time zone settings on your system.

If you do not need Daylight Saving Time, you can set the time using the DBS Control utility settings SystemTimeZoneHour and SystemTimeZoneMinute.

To redefine your time zone, check the list of available time zone strings and their rules in /usr/tdbms/etc/tdlcaldef_tzrules.txt. These rules are also in the GetTimeZoneDisplacement UDF. To change a time zone string and rules, you must change both the GetTimeZoneDisplacement UDF and tdlcaldef.

For more information, see “Adding or Modifying Time Zone Strings” under the topic “GetTimeZoneDisplacement” in *Teradata Vantage™ - SQL Date and Time Functions and Expressions*, B035-1211 and DBS Control utility and Teradata Locale Definition utility in *Teradata Vantage™ - Database Utilities*, B035-1102.

Client Configuration Overview

There are two types of clients.

Type	Description
Mainframe-attached	Mainframe computing system (IBM z/OS only).
Workstation-attached	Windows, Novell Linux, Red Hat Linux, Oracle Solaris, and IBM AIX

A session is the transfer of data between a user on a client and Vantage on the server, including the logical and physical connections that make data transfer possible.

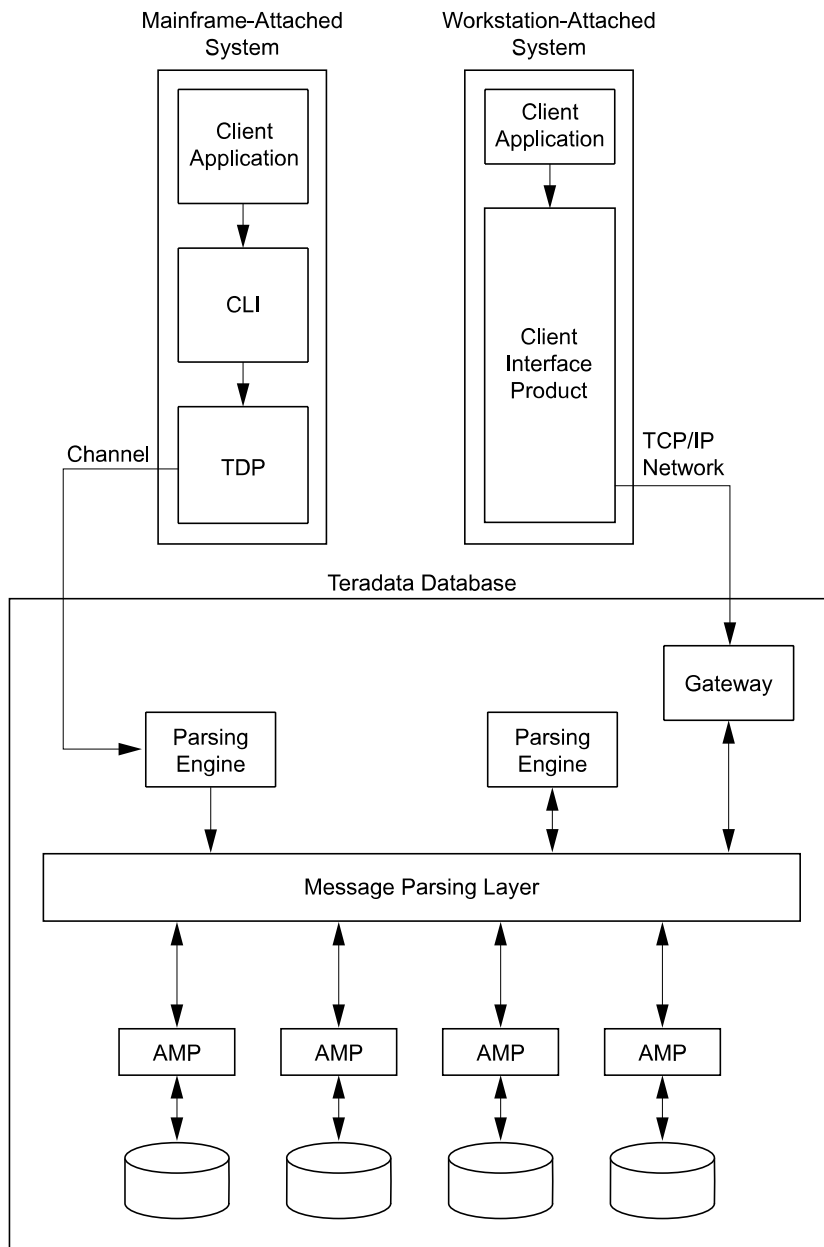
Teradata Director Program for Mainframe-Attached Clients

The mainframe interface enables communication between a mainframe client and Vantage. Teradata Director Program establishes a session with the associated PE and manages communications between the client and Vantage. When a user or application submits an SQL request, the TDP transfers it to Vantage, receives the response, and transfers the response to the application.

Workstation-Attached Clients

In a network environment, the client interface product (such as CLIV2 or the Teradata JDBC Driver) establishes a session with the gateway and manages the routing of user request/server response parcels to and from the gateway. The gateway manages the routing of parcels to and from the PE.

The following figure provides a functional overview of mainframe-attached and workstation-attached clients and Vantage.



Communicating with Teradata Vantage

Sessions Defined

A session is a logical connection between the user and Analytics Database. A session permits a user to submit one transaction at a time and receive one response at a time, and the current session can have only one transaction outstanding at any time. A user may communicate through one or more active sessions concurrently.

A session is explicitly logged on and off from Analytics Database and is established when the Analytics Database server accepts the logon string of the user. When a session is logged off, the system discards the user-session identification and does not accept additional Teradata SQL statements from that session.

Request and Response Messages

When a session logs on, Analytics Database communicates with the host through request and response messages containing parcels of information. The maximum request message size is 7 MB. The maximum response message size is 16 MB.

How Request Parcels Are Handled

To access Analytics Database from a mainframe client, the user or application program logs on through a Teradata client interface product and submits a request. The request is processed by the interface and directed to a TDP.

Each request is handled as follows:

- In a mainframe environment, mainframe CLlv2 builds the request parcel and sends it to the TDP, which sends it through the mainframe connection to the PE associated with that TDP.
- In a network environment, the client interface product (such as CLlv2 or the Teradata JDBC Driver) builds the request parcel and sends it over the network to the gateway. The gateway sends the parcel to the PE for the session, which may or may not reside on the same node as the gateway.

Response Parcels

The result of a query or a status becomes a returned answer set. The PE turns the answer sets for a request into one or more response parcels and returns them to:

- The TDP, in a mainframe environment. The TDP and mainframe CLlv2 returns it to the client utility or program.
- The gateway, in a network environment. The gateway sends it to the client interface product (such as CLlv2 or the Teradata ODBC Driver). The client interface product returns the response to the client utility or program.

Controlling Session Defaults

You can control session defaults by including the DEFAULT DATABASE, COLLATION, or ROLE options in the CREATE USER or MODIFY USER statement, or by issuing a SET statement during a session.

You can also control session conditions for particular users by defining a STARTUP clause containing a string of one or more Teradata SQL statements. A STARTUP string is subject to the same restrictions as a macro.

Client-Server Applications Connectivity

Applications performing embedded SQL statements need to be preprocessed by the Teradata Preprocessor2, which runs on a client system.

You can run the preprocessor against an application without connecting to Vantage if you specify the SQLCHECK (-sc) option as NOSYNTAX. However, to precompile and run an application, a separate connection to Vantage is required for the precompilation and the runtime event.

For instructions on how to prepare, precompile, and run an application on Vantage with embedded SQL statements, see *Teradata Vantage™ - SQL Stored Procedures and Embedded SQL*, B035-1148 and *Teradata® Preprocessor2 for Embedded SQL Programmer Guide*, B035-2446.

Also see:

- *Teradata® Call-Level Interface Version 2 Reference for Workstation-Attached Systems*, B035-2418
- *Teradata® Call-Level Interface Version 2 Reference for Mainframe-Attached Systems*, B035-2417

Mainframe Environment

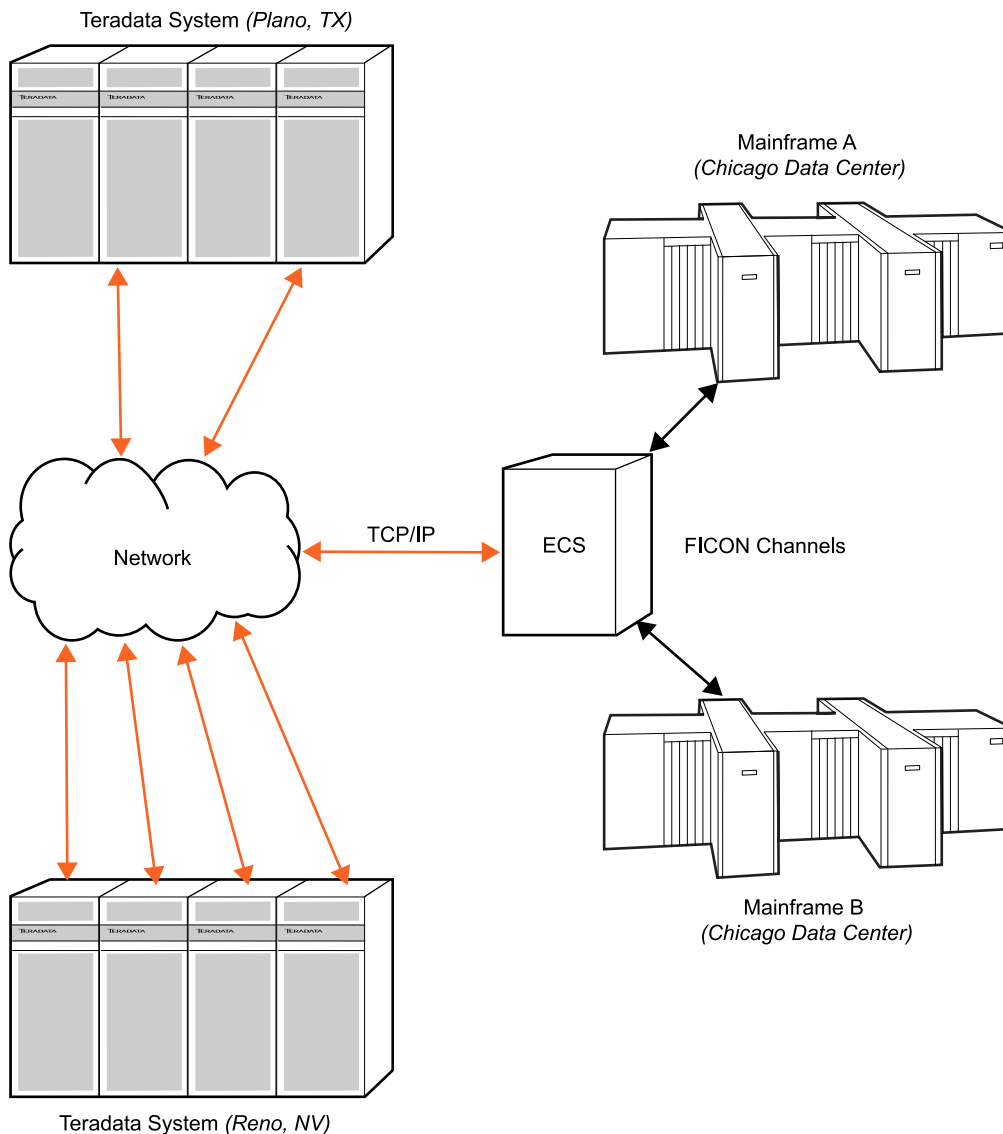
Teradata utilities and software programs support Analytics Database access in mainframe environments. These utilities and programs run under the operating system of the client and provide the functionality for a user to access the database system.

Background

The Teradata Mainframe Interface enables communication between a mainframe client and a server using a channel with either a parallel or serial I/O interface. In the case of the Extended Channel Solution (ECS), the Teradata Mainframe Interface uses a channel plus TCP/IP.

The ECS interface includes:

- The TCHN software on the TPA node, which manages channel devices and communicates with Vantage
- The APSV software on the ECS node, which encapsulates channel write blocks and sends them to the Teradata system via TCP/IP



Example of a Channel Extension with Teradata ECS

The Teradata Channel Interface hardware and the driver may be located on a TPA node or on the Extended Channel Solution (ECS) node. The ECS node provides the following benefits:

- Reduces exposure to Teradata outages induced by adapter faults
- Enables improved mainframe connectivity to a greater number of nodes within the Teradata system
- Creates more options for users connecting Vantage to a mainframe

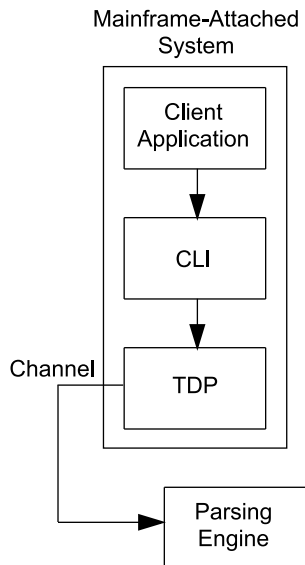
Current platforms can be connected with the PEFA—PCI Bus FICON Host Channel Adapter.

CP and CUA

Currently, each pair of devices, whatever their implementation, is now referred to as a Channel Processor (CP). The even and odd address pair is also known as a channel unit address (CUA).

Software Components

The following figure illustrates the software components in mainframe-attached clients that play important roles in getting requests to and from Analytics Database.



The following table describes these software components.

Component	Description
Client application	<ul style="list-style-type: none"> Written by a developer in your company Written under contract to you by Teradata One of the Vantage-provided utilities <p>Users use these applications to submit SQL statements, maintain files, and generate reports.</p> <p>For details on supported programming languages, the Teradata Preprocessor2, and embedded SQL requirements, see <i>Teradata® Preprocessor2 for Embedded SQL Programmer Guide</i>, B035-2446.</p>
CLIV2	<p>A low-level interface to Vantage. It consists of system calls that:</p> <ul style="list-style-type: none"> Create sessions Allocate request and response buffers Create and deblock parcels of information Fetch response information for the requesting client. <p>For more information, see <i>Teradata® Call-Level Interface Version 2 Reference for Mainframe-Attached Systems</i>, B035-2417 or <i>Teradata® Call-Level Interface Version 2 Reference for Workstation-Attached Systems</i>, B035-2418.</p>
TDP	<p>Manages communication between mainframe clients and Vantage. Functions include:</p> <ul style="list-style-type: none"> Session initiation and termination Logging, verification, recovery, and restart of client applications

Component	Description
	<ul style="list-style-type: none"> Physical input to or output from PE vprocs Security Session balancing across multiple PEs Control of the default client character set to be used during sessions originating through this TDP (unless overridden by the user with a BTEQ.SET SESSION CHARSET command) <p>For more information, see TDP Functionality and <i>Teradata® Director Program Reference</i>, B035-2416.</p>

Mainframe Sessions

A session number uniquely identifies the work stream of a session for a given TDP. A logical client number uniquely identifies each TDP within an z/OS client or multiple clients. A session number and a logical client number identify each session to the z/OS client.

You can request DBCTIME timestamps to record when the:

- TDP receives the request.
- Request was queued to the server.
- Server received the response.
- Response was queued to the cross memory task.
- Response was returned to the input (response) buffer of the user.

Session Pools

A session pool is a number of sessions that are logged onto the Vantage server as a unit, using the same logon string via a TDP START POOL command.

Note:

Session pools can be created only by TDP trusted z/OS system operators and authorized Time Sharing Option (TSO) users.

Unlike ordinary sessions, pool sessions are automatically assigned to applications that initiate a logon using the same logon string as that established for the pool. Every session in a pool is assigned to a specific PE and stays with that PE until the session ends.

When the pool is established, all sessions are not in use. When an application sends a logon request whose string matches that of the session pool, the application is assigned an available session from the pool.

That session is marked in-use and cannot be reassigned to another application until the current application logs off. Sessions pools remain logged on until you log them off with the STOP POOL or LOGOFF POOL commands.

For more information on session pools and TDP commands, see *Teradata® Director Program Reference*, B035-2416.

TDP Functionality

All messages that a mainframe client sends or receives to the Analytics Database normally pass through the TDP. In addition to session, packet, and security control, mainframe TDPs are responsible for:

- Balancing sessions across assigned parsing engines
- Routing responses back to the originating address space

Also, you can request DBCTIME time stamps to record when:

- The TDP receives the request.
- The request was queued to the server.
- The server received the response.
- The response was queued to the cross-memory task.
- The response was returned to the input (response) buffer of the user.

TDP Exits

An exit is a point at which a user request temporarily leaves the existing code to perform a user-specified task before continuing on with normal processing. You can define routines to perform some function or alteration of normal processing.

You can customize the TDP to perform a user-defined exit routine. Customizing the TDP can assist you in collecting information for performance or functional analysis.

The TDP User Transaction Collection Exit (TDPUTCE) is a routine that allows you to collect statistics about all of the requests and responses that pass through the TDP. TDPUTCE is an exit taken from the Transaction Monitor.

Memory Management

To provide for memory acquisition during system operation without incurring the high overhead associated with the operating system memory services, the TDP acquires units of main memory, or cells, from its own more efficient memory management.

During startup, the memory manager pre-allocates a number of cells in sizes that are convenient for use by the TDP.

The sizes of the cells are internal constants. The initial number of cells is an internal default.

If a TDP subtask requests a cell from the memory manager, but other TDP subtasks are using all available cells, the memory manager takes one of the following actions:

- Obtains a new cell from the operating system
- Places the requesting subtask into a wait for memory state.

If the requester is placed into a wait state, the wait ends when another TDP subtask releases a cell. The decision to obtain a new cell or wait for an existing cell is based on TDP considerations.

The TDP typically uses a virtual region of about 4 to 5 MB. To avoid overhead calls to the operating system, the TDP divides its work areas in cells. A warning message (TDP0021) displays when 80% of the cells of a certain size are in use.

Also see [Performance and Memory Management](#).

Using TDP Commands

Commands you enter from the console are not executed until you execute the RUN command. The TDP accepts operator commands from the following:

- The z/OS console
- z/OS and TSO users
- Client interface program applications (such as CLlV2 and the Teradata ODBC and JDBC Drivers)

Messages that result from executing operator commands entered from a console are returned to the console.

For more information, see *Teradata® Director Program Reference*, B035-2416.

Network Environment

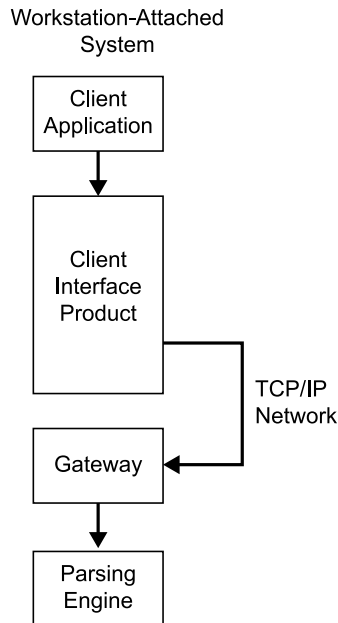
In a network environment, each workstation has a copy of Teradata client software, including the utilities, programs, and drivers needed to access Vantage.

Functionality

In a network environment, the client interface product (such as CLlV2 or the Teradata JDBC Driver) and gateway software in the node handle the functions that TDP handles in a mainframe-attached environment.

Software Components

This figure illustrates the software components in workstation-attached clients that play important roles in getting and executing requests to and from Vantage.



The following table describes these software components.

Component	Description	References
Client application	Either: <ul style="list-style-type: none"> • Written by a developer in your company • Written under contract to you by Teradata • One of the Vantage-provided utilities Users use these applications to submit SQL statements, maintain files, and generate reports.	<i>Teradata® Preprocessor2 for Embedded SQL Programmer Guide</i> , B035-2446
Teradata CLI	A low-level interface to Vantage, CLI consists of routines that perform functions similar to CLI system calls on mainframe-attached clients, including: <ul style="list-style-type: none"> • Logging sessions on and off • Submitting SQL queries • Receiving responses with the answer set 	<i>Teradata® Call-Level Interface Version 2 Reference for Workstation-Attached Systems</i> , B035-2418
Database connectivity drivers	A variety of drivers for open connectivity products, such as ODBC and JDBC.	<i>Teradata JDBC Driver Reference</i> , available at https://teradata-docs.s3.amazonaws.com/doc/connectivity/jdbc/reference/current/frameset.html <i>ODBC Driver for Teradata® User Guide</i> , B035-2526

The Teradata Gateway

The gateway software is the interface between the network and Vantage. It runs on the server as a separate operating system task. Client sessions that communicate through the gateway to Vantage are installed and running on workstations.

In contrast, sessions originating from a mainframe-attached system access Vantage through mainframe connections and TDP software and bypass the gateway completely.

The following table describes tools available to administer your gateway.

Tool	Name	Function
Gateway Control utility	gtwcontrol	Recommended for use only by Teradata field support engineers to. <ul style="list-style-type: none"> • Determine the format of some user names • Configure security features. • Control gateway resource usage. • Investigate gateway problems For more information, see <i>Teradata Vantage™ - Analytics Database Security Administration</i> , B035-1100.
Gateway Global utility	gtwglobal	<ul style="list-style-type: none"> • Monitor network sessions and traffic • Disable logons • Force off a rogue session • Investigate gateway problems
Data Dictionary table	DBC. EventLog	The table logs entries for logon and logoff events found by session control, including logon failures detected external to session control (such as when the Gateway receives an authentication failure).

Displaying Network and Session Information

Using the Gateway Global utility, you can find the information listed in the following table.

Command	Description
DISPLAY NETWORK	Displays your network configuration.
DISPLAY GTW	Displays all sessions connected to the gateway.
DISPLAY SESSION	Displays information about a specific session on the gateway.
DISPLAY DISCONNECT	Displays a list of sessions that have disconnected and not yet reconnected.
DISPLAY FORCE	Displays sessions that have been forced off the system by Gateway Global or a security violation.

Controlling Network Sessions

Use the Gateway Global utility to perform the actions described in the following table.

Command	Description
DISABLE LOGONS	Disable logons through the gateway. This includes logons through DBC.
ENABLE LOGONS	Enable logons via the gateway.
KILL USER	Terminates all sessions of a specific user.
KILL SESSION	Terminates a specific session. Must know session number.

Controlling Trace Logs

Use the commands listed in the following table to turn tracing on or off or to flush the trace buffers to the Event log.

Command	Description
ENABLE TRACE	Records internal gateway events.
DISABLE TRACE	Turns off the writing of event log entries.
FLUSH TRACE	Directs the gateway to write the contents of its internal trace buffers to the event log file.

Related Information

For information on the Gateway Control or Gateway Global utilities, see *Teradata Vantage™ - Database Utilities*, B035-1102. For information on how to configure your Gateway, see *Parallel Upgrade Tool (PUT) Reference*, B035-5716.

Troubleshooting: Operational DBAs

Teradata Vital Infrastructure (TVI) (or Customer Care Link (CCL) on older systems), will send alerts to you and to Teradata when there is a problem in the system. However, there are some areas where you must monitor the system. You can use Teradata Viewpoint, or write custom scripts to monitor blocking, locate idling sessions, and determine resource utilization.

This section suggests some tools for resolving problems and points to specific references for information on how to use these tools. Some of these tools can help you find and analyze a problem. For information on how to handle system slowdowns or hangs, see [Investigating Query Blocks and Delays](#).

Note:

Access to administrator utilities is usually limited to privileges specifically granted in the database. Users of these utilities should log on using their database usernames. Users that are externally authenticated (with directory or Kerberos usernames, for example) may not have access to administrator utilities. For more information about the privileges of externally authenticated users, see *Teradata Vantage™ - Analytics Database Security Administration*, B035-1100.

Tools for Troubleshooting and Administrating

The following table lists some common tools you can use to administrate your system. The table also lists tools intended for use by Teradata field engineers or Vantage system developers. If the tools are described as for use by Teradata personnel only, do not use them unless instructed by Teradata Support.

For more information on the respective tool, see the documentation referenced. For a list of the utilities by suggested function (such as maintenance, troubleshooting, or installation), see the “Alphabetical Listing of Utilities” in *Teradata Vantage™ - Database Utilities*, B035-1102.

What to Troubleshoot or Administrate	Tool and Description	Reference
Administration Workstation	The AWS console for an MPP installation displays the status of each physical component, including nodes, DBW consoles, and the BYNET. It provides many of the functions the System Console would for an SMP.	AWS manuals
AMPs and AMP Worker Tasks	<p>The ampload utility reports AMP bottlenecks due to unavailable free AWTs.</p> <p>The AWT Monitor (awtmon) utility quickly views how many AMP worker tasks are active (in use) and determines “hot AMPs” so you can troubleshoot performance problems.</p> <p>awtmon is a front-end tool to the “puma -c” command and prints AWT in-use count information in a summary format.</p>	<ul style="list-style-type: none"> “AMP Load (ampload)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 “AWT Monitor (awtmon)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102

What to Troubleshoot or Administrate	Tool and Description	Reference
	<p>By default, awtmon displays the AWT in-use count of the local node. With the -s option, print system-wide information and locate hot AMPs on the nodes of the entire system. When hot AMPs are identified, run awtmon on those hot nodes using pcl or psh.</p> <p>You can also use the ResUsageSawt table to see AMP worker tasks in-use counts by work type and determine when AMPs enter the state of flow control.</p> <p>To determine which workload is responsible for I/O skew, use the ResUsageSps table and then use DBQL to identify specific queries.</p>	<ul style="list-style-type: none"> • “ResUsageSawt Table” and “ResUsageSps Table” in <i>Teradata Vantage™ - Resource Usage Macros and Tables</i>, B035-1099
Data corruption and integrity	<p>Checksums check the integrity of disk I/O operations. A checksum is a numeric value computed from data. For a given set of data, the checksum value is always the same, provided the data is unchanged.</p> <p>Because calculating checksums requires system resources and may affect system performance, system-wide checksumming is disabled by default on most platforms. Contact Teradata Support if you suspect disk corruption.</p> <p>Use the CHECKSUM option of CREATE TABLE, ALTER TABLE, CREATE JOIN INDEX, and CREATE HASH INDEX statements to enable checksums for a table, hash index, or join index.</p>	“Checksum Fields” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
	<p>The SCANDISK command of the Ferret utility checks the integrity of the file system. This includes the master index, cylinder index, and data block structures (including structures associated with WAL logs).</p> <p>Previously halted SCANDISK operations can be restarted using a Perl script.</p> <p>Note:</p> <p>If further events occur after SCANDISK has been running for some time, restarting SCANDISK where it left off the last time will not find errors in previously scanned data that were caused by the most recent failures. Therefore, be very careful when deciding to restart a pending SCANDISK operation versus starting the entire operation again from the beginning.</p>	“Ferret Utility (ferret)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Disk space	<p>The Ferret utility allows field engineers and Teradata support people to monitor and control disk usage.</p> <p>The commands of this utility allow the system to combine free sectors on a cylinder, reconfigure contents on a disk leaving a specified percentage of space free for future use, report which tables qualify for packing and display disk cylinder utilization and available free cylinders.</p> <p>Note:</p> <p>This utility is used by Vantage engineers to perform routine and special diagnostics.</p>	“Ferret Utility (ferret)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
	<p>Update DBC utility recalculates PermSpace, SpoolSpace, and TempSpace for user DBC in DBase table. Then based on DBase values, Update DBC utility recalculates MaxPermSpace and MaxSpoolSpace in DBC.DataBaseSpace for all databases.</p>	“Update DBC (updatedbc)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102

What to Troubleshoot or Administrate	Tool and Description	Reference
	Use Update DBC only to correct inconsistency in the DBase or DataBaseSpace tables, which might occur as a result of rare types of system failures.	
	Update Space recalculates permanent, temporary, or spool spaces for one database or all databases.	“Update Space (updatespace)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
	The FixCurrentSpace SQL stored procedure performs the same functions as the Update Space utility.fixcurrentspace	FixCurrentSpace Procedure
	The FixAllocatedSpace SQL stored procedure repairs the dynamic need-based allocations for databases that use global space accounting. This procedure is fast because it assumes actual current usage is correct.	FixAllocatedSpace Procedure
	The DBC.DiskSpaceV view provides disk space usage information per AMP. This includes permanent and spool data by database or account for each AMP. Use this view to track large spool usage and available PERM space.	<ul style="list-style-type: none"> • Querying the DiskSpaceV View to Find System Space Issues. • <i>Teradata Vantage™ - Data Dictionary</i>, B035-1092
Down AMP	If you receive an error that an AMP is down (for example from a MultiLoad or BTEQ job) use Vprocmanager to reboot a fatal AMP or bring a offline down AMP up.	“Vproc Manager (vprocmanager)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Down Table	<p>The system will mark a data or index subtable as down if an error occurs in several regions of the file system structures of the table. This error could be due to hardware or bad I/O. To address or fix a down region in a subtable, do one of the following:</p> <ul style="list-style-type: none"> • Use the DELETE ALL/DROP table statement. • Rebuild the table using the Table Rebuild utility. • Restore the table from backup. • Drop and recreate the index. • Drop and recreate the fallback subtable. <p>After repairing a down table, use the ALTER TABLE ... RESET DOWN statement to reset a down table.</p> <p>In the case of hardware errors, a fallback table can be rebuilt to copy the rows from the fallback subtable, to repair the rows in the down regions of the primary subtable, and vice versa. If the down table is rebuilt or restored from an earlier back up, the down status flag for the various subtables will be reset, and the rows in them will also be moved to unmarked data blocks and cylinders from the fallback copies.</p> <p>To change the number of regions per AMP that must fail before a table is marked as down, use the MaxDownRegions field in DBS Control.</p>	<ul style="list-style-type: none"> • “ALTER TABLE” in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144 • “MaxDownRegions” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102

What to Troubleshoot or Administrate	Tool and Description	Reference
	See “Table” in this table for more information.	
Global parameters	DBS Control displays and modifies the tunable global parameters in the DBS Control record.	“DBS Control (dbscontrol)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Gateway settings	<p>The Gateway Control utility allows you to set Gateway settings such as how to handle encryption, number of session allowed for host group, time out values and more.</p> <p>You can use options in the Gateway Control utility to select things like connection time out length, external authentication settings, or whether to use the system default or customer-settable defaults after a configuration or addition of new host groups and gateway vprocs, and more.</p> <p>Note:</p> <p>This utility is used by Vantage engineers to perform routine and special diagnostics.</p>	<ul style="list-style-type: none"> • Tools for Troubleshooting Client Connections • “Gateway Control (gtwcontrol)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 • “Gateway Global (gtwglobal)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102
	The Gateway Global utility monitors and controls sessions for LAN-connected users. You can monitor traffic, control network sessions, display which users are using which sessions, or abort specific sessions.	
Hangs or slowdowns	<p>See the topics “Sessions that are hung”, “AMPs and AMP Worker Tasks”, and “Resource usage” in this table.</p> <p>Resource Check Tools detect hangs or slowdowns as follows:</p> <ul style="list-style-type: none"> • mboxchk: Monitors response time. If the response time indicates the database is slow or unresponsive, mboxchk runs a Teradata Support Center script to gather system information, logs an event, and contacts Teradata Vital Infrastructure (TVI). • nodecheck: Displays local, node-level resources only. Provides summary data to syscheck for analysis. • syscheck: Monitors the system for signs of congestion that might lead to system slowdowns or perceived hangs and notifies you when Warning or Alert thresholds are reached. <p>A text file called syscheckrc resides on each node. You can set it up to detect and report any resource that falls below a threshold.</p> <p>The resource check tools are located in the /usr/pde/bin directory.</p>	mboxchk, nodecheck, syscheck in man pages or pdehelp
Loads into Large Tables	<p>To improve the speed of a batch INSERT/SELECT request into a large table, partition the table by transaction date, which groups the inserts into data blocks within the partition instead of throughout the table. This reduces the number of I/Os.</p> <p>You can also use the DBQL utility job log table, DBC.DBQLUtilityTbl, to troubleshoot utility job performance.</p>	<ul style="list-style-type: none"> • Tools for Troubleshooting Client Connections • <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144 • Utility Job Performance Analysis and Capacity Planning

What to Troubleshoot or Administrate	Tool and Description	Reference
Lock contentions or held locks	There are several things that might cause lock contention: a pending lock, a very congested system, a number of transactions exceeding the machine capacity, or a conflict with DSA. Careful planning of what sessions run in which order or at what time of day can help prevent lock contention. Use the EXPLAIN. Consider running a request that needs an EXCLUSIVE lock and will take a while to process during off hours.	Lock Contentions
	The Lock Display utility shows all real-time locks in use. This includes locks for concurrency control. You can identify which transactions are blocked and which transactions are doing the blocking.	“Lock Display (lokdisp)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
	You may have locking issues on DBC.AccessRights if you are running concurrent jobs with the same userid in the same database and dropping or creating objects. To reduce locking issues on DBC.AccessRights: <ul style="list-style-type: none"> • Run the jobs with a few different userids and distribute them randomly among the connection strings to reduce the occurrence of two userids running concurrently. • Determine if housekeeping needs to be performed on DBC.AccessRights for the UserID/DatabaseID involved. • Determine if statistics need to be collected on the Data Dictionary tables. 	Teradata knowledge articles on reducing locking issues with specific applications
	The DBQL Lock Log logs in XML format in DBQLXMLLOCKTbl any lock contention longer than a user-specified threshold. You can access the DBQL lock log through the Viewpoint lock viewer portlet or query the system table DBC.DBQLXMLLOCKTbl or the view DBC.QrylockLogXMLV.	XML Lock Log Table: DBQLXMLLockTbl .
	Showlocks utility identifies and displays all active host utility (HUT) locks placed on databases and tables by DSA operations. HUT locks may interfere with application processing and are normally released after the utility process is complete. If locks interfere with application processing, you can remove them by invoking the RELEASE LOCK statement. It is available through the DSA utility or as an SQL statement. Showlocks can be started from the DB Window Supervisor screen with the command start showlocks.	<ul style="list-style-type: none"> • <i>Teradata® DSA User Guide</i>, B035-3150 • “Show Locks (showlocks)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102
Memory (insufficient memory resulting in Error 3710)	Teradata recommends that you use the default value set for the MaxParseTreeSegs field in DBS Control. If you run into this error, first try simplifying your queries and/or dropping unnecessary join indexes. If this still does not work, verify the value you have set for MaxParseTreeSegs and contact Teradata Support.	“MaxParseTreeSegs” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102

What to Troubleshoot or Administrate	Tool and Description	Reference
	<p>Note:</p> <p>The optimal value depends on your system and what release and fix level you are running. Increasing MaxParseTreeSegs to a value larger than the default may allow a previously failed 3710 error to run but could result in other problems such as a long parsing time or in extreme cases, a Teradata reset.</p>	
Priority of jobs	<p>TASM assigns different priorities to different types of jobs. Higher priority work receives access to CPU and I/O more frequently.</p> <p>Use the ResUsageSps table to understand how CPU is being consumed by different workloads. You can also use this table to understand the pattern of database activity across the processing day.</p>	“ResUsageSps Table” in <i>Teradata Vantage™ - Resource Usage Macros and Tables</i> , B035-1099
Queries	<p>Use BEGIN/REPLACE QUERY LOGGING WITH XMLPLAN or use the EXPLAIN IN XML request modifier.</p> <p>Alternately, use the STEPINFO option of BEGIN/REPLACE QUERY LOGGING to determine which step uses the most resources and if there are any large differences between estimated and actual CPU, I/O, and number of rows.</p> <p>Manage your queries using Teradata Viewpoint. You can:</p> <ul style="list-style-type: none"> • Filter queries against object-access or query-resource rules. • Control the flow of queries coming into the system for concurrency reasons and delay them if necessary. • Log any system exception actions. 	
Resource usage <i>Teradata Vantage™ - Data Dictionary</i>	Using resource usage macros and tables to obtain information about which nodes or which vprocs are heavily using which resources. This can help you identify bottlenecks.	<i>Teradata Vantage™ - Resource Usage Macros and Tables</i> , B035-1099
	The DBQL view DBC.QryLogV reports things such as the AMP using the most CPU, the AMP with the most I/O, or maximum amount of spool used when processing a query.	See QryLog[V] in <i>Teradata Vantage™ - Data Dictionary</i> , B035-1092
Sessions that are hung	Use Query Session utility to determine the state of each session and then use Lock Display utility to find lock contentions.	Troubleshooting Problems with Sessions
Session information when logged onto multiple sessions	<p>To find the Thread ID or Job ID of the session from which you are querying, first submit a SELECT SESSION;</p> <p>Then after that query returns the session number <i>n</i>, use that number in the following query:</p> <pre> sel logonsource from sessioninfovx where sessionno = n; </pre> <p>Result:</p>	“SessionInfo[V][X]” in <i>Teradata Vantage™ - Data Dictionary</i> , B035-1092

What to Troubleshoot or Administrate	Tool and Description	Reference
	<p>*** Query completed. One row found. One column returned. *** Total elapsed time was 1 second.</p> <p>LogonSource ----- (TCP/IP) 09AB 127.0.0.1 DBC 8804 USER BTEQ 01 LSS</p> <p>This helps you determine which Client process/Thread ID belongs to which session. This is especially useful if you want to find out which Thread ID belongs to which session because you are logged on to multiple sessions.</p>	
	Query Session utility displays the state of load utility and query sessions. Details can include statuses such as Parsing, Active, Blocked, Response, whether stored procedures are being processed, and so on.	“Query Session (qrysessn)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Skew	<p>There are many ways to detect skew:</p> <ul style="list-style-type: none"> • Use Teradata Viewpoint to determine which user is using the most CPU. In general, a bad query can be detected by finding users with high CPU relative to I/O access. • Look at the EXPLAIN text to determine the number of steps used for the query and the type of operation. You may find something like a product join in a poorly written query. You can abort the query and investigate further or fix the query. • To identify node-level skew, use the ResUsageSpma table, comparing maximum CPU usage to the average CPU consumed across all nodes. If parallel efficiency as shown in the ResUsage Spma table is not approaching 100%, look for the reasons for the imbalance. • To find AMP-level skew, there are several options. Use the ResCPUByAMP macro to find AMP-level CPU skew. You can look at the ResUsageSpvr table and compare resource usage across different vprocs. You can also look at the number of requests field in the ResUsageSps table to see if workloads on one node are actively supporting more requests than on other nodes, then drill down to AMPs. Also using the ResUsageSps table, you can examine queue wait and service time numbers to find backed up queries by workload or allocation group. • To find skew at the request level, you can compare DBQLogTbl fields MaxAMPCPUTime with MinAMPCPU. Because user logging is often incomplete with DBQL, ResUsage tables and macros may provide more accurate information. • Another option is to use PM/API to develop your own application to monitor for high CPU consumers. 	<ul style="list-style-type: none"> • “EXPLAIN Request Modifier” in <i>Teradata Vantage™ - SQL Data Manipulation Language</i>, B035-1146 • “ResUsageSps Table,” “ResUsageSpma Table,” “ResUsageSpvr table,” and “ResCPUByAMP Macros” in <i>Teradata Vantage™ - Resource Usage Macros and Tables</i>, B035-1099
Slowdowns	See the topic “Hangs or slowdowns” in this table.	
Slow queries to DBC tables	If your site uses a lot of custom applications or tools that often query the DBC tables, consider collecting statistics on these tables	About Collecting Statistics on Data Dictionary Tables

What to Troubleshoot or Administrate	Tool and Description	Reference
Space Exceeded	<p>If a space-intensive operation, such as a load utility or restoring from an archive, exceeds available perm space, two DBS Control utility fields can be set temporarily to allow operations to complete and avoid out of space errors.</p> <p>The settings apply to all operations and all instances of space accounting, such as table inserts, so Teradata recommends setting these fields to a non-zero value only temporarily.</p>	<p>OverflowMaxPermSpace-Percentage and OverflowMaxPermSpace-KByteCount fields of the DBS Control utility in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102</p>
	<p>Alternatively, use global space accounting, which provides extra space to AMPs when needed, preventing space-intensive operations from failing due to lack of space.</p>	<p>Global Space Accounting</p>
Spool space depletion	<p>There are several reasons that could contribute to limited spool usage:</p> <ul style="list-style-type: none"> • Not using global space accounting, which allows extra space to AMPs when needed. • Poorly written queries can sometimes consume excessive spool space. Use Teradata Viewpoint to check queries before allowing them to run. You can check the EXPLAIN to see how much spool is used and prevent the system from getting overloaded with bad queries. • Outdated statistics can cause bad queries. The Optimizer may calculate query plans differently than it ought if statistics it uses does not accurately reflect the system. The Optimizer also determines how much spool it needs or does not need based on collected statistics. Refresh collected statistics regularly. • If you are using OLAP statistical functions and you run out of spool, check the syntax of your queries. For example, when using the PARTITION BY clause, if a large number of identical values in the partitioning column hash to the same AMP, this can result in out-of-spool errors. Choose a column that results in rows being distributed over a broader set of AMPs. • Poorly skewed data. Poor data distribution means one AMP is responsible for a larger proportion of the data. Because the spool space is divided by the number of AMPs, reaching the maximum spool space limit on one AMP means your query will fail even despite the other unused spool space on the other AMPs. • Poorly chosen PI that leads to AMP skew. Because the spool space is divided by the number of AMPs, reaching the maximum spool space limit on one AMP means your query will fail even despite the other unused spool space on the other AMPs. • Adding nodes or AMPs without reassigning spool space means spreading out the spool amongst more AMPs. • The limit for spool space as defined in the profile overrides the user settings. Check the limit for the profile to see if it is lower than the limit defined for the user. You can compare the values of MaxProfileSpool and MaxSpool in DBC.DiskSpace.V <p>Limiting spool space for new users helps reduce the impact of possibly bad queries (such as product joins). With a reasonable limit, the user will get out-of-spool messages before a bad query ties up the system.</p>	<ul style="list-style-type: none"> • Global Space Accounting • Troubleshooting Spool Space Problems • “EXPLAIN Request Modifier” in <i>Teradata Vantage™ - SQL Data Manipulation Language</i>, B035-1146 • Teradata Viewpoint • “Ordered Analytical/Window Aggregate Functions” in <i>Teradata Vantage™ - SQL Functions, Expressions, and Predicates</i>, B035-1145

What to Troubleshoot or Administrate	Tool and Description	Reference
	Furthermore, adding spool to a bad query will allow the query to run longer but a bad query may still not complete.	
Space allocated to storage	Use the Teradata Virtual Allocator Manager (TVAM) utility. Allows you to display mapping or force migration of cylinders on the system. Note: Do not use the TVAM utility unless instructed by Teradata Support.	Man page or online help
System crashes or failures	Screen Debug and Screen Dump controls how and what your system records for crashdumps. Teradata recommends that default settings for all dumps be changed only when requested by a system support representative or Teradata Support.	“Control GDO Editor (ctl)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
	DBC.Software_Event_LogV is a system view that provides detailed information about errors or system failures and their related node, vproc, partition, task, function, software version, optional backtrace, diagnostic information, and so on. The Event_Tag field of the view reports the error message number.	<ul style="list-style-type: none"> • <i>Teradata Vantage™ - Data Dictionary</i>, B035-1092 • <i>Teradata Vantage™ - Database Messages</i>, B035-1096
	DUL utility saves or restores system dump tables onto tape.	“Dump Unload/Load Utility (dul)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
System recovery	Recovery Manager utility monitors and reports the progress of a system recovery after a crash or a user abort. You can use this utility to monitor transaction rollbacks, cancel rollbacks, and more.	“Recovery Manager (rcvmanager)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
Table	The Table Rebuild utility rebuilds tables Vantage cannot recover automatically. Table Rebuild can rebuild the following for an AMP: <ul style="list-style-type: none"> • The primary or fallback portion of a table • An entire table (both primary and fallback portions) • A particular range of rows of a subtable • The corrupted regions of a table • A corrupted or missing table header • All tables • All fallback tables • All tables in a database • Single tables sequentially or tables of each database in parallel. 	<ul style="list-style-type: none"> • “Table Rebuild (rebuild)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 • See also “Table inconsistencies or corruption”
Table inconsistencies or corruption	CheckTable is a diagnostic tool that checks for: <ul style="list-style-type: none"> • Table and dictionary inconsistencies, such as differing table versions, ParentCount and ChildCount data, and partitioning definitions. 	“CheckTable (checktable)” in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102

What to Troubleshoot or Administrate	Tool and Description	Reference
	<ul style="list-style-type: none"> Table corruption, such as duplicate rows or unique values and data inconsistency, of primary and fallback data and stored procedure tables. Inconsistencies in internal data structures such as table headers, row identifiers, secondary indexes, and reference indexes. Invalid row hash values and partition numbers. <p>Options for CheckTable include the following:</p> <ul style="list-style-type: none"> Four levels of data integrity checking (PENDINGOP, ONE, TWO, and THREE). Each successive level performs additional checks that are more thorough and consequently more time consuming. The DOWN ONLY option displays status information for down tables only. The DOWN ONLY option works for all levels except PENDINGOP. The ERROR ONLY option causes CheckTable to display only skipped tables, down tables, and tables with errors and warnings. This allows you to quickly identify and address problems found by CheckTable. The OUTPUT command allows you to redirect CheckTable output to a log file. The ECHO option for this command allows you to direct output to both the console and a log file. <p>CHECKTABLEB is a non-interactive batch mode version of CheckTable that can be run through the cnstrun utility. CHECKTABLEB is identical to CheckTable except that it is meant to run from a script.</p>	
	<p>The Table Rebuild utility allows you to fix a primary or fallback copy of a table or only specific corrupted parts. When the rebuild operation successfully completes on an AMP, TableRebuild will remove the down region information from the table header. If, however, the fallback subtable has any down regions defined on it, the rebuild process aborts and the system returns an error.</p> <p>You cannot rebuild a table with parts of its fallback down. While you can rebuild stored procedures, UDFs, and UDMS, you cannot rebuild a join index or hash index defined on a table being rebuilt.</p>	<ul style="list-style-type: none"> “Table Rebuild (rebuild)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 See also “Table inconsistencies or corruption”
Vprocs	<p>The Vproc Manager utility allows you to:</p> <ul style="list-style-type: none"> View or change vproc states Initialize and boot a specific vproc Initialize the storage associated with a specific vproc Force a manual restart <p>A few of the vprocs statuses include:</p> <ul style="list-style-type: none"> FATAL: the vproc or its associated storage is not operational. The system marks it fatal because of repeated crashes or if data integrity is in danger. Multiple crashes are indicated by the crash count and indicate a repeatable condition of data corruption, bad SQL or a software bug. Contact Teradata Support. FATAL** : this is an “fsgwizard” data integrity situation. Contact Teradata Support. OFFLINE: the vproc is operational, but either set offline purposely by an operator or if, for example, all AMPs in a clique are OFFLINE, they 	<p>“Vproc Manager (vprocmanager)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102</p>

What to Troubleshoot or Administrate	Tool and Description	Reference
	<p>were probably set OFFLINE by the system because none of the nodes in its clique were up. This usually occurs during a system-wide reboot of all nodes, but nodes come up at different times.</p> <p>Note: A local crash of an OFFLINE vproc will cause the database to restart.</p> <ul style="list-style-type: none"> UTILITY Catchup (offline catchup): the vproc was previously down (OFFLINE or FATAL or some other state). If its fallback data was written to another vproc in its cluster, the vproc is now back up (but not online) while it catches up its fallback data. 	
Vprocs and storage	<p>From the Supervisor screen of the Database Window, enter GET CONFIG to view the current system configuration.</p> <p>To administrate vprocs and disks, use the Vproc Manager utility.</p>	<ul style="list-style-type: none"> “Database Window (xdbw)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 “Vproc Manager (vprocmanager)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102
	<p>The Query Configuration utility reports the status of vprocs managed by node or for the entire system.</p>	<p>“Query Configuration (qryconfig)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102</p>
Vproc Manager (vprocmanager) utility	<p>If vprocmanager exits with an error...</p> <pre># vprocmanager PDE is not running: Operation not permitted PDE event 10117 reported when task is not attached to PDE or PDE is down. Task should stop on this error.</pre> <p>then you are probably attempting to run it from a non-TPA node. Run vprocmanager from a TPA node (a node running Vantage software).</p>	<p>“Vproc Manager (vprocmanager)” in <i>Teradata Vantage™ - Database Utilities</i>, B035-1102</p>

Tools for Troubleshooting Client Connections

Mainframe Connection Tools

The following tools monitor session and TDP activity on mainframe-connected clients.

- HSI timestamp – Host System Interface (HSI) timestamps tell you when TDP receives a request, when the request parcel is sent to or queued for Vantage, and when the response parcel is received from Vantage.

- TDPUTCE – TDP User Transaction Collection Exit (TDPUTCE) collects statistics about all of the requests and responses controlled by the TDP, including user, session/request parcels, timestamps, request type, and request/response parcels.
- Your site is responsible for processing and analyzing the data collected by TDPUTCE.
- z/OS System Management Facility (SMF) is a feature in the z/OS operating system that can record accounting and performance information, such as:
 - Statistical information about the processing activity of a PE recorded at shutdown.
 - Log-off session information.
 - Logon violations and security violations records.
 - Statistical data about the processing activity of the TDP, recorded at shutdown.
 - Statistical information about the processing activity of a Channel Processor (CP) either recorded at shutdown or started with the Channel Communication Unit (CCU) operand.

For more information on TDP, see *Teradata® Director Program Reference*, B035-2416.

Network Connection Tools

The following tools monitor and control sessions originating from workstation-attached clients.

Tool	Description	Reference
Gateway Control (gtwcontrol)	Utility that allows you to monitor network and session information. You can do the following: <ul style="list-style-type: none"> • Get network configuration information • See all sessions connected via the gateway • See status information for a selected gateway session • Force off one or more network sessions 	<ul style="list-style-type: none"> • The Teradata Gateway • <i>Teradata Vantage™ - Database Utilities</i>, B035-1102
Teradata Network Statistics (tdnstat)	Utility that gives you a snapshot, or a snapshot differences summary, of statistics specific to Teradata Network Services. You also can clear the current network statistics.	Man or pdehelp pages

Troubleshooting Spool Space Problems

The system automatically creates and drops spool files that it uses as temporary work tables to execute queries. Sometimes, one of two potential problems can occur with the spool files:

- **Leftover spool** – Leftover spool occurs when the system fails to properly drop the spool file after the execution of a query completes. The spool that the query was using appears frozen without a session. This may result in error 2667 which aborts the transaction. Note that with error 2667, the system terminates the query not because the query was faulty, but as a result of the leftover spool problem.
- **“Phantom” spool** – Phantom spool occurs when the DBC.DatabaseSpace table is not properly updated to reflect current and actual space usage.

To check for either types of spool problems, run the following query when the system is most quiescent and when there are the least amount of users logged on:

```
SELECT DATABASENAME, VPROC, CURRENTSPOOL
FROM DBC.DISKSPACE
WHERE DATABASENAME NOT IN (SEL USERNAME FROM DBC.SESSIONINFO)
AND CURRENTSPOOL > 0
ORDER BY 1,2
WITH SUM(currentspool);
```

Note:

If the user whose query unintentionally caused the spool space problem is logged on at the time you run this query, the query will not detect the user because the query is looking for spool usage for a user that is holding spool space but is not logged on.

If the request returns rows, do one of the following things:

- Run the Update Space utility to update the spool space accounting in DBC.DatabaseSpace for each of the databases.
- Run the FixCurrentSpace procedure. This procedure performs the same functions as the UpdateSpace utility. For more information, see [The FixCurrentSpace Procedure](#).

For assistance with spool space problems, contact Teradata Support.

Using the Update Space Utility

The Update Space utility (updatespace) recalculates the permanent, temporary, or spool space used by either of the following:

- A single database and its individual tables
- All databases in a system and their individual tables

The Update Space utility accomplishes this by:

- Examining storage descriptors and adding up space for each table.
- Setting values in CurrentPermSpace, CurrentTempSpace, or CurrentSpoolSpace in the DBC.DatabaseSpace table for each table and for the containing database as a whole.

A different utility, Update DBC (updatedbc), recalculates the maximum allowed values for permanent, temporary, and spool space in the DBC.Dbase and DBC.DatabaseSpace tables.

The following are the difference between the Update DBC and Update Space utilities:

- Update DBC recalculates maximum allowed values for permanent, temporary, and spool space.
- Update Space recalculates current usage for permanent, temporary, and spool space.

The only reason to use Update Space is to correct inconsistencies in the DBC.DatabaseSpace table, which might occur as the result of rare types of system failures.

The format of the command to use with the “Update Space” utility is:

```
UPDATE {
  { TEMPORARY | ALL } SPACE FOR { ALL DATABASES | dbname | ALL PROXYUSERS } |
  { SPOOL | PPOOL } SPACE FOR { dbname | ALL PROXYUSERS }
} [;]
```

If the problem is fixed using Update Space, then the problem was phantom spool. If running Update Space does not fix the problem, you have leftover spool and the only way to fix the problem is to restart the system.

For assistance with spool space problems, contact Teradata Support.

Related Information

Topic	Resources for Further Information
Command syntax for the Update Space utility	"Update Space" (updatespace)" in <i>Teradata Vantage™ - Database Utilities</i> , B035-1102
The spool space problems	Knowledge Article SD1000C126E
DBC.DatabaseSpace table	<i>Teradata Vantage™ - Data Dictionary</i> , B035-1092

Adjusting for Low Available Free Memory

When the amount of available free memory dips too far below 100 MB (25,000 pages), some sites have experienced issues. 40 MB of free memory is the lowest acceptable amount. You can usually avoid problems if you configure your AMPs to have at least 135 MB of OS-managed memory per AMP. You can adjust the amount of available free memory by performing the following:

- Use `ctl` to adjust the FSG Cache percent downward to make more memory available to free memory. If the system takes too much memory for FSG Cache and the OS does not use that memory, the free memory is wasted.
- If available free memory goes below 100 MB during heavy periods of redistribution (as explained later in this section), lower the value of the `RedistBufSize` field in the DBS Control Record (see “RedistBufSize” in *Teradata Vantage™ - Database Utilities*, B035-1102).

Solving Partitioning and RI Validation Errors

Use the following procedures and tools to detect and correct errors in tables that use partitioning and referencing. (For more information, see *Teradata Vantage™ - Database Design*, B035-1094.)

IF you want to ...	THEN ...
correct a partitioning expression that is causing transaction	do one of the following: <ul style="list-style-type: none"> • Change the partitioning expression • Delete the rows causing the problem

IF you want to ...	THEN ...
rollbacks due to an evaluation error (such as divide by zero)	<ul style="list-style-type: none"> Remove partitioning from the table Drop the table
find invalid table states or internal structures	run the CheckTable utility LEVEL 3 command.
regenerate only the headers in a table with partitioning	use the ALTER TABLE ... REVALIDATE statement.
validate a column-partitioned table or join index	use the REVALIDATE option of the ALTER TABLE request.
for a table with partitioning: <ul style="list-style-type: none"> Regenerate table headers Re-evaluate partition expressions Recalculate row hash values Move rows to proper AMPs and row partitions Update any SI, JI, and HI defined for the table 	use the ALTER TABLE ... REVALIDATE WITH DELETE/INSERT [INTO] statement. <ul style="list-style-type: none"> WITH DELETE deletes any rows with a partition number that is null or outside the valid range. WITH INSERT [INTO] deletes any rows with a partition number that is null or outside the valid range and inserts them into save_table. <p>Note: REVALIDATE changes the table version.</p>
find corrupt rows after running an update or delete operation using WITH NO CHECK OPTION on tables with RI constraints	submit the RI Validation Query, structured as: <pre>SELECT DISTINCTchildtablename .* FROM childtablename,parenttablename WHERE childtablename .fkcol NOT IN (SELECT pkcol FROM parenttablename) AND childtablename .fkcol IS NOT NULL;</pre> <p>This query reports every row in the Child table with an FK value that does not have a matching PK value. (FK nulls are excluded because it is not possible to determine the values they represent.)</p>
purify a Child table for which corrupt rows were reported by the RI Validation Query	delete from the Child table any reported rows as soon as possible to maintain the integrity of your database.

Incident Information Checklist

If you notice your system is having problems and you want to contact Teradata Support to report an incident, collect as much information about your system as possible. The following list is *not* exhaustive and is meant only to help you begin gathering details.

Note that not all of the questions may be applicable to your system and that other aspects of performance problems may not be addressed in this list. It is important to provide any other information that is not listed here but that you feel is important.

- Describe any performance problems and its impact.
 - Is the system hung?

- Is there a general system-wide slowdown?
- Is there a slow query or class of queries?
- Has there been a gradual deterioration?
- Are there any time-outs?

Example: The entire system runs fine for a while, then performance gradually deteriorates over a span of a few days. Marketing XYZ queries time out more often until eventually the entire system hangs. A Teradata reset clears the problem.

2. Is performance degraded for the entire system or part of the system?

- A particular node(s) or clique(s)?
- A particular subsystem like BYNET or disk array?
- A particular application or client?
- The network?
- A particular group or class of users?
- A particular job or query?
- Do you have a performance baseline to compare against?

Example: The Marketing XYZ queries, when run from a gateway connected session via a Web interface, had degraded response time from 10 minutes to approximately 30 minutes.

3. When does the problem occur?

- All the time?
- A particular time of day, day of week, week of month, month of year?
- When did this problem start? Did it start suddenly?
- Is it getting progressively worse?
- Is the performance problem intermittent and unpredictable?
- Has this problem happened before?
- Are there any other past incidents that might be relevant to the current performance situation at this site?
- Is the problem reproducible? If so, how?
- What else is running while this problem occurs?

4. Other than performance, do you observe any other symptoms? For example:

- Any unusual errors, warnings or messages, especially lately?
- MiniCylPacks?
- File space problems?
- Memory depletion?
- Lock contention?
- Network congestion, collisions, bottlenecks?
- z/OS errors, cell problems, in particular?
- Are users having problems logging into the system and the database?

- Are any third party incidents currently open?
5. What recent changes have been made to the system?
 - Recent configuration changes?
 - New hardware or new software?
 - New applications or new queries?
 - Different job mix and scheduling?
 - Have you just newly switched to a production environment?
 - Increased or different workload?
 - New or additional users?
 - New or additional data?
 - System tuning changes?
 - Changes to workload management methods?
 6. Describe the data processing environment and where Teradata fits into the context.
 - What is your system generally used for? (Decision Support, OLTP, OLAP, Ad-Hoc Queries, a mix?)
 - What other vendor applications are you using?
 - Which applications are your most critical applications?
 - Who are the users? Are there different groups of users running different applications?
 - What types of data and how often is data loaded?
 - Can you provide a high-level block diagram illustrating the Teradata system, interfaces, networks, hosts, data marts, intermediate Servers, clients, sessions, users, ODBC, and so on?
 7. Describe the Teradata System Configuration.
 - What are the versions of DBS, PDE, TCHN, TGTW, TDGSS, BYNET, and storage devices?
 - What is the number of nodes, CPU speed, AMPs per node, memory size per node, and storage per AMP?
 - Have the workload management methods been set up differently than the default settings?
 - Are there any unusual or asymmetrical aspects to the system configuration, such as different node types, different CPU speeds, different size cliques, different numbers of VPROCs per node, different storage subsystem, BYNET or other I/O components?
 8. What supporting evidence and data do you have to assist Teradata Support in troubleshooting?
 - Do you have logs for: resource usage data, access logging, the Viewpoint Lock Viewer portlet, system accounting, syschk and mboxchk, blmstats, tdnstats, event logs and other error logs.
 - Do you have baseline data?
 - If you are trying to troubleshoot a particular query, have you measured performance of the same query from different clients (that is, BTEQ versus network attached client or have you tried to isolate bottlenecks)?
 - Has there been network congestion or collisions?

Monitoring Transaction Recovery

The Rcvmanager (Recovery Manager) utility lets you monitor the backing out of incomplete transactions on tables that may be locked for access. The resulting journal is called the Transaction Journal. Rcvmanager also shows the count of records of data presently in the Down AMP Recovery Journal. This journal represents the data rows an AMP vproc must recover from the other AMP vprocs in the cluster before coming back online to the database.

The Recovery Manager utility runs only when the system is in one of the following states:

- Logon
- Logon/Quiet
- Logoff
- Logoff/Quiet
- Startup (if system has completed voting for transaction recovery)

If the system is not in one of these states, Recovery Manager will terminate immediately after you start it.

Starting Recovery Manager

To start Recovery Manager, enter `start rcvmanager` in the Supervisor interactive area.

Stopping Recovery Manager

After you start Recovery Manager, you cannot stop it with the Supervisor program STOP command. You must use the Rcvmanager QUIT; command to stop the program.

Note:

All Rcvmanager commands end with a semicolon (;).

Recovery Manager Commands

The LIST STATUS command displays information about recovery operations in progress. The processor id option provides additional detailed information about a specific down AMP.

The LIST LOCKS command displays a list of all locks currently held by online transaction recovery.

The LIST ROLLBACK TABLES command displays the list of tables which are currently being rolled back in the system. Separate listings are generated to distinguish between online transaction recovery and system recovery. Table ids can be selected from this listing for executing the CANCEL ROLLBACK ON TABLE command. In case a '*' character follows the table names then they cannot be specified in the CANCEL ROLLBACK ON TABLE command.

The LIST CANCEL ROLLBACK TABLES command displays the list of tables for which rollback processing is pending cancellation during the online transaction recovery. These tables are removed from the list when all the journal rows corresponding to the tables have been skipped on all the AMPs.

The DEFAULT PRIORITY command sets the recovery priority to Low and the rebuild priority to Medium.

The CANCEL ROLLBACK ON TABLE command is used to specify a table for which rollback processing is to be canceled for an online user requested abort or during system recovery. The DBC password is required to execute this command. Multiple tables can be specified by separating their table ids with commas.

NOTICE

The target table will be unusable after this command is issued and will become usable only when the table is dropped and created again, when the table is restored from an archived backup, or when a DELETE ALL operation is performed on that table. The CANCEL ROLLBACK command should only be used in cases where the rollback will take longer than the restore of the table or where the table is unimportant (such as a temporary table). A single table retrieve operation can be performed on the target table by using the READ OVERRIDE locking modifier on it.

Resolving Orphan or Phantom Spool Issues

In rare cases when a request completes (or is aborted), the spool file is not dropped. The leftover spool may be:

- Orphan: The spool file (tableid) incorrectly remains in existence.
- Phantom: The spool file tableid is gone, but the spool space is still allocated.

Execute the following query to determine the presence of leftover spool of either type:

```
SELECT Databasename, Vproc, CurrentSpool
FROM DBC.DiskSpace
WHERE Databasename NOT IN (SELECT Username FROM DBC.SessionInfo)
AND CurrentSpool > 0
ORDER BY 1,2
WITH SUM (CurrentSpool);
```

If this request returns rows, the next step is to run the Update Space (updatespace) utility.

The Update Space utility can also be used to correct inconsistencies in the DBC.DatabaseSpace table, which might occur because of unusual types of system failures.

From the Supervisor window issue:

```
START UPDATESPACE
```

Repairing Data Corruption

Table Rebuild is a utility that repairs data corruption. It does so by rebuilding tables on a specific AMP vproc based on data located on the other AMP vprocs in the fallback cluster.

Table Rebuild can rebuild data in the following subsets:

- The primary or fallback portion of a table
- An entire table (both primary and fallback portions)
- All tables in a database
- All tables that reside on an AMP vproc

Table Rebuild performs differently based on the type of table (for example, fallback or not).

Type of Table	Action
Fallback Tables	Delete the table header (one row table which defines a user data table) Delete specified portion of the table being rebuilt Rebuild the table header Rebuild the specified portion of the table
Non-fallback Tables	Delete the table header. Rebuild the table header.
Permanent Journal Tables	Delete data for the table being rebuilt. Rebuild the table header. Locks the table pending rebuild. Note: You must restore non-fallback data.

Note:

You can start REBUILD when the failed AMP has been fixed and brought back to the OFFLINE state.

Using Maps to Position Table Data across AMPs: All DBAs

Map Overview

Every table uses a *map* that specifies which AMPs store the rows of the table. For tables with fallback, the primary and fallback copies of the row are stored on different AMPs in the map.

You can use maps to optimize the placement of tables on the AMPs in your system or to redistribute table rows after a system expansion. Before doing this, it is important to consider some general concepts.

Vantage uses two types of maps to track which rows of a table belong on which AMP:

Contiguous map

This type of map includes all AMPs within a specified range. By default, every system has one contiguous map that includes every AMP in the system. Vantage creates contiguous maps during a system initialization, configuration, or reconfiguration.

Sparse map

This type of map includes a subset of AMPs from a contiguous map. By default, each system that enables MAPS Architecture has a 1-AMP sparse map and an n -AMP sparse map, where n is the number of nodes in the system. You can also create new sparse maps if you have appropriate database privileges.

Secondary index tables use the same map as their base (indexed) table. Tables and join indexes are assigned a map either explicitly or by default.

Sparse maps are efficient for small tables. For example, consider a table with just one row on a 1,000 AMP system. A request requiring a full-table scan would require all 1,000 AMPs in a contiguous map to read their rows. Because the table has only one row, 999 AMPs use resources to determine and report that they have no rows. However, if the one-row table uses a one-AMP sparse map, Teradata knows which AMP to read and can respond faster to a request on that table.

To determine which maps are currently defined on the system, query the DBC.MapsV or DBC.MapsVX views. For more information on Data Dictionary views, see *Teradata Vantage™ - Data Dictionary*, B035-1092.

Sparse Maps and Table Colocation

Rows from tables that have the same PI or PA and that share the same contiguous map are distributed to the same AMPs. This is called table *colocation*. Colocation provides a performance advantage when tables are joined on their PI or PA columns, because the join processing for corresponding rows happens within the same AMP.

For tables using sparse maps, there is an additional requirement for joins to take advantage of colocation.

Because tables using the same sparse map may not be stored on the same subset of AMPs, even if these tables have the same PI or PA, their corresponding rows would not benefit from colocation during joins involving the PI or PA columns. You can force rows of frequently joined tables to be distributed to the same subset of AMPs by specifying a *colocation name* when you associate the tables with the sparse map. The colocation name forces tables that use the same sparse map to be stored on the same subset of AMPs.

Scenarios for Using Maps to Move Table Data

Some tables stay the same size after they are created, while others change frequently. A table should use a map that has the number of AMPs appropriate to the table size, which leads to an even distribution of the table rows among the AMPs in the map. Consequently, it is a good idea to periodically reassess a table to ensure that it uses a map appropriate for the current size and for expected growth.

Here are some scenarios in which a DBA might want to use maps to move table data from one set of AMPs to another.

System Expansion

After you add AMPs or nodes to a system, if MAPS architecture is enabled, the Reconfiguration utility (Reconfig) creates a new map that reflects the expanded system. A DBA can then choose to move table rows to the new AMPs using either Reconfig or MAPS. There are several advantages to using MAPS instead of Reconfig:

- Reconfig requires more planned downtime.
- Reconfig requires you to move all tables in the system immediately to evenly distribute rows between AMPs. However, if you use MAPS, you can move tables into the new configuration gradually while the system is online. Some tables can be in the old map and some tables can be in the new, expanded map.

Note:

The Configuration and Reconfiguration utilities must be run under the supervision of Teradata support center personnel.

Note:

You can also use the MAPS Advisor procedures to model the moves before a system expansion. You can redo the analysis as many times as needed.

Upgrade

When you upgrade your system to Release 16.10 or later and you enable Teradata MAPS architecture, you can look for small tables that would benefit from using a sparse map.

Enabling Teradata MAPS Architecture

Systems initialized at Release 16.10 or later automatically include the ability to use more than the default map for distributing table data. However, systems upgraded to Release 16.10 or later have this ability

disabled. To be able to use more than the default map, ask your Teradata representative to enable Teradata MAPS Architecture.

Initial Setup for Managing Maps

There are prerequisites for using maps to distribute table data across AMPs:

- User DBC must grant the LOGON privilege to user TDMaps. TDMaps is a system database that contains the information and SQL stored procedures used for managing maps.
- After the first logon to TDMaps (the initial password is tdmadmin), change the password for TDMaps.
- Give TDMaps enough space to store rows in the ActionsTbl and ActionHistoryTbl. A rough guideline is the size of TVM. The following example determines the size of TVM:

```
SELECT SUM(CurrentPerm) FROM DBC.TableSizeV WHERE TableName='TVM';
```

- Grant DBAs privileges on TDMaps:
 - To grant the use of TDMaps procedures:

```
GRANT EXECUTE PROCEDURE ON TDMaps TO <User>;
```

- To grant the ability to query TDMaps tables:

```
GRANT SELECT ON TDMaps TO <User>;
```

- To grant the ability to modify TDMaps tables:

```
GRANT INSERT, UPDATE, DELETE ON TDMaps TO <User>;
```

- Grant DBAs privileges on maps:
 - To grant the privileges to create and drop maps, for example:

```
GRANT CREATE MAP TO Roger WITH GRANT OPTION;
```

```
GRANT DROP MAP TO Roger WITH GRANT OPTION;
```

- To grant the existing contiguous or sparse map to a user or role, for example:

```
GRANT MAP SomeMapName TO Roger WITH GRANT OPTION;
```

Maps must be granted before they can be used in an SQL statement. This requirement does not apply to default maps.

When a user creates a map, the user is granted that map by default. User TDMaps is automatically granted the default sparse maps.

A Simple Process for Changing the Map for a Table

The following process is a simple way to change the map a table uses:

- Query the view TableToSparseMapSizingV(X). This view recommends a map for each table and identifies when the current map should be changed.

For example, the following request produces a list of tables that Teradata recommends use a sparse map but are now using a contiguous map:

```
SELECT DatabaseName, TableName FROM TDMaps.TableToSparseMapSizingV WHERE
RecommendedMap LIKE '%Sparse%' AND CurrentMapKind = 'C';
```

The following request produces a list of tables that Teradata recommends use a different map type (either sparse or contiguous) than what they are using now:

```
SELECT DatabaseName, TableName FROM TDMaps.TableToSparseMapSizingV WHERE
MapOk = 'N';
```

- If you decide that a table should use a different map, either issue an ALTER TABLE request to name a different map for the table or use the Mover stored procedure, which is described later in this section. To use ALTER TABLE, you must have the DROP privilege on the table you want to alter. For more information on the ALTER TABLE request for maps, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Viewpoint Portlet for Analyzing and Optimizing Map Placement

Use the Teradata Viewpoint portlet MAPS Manager to determine if tables are using optimal maps and, if necessary, switch them to more suitable maps on a flexible schedule.

For more information on this portlet, see the *Teradata® Viewpoint User Guide*, B035-2206.

Introduction to the Advisor and Mover Tools

Within the TDMaps system database, there are many SQL stored procedures for automating system expansion and map assignments. The procedures in TDMaps fall into two main categories: Advisor and Mover.

- **Advisor** procedures analyze tables to recommend the best map for each table. The recommendations can optionally be customized and used as input for the Mover.
- **Mover** procedures redistribute the data for a specified list of tables based on new maps. Mover can use multiple worker sessions to achieve the desired level of concurrency. One manager session schedules the execution of groups of related actions and coordinates the worker sessions.

Enabling Step Logging To Aid in Map Analysis

Before analyzing how tables use maps, you can optionally enable DBQL step logging. The information step logging collects helps Teradata decide which tables to keep together for join processing efficiency when redistributing tables. For example:

```
BEGIN QUERY LOGGING WITH STEPINFO ON ALL;
```

Teradata recommends leaving logging on for 7 days to capture a set of queries that are representative of the workload on your system.

If there is no step logging, Teradata groups tables by database when redistributing them.

Excluding Objects from Map Reassignments

To exclude tables, join indexes, or hash indexes from the list to analyze and reassign to different maps, use procedures to create an empty list of exclusions and then add objects to it. In the following example, a DBA creates an exclusion list named 'MyExclusions' consisting of all tables in database 'Payroll', an individual table 'Employee.Withholding', and all tables in database 'Taxes' whose names begin with 'Rate'.

```
CALL TDMaps.CreateExclusionListSP('MyExclusions', NULL, :ExclusionListId);
CALL TDMaps.AddExclusionListEntrySP('MyExclusions',
'Payroll', NULL, :NumObjectsAdded);
CALL TDMaps.AddExclusionListEntrySP('MyExclusions',
'Employee', 'Withholding', :NumObjectsAdded);
CALL TDMaps.AddExclusionListEntrySP('MyExclusions',
'Taxes', 'Rate%', :NumObjectsAdded);
```

The result is that these tables are marked as exclude in the ActionsTbl and will not be considered for redistribution to other AMPs. The list of excluded objects is visible in the view TDMaps.ExclusionLists[V][X].

For more information on these procedures, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Analyzing Maps

For maximum processing efficiency, tables need to be on the AMPs best suited to their size and join functions. To determine if tables should be moved to other AMPs, follow these steps to analyze how effectively tables are assigned to maps. These maps determine table placement on AMPs:

1. Decide which tables to analyze for map efficiency.
Consider analyzing table groups by either database, application, or query band.
2. Create a list of maps.
To do this, call CreateMapListSP to create the empty list. For example:


```
CALL tdmmaps.CreateMapListSP('MyMapList', NULL, :ListId);
```

3. If you are doing this analysis to help you plan a system expansion, include pre-expansion (planned) maps in the map list and define the number of nodes and AMPs in the planned expanded system. To do this, call `CreateExpansionMaps` to insert the three planned maps into `TDMaps.Maps`. For example, if you are expanding the system from two to four nodes, and the number of AMPs on the expanded system is 40:

```
CALL
TDMaps.CreateExpansionMaps(4,40,:ContiguousMapString,:OneAmpSparseMapString,:TotalNodesSparseMapString);
```

This creates the following placeholder names for the three planned maps, which must be renamed after the system expansion (see [Renaming Maps after a System Expansion](#)):

- `PREEXPANSIONMAP_CONTIGUOUSMAP_4NODES`
- `PREEXPANSIONMAP_1AMPSPARSEMAP_4NODES`
- `PREEXPANSIONMAP_4AMPSSPARSEMAP_4NODES`

4. Add an entry to the map list, which can be a list of maps or a map name. To do this, call `AddMapListEntrySP`. For example:

```
CALL tdmmaps.AddMapListEntrySP('MyMapList', 'MySparseMap');
```

5. Build a list of objects to analyze. To do this, first call `CreateObjectListSP` to create the empty list. For example:

```
CALL tdmmaps.CreateObjectListSP('MyObjectList', NULL, :ObjectListId);
```

Next, call `AddObjectListEntrySP` to add tables to the list. For example:

```
CALL tdmmaps.AddObjectListEntrySP('BillsList','Bill%',NULL,:NumObjectsAdded);
```

6. Optionally, exclude tables from the list. This is useful when you want to consider an entire database for map reassignment but exclude a few tables in that database. First, call `CreateExclusionListSP` to create an empty list of tables to exclude. Then call `AddExclusionListEntrySP` to add selected tables to the exclusion list. For example:

```
CALL tdmmaps.CreateExclusionListSP('BillsList', NULL, :ListId);
```

7. Create a list of recommendations for reassigning tables to different maps. To do this, call `AnalyzeSP` to put recommended actions in `TDMaps.ActionsTbl`. For example:

```
-- Analyze tables in Personnel object list for queries logged over the
-- last 7 days and recommend actions for moving them into MyNewMap
--
CALL TDMaps.AnalyzeSP(
    'MyNewMapList', 'Personnel',NULL,
```



```
CAST(CURRENT_TIMESTAMP - INTERVAL '7' day AS TIMESTAMP),
CURRENT_TIMESTAMP,
'DBC', 'MyNewMapActions', :NumAlters, :NumExcludes, :NumLogEntries);
```

Analysis is typically very fast if you did not enable DBQL step logging. If the analysis is considering DBQL input, the speed of the analysis depends on how many rows are in the table.

Note:

For more information on these procedures, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

ActionsTbl

The results of AnalyzeSP appear in TDMaps.ActionsTbl, with one row for every table that was analyzed for optimal map use. Tables are either marked as ALTER (meaning that Teradata recommends assigning the table to a new map) or EXCLUDE (meaning that the table is already assigned to a suitable map or does not meet the criteria for map reassignment). If DBQL step logging was considered in the analysis, tables are grouped by recommended proximity. Without DBQL input, tables are grouped by database. Within each grouping, the order is based on table size.

Teradata recommends that tables are assigned to the smallest sparse map possible. If the table is too large for a sparse map, Teradata recommends that the table is assigned to a contiguous map. ActionsTbl also describes issues that may prevent recommended map reassignments from occurring.

To examine the Advisor results, query ActionsTbl. For example:

```
SELECT Action, DatabaseName, TableName, GroupOrder
FROM TDMaps.ActionsTbl
WHERE ActionListName = 'MyMoveTableList'
ORDER BY 1, 4, 3, 2;
```

Result:

Action	DatabaseName	TableName	GroupOrder
Alter	db2	JoinTab1	1
Alter	db2	JoinTab2	1
Alter	db2	OtherTab	2
Alter	db1	LargeTab	3
Alter	db1	MediumTab	3
Exclude	db1	TinyTab	NULL
Exclude	db2	SmallTab	NULL
Exclude	db3	SmallToMediumTab	NULL

Reviewing Recommended Map Actions

It is important to review the actions Teradata recommends for reassigning tables to different maps. Look for issues that may need to be resolved before any recommended changes can occur. Also consider whether the recommended actions are best for your site.

1. Isolate any issues found during map analysis.
Query the issues in ActionsTbl. For example:

```
SELECT * FROM tdmmaps.ActionsVX WHERE Issues='Y' OR Action <> 'alter';
```

The Description column describes the issue.

2. Check for space issues that might prevent recommended map reassignments. In particular, see FractionOfDBFreePerm, which lists the space that will be left in a database after tables are redistributed. For example, if your database has 100 MB of perm space and the table being redistributed is 1 MB, FractionOfDBFreePerm is .99, indicating that 99% of permanent space in the database remains free after table redistribution.

It is particularly important to notice this value after a system expansion because the table header has to be written to each AMP. For example, if you upgrade from a 2-AMP system to a 100-AMP system the table header will use 50 times more space than before, even though the table is the same size.

Reassigning a table to another map requires an INSERT-SELECT, which uses 2 times the amount of permanent space the table requires.

3. Give the table being reassigned additional permanent space if necessary.
4. Look for other issues.

These can include the following common issue types, for example:

- The table uses a sparse map and will be reassigned to a contiguous map.
- The table has permanent journaling enabled, so the map reassignment will fail. Tables with permanent journals must use the default map.
- The join or hash index will be invalid when the table is assigned to a new map. The DBA must drop and recreate the join or hash index after the map reassignment. This occurs when the join or hash index is based on these types of tables:
 - Primary AMP index (PA)
 - Nonpartitioned primary index (NOPI)
 - Column-partitioned primary index (CPPI)
- A user-created join or hash index does not transfer when a table is assigned to a new map. The DBA must issue a separate ALTER TABLE request for a user-created join or hash index.

Note:

System-defined join indexes automatically use the new map.

Customizing the List of Recommended Map Actions

You may want to customize the list of recommended actions in ActionsTbl before reassigning tables to different maps. The reason for this is ActionsTbl is input to the process that changes the map that a table uses. You may want to change any of these items in ActionsTbl:

- Table grouping
- Table order within the group
- The list of recommended actions

To change the planned actions in the ActionsTbl, either update the table or delete rows. To add actions, insert rows. When executing DML statements directly against ActionsTbl, users must include a WHERE condition on column *ActionListName* to limit the changes to a designated action list. The following steps are optional.

1. Change the recommended actions.

Update the ActionsTbl. For example, if user Fisk has a customer table that has only a few rows but a lot more will be added, AnalyzeSP may recommend a sparse table for it:

```
sel * from tdmappings.actionsv where ActionListName = 'Fiskobject' and
databasename='Fisk' and tablename='customer';
```

Result:

```
*** Query completed. One row found. 21 columns returned.
*** Total elapsed time was 1 second.

      ZoneName ?
      Action Alter
      Origin A
DatabaseName FISK
      TableName customer
      SourceMap TD_Map2
DestinationMap TD1AMPSPARSEMAP_1NODES
      CoLocateName ?
ActionSQLText ALTER TABLE FISK.customer, MAP=TD1AMPSPARSEMAP_1NODES
ActionListName Fiskobject
      GroupOrder 1.00
      ActionOrder 14.00
      TableSize 1024
FractionOfPermDBFree 9.99999630767549E-001
      UDICnt ?
      QueryCnt ?
      PKJoinCnt ?
LastAlterTimeStamp 2017-02-06 14:55:02.440000-08:00
LastAlterUID 10000000
```


Issues N

ActionsTblDescription Table is very small. One-AMP sparse map is recommended. This Action is from running Analyzer without DBQL analysis.

The DBA knows that the table is growing too fast to use a sparse map. This SQL can change the recommended map for the table:

```
UPDATE tdmaps.ActionsTbl
  SET ActionSQLText = 'ALTER TABLE FISK.customer, MAP=TD_Map3',
      DestinationMap = 'TD_Map3',
      Origin         = 'U'
 WHERE ActionListName = 'fiskobject'
    AND DatabaseName  = 'fisk'
    AND TableName     = 'customer';
```

The following example illustrates another situation in which you might want to update the ActionsTbl:

```
UPDATE tdmaps.ActionsTbl SET Action = 'Exclude', ActionsTblDescription =
'Slocum marks this exclude because this table will be dropped tomorrow so why
move it?'
WHERE DatabaseName = 'Slocum' AND TableName = 'Customer' AND ActionListName
= 'MoveTablesToMap2';
```

Note:

Teradata recommends using UPDATE instead of DELETE because UPDATE logs the reason for the change in the history table.

2. Change the order in which groups of tables are reassigned or change the order in which tables are reassigned within a group. Tables are grouped by database if you do not use DBQL step logging. If you do use DBQL step logging, tables are grouped by how often they are joined together. To add a new group between existing ones, use a fractional value in the GroupOrder column. To add new actions within a group, use a fractional value in the ActionOrder column.

The following is an example of ActionsTbl rows that are listed from top to bottom in their priority of execution. Note the following information about the table:

- Rows whose Origin value is 'A' were generated by AnalyzeSP and those with a 'U' (or NULL) value were added later by manual INSERT statements.
- Rows for tables 't6' and 't7' represent user-defined actions within a new group whose assigned GroupOrder value of 1.5 falls between existing groups 1 and 2.
- Rows for tables 't10' through 't20' represent user-defined actions added to existing group 2. The fractional digits assigned to their ActionOrder values (1.1 through 1.11) result in them being ordered between existing ActionOrder values 1 and 2.

Action	Origin	TableName	GroupOrder	ActionOrder
--------	--------	-----------	------------	-------------

Alter	A	t1	1	1
Alter	A	t2	1	2
Alter	A	t3	1	3
Alter	U	t6	1.5	1
Alter	U	t7	1.5	2
Alter	A	t4	2	1
Alter	U	t10	2	1.1
Alter	U	t11	2	1.2
Alter	U	t12	2	1.3
Alter	U	t13	2	1.4
Alter	U	t14	2	1.5
Alter	U	t15	2	1.6
Alter	U	t16	2	1.7
Alter	U	t17	2	1.8
Alter	U	t18	2	1.9
Alter	U	t19	2	1.10
Alter	U	t20	2	1.11
Alter	A	t5	2	2

Renaming Maps after a System Expansion

If you are expanding a system, run the map analysis procedures at either of these times:

- After the system expansion
- Before the system expansion

Perform the following procedure only in the second case, when you ran the analysis procedures before the system expansion and are not going to run them again afterward.

1. Rename the planned maps created with the CreateExpansionMaps procedure with the actual map names to be used in the Mover procedures. To do this, call PostExpansionAction. In the following example, you have expanded the system to 4 nodes and 40 AMPs. The PostExpansionMap contiguous map is called TD_Map1005:

```
CALL TDMaps.PostExpansionAction(4, 'TD_Map1005');
```


Sessions Used for Reassigning Maps

The stored procedures that change the table-to-map assignments run in SQL command-line tools, such as Teradata Studio or BTEQ. You will need to open these BTEQ or Teradata Studio sessions before changing the maps that tables use:

- One session for the managing process
- One session for each of the worker processes

For example, if you want 4 worker sessions, open 1 BTEQ window for the manager and another to create all the worker sessions. You could achieve that like this:

```
.SET SESSIONS 3
.LOGON machine/user,password
.REPEAT 3
```

Note:

Teradata recommends using 3 worker sessions for the Mover procedure.

Estimating the Time Needed to Reassign Maps

To see an estimate of the time needed for planned map reassignments, you can use any of the following methods:

- Add the table to DBC.ReconfigRedistOrderTbl and run the Reconfig estimator.
- Experiment with INSERT-SELECT or ALTER TABLE on sample data.
- Use DBQL step logging to log CPU and I/O for each step. Scale up the numbers to arrive at an estimate.

Use MONITOR SESSION to compare estimates with actual times.

Decisions Required before Reassigning Maps

Before you can change which maps tables use, you need to decide how these changes will occur.

The Time Limit Decision

The Mover process can operate either with or without a time limit. The time limit is set using the TimeLimit parameter of ManageMoveTablesSP.

- Without a time limit, planned map reassignments listed in ActionsTbl will occur until either all the planned reassignments complete or until the DBA manually stops the reassignments.
- With a time limit, the Mover process might skip a group of tables if the time limit is almost reached and instead reassign another, smaller group of tables that can be processed in the available time. One example of using a time limit is if you want to limit actions to the weekend and have map reassignment work expire during normal business hours.

The Serial or Parallel Processing Decision

Before you can start the Mover process, you need to decide whether Mover should operate in serial or parallel. To specify that, choose either the ManageMoveTablesSP output parameter SerialTableActionList or ParallelTableActionList.

- **Serial processing:** It is a good idea to use serial processing when changing maps for large tables. Changing the map for one large table at a time is a strategy less likely to run out of space than changing the maps for several large tables at once. The ALTER TABLE request that changes the map for a table first makes a copy of the table and then deletes the old copy. ALTER TABLE requires 2 times the space of the original table to complete map reassignment. Column-partitioned and NoPI tables require 3 times the space of the original table. If you use only 1 worker session, Mover reassigns only one large table at a time. If you try to do more than that, you risk running out of space.
- **Parallel processing:** It is a good idea to use parallel processing when you are changing the map for many smaller or medium-sized tables. Parallel processing is faster than serial processing and is not risky when table sizes are moderate.

The Workload Decision (Optional)

If you use Viewpoint to reassign tables to maps, Viewpoint sets the query band automatically. If you execute procedures to reassign tables to maps, you can set the query band to place this work into a dedicated workload. To do that, in each manager and worker Mover session, use this query band name-value pair: WDClassification=MoveTableToMap. For example:

```
SET QUERY_BAND = 'WDClassification=MoveTableToMap;' FOR SESSION;
```

- **Results of specifying the workload:** If you set the query band as shown in the preceding example, the Mover session will execute in the WD-MapsMover workload. This workload is dedicated to reassigning tables from one map to another. The default priority of this workload is Timeshare High.
- **Results of not specifying the workload:** If you do not set the query band, TASM workload classification rules will determine the workload and the corresponding priority of the Mover process. For more information, see *Teradata Vantage™ - Workload Management User Guide*, B035-1197.

You can move the WD-MapsMover workload to a different Timeshare access level using Viewpoint Workload Designer. For example, if you wanted to run map reassignments in the background, you could move WD-MapsMover to Timeshare Low. If you wanted to run this work at a higher priority, you could move WD-MapsMover to Timeshare Top.

Reassigning the Map for a Table

Use the following procedure to reassign a table to another map to maximize processing efficiency. The Mover process uses stored procedures to act on the recommendations in ActionsTbl.

1. In a BTEQ or Teradata Studio window, call the manager procedure ManageMoveTablesSP to schedule the execution of map actions previously generated by AnalyzeSP.

This procedure moves a group of tables out of ActionsTbl and into a queue table. In the following example, the DBA schedules execution of the actions in MyNewMapActions and imposes a time limit of 12 hours (720 minutes):

```
Session #1:
CALL TDMaps.ManageMoveTablesSP(NULL, NULL, 'MyNewMapActions', 720, 'Y',
    :ActionsCompleted, :NumErrors, :TimeExpired);
```

In the following example, a DBA uses a scheduler to call the manager procedure in BTEQ:

```
CALL TDMaps.ManageMoveTablesSP(DATE+SESSION, NULL, 'MyNewMapActions', 4*60,
'Y', :ActionsCompleted, :NumErrors, :TimeExpired);
```

Note:

There can be only one instance of ManageMoveTablesSP active in Vantage at a time.

For more information on ManageMoveTablesSP, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

2. Call the worker procedure MoveTablesSP in one or more BTEQ or Teradata Studio windows to start reassigning maps.
Each call to MoveTablesSP requires a separate session. One way to do this is by using the BTEQ commands SET SESSIONS and REPEAT. MoveTablesSP issues an ALTER TABLE request for each row in the queue table. In the following example, the DBA starts two concurrent worker sessions, one in each BTEQ window:

Session 2:

```
CALL TDMaps.MoveTablesSP('P',NULL,NULL,NULL,NULL);
```

Result:

```
Procedure has been executed.
*** Total elapsed time was 11 hours 3 minutes and 5 seconds.
```

Session 3:

```
CALL TDMaps.MoveTablesSP('P',NULL,NULL,NULL,NULL);
```

Result:

```
Procedure has been executed.
*** Total elapsed time was 11 hours 19 minutes and 45 seconds.
```


This procedure records completed or in progress actions in TDMaps.ActionHistoryTbl.

For more information on MoveTablesSP, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

3. Optionally, call MonitorMoveTablesSP to monitor the progress of the map reassignments. This procedure reports on the progress of both the manager and worker sessions. It reports the number of tables altered, the number left to alter, and the percentage complete. For example:

```
CALL tdmaps.MonitorMoveTablesSP('MyNewMapActions', :NumTables,
                                :NumComplete, :NumInProgress, :NumWaitingToStart, :PercentComplete);
```

Result:

```
*** Procedure has been executed.
*** Warning: 3212 The stored procedure returned one or more result sets.
*** Total elapsed time was 1 second.
```

```
      NumTables      2712
      NumComplete    2526
      NumInProgress   1
      NumWaitingToStart 185

PercentComplete      .93
```

```
*** ResultSet# 1 : 1 rows returned by "TDMAPS.MONITORMOVETABLELESSP".
```

```
      ZoneName ?
      JobNumber      2016042801
      Action Alter
      Status InProgress
      DatabaseName MHM
      TableName TAB21_PI
      GroupOrder      1.00
      SourceMap TD_Map1
      DestinationMap TD_Map2
      ActionSQLText ALTER TABLE MHM.TAB21_PI, MAP=td_map2
      ActionListName MyNewMapActions
      TableSize      1118482
      FractionOfPermDBFree 9.99999999610461E-001
      Issues N
      PrevGroupsBytesPerSec
      EstElapsedTimeInSeconds ?
      ElapsedTime ?
```



```

StartTime 2016-02-02 12:22:45.270000+00:00
EndTime                                     ?
ErrorCode                                0
RunUID 0000F903
ActionsTblDescription Move table for system expansion.
Description ?
WorkerId                                1

```

For more information on MonitorMoveTablesSP, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Limiting the Resources for Map Reassignments

To limit the resources used by an active map reassignment job, reduce the number of sessions processing the map use changes. In the following example, it is now 8 a.m., the system is busy, and the DBA stops 5 of the 8 Mover sessions that started at 6 p.m. last night:

```
CALL TDMaps.StopMoverSP(5);
```

StopMoverSP does not abort any ALTER TABLE requests in progress.

For more information on StopMoverSP, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Stopping Map Reassignments

You can stop the process of changing which maps tables use either gracefully or abruptly.

Stopping the Map Reassignments Gracefully

To stop a map reassignment job gracefully, call the StopMoveTablesSP procedure and specify N for the StopQueuedActions parameter. For example:

```
CALL TDMaps.StopMoveTablesSP('N',
                             :NumActionsStopped, :NumQueuedActionsStopped);
```

This allows all tables in the currently queued group to complete their map changes. A graceful stop keeps related groups of tables using the same map.

The StopMoveTablesSP procedure is useful if you did not specify a time limit for map reassignments or you want to stop a map reassignment job early before the time limit expires.

To resume reassigning maps after a stop, call ManageMoveTablesSP again with the same TableActionList. ManageMoveTablesSP automatically restarts the stopped and still pending actions in the list.

For more information on StopMoveTablesSP, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Stopping Map Reassignments Abruptly

To stop a map reassignment job immediately, call the procedure `StopMoveTablesSP` and specify `Y` for the `StopQueuedActions` parameter. For example:

```
CALL TDMaps.StopMoveTablesSP('Y',
                             :NumActionsStopped, :NumQueuedActionsStopped);
```

The `ActionHistoryTbl` records any already queued actions that are stopped .

Note:

If a table group is interrupted during the map reassignment process, related tables may be assigned to different maps and may reside on different AMPs for a long time, slowing down request processing. For this reason, Teradata recommends stopping map reassignments gracefully, allowing groups of related tables to stay together.

Do not stop the map reassignment job by issuing `ABORT SESSION LOGOFF`. That causes the manager session to hang. Calling `StopMoveTablesSP` releases the manager session. After running `StopMoverSP`, you need to manually abort the worker sessions.

For more information on `StopMoveTablesSP`, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Managing Restarts

If the system restarts during the Advisor or Mover process, a DBA has to clean up internal tables before the interrupted process can be restarted.

Restart Cleanup during the Advisor Process

If an abort or system restart interrupts the Advisor process, call `CleanupAnalyzerSP` to clean up internal tables. For example:

```
CALL TDMaps.CleanupAnalyzerSP(NULL);
```

For more information on `CleanupAnalyzerSP`, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Restart Cleanup during the Mover Process

If an abort or system restart interrupts the Mover process, any transactions in progress are rolled back when the database is back online. Call `CleanupMoveTablesSP` to clean up internal tables left over after the abort or restart. For example:

```
CALL TDMaps.CleanupMoveTablesSP();
```


If the system restarts, a DBA needs to manually check the status of the tables that were being assigned to new maps when the restart occurred. The tables may now use new maps, but the status in ActionHistoryTbl may not have been updated yet.

Note:

The CleanUpMoveTablesSP procedure may mark tables as aborted that were actually completed. To remedy this, issue a SHOW TABLE request on any tables that are shown as being in progress in the ActionHistoryTbl. The map SHOW TABLE/JOIN/HASH INDEX returns indicates whether or not the map reassignment completed before the status was marked Complete.

For more information on CleanUpMoveTablesSP, including the syntax, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Other Cleanup Tasks

After you have completed assigning tables to other maps, cleanup work may be required, such as handling a secondary contiguous map and cleaning up the ActionHistoryTbl.

Handling a Second Contiguous Map after a System Expansion

After a system expansion, there are 2 contiguous maps: the old map without the new nodes and associated AMPs and the new map with the new resources. After all the tables and databases are reassigned to the new map, drop the older contiguous map (the default is TD_Map1) using a DROP MAP request to prevent accidental use of this old map.

Removing Historical Data

Historical data in TDMaps tables can take up too much space. To clean up old data and regain space, periodically run TruncateHistorySP. For example:

```
call tdmmaps.truncatehistorysp(date '2016-01-01', :numdeletedrows);
```

Result:

```
*** Procedure has been executed.
*** Warning:  3212: The stored procedure returned one or more result sets.

NumDeletedRows
-----
              35

*** ResultSet# 1 : 3 rows returned by "TDMAPS.TRUNCATEHISTORYSP".
```


Table	NumRowsDeleted
-----	-----
ActionHistory	10
LogTbl	20
AnalyzeLogTbl	5

Adjusting Space Limits and Skew Settings

After you have expanded your system, analyzed tables for map usage, and moved tables to other AMPs as needed, analyze whether space limits and skew settings need adjustment. Use the AdjustSpace procedure to adjust the permanent space quota and perm skew limit for one or more databases. This procedure adjusts the perm space quota to be as close as possible to current perm use while leaving some extra buffer space for future additions to the database. The following example adjusts space settings for all the databases in the system with the default buffer percentage:

```
CALL TDMaps.AdjustSpace(NULL, NULL, NULL, OutInfo);
```

This example sets the extra buffer space for the FINANCE database to 5%:

```
CALL TDMaps.AdjustSpace('FINANCE', 5, NULL, OutInfo);
```

TDMaps.SpaceSettingsTbl contains the default values for the AdjustSpace procedure, which a DBA can change as needed. The default buffer space is 10% and the rounding multiple for skew is 5%.

DBAs can use the AddDBInExcludeList macro to exclude databases that do not need space readjustments from the AdjustSpace procedure. The following example adds database SALES_DB to an exclusion list, so AdjustSpace skips SALES_DB when it is executing on all databases:

```
EXEC TDMaps.AddDBInExcludeList('SALES_DB');
```

DBAs can also use the DelDBInExcludeList macro to include a database previously excluded from the AdjustSpace procedure. The following example removes SALES_DB from an exclusion list so that it is not skipped when AdjustSpace is executing on all databases:

```
EXEC TDMaps.DelDBInExcludeList('SALES_DB');
```

Note:

You can use the AdjustSpace procedure on a system that has disabled global space accounting (the DBS Control utility flag LegacySpaceAcctg is set to TRUE). However, AdjustSpace will report an error if it tries to set a non-zero skew value for a system using legacy space accounting.

For more information, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Performance Management: All DBAs

This section presents a quick reference guide for the setup, monitoring, and management of database performance.

Implementing Performance Management

- Follow recommended database and query design practices:
 - For best practice design of database components for optimal performance, see *Teradata Vantage™ - Database Design*, B035-1094.
 - For best practice design of database queries, see *Teradata Vantage™ - SQL Fundamentals*, B035-1141, *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142, and *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145.
- Set up recommended performance management tools. See [Performance Management Setup](#).
- Manage query performance. See [Managing Query Performance](#).
- Manage the database workload. See [Managing the Database Workload](#).
- Manage database resources. See [Managing Database Resources to Enhance Performance](#).
- Do performance-related periodic maintenance. See [Periodic Maintenance and Performance](#).

Performance Management Setup

The following table describes set up tasks that must be completed to use the performance management techniques described elsewhere in this section.

Tool	Setup Task
Resource Usage Data (ResUsage)	<p>Turn on ResUsage data collectors to enable system data gathering and provide information to various Teradata Viewpoint system management portlets.</p> <p>Note:</p> <p>In addition to Teradata Viewpoint, you can use ResUsage macros to examine the collected data.</p> <p>For information, see <i>Teradata Vantage™ - Resource Usage Macros and Tables</i>, B035-1099.</p>
Lock Logging	<p>Use the WITH LOCK =<i>n</i> log option for the BEGIN/REPLACE QUERY LOGGING statement to log any lock contention longer than <i>n</i> centiseconds. The minimum acceptable value for <i>n</i> is 5.</p> <p>You can access the DBQL lock log through the Viewpoint lock viewer portlet or query the system table DBC.DBQLXMLLOCKTbl or the view DBC.QrylockLogXMLV.</p> <p>For information, see Periodic Maintenance and Performance.</p>

Tool	Setup Task
Teradata Viewpoint	Configure Teradata Viewpoint, the primary Vantage system management tool, to: <ul style="list-style-type: none"> • Monitor resource usage, query activity, workload and detect problems. • Set thresholds that help to identify performance problems. • Set up alerts to inform you when thresholds are exceeded. For details, see the <i>Teradata® Viewpoint User Guide</i> , B035-2206.
Space allocation	Allocate space according to best practice recommendations, including a spool space reserve. See: <ul style="list-style-type: none"> • Managing Space: Operational DBAs. • “Working with System-Level Space Allocation” in <i>Teradata Vantage™ - Analytics Database Security Administration</i>, B035-1100.
Collect Statistics	Collect statistics on table columns and recollect when necessary to provide the Optimizer with the information necessary to create optimal query execution plans. See Improving Query Performance Using COLLECT STATISTICS: Application DBAs .
Database Query Logging (DBQL)	Set up DBQL to collect data about queries for use in query optimization and detection of bad queries. See How to Use DBQL .

For information on other useful tools, see [Troubleshooting: Operational DBAs](#).

Monitoring the System

Measuring System Conditions

The following table summarizes recommended system conditions to measure.

System Conditions	What to Use	What to Collect	How to Use
Response time	Heartbeat queries	<ul style="list-style-type: none"> • Response-time samples • System-level information 	Saved samples can be used to: <ul style="list-style-type: none"> • Track trends • Monitor and alert • Validate workload management methods
	Baseline testing	<ul style="list-style-type: none"> • Response-time samples • Execution plans 	<ul style="list-style-type: none"> • Check for differences +/- in response time. • Check EXPLAINs if degradation is present. • Collect / keep information for comparison when there are changes to the system.
	Database Query Log (DBQL)	Application response time patterns	Track trends: <ul style="list-style-type: none"> • To identify anomalies

System Conditions	What to Use	What to Collect	How to Use
			<ul style="list-style-type: none"> To identify performance-tuning opportunities For capacity planning
Resource utilization	Resource Usage	<ul style="list-style-type: none"> ResNode Macro set SPMA summarized to one row per node 	Look for: <ul style="list-style-type: none"> Peaks Skewing, balance
	AMPUUsage	CPU seconds and I/O for each unique account	Use this to quantify heavy users.
	DBQL	CPU, I/O, spool used, rows returned	
Data growth	<ul style="list-style-type: none"> Script row counts Permspace 	Summary row per table once a month	Look at trends.
Changes in data access	DBQL	Summary row per table and the number of accesses once a month	Look for increases in access, trends.
Increase in the number of active sessions	LogOnOffV, acctg	Monthly and summarized number of sessions	Look for increases in concurrency and active users.
Increase in system demand	DBQL	Query counts and response times, plus other query information	Look for trends, including growth trends, and measure against goals.
	Resource Usage	System demand	
	AMPUUsage	Queries with errors or queries that have been canceled	

Using Alerts to Monitor the System

The Teradata Viewpoint alerting monitors Vantage and automatically invokes actions when critical (or otherwise interesting) events occur.

- Teradata Viewpoint alerting can generate alerts based on the following events:
 - System
 - Node
 - Vproc
 - Session
 - SQL

- Manual
- Canary query
- System health
- Teradata Viewpoint alerting can:
 - Send email to an administrator
 - Send an SNMP trap
 - Run a program
 - Run a BTEQ script
 - Write to the alert log

Suggested Alerts and Thresholds

The following table lists key events and the values that constitute either warning or alerts.

Type	Event	Warning	Critical
CPU saturation	Average system CPU > x%	(x = 95)	
I/O saturation	CPU+WIO > x% and WIO > y%	(x = 90, y = 20)	
Query blocked	Query or Session blocked on a resource for longer than x minutes, and by whom	(x = 60)	
Entire system blocked	Total number of blocked processes > x		(x = 10)
User exceeding normal usage	Number of sessions per user > x (with an exclusion list, and custom code to roll up sessions by user)	(x = 4)	
“Hot Node” or “Hot AMP” problem	Inter-node or inter-AMP parallelism is less than x% for more than 10 minutes	(x=10)	
Disk space	Disk Use > x% (vproc)	(x=90)	(x=95)
Product Join	Average BYNET > x% (system)	(x=50)	
System restart	Restart		SNMP
Node down	Node is down		SNMP
Heartbeat query	Timeout		SNMP

Weekly and/or Daily Reports

Weekly reports provide data on performance trends.

To make weekly and/or daily reporting effective, Teradata recommends that you establish threshold limits. Note that filtering on key windows having a 24-hour aggregate of weekly aggregates that includes low-used weekends and night hours distorts the report.

Furthermore, Teradata recommends that you analyze the longest period possible. This avoids misleading trend data by softening temporary highs and lows.

Examples of some weekly reports and their sources include:

- CPU AV & Max from ResUsage
This report provides data on the relationship between resources and demand.
- CPU seconds by workload (account) from AMPUsage.
- Throughput by Workload from DBQL
This report provides data on how much demand there is.
- General Response times from Heartbeat Query Log
This report provides data on the general responsiveness of the system.
- Response Time by Workload from DBQL
This report provides data on how fast individual queries were responded to.
- Active Query & Session Counts by Workload
This report provides data on how many users and queries were active and concurrent.
- CurrentPERM
This report provides data on how data volume is or is not growing.
- Spool
This report provides data on how spool usage is or is not growing.

Note:

These reports can, of course, be run daily.

Collecting Statistics

Guidelines for Collecting Statistics

Use the COLLECT STATISTICS statement to collect statistics on the columns used in predicates or GROUP BY clauses. When the table size grows more than 10%, you should recollect statistics on the table. Recollection covers all columns on which statistics were initially collected.

Task	Guideline
Collect UPI statistics on small tables	If you collect no other statistics on the table, collect UPI statistics. The table is small (that is, 100 rows/AMP).
Collect NUPI statistics	The NUPI is: <ul style="list-style-type: none"> • Fairly or highly unique, and

Task	Guideline
	<ul style="list-style-type: none"> • Used commonly in joins, or • Skewed • Small
Collect NUSI statistics on all NUSIs	<p>The Optimizer can use the NUSI in range scans (BETWEEN ... AND). With statistics available, the system can decide to hash on the values in the range if demographics indicate that to do so would be less costly than a full table scan. Without statistics, the index will likely not be used and, in some cases, the NUSI may be used when it is not efficient to do so.</p>
Collect NUSI statistics on covering (ALL option) NUSIs	<p>If an SI is defined with the intent of covering queries (the ALL option is specified), you should consider collecting statistics even if the indexed columns do not appear in WHERE conditions.</p> <p>Collecting statistics on a potentially covering NUSI provides the Optimizer with the total number of rows in the NUSI subtable and allows the Optimizer to make better decisions regarding the cost savings from covering.</p>
Collect NUSI statistics on NUSIs with ORDER BY	<p>If a sort key is specified in a NUSI definition with the ORDER BY option, collect statistics on that column so the Optimizer can compare the cost of using a NUSI-based access path with a range or equality condition on the sort key column.</p>
Collect non-index column statistics on all non-indexed columns used for table selection	<p>Collecting statistics on a group of columns allows the Optimizer to estimate the number of qualifying rows for queries that have search conditions on each of the columns or that have a join condition on each of the columns.</p>
Refresh statistics after updates	<p>When:</p> <ul style="list-style-type: none"> • The number of rows changed is greater than 10%. • For row partitioned tables >10% for any partition. • The demographics of columns with collected statistics changes.
Drop statistics	<p>If statistics are no longer needed because no queries are using them, they should be dropped.</p>

Collecting Statistics on Skewed Data

For collecting statistics on skewed data, the Optimizer can accommodate exceptional values. Statistics take into account the most nonunique value in a table and the most nonunique value in a value range.

Without statistics, the system relies on dynamic AMP sampling, which can be quite inaccurate when the table is small or unevenly distributed. Small tables often distribute unevenly.

Collecting Statistics on Join Index Columns

- For non-sparse join and hash indexes, the Optimizer inherits the row count and all histograms from the base tables. Teradata recommends that you collect statistics on base tables only and not on the non-sparse join or the hash indexes.

- For sparse join indexes, Teradata recommends that you collect statistics on base tables and sparse join indexes separately.
- For non-compressed join indexes, the Optimizer performs all-AMP sampling to estimate the join index row count accurately.
- For compressed sparse join indexes, Teradata recommends that you collect PI statistics. The All-AMP random sampling that the Optimizer performs may not reflect an accurate row count because of the compression.

PARTITION Statistics

Vantage provides the user with the ability to collect with “PARTITION statistics” based on partition numbers rather than column values. This enables the Optimizer to more accurately estimate cost operations involving a partitioned table.

The Optimizer is provided with:

- The number of partitions that are nonempty
- How the rows are distributed among the nonempty partitions.

Partition statistics can be collected for just the PARTITION column (single-column partition statistics) or on the PARTITION column and other table columns (multicolumn PARTITION statistics). Collecting statistics on the partitioning columns is also recommended. When the Optimizer has this information, it can better calculate the relative cost of various methods of optimizing a query for a partitioned table.

Having PARTITION statistics allows the Optimizer to generate a better plan for partitioned tables. For example, the Optimizer can cost joins involving partitioned tables more accurately with PARTITION statistics.

Setting Up Database Query Logging

Database Query Logging (DBQL) provides a series of predefined tables and views that store historical records of queries based on rules you specify.

For more information on DBQL, see [Tracking Query Behavior with Database Query Logging: Operational DBAs](#).

Performance Effects of Query Logging

Because DBQL operates asynchronously, the logging activity has a much lower impact on the overall response time of given transactions. Furthermore, DBQL writes its information to internal memory buffers or caches. These are flushed to disk when the buffer is full or at the time indicated by the DBS Control Record field DBQLFlushRate. The default rate is every 10 minutes for normal operation, but the Database Administrator (DBA) can change the rate to be more frequent to analyze performance issues.

For information on DBQLFlushRate, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Performance Information in DBQL Tables

DBQL Table	Performance Management Information
DBQLLogTbl	Data on individual queries, including: <ul style="list-style-type: none"> • Query origination, start, stop, and other timings • CPU and logical I/O usage • Error codes • SQL (truncated) • Step counts • The number of rows inserted, updated, and deleted
DBQLSummaryTbl	A summary of short-running queries. For high-volume queries, it provides query origination, response time summaries, query counts, and CPU and logical I/O usage.
DBQLStepTbl	Query step timings, CPU and I/O usage, row counts, and step actions, among other things
DBQLObjTbl	Usage information about database objects such as tables, columns, indexes, and databases.
DBQLSqlTbl	The complete SQL text.
DBQLExplainTbl	EXPLAIN output.
DBQLXMLTbl	Query plans stored in Extensible Markup Language (XML).

Recommended DBQL Logging Scenarios

User Type 1: Short Subsecond Known Work (Tactical Work)

- Log as Summary
- BEGIN QUERY LOGGING LIMIT SUMMARY = 1,3,5 ON ALL ACCOUNT = 'acctname';
The numbers 1,3,5 are clock seconds not CPU seconds
- No SQL gets logged
- No Objects get logged

For this workload, high-speed performance and minimal response time are the primary objectives. Typically, this workload tends to be very predictable in nature with queries typically designed to be single AMP retrievals.

Logging of this workload at the request level may be unnecessary for two reasons:

- The transactions are well-tuned, known, and repeated over and over again.
- The additional overhead required to record SQL for each request would represent a meaningful portion of the overall work performed on behalf of the transaction, that is, the additional overhead could materially impact request response time.

The objective is to capture only summarized information about these SQL requests. Since the expectation for this workload type is that the work is predictable, repeatable and does not vary much, summary logging is sufficient. If however, there could be unauthorized use of this ID, or on occasion, a longer running query is run, threshold logging could be used.

User Type 2: Mostly Long Running Work

- Log detail with SQL and objects
- BEGIN QUERY LOGGING WITH SQL, OBJECTS LIMIT SQLTEXT =0 ON ALL ACCOUNT = 'acctname';

If there are 10s of thousands of subsecond work, additional overhead will be incurred

Teradata recommends that a high degree of detailed data be captured for analysis of this workload category. The data generated from this DBQL logging option generates the critical detailed information needed for effective performance management and tuning.

The logging captures the entire SQL text for each request along with the objects used in processing the request. The SQL text and Object data is critical for performing query access path analysis. The DBQL detail data provides not only CPU and IO data, but also the data to calculate whether a query is skewed, does a large scan, or has the characteristics of a large product join, the keys to high impact performance tuning.

User Type 3: Mostly Short Subsecond Requests with Occasional Long Running or Unknown Work

- Log Threshold
- BEGIN QUERY LOGGING WITH SQL, OBJECTS LIMIT THRESHOLD =100 CPUTIME AND SQLTEXT =10000 ON ALL ACCOUNT = 'acctname';
- The threshold number 100 represents hundredths of CPU seconds and causes queries that require more than one second of CPU time to be logged in DBQLSQLTbl, DBQLObjTbl and DBQLLogTbl, with details. Queries with less than one second CPU time are summarized.

With threshold logging, DBQL cannot log to separate Explain and XML tables, even for those queries taking longer than the specified criteria. SQL, STEPINFO, and OBJECTS can be logged during threshold logging, even for those queries taking longer than the specified clock seconds.

This logging scenario captures data at a more detailed level for unknown or long running requests, while still logging the bulk of the work, the subsecond requests, at a summary level.

DBQL Collection Recommendations

- All usage should be logged in DBQL 24x7.
- Flush the DBQL cache every 10 minutes. See [Options for Flushing the DBQL Cache](#).
- Log usage in DBQL at the user account level:

Recommendation for account string setup is as follows: \$W00MS/R&D&H, where:

- *\$W00* = Workload performance category. Possible values include *\$R00* for Timeshare Top, *\$H00* for Timeshare High, *\$M00* for Timeshare Medium, and *\$L00* for Timeshare Low.

Note:

This optional classification is in effect only when TASM or TIWM classification processes cannot match the request to a user-defined workload.

- *MSIR* = Application/workload name, such as FINB for Finance and Batch.
- *&D&H* = ASE variables: date and hour

Using DBQL Tables

You can use DBQL tables to collect and evaluate:

- System throughput
- Response time
- Query details, such as step information, request source, SQL, answer size, rejected queries, resource consumption, and the number of rows inserted, updated, and deleted
- Objects accessed by the query

Such information enables you to:

- Identify a workload that does not meet response time Service Level Goals (SLGs)
- Drill down into DBQL after targeting workloads using:

Use this resource...	To identify details about...
AMPUsage	Top consumers
Spool Log	High spool users

Collecting Useful DBQL Historical Data

DBQL History summaries key DBQL data to 1 row per user / account / application ID / client ID per time period. Teradata recommends retaining 13 months of copied detail. That is, Teradata recommends copying detail to another table daily. You should delete the source of copied detail daily to keep online summary collection efficient.

Managing Query Performance

You can monitor queries using the Teradata Viewpoint Query Monitor portlet and other tools and utilities to identify poorly performing queries. Poor query performance is usually due to:

- Stale statistics. See [Collecting Statistics](#).
- Poor query construction. See [Finding and Fixing Problem Queries](#).

- Skewed query processing. See [Managing Skewed Processing](#).
- Skewed table distribution. See [Identifying and Fixing Skewed Tables](#).
- Locks that block queries from accessing needed data. See [Identifying and Managing Blocked Queries](#).

This section also covers the performance issues related to transaction rollbacks, which occur when a query fails to complete. See [Working with Transaction Rollbacks](#).

Finding and Fixing Problem Queries

You can use the Teradata Viewpoint Query Monitor portlet for regular monitoring of database queries and to investigate whether a reported delay is the result of a bad query.

Note:

The Query Monitor portlet has many functions beyond what is shown in this example procedure. See *Teradata® Viewpoint User Guide*, B035-2206 for detailed setup and options.

1. Monitor query activity using the **By Session** tab.
 2. Look for bad queries using the following Query Monitor parameters:
 - High CPU Use% combined with low REQ I/O accesses. If the CPU divided by the I/O is > 100, the query is probably bad.
 - High CPU Skew% with significant Delta CPU.
 - High spool usage.
-

Note:

You can click on and sort the display by a parameter instead of by session number.

3. Select a session row that shows CPU characteristics indicating a bad query to display the **Details View**, which provides information about the query, including the user and account that submitted the query.
4. Select the **SQL** tab to display the query. Examine the SQL for errors and poor construction.
5. Select the **Explain** tab to display an abbreviated version of the step statistics and explain text (generated from a full EXPLAIN request), along with a listing of the active steps, completed steps, and steps yet to run.
6. Check for the following in the Explain:
 - The steps required to execute the query
 - The type of operation being performed, and whether it includes a product join

Often the query is stuck on the step where the product join occurs.
7. You can copy the query from the SQL tab, run it from Teradata Studio, and request a full EXPLAIN to get more details and validate the source of the problem.
8. If the query appears to be constructed correctly, other problems may be indicated.

- System conditions may cause uneven processing. See [Skewed Query Processing Across Nodes](#) and [Skewed Query Processing Across AMPs](#).
- The data table may have skewed distribution. See [Finding and Fixing Skewed Tables](#).
- The query may be blocked from accessing the data. See the sections beginning with [Identifying and Managing Blocked Queries](#).

If a bad query causes problems in the database, you can:

- Use Query Monitor to abort the query. See *Teradata® Viewpoint User Guide*, B035-2206.
- Use the SET SESSION ACCOUNT request to reassign a poorly-performing query to a lower priority account. For information on:
 - Using accounts to set job priorities.
 - SET SESSION ACCOUNT, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Alternate Query Investigation Methods

Problem	Action
If the system is CPU-bound or disk-bound, or the processing is skewed.	<ul style="list-style-type: none"> • Look at physical resource utilization: Use Teradata Viewpoint. Monitor physical resources to view CPU/disk usage by node. Monitor virtual resources to view CPU/disk usage by vproc. Look at DBC.ResusageSpma to view CPU, disk, and BYNET usage. • Look at workload utilization: Use DBQL to find the user with the skewed processing. • Look at data distribution: Use DBC.TableSizeV to identify tables with skewed data distribution.
If the processing is skewed, identify the user.	<ul style="list-style-type: none"> • Use Teradata Viewpoint. • Check DBC.SessionInfoV or run Query Session to identify the current user. • Check DBC.LogOnOffV to view historical users. • Check DBC.AMPUsage to identify the heavy resource user. • Check DBC.DiskSpaceV to identify large spool user.
If the processing is skewed, get information on the problematic query.	<ul style="list-style-type: none"> • To identify the query, ask the user or check query log. • Run EXPLAIN on the query to determine if a bad join plan exists. Look for unexpected product joins. Verify that large tables are not joined before small tables. Look for estimates that are too high or too low.

Managing Skewed Processing

Skewed Query Processing Across Nodes

Parallel node efficiency is a matter of how evenly the workload is shared among nodes. The more evenly the nodes share the work, the higher the efficiency.

You can calculate parallel node efficiency by dividing average node utilization by maximum node utilization, and identify skewed processing across nodes.

IF parallel node efficiency ...	THEN ...
is nearly 100%	the better the nodes are working together.
falls significantly below 100%	in that time period, one or a few nodes are working harder than the others.
falls below 60% more often than a couple of sampling periods	your installation is not taking advantage of the parallel architecture.

Possible causes of skewed node processing include:

- Down node
- Non-Vantage application running on a TPA node
- Co-existence system where the number of AMPs per node is not optimally balanced with different node powers.
- Poor AMP efficiency

Skewed Query Processing Across AMPs

AMP vprocs always run in parallel, but the way data rows or column values are striped across the disks affect the parallel operation of AMP step processing.

Unbalanced, or skewed, or spiked, disk loads can cause one or a few AMPs to be doing most of the I/Os. For example, when a numeric column allows zeros and/or nulls, the majority of rows might hash to the same AMP.

If your disk loads are poorly balanced, discuss with operations ways to correct the situation. For example:

- Perhaps queries or views against a column with zeros/nulls could use “WHERE NOT NULL” or “NOT= 0” qualifiers.
- If the cause is a NUPI, consider redefining the index, especially for a very large table, to achieve a higher percentage of uniqueness.

Common Skew Issues

Issue	Cause	Results
Same row hash value for an excessive number of rows <ul style="list-style-type: none"> • Rows with same row hash value cannot fit in a single data block. • Rows spill over into additional data blocks. 	The table is skewed due to a highly nonunique primary index (PI). As an estimate, more than 1000 occurrences/NUPI value begin to cause performance degradation problems. This figure is based on all the rows for the same NUPI value spilling over into more than five data blocks.	<ul style="list-style-type: none"> • Increased I/Os for updates. • Increased compares for inserts and FastLoad (more Central Processing Unit (CPU) and I/Os). • Performance degradation in the Restore and Table Rebuild utilities.

Issue	Cause	Results
	See Identifying and Fixing Skewed Tables .	
	<p>The data block size is set too low. See Managing Data Block Usage.</p> <p>Note: Data block size should not be a problem on systems using block-level compression because the size should be set to the maximum.</p>	
Some AMPs have many more rows of a table than do other AMPs.	<ul style="list-style-type: none"> One or a few NUPI values have many more rows than all the other NUPI values. A highly skewed join field in a join or aggregation, for example a join on a city code in a customer table where a large number of customers may be located in a few cities. A highly skewed referencing column (for instance, a city code) when using referential integrity (RI). 	<ul style="list-style-type: none"> Poor CPU parallel efficiency on the AMPs during full table scans and bulk inserts. Increased I/O for updates, and increased compares for inserts (also more I/O).

Identifying and Fixing Skewed Tables

Performance Effects of Skewed Row Distribution

Uneven distribution of table rows among AMPs (skew) can prevent efficient query processing.

Skewed distribution results in:

- Poor CPU parallel efficiency on full table scans and bulk inserts
- Increased I/O for updates and inserts of over-represented values

The effects of a skewed table appear in several types of operations. For example:

- In full table scans, the AMPs with fewer rows of the target table must wait for the AMPs with disproportionately high numbers of rows to finish. Node CPU utilization reflects these differences because a node is only as fast as the slowest AMP of that node.
- In the case of bulk inserts to a skewed table, consider the extra burden placed on an AMP with a high number of multiple rows for the same NUPI value.

For example, assume you have a 5 million row table, with 5,000 rows having the same NUPI value. You are inserting 100,000 rows into that table, with 100 of those insert rows having the same NUPI value. The AMP holding the 5,000 rows with that NUPI value has to perform one half million duplicate row checks ($5,000 * 100$) for this NUPI. This operation results in poor parallel efficiency.

Identifying Skewed Distribution of Table Data

You can do the following to identify uneven row distribution:

- Check space usage using Teradata Viewpoint. See [Using Teradata Viewpoint to Find Skewed Tables](#).
- Run the ResVproc macro to check the AMP parallel efficiency so that you can identify the AMP affecting that node. See *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.
- Check table distribution information at the AMP level by running an SQL query against DBC.TableSizeV. See [Example of Finding Skewed Tables by Querying the TableSizeV View](#).
- Check hash distribution among AMPs. See [Finding Uneven Distribution Using Hash Functions](#).

System Tables Subject to Skewing

- Journal tables take the require spool and other transient storage from free space. A common practice is to allocate a database with a certain amount of PERM, but never use it. This ensures that there is free space for journals and so on. When your journal tables are skewed, you get an error when running queries, letting you know the system needs the journal space but cannot obtain it.
- The DBC.AccessRights table is notorious for becoming skewed. If it grows too large, it will affect the space limits dramatically. To reduce this skewing, either clean up DBC.AccessRights by removing redundant rows or convert to using roles and then remove the now redundant rows.
- DBC.AccLogTbl table. Vantage generates at least one entry every time a user defined for logging attempts to access an object. This table can grow very large and consume a lot of disk space.
- Resource usage logging tables or query logging tables. Consider daily offloading data from the actual DBC tables into temporary staging tables and ultimately history tables for analysis or archive.

Any table can becoming skewed if the primary index is poorly designed or if statistics have not been collected and the Optimizer is using stale data. A usual tell-tale sign is if one or more queries are doing a lot of maintenance on typically a single AMP. If this happens, you need to track that query down and handle it like any other skewed processing.

Using Teradata Viewpoint to Find Skewed Tables

You can use the Details view in the Teradata Viewpoint Space Usage portlet to examine skew percentage for the tables in a database.

Finding Skewed Tables in DBC.TableSizeV

You can query the DBC.TableSizeV view to determine if data for a given table is evenly distributed across all AMP vprocs.

```
SELECT vproc,CurrentPerm
FROM DBC.TableSizeV
WHERE Databasename = 'RST'
```



```
AND Tablename = 'Message'
ORDER BY 2 desc;
```

The output lists the AMPs in order of the space they use for the Message table, from the most to least space used. See [Example of Finding Skewed Tables by Querying the TableSizeV View](#).

Finding Uneven Distribution Using Hash Functions

Use the following functions to identify uneven hash distribution of data.

Function	Definition
HASHAMP	AMP that owns the hash bucket
HASHBACKAMP	Fallback AMP that owns the hash bucket
HASHBUCKET	Grouping for the specific hash value
HASHROW	32 bits of row hash ID without the uniqueness field

HASHAMP Example

If you suspect distribution problems (skewing) among AMPs, the following is a sample of what you might enter for a three-column PI:

```
SELECT HASHAMP (HASHBUCKET (HASHROW (col_x, col_y,
    col_z))), count (*)
FROM hash15
GROUP BY 1
ORDER BY 2 desc;
```

HASHROW Example

If you suspect collisions in a row hash, the following is a sample of what you might enter for a three-column PI:

```
SELECT HASHROW (col_x, col_y, col_z), count (*)
FROM hash15
GROUP BY 1
ORDER BY 2 desc
HAVING count(*) > 10;
```

Primary Index and Skewed Row Distribution

The hash value of the PI for a row determines where the row is stored on the AMP. In a normal environment, hash values are evenly distributed across the nodes and the AMPs within a node.

The less unique the values are for the index, the less evenly the rows of that table are distributed across the AMPs. If a table has a NUPI with thousands of instances of a single value, the table can become skewed.

Performance Effects of a Primary Index

- The more unique the PI, the more unique the row hash value.
- The more unique the row hash value, the more even the data distribution across all the AMP vprocs. Even data distribution enhances parallel efficiency on full table scan operations.
- UPIs generate the most even distribution of table rows across all AMP vprocs.
- NUPIs generate even row distribution to the same degree that values for a column or columns are unique. Rows with the same row hash always go to the same AMP, whether they are from the same table or from different tables.

To determine the best PI for a table, factor in the:

- Extent of update activity
- Number of full table scans
- Join activity against the PI definition
- Frequency of PI as a selectivity column and, therefore, a potential access path

If a PI causes uneven data distribution, you should re-specify it according to the primary index design best practices shown in *Teradata Vantage™ - Database Design*, B035-1094.

Identifying and Managing Blocked Queries

Access to data for a query can be blocked by other queries which are temporarily given a lock on the data as a normal part of database operation. However some blocks, such as those due to a hung query, indicate database problems.

Also see [Finding and Resolving Lock Contentions](#).

Locking Overview

When multiple transactions need to perform work that requires a nonshareable lock on the same object, Analytics Database controls concurrency by:

- Granting a lock to the transaction that requests access to first.
- Queuing subsequent transactions in order of their arrival, such that they wait indefinitely until preceding query completes and a new lock can be granted, with these exceptions:
 - Requests identified with the LOCKING modifier NOWAIT option immediately abort rather than join the queue.
 - MultiLoad transactions can timeout after waiting for over 50 seconds. See *Teradata® MultiLoad Reference*, B035-2409.
- Aborting the younger request(s) among deadlocked requests. Deadlocking occurs when multiple requests need access to several of the same objects, each request has a lock on one of the objects, and is locked out of at least one of the objects, such that none of the requests can complete.

- Releasing the lock and granting a new lock to the oldest transaction in the queue, when the current transaction completes.

For a complete discussion of locks and locking, see *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142.

Investigating Query Blocks and Delays

Most blocks are momentary and do not require attention. However, if a block persists, you can investigate it to identify the cause and determine whether further action is required.

1. Use the Teradata Viewpoint Query Monitor portlet to monitor the parameters related to query blocks and delays.

State	Description
Blocked	Indicates that a session query is held up by a lock on an object that the query is attempting to access.
Blocked Time	How long the query has been blocked. Momentary locks are a normal part of database operation. Persistent locks may indicate: <ul style="list-style-type: none"> • A runaway query or hung query that cannot complete and release its locks, and which should be aborted • A long-executing query, such as large-scale update or backup operation, which should be rescheduled to avoid resource contention
Delayed	Indicates that the query is in a delay queue caused by a workload rule.

2. Click the session ID for the query to access the **Details View**.
 - If a query is blocked, see the **Blocked By** tab for details
 - If the query is delayed, the **Delay** tab provides details about the cause of the delay
3. You can also use the Lock Viewer portlet to find out additional details about the block:
 - Block Time
 - Database
 - Table
 - Delay
 - Blocked User
 - Blocking Level
 - Blocking User

Additional Tools for Analyzing Lock Problems

The following table provides suggestions for analyzing and solving lock problems.

Tool	Analysis	Solution
Lock Display	Transaction locks	Determine which session is holding the lock that blocks others.
Query Session	Blocked session	Abort the session causing blocked transactions.
Show Locks	Host utility (HUT) locks; that is, locks placed by a client-based utility, such as DSA	Submit RELEASE LOCKS as either an Archive and Recovery command or an SQL statement.

Reducing Deadlocks

- Except with CREATE INDEX, use LOCKING ... FOR ACCESS if dirty reads are acceptable.
- Consider BTEQ handling of transaction processing. After transaction rollback, BTEQ continues the transaction from the point of failure, not at the beginning of the transaction.
- Set the DeadLockTimeout field via the DBS Control utility to 30 seconds if you have a mix of DSS and PI updates on fallback tables.
- Be sure to use RELEASE LOCKS on Archive/Recovery jobs.
- Use the LOCKING ROW [FOR] WRITE/EXCLUSIVE phrase preceding a transaction. This phrase does not override any lock already being held on the target table. LOCKING ROW is appropriate only for single table selects that are based on a PI or SI constraint. For example:

```
LOCKING ROW FOR WRITE
SELECT y FROM tableA WHERE pi =1;
UPDATE tableA SET y=0 WHERE pi =1;
```

- In macros, use multistatement requests instead of Begin Transactions (BT)/End Transactions (ET) to minimize table-level deadlocking. For example:

```
LOCKING ROW FOR WRITE
SELECT y FROM tableA WHERE pi =1
; UPDATE tableA SET y=0 WHERE pi =1 ;
```

This causes all the necessary locks to be applied at the start, which avoids the potential for a deadlock. Use the EXPLAIN modifier to check out the processing sequence.

- Be aware that when you are updating tables where several rows in the base table share a row in the join index subtable (such as in most aggregate join index cases), there can be considerable collision on the same row.

Handling Blocked Internal DBQL Requests

DBQL uses express requests to do internal work on DBQL log tables. If you are performing work on the same log tables, such as deleting rows, this may block DBQL internal requests because you have a lock

on the table. The internal requests will then consume AMP worker tasks until your work finishes or until you disable the DBQL logging.

Note:

You cannot abort or otherwise do anything to this internal session record. It exists only to show that internal DBQL requests are blocked.

If DBQL internal requests remain blocked, the system deadlock detection may abort them. If the system aborts these internal sessions, the logging data will be lost. You must submit the appropriate `END QUERY LOGGING` statement to end logging before you do any kind of maintenance on DBQL tables.

If you notice frequently blocked DBQL internal requests reported by the internal session, consider scheduling your queries (such as table maintenance) on DBQL tables when there is no logging activity. Or use locks that hold the logging tables for a minimal amount of time.

Working with Transaction Rollbacks

A rollback is a reversal of an incomplete database transaction due to problems. A transaction may fail to complete and be aborted due to:

- A deadlocked query
- A database restart
- An `ABORT SESSION` command

The rollback removes any partially completed database updates from the affected user tables. The system uses the TJ (`dbc.transientjournal`), which contains data about incomplete transactions, including a “before image” of each modified table row, to maintain data integrity.

Also see the rollback topics in [Archiving, Restoring, and Recovering Data: Operational DBAs](#).

Effects on Performance

A rollback can affect the performance and availability of resources in Vantage while the rollback is in progress. The rollback competes for CPU with other users. Moreover, a rollback can keep locks on affected tables for hours, or even for days, until the rollback is complete. The `RollbackPriority` field in the DBS Control utility determines the priority given to rollback operations.

For more information on the DBS Control utility, see *Teradata Vantage™ - Database Utilities*, B035-1102.

RollbackPriority

RollbackPriority Performance Implications

Because rollbacks can involve millions or billions of rows, competing for CPU and other system resources, rollbacks can impact system performance. Rollbacks can keep locks on affected tables for hours or days until the rollback is complete. During a rollback, a trade-off occurs between overall system performance and table availability.

How RollbackPriority affects performance is not always straightforward. It is related to the TASM/Viewpoint Ruleset, job mix, and other processing dynamics. The RollbackPriority setting should only be changed after full consideration of the performance consequences:

- When RollbackPriority is set to FALSE, rollbacks are performed at system priority, a special priority higher than any user-assigned priority, that is reserved for critical internal work. As a result, faster rollbacks occur at the expense of other online performance.

The default setting of FALSE is especially appropriate when rollbacks are large, occurring to critical tables that are accessed by many users. It is better to complete these rollbacks as quickly as possible to maximize table availability.

- When RollbackPriority is set to TRUE, rollbacks are executed within the aborted job's workload. This isolates the rollback processing to the aborted job's priority, and minimizes the effect on the performance of the rest of the system. However, if the rollback places locks on tables that other users are waiting for, this causes a greater performance impact for those users, especially if the rollback is running at a low priority.

A setting of TRUE is appropriate when rollbacks are typically smaller, occurring to smaller tables that are less critical, and less extensively used.

Rollbacks can be affected by Workload Management Capacity On Demand (WM COD) and Hard Limits, depending on their running context:

- When RollbackPriority is FALSE, the rollback runs under the system priority, which is subject to WM COD throttling.
- When RollbackPriority is TRUE, the rollback runs under the current user workload, which is also subject to WM COD throttling. The current workload may be further throttled if it is running under a Virtual Partition with a fixed limit, or a SLG Tier Workload Management Method with a hard limit.

Minimizing or Avoiding Rollbacks

If you are doing a lot of deletes on rows from a table, consider using MultiLoad rather than BTEQ. MultiLoad completely avoids use of the TJ and is restartable.

To minimize the number of TJ rows being generated on an INSERT ... SELECT request, consider doing a multistatement INSERT ... SELECT request into an empty table from both of the other tables. This only puts one entry into the transient journal to let the DBS know that the target table is an empty table and to drop all the rows in the target table if the need for a rollback is encountered. After the new table is created, drop the old table and rename the new table to the name of the old table.

More details on both the delete rows and INSERT ... SELECT request are available online in the Support Link Knowledge Base.

Detecting a Rollback in Progress

Sessions in rollback may appear to have logged off in both DBC.LogOnOffV and DBC.AccessLogV, but this is not always the case. The logoff would depend on the manner in which the job was aborted. If you specify one of the following:


```
ABORT SESSION hostid.username LOGOFF
ABORT SESSION *.username LOGOFF
ABORT SESSION hostid.* LOGOFF
```

then the LOGOFF option would terminate the session.

Without it, the session should continue until the abort completes or the Supervisor issues a LOGOFF request. Unless an SQL job is explicitly coded to do otherwise, a session will also appear to have logged off if the system has undergone a restart.

The rollback or abort is independent of the session. It is actually handled by a completely different mechanism with internally allocated AMP worker tasks.

Example

To activate the RCVmanager, go to the Database Window and type “start rcvmanager”. Then issue the “list rollback tables” command. It will show you each table that is being rolled back at that point in time, how many TJ rows have been rolled back and how many rows are remaining.

If you run this command twice, you can then make an estimate how long it will take the rollback to complete, based on the rows processed and rows remaining and the time between the two snapshots.

```
list rollback tables;
```

Result:

```
TABLES BEING ROLLED BACK AT 10:01:26 04/09/20

ONLINE USER ROLLBACK TABLE LIST
```

Host	Session	User ID	Workload Definition	AMP W/Count
1	234324	0000:0001		24

TJ Rows Left	TJ Rows Done	Time Est.
53638	1814	00:09:51

Table ID	Name
0000:16A6	"FINANCE_T"."Order_Header"


```
SYSTEM RECOVERY ROLLBACK TABLE LIST
```

Host	Session	TJ Row Count
------	---------	--------------


```

-----
Table ID      Name
-----

Enter command, "QUIT;" or "HELP;" :
list rollback tables;

TABLES BEING ROLLED BACK AT 10:01:37 04/09/20

ONLINE USER ROLLBACK TABLE LIST

Host  Session  User ID      Workload Definition  AMP W/Count
-----
   1    234324  0000:0001                      24

TJ Rows Left  TJ Rows Done  Time Est.
-----
      52663      2789  00:09:45

Table ID      Name
-----
0000:16A6  "FINANCE_T"."Order_Header"

SYSTEM RECOVERY ROLLBACK TABLE LIST

Host  Session  TJ Row Count
-----

Table ID      Name
-----

Enter command, "QUIT;" or "HELP;" :

```

Canceling Rollbacks

When a transaction fails to complete because the database restarts or the job of the transaction has failed, the system must remove any partially completed database updates and roll back to the pre-transaction condition to assure data integrity.

Because rollbacks can lock affected tables until the rollback completes, you can restore specific tables if it is quicker than waiting for a rollback to complete. If you decide to use the restore method, cancel the

rollback using the Recovery Manager utility. For more information, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Note:

You should list rollback tables using the LIST ROLLBACK TABLES command and then cancel rollback from that list within the same Recovery Manager session.

Noticing a Slow Rollback of an ALTER TABLE Statement

Because Teradata ensures that all transactions must succeed or fail in their entirety, the system does not allow partial results. In most cases, if a transaction fails, Teradata rolls back all the previous actions of a transaction. This requires that the system logs the before state of every action so that the action can, if necessary, be undone later. For instance, when you delete a row, Teradata must first log a copy of the row, so that it can be restored if the transaction fails.

Sometimes, logging is extremely expensive compared to the cost of the action, for example, dropping a large table. If the system does not have to roll the table back later, the system may simply release all table data, deallocating entire cylinders with a single action. In this case, Teradata delays performing the action until it is certain that the transaction will succeed. In effect, Teradata commits the transaction before executing the deferred action.

During this special post-commit phase, the transaction is still working and holding locks, but it cannot be aborted (since it is already committed). In most cases, this post-commit phase finishes quickly and is unnoticeable.

The exception is the ALTER TABLE statement. It may take longer when used to add or remove columns of a table. ALTER TABLE requires that every row in the table be rebuilt and is, therefore, an expensive operation. It can take hours on a large table, even with the highly optimized, block-oriented method that Teradata uses.

When you notice that a rollback of an ALTER TABLE statement is taking a very long time but you cannot abort it, the system is deferring an abort and will need to complete it.

Managing the Database Workload

Also see [Managing Database Resources: Operational DBAs](#).

Evaluating Peak Utilization Periods

Once you have identified the peak period, you can look for the bottleneck. Examining ResUsage data is one technique helpful in finding bottlenecks.

Some possible bottlenecks are:

- CPU saturation
- Disk saturation

- Free and/or FSG cache memory
- BYNET
- Vprocs (hot AMPs, coexistence, load balancing)
- Channel (number of sessions or channel speed)
- LAN/gateway (number of sessions or network connections)
- Lock contention

Resource Usage by Workload

ResSpsView, a view of the ResUsageSps table, allows you to see the percentage of CPU used by different workloads, determine which workload is responsible for I/O skew, and examine queue wait time and service time numbers to find backed up queries.

For ResUsageSps table column definitions, see *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

Assessing a Busy System

CPU Saturation

Systems that run frequently at or near capacity are often the subject of assessments in which attempts are made to determine the extent of resource exhaustion.

When CPU is the binding factor and the scarce resource on a system, it is useful to determine the highest level of saturation of the system, that is, the point to which you can drive the system before it becomes so overloaded that it appears to be hanging.

Once a system reaches this level of saturation, the system should still be able to work itself out, but that may require extra time to do so.

Without appropriate performance monitoring, users may start, once the system appears to be hanging, to:

- Abort jobs that could cause rollbacks, or
- Submit duplicate queries that create more work on an already-exhausted system.

While ResUsage data provides the bulk of the information needed to know that a system is at 100% busy with respect to CPU or CPU+I/O Wait, other information may also be needed in order to examine the extent of CPU saturation and system congestion. In other words, one can know that the system is running at capacity, but not the extent of the overload.

When the system becomes so busy that logons become slow or hung, performance monitoring is not able to determine whether the system is actually hung or simply overloaded without using other tools.

Suggested Monitoring Techniques

Since the goal is to be able to drive the system as hard as possible without overloading it, some techniques for assessing the level of busy can be used when CPU usage is high:

1. Check AWT utilization. If the number is constantly at or near maximum, then
2. Check the message flow control. If there are tasks apparently in flow control, then

3. Check run queue ResUsageSawt MailBoxDepth data. If the run queue is growing longer and longer, the system is too busy and will slow down dramatically.

While it is not unusual to see a busy system with high AWT counts, the presence of flow control means that some tasks are currently being blocked from sending more work to busy AMPs.

Finding a Saturated Resource

Use Resource Check Tools, located in the `/usr/pde/bin` directory, to check for saturated resources.

IF ...	THEN ...
mboxchk is not already running a background task	run mboxchk to check current response time.
the mboxchk log shows a slow response or timeout	run syscheck to get a report showing attributes that falls below the specified danger level.
no attribute is reported at the WARN level	check disk and AMP CPU usage.

For information about mboxchk and other PDE resource check tools, such as nodecheck and syscheck, see man pages or pdehelp.

High I/O Wait

Teradata recommends configurations with CPU to I/O bandwidth ratios according to typical database workload demands in order to avoid CPU starvation. If these guidelines are not followed, or customer workload is unusually heavy with respect to I/O, it is possible that CPU starvation may still occur, as reflected by high I/O WAIT on the system.

If ResUsage data shows that I/O WAIT is rising while CPU busy is falling, it is an indication that the system is not able to use the available CPU because I/O has become the limiter.

If the onset of high I/O wait is sudden:

1. Determine if the high I/O wait is due to disk I/O, waiting for AMPs from other nodes (because of skewing or coexistence imbalance), low system demand, or BYNET I/O.

CPU+WIO less than 90% may suggest low system demand without a true I/O bottleneck. Look at node efficiency to determine if the I/O wait is due to node waiting.

2. Look at the actual disk I/O wait queue using `sar -d`, and examine:

- `await`

Average time in milliseconds that transfer requests wait idly on queue for response (in the FibreChannel driver queue or disk queue).

- `avserv`

Average time to be serviced (includes seek, rotational latency and data transfer times).

For more information on the `sar` utility, see the Linux operating system documentation.

Job Scheduling Around Peak Utilization

Rescheduling Jobs

Once you determine your peak system utilization times, you can recommend that some jobs be moved to other time slots.

For example, if peak periods are 9 A.M. to 5 P.M., you can schedule batch and load jobs overnight so they do not interfere with peak daytime demand.

Bound Jobs

Since some jobs tend to be CPU-bound and others I/O-bound, it is a good idea to determine which jobs fit into each category. You can determine this by means of AMPUsage data analysis.

You can schedule a CPU-bound job with an I/O bound job so that the resource underutilized by one job can be used more fully by the other.

TASM and Concurrency

TASM throttle rules can be useful in controlling the concurrency of certain types of queries during peak utilization times.

In addition, TASM filter rules can prevent queries with certain characteristics from even starting to execute during specific windows of time. This can help keep utilization levels under control at times of high contention.

Managing I/O-Intensive Workloads

Suggestions for balancing resource usage when the system is I/O-bound follow:

- Identify I/O-intensive portions of the total work using AMPUsage reports and DBQL.
- Reschedule I/O-Intensive work to off-hours.
- Look for query or database tuning opportunities, including:
 - Collecting/refreshing statistics on all join and selection columns
 - Adding indexes, join indexes, or sparse indexes
 - Using MVC to reduce row size and get more rows per block
 - Using PPI
 - Increasing block sizes
 - Using 3NF data model to obtain narrower rows, more rows / block, fewer I/Os, and then denormalizing as needed
 - Increasing node memory, in order to expand the size of the FSG cache

Using Canary Queries

A canary query is a common SQL statement that is run at specific intervals and monitored for data such as response time to characterize the current system workload. Each canary query is fixed to do a consistent amount of work per execution.

You can use Teradata Viewpoint to send canary queries to:

- Measure response time as an indicator of system demand or system or database hangs.
- Measure response time for various TASM workloads and performance tiers.
- Initiate an alert system if response time degrades so that you can take appropriate action.
- Establish response time Service Level Agreements (SLAs) based on canary response times.

System Canary Queries

Use system canary queries to check for overall system or database hangs, to take some kind of action when response times reach certain thresholds, or when stalled, such as send alert and/or capture system level information.

More than just a check, a system canary query should execute diagnostics that capture the state of the system if performance stalls.

System canary queries are intended specifically to focus on the core system. They should be short-running (one second), low impact queries on tables that are normally not write locked.

System canary queries are most useful when run frequently. For example, some sites run them every 3 to 5 minutes; other sites find every 5 to 10 minutes adequate.

They should be run on a system node. This will eliminate other factors, such as middle tiers, network connections.

Depending on their makeup, canary queries can add to contention for resources. Use them selectively, where needed, with shorter queries preferable.

Sample System Canary Query

The simplest canary monitor query is the following:

```
SELECT * from DBC.DBCInfoV;
```

As the query runs, Teradata Viewpoint can monitor the query, logging start and end times. If the query runs longer than the indicated threshold, an alert and perhaps diagnostic scripts are automatically executed.

Production Canary Queries

Production canary queries may be used to:

- Take response time samplings, storing them for tracking purposes, or
- Monitor the expected response times of specific groups of queries, such as short-running tactical queries running in high priority.

Response times are an indicator of system demand. When system demand is high, canary response is high.

From a user perspective, a sudden deviation in response times would have an immediate impact, since users of consistently short running queries would be the first to notice performance degradation.

Production canary queries have wider uses than system canary queries and can be used in a variety of ways. For example, they:

- Can be run on production user tables.
- Could be run from other endpoints in the system architecture, such as a network client PC or z/OS client to expand scope of monitoring.
- Monitor overall response.
- Monitor specific area of the job mix.
- Are run less frequently than system canary queries, usually once every 20 to 60 minutes.

In using a production query from a non-TPA node location, other things, such as network and middle-tier monitoring, are also covered, but when it stalls, you need to investigate further to determine where the bottleneck is located.

Once the response time for a canary query is stored in a table, it can be summarized for use in tracking trends.

ASE Impact on PE and AMP Performance

- ASE has little impact on PE performance. The cost incurred for analyzing the account string amounts to only a few microseconds.
- ASE may have impact on AMP performance.

The AMP has the burden of additional DBC.AMPUsage logging. Depending on the number of users and the ASE options selected, the added burden may vary from very slight to enough to degrade performance. In general, the &D, &H, and &L options do not have major effects on performance.

Be cautious where you use the &T option. It can generate an AMPUsage row for virtually every Teradata SQL request. The &T option can have a much greater effect on performance. Therefore, do not use the &T option:

- In default account ID strings
- In conjunction with tactical queries
- With TPump

The &T option should not be a problem for long-running DSS requests, but could be a performance issue if users are running numerous small requests. &T is site-dependent; it should generate no more than 10 to 20 AMPUsage rows per minute.

Note:

Because ASE causes the system to write more entries to DBC.AMPUsage, you must manage the table more often.

For more information on account string expansion, see [Logging Resource Usage Data with Account String Variables](#).

AMPUsage Logging with ASE Parameters

AMPUsage logging may have both performance and data storage impacts.

The following table summarizes potential impact.

ASE Parameter	Performance Impact	Data Capacity Impact
None	Negligible	1 row per account per AMP
&D	Negligible	1 row per account per day per AMP
&H	Negligible	1 row per account per hour per AMP
&D&H	Negligible	1 row per account per hour per day per AMP
&L	Negligible	1 row per session pool
&I	Negligible	1 row per SQL request
&T	Potentially non-negligible	1 row per query per AMP

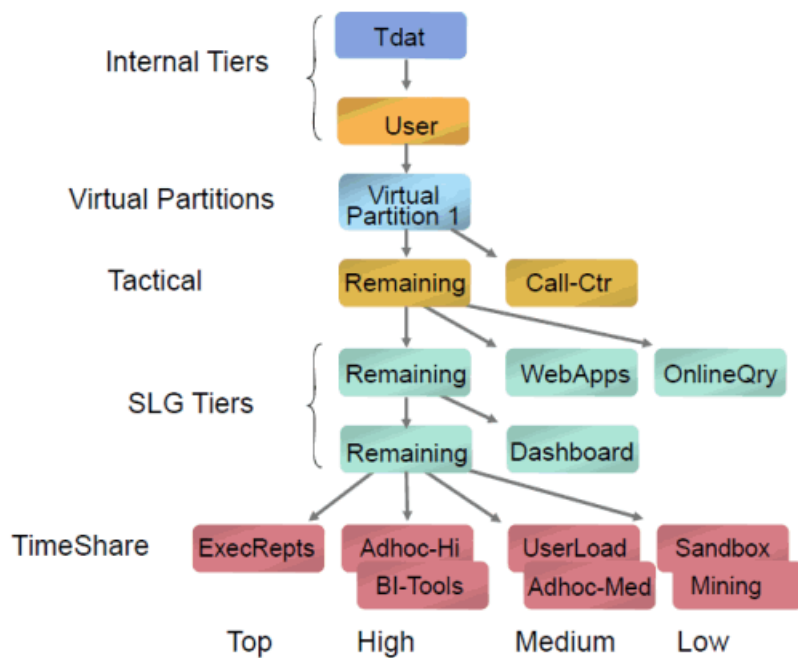
Workload Priority Management

When you define a workload for a Linux SLES system, the **Workload Designer** portlet prompts you to choose a workload management method. Select one of the following workload management methods to give a workload its place in the resource consumption hierarchy:

- Tactical
- SLG (up to 5 optional tiers)
- Timeshare (Top, High, Medium, and Low tiers)

TASM uses these tiers to prioritize requests in a workload and, optionally, to control when they can begin to execute.

The following figure illustrates the priority hierarchy levels.



Charge Back

ASE can be used to implement a simple charge back utility. It provides a way to determine how much system CPU time and disk activity is consumed by a user each day. By adding a few special characters to a user's account name, you can extract detailed information from system tables about what that user has done. Vantage expands these special characters into such things as the session number, request number, date or time when the account name is written to a system table.

At the completion of each SQL step, Vantage always updates the DBC.Acctg table with statistics about the request. These statistics include the total CPU time in seconds and the number of logical disk I/Os used by the request. This statistical information is summarized by adding it to an existing row that contains the same username and account name.

Configuration

The following example modifies the account string of each user you want to track. You preface each account string with the text CB&D. You can add any additional account information after these four characters if you wish.

When you add a date to the account name, the account name effectively changes each day and a new row is written to the DBC.Acctg table. This row contains the total number of CPU seconds and total number disk I/Os for each request that was submitted on that date.

The &D is an ASE token that expands to the current date in the format YYMMDD. You can use CB to indicate that the account is being tracked for charge back.

You can modify an existing account string for a user using the following SQL command:


```
MODIFY USER JANETJONES AS ACCOUNT = ('CB&D');
```

Note:

Workload performance characters (\$R00, \$H00, \$M00, \$L00), if used, must be the first characters in the account string.

Example

```
SELECT ACCOUNTNAME, USERNAME, SUM(CPUTIME), SUM(DISKIO) FROM DBC.AMPUSAGE
WHERE SUBSTR(ACCOUNTNAME, 1, 2) = 'CB'
GROUP BY USERNAME, ACCOUNTNAME
ORDER BY USERNAME, ACCOUNTNAME;
```

Result:

```
*** Query completed. 11 rows found. 4 columns returned.
*** Total elapsed time was 2 seconds.
```

AccountName	UserName	Sum(CpuTime)	Sum(DiskIO)
-----	-----	-----	-----
CB060902	JANETJONES	1,498.64	3,444,236
CB060903	JANETJONES	934.23	1,588,764
CB060904	JANETJONES	883.74	924,262
CB060905	JANETJONES	214.99	200,657
CB060902	JOHNSMITH	440.05	396,338
CB060903	JOHNSMITH	380.12	229,730
CB060904	JOHNSMITH	112.17	184,922
CB060905	JOHNSMITH	56.88	99,677
CB060902	SAMOREILLY	340.34	410,178
CB060903	SAMOREILLY	70.74	56,637
CB060902	WEEKLY	3,498.03	7,311,733

If we wanted to charge \$0.25 per CPU second and bill for the month of September 2006, we could use the following query to generate the bill:

```
SELECT USERNAME, SUM(CPUTIME)*0.25 (FORMAT '$$ZZZ,ZZZ,ZZ9.99')
FROM DBC.AMPUSAGE
WHERE SUBSTR(ACCOUNTNAME, 1, 6) = 'CB0609'
GROUP BY 1
ORDER BY 1
WITH SUM(CPUTIME)*0.25 (FORMAT '$$ZZZ,ZZZ,ZZ9.99', TITLE 'Grand Total:');
```


Result:

```
*** Query completed. 4 rows found. 2 columns returned.
*** Total elapsed time was 2 seconds.
```

UserName	(Sum(CpuTime)*0.25)
JANETJONES	\$882.90
JOHNSMITH	\$247.33
SAMOREILLY	\$102.77
WEEKLY	\$874.51
Grand Total:	\$2,107.51

Charge Back and Overhead

From a CPU perspective charge back entails very little overhead. The accounting table is already being updated at the completion of each statement. The only cost is the creation of a new row in the table for each user each day. From a space perspective, the accounting table will grow by one row for each user each day. Periodic cleanup can constrain this growth.

Managing Database Resources to Enhance Performance

Managing these resources efficiently can enhance performance:

- Space
 - Disk
 - Spool
 - PackDisk
 - FreeSpace FSP
- Memory
 - Monitoring memory
 - Free Memory
 - Memory consuming features
 - FSGCache
- Processing Resources
 - Canary queries
 - Monitor ResUsage data (Macros, Viewpoint)
 - DBC.AMPUsage
 - Compression
 - DataBlockSize

Performance and Space Management

Also see [Managing Space: Operational DBAs](#).

Managing System Disk Space

System disk space is initially allocated as:

- A spool space reserve, which guarantees a certain amount of space remains unused for data storage so it is available for query execution.
- Permanent space assigned to a top level administrative user to contain other users, databases, and tables.

You should monitor disk space usage and set up alerts that track usage patterns, for the capacity planning purposes.

Managing Spool Space

Managing spool space allocation for users can be a method to control both space utilization and potentially bad (that is, non-optimized) queries.

Spool Space and Perm Space

Spool space is allocated to a user. If several users are active under the same logon and one query is executed that exceeds the limit of the spool space allocated, all active queries for that user that require spool will likewise be denied additional spool and will be aborted.

If space is an issue, it is better to run out of spool space than to run out of permanent space. A user requesting additional permanent space will do so to execute queries that change tables (inserts or updates, for example). Additional spool requests are almost always done to support a SELECT. Selects are not subject to rollback.

To configure this, see “Cylinders Saved for PERM” in *Teradata Vantage™ - Database Utilities*, B035-1102.

Spool Space Accounting

Spool space for users is updated in the DatabaseSpace table. However, given the possibility of phantom spool, you may need to do one of the following things to clear spool space for users that have logged off:

- Run the Update Space utility or execute the SQL stored procedure FixCurrentSpace, which performs the same functions as the Update Space utility; see [FixCurrentSpace Procedure](#)
- Execute the SQL stored procedure FixAllocatedSpaceSpace to fix the inconsistencies between the AMP-level space allocations in the DBC.DatabaseSpace table and the global-level space allocations in the DBC.GlobalDBSpace table; see [FixAllocatedSpace Procedure](#)

After running Update Space or FixCurrentSpace, spool space values in DBC.DatabaseSpace reflect actual spool usage.

Phantom spool cases are those in which the DBC.DatabaseSpace table indicates that there is spool, although no spool exists on the disk. Phantom spool cases are not the same as “left-over spool” cases. Left-over spool cases are those in which spool is actually created and exists on the disk, although a request completed its execution.

Using Spool Space as a Trip Wire

Lowering spool space may be a way to catch resource-intensive queries that will never finish or that will run the entire system out of free space if the user is allocated a very high spool space limit.

In the interest of system performance, do not allocate high spool space limits to all users and, in general, be very conservative in setting spool space limits.

Note:

Consider defining a spool skew factor for databases and users when requests are likely to be skewed across AMPs. However, remember that when skew is non-zero, the system needs to perform more background work to manage the spool limit globally. To maintain system performance, do not give spool space unlimited skew. For more information on skew factors and global space accounting, see [Global Space Accounting](#).

Managing Cylinders

Disk space is divided into cylinders, with each cylinder being made up of a number of data blocks, each of which contains one or more data rows. The database allocates cylinders as follows.

Usage	Description
Contiguous sectors on a cylinder	datablocks are stored on adjacent sectors in a cylinder. If a cylinder has 20 available sectors, but only 10 are contiguous, a 15-sector block must be stored on another cylinder.
Free cylinders	Vantage performs better if permanent data is distributed across multiple cylinders. However, permanent data and spool data cannot share the same cylinder. Therefore, a system must always have empty cylinders that can be used for spool space.

Vantage does not run out of disk space until it allocates and fully utilizes all cylinders.

Low Cylinder Utilization

Performance degradations can occur, however, as soon as the system gets close to exhausting the free cylinder pool. This happens because the system performs MiniCylPacks on cylinders with low utilization in order to reclaim the unused disk space. Therefore, you should be aware if you are running out of space due to a preponderance of under-utilized cylinders.

Low utilization of cylinders can occur when:

- You FastLoad a table using a small FreeSpacePercent (FSP) and then insert additional data to the table that is greater than the FSP.
- You delete a significant percent of a table but have not yet run Ferret PACKDISK to reclaim the space.

Frequently Updated Tables

With frequently updated tables, the free space on the cylinder can become so fragmented that it cannot be used.

When this occurs, the system could allocate additional cylinders to the table. To avoid this problem, the system sometimes performs a cylinder defragmentation to make the free space on the cylinder usable again.

AutoCylPack

AutoCylPack (automatic background cylinder packing) manages space on cylinders, finding cylinders with low or high utilization and returning them to their user-defined FSP (Free Space Percent), that is, the percentage of storage space within a cylinder that AutoCylPack leaves free of data to allow for future table growth.

There is a system wide default for AutoCylPack. It can be overridden on a table-by-table basis by specifying the FSP clause in a CREATE TABLE or ALTER TABLE statement.

It is important for the file system to maintain enough free cylinders and to manage the space on the cylinders.

Although AutoCylPack runs as a background task issuing I/Os, you can adjust its level of impact. For the DBS Control fields that support AutoCylPack, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Performance

AutoCylPack helps reduce:

- The chances that MiniCylPacks run. MiniCylPacks are strictly concerned with reclaiming whole cylinders.
- The need for you to perform regular Ferret PACKDISK operations.

PACKDISK can be performed on a table, a range of tables or an entire system, but it affects performance of the foreground tasks and thus system performance.

If cylinders have higher utilization:

- System performance improves because there is a higher data block to cylinder index ratio and more effective use of Cylinder Read. There will be less unoccupied sectors read in.
- A table occupies less cylinders. This leaves more cylinders available for other uses.

MiniCylPack

A MiniCylPack moves data blocks in logical sequence from cylinder to cylinder, stopping when the required number of free cylinders is available. A single MiniCylPack may affect two to 20 cylinders on an AMP.

The process continues until one cylinder is completely emptied. The master index begins the next required MiniCylPack at the location that the last MiniCylPack completed.

The file system will start to MiniCylPack when the number of free cylinders drops to the value set by MiniCylPackLowCylProd. The default is 10.

The File Information Block (FIB) keeps a history of the last five cylinders allocated to avoid MiniCylPacks on them.

Note:

Spool files are never cylinder packed.

Use the DBS Control (see “MiniCylPackLowCylProd” in *Teradata Vantage™ - Database Utilities*, B035-1102) to specify the free cylinder threshold that causes a MiniCylPack. If the system needs a free cylinder and none are available, a MiniCylPack occurs spontaneously.

Migrating Data Blocks

1. If space can be made available either by migrating blocks forward to the next cylinder or backwards to the previous cylinder, choose the direction that would require moving the fewest blocks.

If the number of blocks is the same, choose the direction of the cylinder with the most number of free sectors.
2. If Step 1 fails to free the desired sectors, try migrating blocks in the other direction.
3. If space can be made available only by allocating a new cylinder, allocate a new cylinder. The preference is to add a new cylinder:
 - Before the current cylinder for permanent tables.
 - After the current cylinder for spool tables and while performing FastLoads.

When migrating either forward or backward, the number of blocks may vary because the system considers different blocks for migration.

Because of the restriction on key ranges within a cylinder, the system, when migrating backward, must move tables and rows with the lowest keys. When migrating forward, the system must move tables and rows with the largest keys.

The system follows special rules for migrating blocks between cylinders to cover special uses, such as sort and restore. There are minor variations of these special rules, such as migrating more data blocks than required in anticipation of additional needs, and looking for subtable breaks on a cylinder to decide how many data blocks to attempt to migrate.

Performance

Although cylinder packing itself has a small impact on performance, it often coincides with other performance impacting conditions or events. When the file system performs a MiniCylPack, the operation frees exactly one cylinder.

The cylinder packing operation itself runs at the priority of the user whose job needed the free cylinder. The cylinder packing operation is the last step the system can take to recover space in order to perform a write operation, and it is a signal that the system is out of space.

Needing to pack cylinders may be a temporary condition in that a query, or group of queries, with very high spool usage consumes all available free space. This is not a desirable condition.

If space is a problem:

- Enable AutoCylPack if you have not done so and specify an FSP of 0% for read-only tables using the CREATE TABLE or ALTER TABLE statement
- Run the Ferret PACKDISK command.

Error Codes

MiniCylPacks are a natural occurrence and serve as a warning that the system may be running short on space. Tightly packed data can encourage future cylinder allocation, which in turn triggers more MiniCylPacks.

The system logs MiniCylPacks in the Software_Event_LogV with the following error codes.

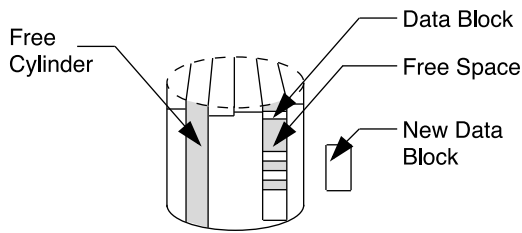
Code	Description
340514100	Summary of MiniCylPacks done at threshold set via the DBS Control.
340514200	A MiniCylPack occurred during processing and a task was waiting for it to complete.
340514300	The system could not free cylinders using MiniCylPack. The MiniCylPack failed. This means that the system is either getting too full or that the free cylinder threshold is set unreasonably high. Investigate this error code immediately.

Frequent 340514200 or 340514300 messages indicate that the configuration is under stress, often from large spool file requirements on all AMPs. MiniCylPacks tend to occur across all AMPs until spool requirements subside. This impacts all running requests.

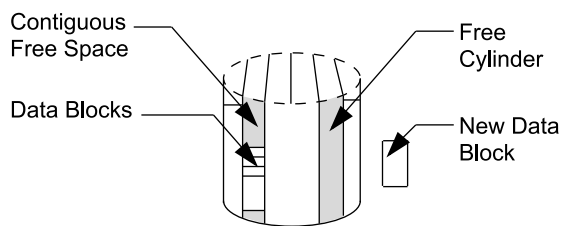
If table data is skewed, you might see MiniCylPacks even if Vantage has not used up most of the disk space.

Defragmentation

As random updates occur over time, empty gaps become scattered between data blocks on the cylinder. This is known as fragmentation. When a cylinder is fragmented, total free space may be sufficient for future updates, but the cylinder may not contain enough contiguous sectors to store a particular data block. This can cause cylinder migrates and even new cylinder allocations when new cylinders may be in short supply. To alleviate this problem, the file system defragments a fragmented cylinder, which collects all free space into contiguous sectors.



Use the DBS Control (see “DefragLowCylProd” in *Teradata Vantage™ - Database Utilities*, B035-1102) to specify the free cylinder threshold that causes defragmentation. When the system reaches this free cylinder threshold, it defragments cylinders as a background task.



To defragment a cylinder, the file system allocates a new cylinder and copies data from the fragmented cylinder to the new one. The old cylinder eventually becomes free, resulting in a defragmented cylinder with no change in the number of available free cylinders.

Since the copy is done in order, this results in the new cylinder having a single, free-sector entry that describes all the free sectors on the cylinder. New sector requests on this cylinder are completed successfully, whereas before they may have failed.

Ferret Defragmentation

The Ferret Defrag command is a defragmentation command. It can be used to scan the entire set of user cylinders and then to defragment the qualified cylinders.

Running Ferret Defrag command, however on a busy customer system affects the performance of the foreground tasks and thus system performance. Moreover, Ferret Defrag has no way to know if there are enough free cylinders already available in the system, so that no more defragmentation is required.

Performance Degradation

While AutoCylPacks, MiniCylPacks and defragmentation help the system reclaim free disk space for further use, they incur a performance degradation. Properly size and tune the system to avoid this overhead.

Freeing Space on Cylinders

In addition to MiniCylPack and AutoCylPack, you can use the following to free space on cylinders or to make more cylinders available to the system:

- FreeSpacePercent (FSP) (see [PACKDISK and FreeSpacePercent](#))
- PACKDISK (see *Teradata Vantage™ - Database Utilities*, B035-1102)
- Cylinders to Save for Perm (see “Cylinders Saved for PERM” in *Teradata Vantage™ - Database Utilities*, B035-1102)
- Temporary space limits on users/profiles.
- Spool space limits on user profile definitions:
 - Set larger spool space limits.
 - Minimize the number for individual users to limit runaway query spool space.

Both of these actions will not necessarily stop space management routines, such as MiniCylPack from running, but it could help manage the area.

- Archival and deletion of aged data

Planned deletion of obsolete rows facilitates space availability. Depending on the nature of your data, you may want to archive before deleting.

- Appropriate data compression
- Additional disk space
- Appropriate data block sizing:
 - Maximum block size allocation
 - Minimum block size allocation
 - Journal data block size allocation

Managing I/O with Cylinder Read

Cylinder Read is a method of loading data blocks in a single database I/O operation. Instead of block I/Os for operations such as table scans and joins that process most or all of the data blocks of a table, Cylinder Read issues an I/O of up to 2 MB.

Because Cylinder Read loads the desired data blocks in a single database I/O operation, Vantage incurs I/O overhead only once per cylinder rather than for each data block.

Cylinder Read and I/O

With larger block sizes, it is possible that the frequency of Cylinder Reads will decrease, particularly for moderate or smaller tables. This may increase block-at-a-time I/O requests, driving up I/O demand.

To be a candidate for Cylinder Read on an AMP, the cylinder must contain 6 or more data blocks for the table being accessed. When you increase the block size drastically, more rows fit into each data block and fewer data blocks are needed to hold the rows of the table on each AMP. For large tables this may not be a problem, but you may end up doing more I/O operations on moderate or smaller tables.

Cylinder Read allows you to keep smaller block sizes, while still having the advantage of reading many rows in one I/O operation when scanning.

Using Cylinder Read for WAL Log Maintenance

Setting Cylinder Read to LogOnly using `ctl` limits cylinder scan to WAL log maintenance only. With LogOnly, the average size of the WAL log should be reduced. Table scan jobs may run slower than before because individual data blocks are read rather than whole cylinders.

Setting Cylinder Read to LogOnly may reintroduce some Cylinder Read performance anomalies that may have caused you to disable Cylinder Read in the first place. But completely disabling Cylinder Read can cause WAL log maintenance to fall behind, thereby causing a general system slowdown.

But by setting Cylinder Read to LogOnly, users only use Cylinder Read for WAL log maintenance and not for table scans, and because WAL log maintenance only runs in the background once a minute, the number of anomalies should be less than those you may have experienced when Cylinder Read was fully enabled.

Tracking Cylinder Read Resource Usage

To track Cylinder Read behavior if you enable resource usage logging, see the Cylinder Read Columns in the “ResUsageSvpr Table” in *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

Managing Free Space

Evaluating Free Space Percent Requirements

Reserved free space allows tables to expand within their currently allocated cylinders. This can prevent or delay the need for additional cylinders to be allocated, which incurs the overhead of moving data to the new cylinders. Avoiding new cylinder allocations can improve overall system performance.

Choosing an appropriate FSP value for a table involves both performance and space considerations, and depends on the growth needs of the table:

- Reference tables that experience no modifications or growth require no FSP, so FSP can be zero for these types of tables. If the system is primarily these types of tables, set `FreeSpacePercent` to zero, and use the `FREESPACE` option of the `CREATE TABLE` and `ALTER TABLE` SQL statements to set a different FSP value for tables that will experience growth.
- Tables that are expected to experience large scale growth require higher FSP values than tables that grow to lesser degrees. However, larger FSP values consume more storage, so FSP choice should balance space and performance considerations.

A table that would require 100 cylinders of storage with 0% FSP, requires 134 cylinders when FSP is set to 25%. If FSP is 75%, that same table would require 400 cylinders. Ensure that the requisite cylinders are available to satisfy the storage overhead added by the FSP, or performance can suffer.

With time, the available free space may change, due to table modifications that do not honor the `FreeSpacePercent` setting. The `AutoCylPack` background task runs periodically to check and restore the FSP for tables. The `Ferret PACKDISK` command can be run manually to force FSP to be restored, or to temporarily set a table FSP to a different value. `MiniCylPack` may change the available free space if there is a shortage of cylinders on the system.

Effects of FreeSpacePercent

FreeSpacePercent (FSP) is a system-wide parameter. FSP does not override a value you specify for an individual table in a CREATE or ALTER TABLE request.

MiniCylPack operations attempt to honor the FSP specified in a CREATE TABLE and ALTER TABLE statements or, if no table-specific FSP is specified, the system default FSP. If MiniCylPack cannot reclaim space while honoring those values, it will keep trying to use a degraded (smaller FSP) value until space can be reclaimed. This continues until MiniCylPack attempts to use an FSP value of 0 for all cylinders.

In some situations, Vantage runs out of free cylinders even though over 20% of the permanent disk storage space is available. This is due to a:

- Higher FSP setting than necessary, which causes the system to allocate unneeded space
- Lower FSP setting than necessary, which causes the system to over-allocate new cylinders
- Low storage density (utilization) on large tables due to cylinder splits

Operations Honoring the FSP

When adding rows to a table, the file system can choose either to use 100% of the storage cylinders available or to honor the FSP. The following operations honor FSP:

- FastLoad
- MultiLoad into empty tables
- Restore
- Table Rebuild
- SQL to add fallback
- SQL to create an SI

Operations That Disregard FSP

The following operations disregard FSP:

- SQL inserts and updates
- T pump
- MultiLoad inserts or updates to populated tables

If your system is tightly packed and you want to apply or reapply FSP, you can:

- Specify the IMMEDIATE clause with the ALTER TABLE statement on your largest tables.
- DROP your largest tables and FastLoad them.
- DUMP your largest tables and RESTORE them.
- In Ferret, set the SCOPE to TABLE and PACKDISK FSP = xxxx

In each case, table re-creation uses utilities that honor the FSP value and fills cylinders to the FSP in effect. These options are only viable if you have the time window in which to accomplish the processing. Consider the following guidelines:

- If READ ONLY data, pack tightly (0%).

- For INSERTs:
 - Estimate growth percentage to get FSP. Add 5% for skewing.
 - After initial growth, FSP has no impact.
 - Reapply FSP with DROP/FASTLOAD, DUMP/RESTORE or PACKDISK operations.
 - Experiment with different FSP values before adding nodes or drives.

Determining a Value for FSP

Use the data in the following table to determine a value for FSP.

IF the majority of tables are...	THEN...
read-only	set the default system-wide FSP value to 0. You can override the FSP for the remaining modifiable tables on an individual table basis with the ALTER TABLE statement.
NOT read-only	the FSP value depends on the percentage of increase in table size due to the added rows. Set the FreeSpacePercent parameter to reflect the net growth rate of the data tables (inserts minus deletes). Common settings are 5 to 15%. A value of 0% is appropriate for tables that are not expected to grow after initially being loaded. For example, if the system keeps a history for 13 weeks, and adds data daily before purging a trailing 14th week, use an FSP of at least 1/13 (8%). To accommodate minor data skews and any increase in weekly volume, you can add an extra 5% (for a total of 13%) FSP.

Because the system dynamically allocates free cylinder space for storage of new or updated rows, leaving space for this during the initial load allows a table to expand with minimal cylinder splits and migrates. However, if you do not expect table expansion, that is, the majority of tables are read-only, use the lowest value (0%) for FSP.

If the system default FSP is zero, do as shown in the following table to minimize problems.

IF you...	THEN...
use read-only tables	change nothing. At load time, or PACKDISK time, the system stores tables at maximum density.
add data via BTEQ or a CLI program	set the FREESPACE value on the CREATE TABLE statement to an appropriate value before loading the table. If the table is loaded, use the ALTER TABLE statement to change FREESPACE to an appropriate value before running PACKDISK.

If you set FSP to a value other than 0, tables are forced to occupy more cylinders than necessary. The extra space is not reclaimed until either you insert rows into the table, use the Ferret utility to initiate PACKDISK on a table, or until MiniCylPack is performed due to a lack of free cylinders.

When the system default FSP is greater than 0, use the information in the following table to minimize problems.

IF you...	THEN...
use read-only tables	set FREESPACE on the CREATE TABLE statement to 0 before loading the table. If the table is loaded, use the ALTER TABLE statement to change FREESPACE to 0 before running PACKDISK.
add data via BTEQ or a CLI program	change nothing. The system adds rows at maximum density.

Adjusting FREESPACE for a Table

You can specify the default value for free space left on a cylinder during certain operations using the FREESPACE option in the CREATE TABLE and ALTER TABLE statements.

You can select a different value for tables that are constantly modified versus tables that are only read after they are loaded. To specify the global free space value, use FreeSpacePercent (see “FreeSpacePercent” in *Teradata Vantage™ - Database Utilities*, B035-1102).

PACKDISK and FreeSpacePercent

When Vantage runs out of free cylinders, you can run Ferret PACKDISK, an expensive overhead operation, to compact data to free up more cylinders.

To reduce the frequency of PACKDISK operations:

- When FastLoading tables to which rows will be subsequently added, set FSP to 5-20% to provide enough free space to add rows.
- For historical data, when adding and deleting data, provide enough free space to add rows.

For example, you add up to 31 days before deleting on a table with six months history.

- Add one month to six months: $1/7 = 14.3\%$
- Safety - plan on 1.5 months, $1.5 / 7.5 = 20\%$

Set Free Space Percent to 20%.

- For historical data and fragmented cylinders:
 - For large tables, either set FSP to 20 - 35%, or set MaxBlockSize to a smaller size (for example, 16 KB).
 - Translate free space to the number of data blocks; at least 6-12 blocks of free space.
- Specify the IMMEDIATE clause with the ALTER TABLE statement.

The table header contains the FSP for each table. If you change the default FSP, the system uses the new default the next time you modify the table. FSP has no effect on block size.

Running Other Utilities with PACKDISK

If you run PACKDISK frequently, use the following tools, two of which are utilities, to determine the amount of free space:

- DBC.DiskSpaceV
- SHOWSPACE, a Ferret command, shows you the percent of free space per cylinder.

If this figure is low, it will impact performance by performing dynamic cylinder packs when the system needs contiguous space.

- SHOWFSP, a Ferret command like SHOWSPACE, helps find tables that need packing.

SHOWFSP shows the number of cylinders that can be freed up for individual tables by specifying a desired free space percent, and is useful in discovering which tables would free the most cylinders if PACKDISK were run on them

Cylinder Splits

A FreeSpacePercent value of 0% indicates that no empty space is reserved on disk cylinders for future growth when a new table is loaded. That is, the current setting causes each data cylinder to be packed 100% full when a new table is loaded.

Unless data is deleted from the table prior to subsequent row inserts, the 0% value guarantees a cylinder split the first time that a row is inserted into the table (following the initial load). Cylinder splits consume system I/O overhead and often result in poor use of data cylinders.

PACKDISK, FSP, and Cylinder Allocations

Packing data for tables being modified too tightly can result in too many cylinder allocations, which reduces the number of free cylinders. For example, if the data on a table occupying 10 cylinders is packed to 0% free space, inserting one row on each cylinder (for a total of only 10 rows) may require Vantage to allocate up to 10 additional new cylinders.

However, if you run PACKDISK on the table with an FSP of 10%, the original table would occupy 11 cylinders. When the same 10 rows are inserted into this table, there will most likely be space available on the existing cylinders and the table would remain, consuming only 11 cylinders. This leaves the other 9 cylinders that were previously used free and available for future cylinder requests. Another way to accomplish the same goal is to update the table first and then perform the PACKDISK operation to free up any cylinders that are not required.

Running the PACKDISK command in this example saved 9 free cylinders, but required manual intervention. Another way to accomplish this is to specify an FSP of 10% in the CREATE TABLE or ALTER TABLE statement, and then let AutoCylPack adjust the table as part of its background activity. If new rows are added to the table in this example on a regular basis, then the AutoCylPack solution is preferable. If new rows are rarely added, then it is better to leave the table packed with 0% free space and run a PACKDISK when required.

Data Compression and Performance

Compression is useful to free up additional disk space, but it may or may not enhance performance depending on the type of compression used, the workload, the frequency of data access, and the system capabilities. Compression generally affects performance as follows:

- The cycle of decompressing and recompressing required to access and store data may significantly increase CPU usage compared to uncompressed data.
- Storing data in dense, compressed form may reduce I/O requirements.

For details about compression methods, see *Teradata Vantage™ - Database Design*, B035-1094.

Multivalue Compression (MVC)

MVC compresses recurring values within a column into a single value in the table header.

Performance Impact

MVC enhances the performance of high data volume applications, like call record detail and click-stream data, and provides significant performance improvement for general ad hoc workloads and full-table scan applications.

Smaller physical row size results in less data blocks and fewer I/Os and improved overall performance, depending upon amount of compression achieved.

Select and delete operations show a proportionate improvement in all cases. Inserts and updates show mixed results. The load utilities benefit from the compressed values.

Algorithmic Compression (ALC)

ALC includes:

- Teradata-supplied UDFs to compress various types of character data
- The ability to create custom UDF algorithms

Performance Impact

Teradata-standard ALC UDFs tend to reduce I/O, but can increase CPU usage. For information on the compression/decompression functions, see *Teradata Vantage™ - SQL Operators and User-Defined Functions*, B035-1210.

Block-Level Compression

BLC compresses data at the data block level.

Because BLC is CPU-intensive, consider the CPU capabilities of the particular Vantage platform when using BLC.

When using BLC on the Teradata appliance systems with very high CPU capabilities, you can:

- Apply BLC to all allowable types of data.

- Load and access data at any time because there is enough available CPU to frequently decompress data

When using BLC on other platforms with less CPU capacity, you may need to:

- Load tables during off-peak hours.
- Limit access to compressed tables during critical throughput periods.

Some operations (including queries, insert/updates, and CheckTable) can use considerably more CPU while operating on compressed tables. Unless the system is very CPU rich, these operations can impact other workloads and lengthen elapsed response times.

Managing Data Block Usage

Global settings for data block size are controlled using DBS Control utility settings:

- PermDBAllocUnit
- PermDBSize

You can also adjust the data block size for individual tables using the DATABLOCKSIZE and MERGEBLOCKRATIO options in CREATE TABLE or ALTER TABLE statements.

PermDBAllocUnit and System Performance

As tables are modified, rows are added, deleted, and changed. Data blocks grow and shrink dynamically to accommodate their current contents. However, data block sizes can change only in units of PermDBAllocUnit. This means there will nearly always be some unused space left at the end of the data block. If table modifications are relatively even, such incremental changes in data block size result in an average of approximately half an allocation unit of space wasted for every data block. (This is a rough approximation, and will depend on many factors that differ from database to database.)

In environments where new rows are added frequently to tables, or where tables with variable length rows are frequently growing, system performance might be improved slightly by increasing the allocation unit size. With a larger allocation unit, data blocks will not need to be enlarged as frequently, because there will already be room for additional changes. However, in environments where new rows are not added frequently, the additional space in each block can degrade performance by increasing the average I/O size.

Make only small changes to this setting at a time, and carefully evaluate the results before committing the change on a production system. Set the allocation unit to a multiple of the average row size of tables that change frequently, rounded up to the nearest sector.

Because the benefit of larger allocation units is often offset by the consequent increase in average wasted space, Teradata recommends that PermDBAllocUnit be left at the default setting.

PermDBSize and System Performance

Database performance can be affected by the relationship of data block size to the type of work typically performed by the database:

- When database queries are tactical in nature, involving one or a few table rows, it is advantageous to have fewer rows stored per data block to speed data access. Online transaction processing (OLTP) is an example of this type of work.
- Alternatively, when database queries are strategic in nature, involving complex queries that involve many table rows per table, it is advantageous to have many rows stored in each data block, to minimize costly data I/O operations. Decision support software (DSS) and complex report generation are examples of this type of work.

PermDBSize sets the default maximum size used by the system for multirow data blocks in permanent tables. Use a larger value if the database is used primarily for strategic work, and a smaller value if the database is used primarily for tactical work.

In a mixed-work environment, determine a value for PermDBSize based on the kind of work typically performed by the database. For tables involved in other types of work, PermDBSize can be overridden on a table-by-table basis using the DATABLOCKSIZE option of the CREATE TABLE and ALTER TABLE SQL statements.

Adjusting DATABLOCKSIZE in Tables

You can control the default size for multirow data blocks for individual tables using the:

- DATABLOCKSIZE option
- MERGEBLOCKRATIO option

in the CREATE TABLE and ALTER TABLE statements, as follows.

IF you specify...	THEN...
DATABLOCKSIZE in CREATE TABLE	the datablock can grow to the size specified in DATABLOCKSIZE instead of being limited to the global PermDBSize. For any row, the system uses only the datablock size required to contain the row.
DATABLOCKSIZE in ALTER TABLE	the datablocks can grow to the size specified in DATABLOCKSIZE where the row size requires the growth. Whether they are adjusted to that new size gradually over a long period of time depends on the use of the IMMEDIATE clause.
MERGEBLOCKRATIO in CREATE TABLE	it limits attempts to combine blocks if the resulting size is larger than the specified percent of the maximum multirow datablock size.
MERGEBLOCKRATIO in ALTER TABLE	the size of the resulting block when multiple existing blocks are being merged has an upper limit. The limit depends on whether logically adjacent blocks are deemed mergeable with the single block being modified. Blocks can still be initially loaded at the PermDBSize specification or the block size specified with the DATABLOCK option. Merges occur only during full table modifications. MERGEBLOCKRATIO and the default size for multirow datablocks are unrelated.

IF you specify...	THEN...
the IMMEDIATE clause	<p>the rows in all existing datablocks of the table are repacked into blocks using the newly specified size. For large tables, this can be a time-consuming operation, requiring spool to accommodate two copies of the table while it is being rebuilt.</p> <p>If you do not specify the IMMEDIATE clause, existing datablocks are not modified. As individual datablocks of the table are modified as a result of user transactions, the new value of DATABLOCKSIZE is used. Thus, the table changes over time to reflect the new block size.</p>

To evaluate actual block sizes of tables, you can run the SHOWBLOCKS command of the Ferret utility. For more information on running this command, see *Teradata Vantage™ - Database Utilities*, B035-1102. To specify the global data block size, use PermDBSize (see “PermDBSize” in *Teradata Vantage™ - Database Utilities*, B035-1102).

You can also use SQL macros to generate SHOWBLOCKS-like information. The PopulateFsysInfoTable macro generates file system data to a table created by the CreateFsysInfoTable macro.

The PopulateFsysInfoTable_ANSI macro generates file system data to a table created by the CreateFsysInfoTable_ANSI macro. For more information, see the section on file system information macros in *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145.

Specifying Maximum Data Block Size

You can set maximum block size two ways.

Operation	Comments
Set PermDBSize via the DBS Control	When you set maximum block size at the system level, a table utilizes a value only until a new value is set system-wide.
Use the CREATE or ALTER TABLE command	When you set maximum block size at the table level, this value remains the same until you execute an ALTER TABLE command to change it.

Larger block sizes enhance full-table scan operations by selecting more rows in a single I/O. The goal for DSS is to minimize the number of I/O operations, thus reducing the overall time spent on transactions.

Smaller block sizes are best used on transaction-oriented systems to minimize overhead by only retrieving what is needed.

For more information on data block size, see “PermDBSize” in *Teradata Vantage™ - Database Utilities*, B035-1102. Rows cannot cross block boundaries. If an INSERT or UPDATE causes a block to expand beyond the defined maximum block size, the system splits the block into two or three blocks depending on the following.

IF...	AND...	THEN...
the new or changed row belongs in the beginning or end of a block	a block containing only that row is larger than the defined	the row is placed in a block by itself.

IF...	AND...	THEN...
	maximum block size for the table	
the new or changed row belongs in the middle of a block	a block containing only that row would be larger than the defined maximum size for the table	then a three-way block split is performed. The existing block is split into two blocks at the point where the row being modified belongs and a new block is created between them containing only the modified row.
the existing block size can be changed to accommodate modification and still not be larger than the defined maximum size for the table	case is empty	a single new block with the existing rows and the new or changed row is created.
the existing block size cannot be changed to accommodate modification and still not be larger than the define maximum size for the table	case is empty	the modification is applied and then the block is split into as many parts as required (and as few parts as possible) so that each part is not larger than the defined maximum for the table.

Additional special rules exist that take precedence over the preceding rules. For example:

- Rows of different subtables never coexist in data blocks.
- Spool tables are almost always created as whole blocks all at once with many rows in them with the maximum size defined for spool (as long as there are enough rows).

Using Data Block Merge

The Datablock Merge operation reduces the number of small data blocks in table(s) by combining them with logically adjacent data blocks. It enables the file system to automatically merging smaller data blocks into a single large data block. Having fewer blocks reduces the number of I/O operations required to read and modify a large set of rows from disk. The Merge:

- Runs automatically.
- Does not require manual investigation of block size histograms.
- Does not require any specific AMP level locking.
- Searches continuously for small blocks to merge, even if some of those blocks do not require updates.
- Only affects the performance of workloads that modify data. Read-only workloads, insert operations into new subtables, and prime key modification requests are not affected.

Moreover, Datablock Merge is only applicable to permanent and permanent journal tables, not to global temporary or volatile tables.

For the DBS Control fields that support datablock merge, see *Teradata Vantage™ - Database Utilities*, B035-1102.

For the option, MergeBlockRatio, which defines the size of the resulting block when multiple existing blocks are being merged, see “CREATE TABLE” and “ALTER TABLE” in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

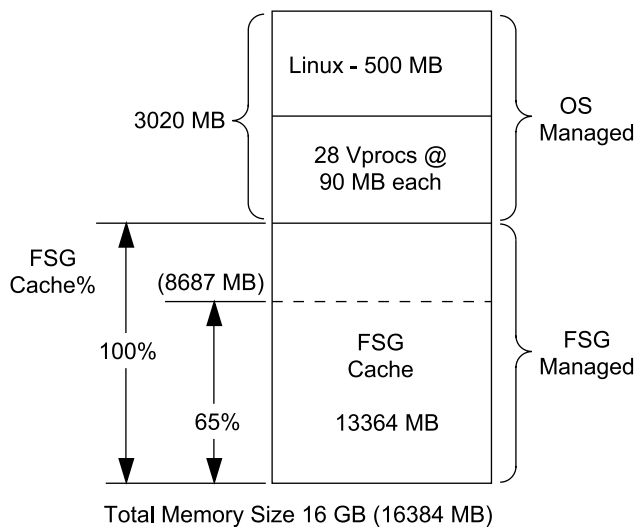
Effects of Journal Data Block Size

Journal data block sizes may affect I/O usage. A larger journal size may result in less I/O or cause wasted datablock space. You can set the JournalDBSize. For more information, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Performance and Memory Management

Shared Memory

The system determines how memory is shared between the OS and FSGCache. For example, the following figure illustrates how the system calculates FSG Cache for a system with 16 GB shared memory.



When Linux boots, PDE:

- Determines how much memory the OS is already using.
In the example, memory is shown to be 500 MB. This number may vary depending on the system configuration.
- Leaves to the OS an amount of memory equal to the number of vprocs times the memory allocated to each vproc.

In the example, this amount is 90 MB for each of the 28 vprocs.

Note:

Vprocs include AMPs, PEs, the Gateway, and the Node vproc.

- Calculates the FSG Cache by (1) subtracting the OS-managed memory from total memory size and then (2) applying a FSG Cache% to that amount.

Note:

To calculate the FSG Cache per AMP, divide the FSG Cache by the number of AMPs.

The amount of OS-managed memory is 3020 MB and the FSG Cache, if the FSG Cache% is 100%, would be 13364 MB.

If the FSG Cache% were set to 65%, then the FSG Cache would be 8687 MB, and the additional free memory would be 13364 MB minus 8687 MB, or 4677 MB.

If you want to run applications with memory requirements unknown to Vantage software, reduce the FSG Cache% to leave memory for the applications.

Note:

FSG Cache is managed in multiples of 4 KB up to 128 KB, and then in increments of 256 KB, 512 KB, and 1 MB.

Reserving Additional Free Memory for Applications

To reserve 20% of the FSG Cache for applications over and above the 90 MB/vproc reserved for the OS, go to the DBS screen and set FSG Cache percent to 80. The system assigns 80% of the FSG Cache to FSG and leaves the remaining 20% for other applications.

For information on ctl, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Memory-Consuming Features

Certain features may require more memory in order to show their optimal performance benefit. Of particular mention are:

- Array INSERT
- Compression
- Cylinder read
- Geospatial data type
- Global and Persistent Data (GLOP)
- Hash joins and product joins
- Join index
- Large objects (LOBs)
- Row and column partitioning
- Table functions
- Stored procedures
- User-defined functions (UDFs)

- XML data type
- 1 MB response buffer
- 1 MB data blocks
- Larger than 1 MB plan cache

Performance gains from implementing these features may be countered by their impact on memory usage. In turn, you may experience more segment swaps and incur additional swap physical disk I/O. To counter this, you can lower the FSG cache percent to ensure that 135 MB per AMP is allocated in OS memory.

Lowering the FSG cache percent may cause fewer cache hits on table data and instead cause a different type of additional physical disk I/O. In general, additional I/O on table data is a lesser performance issue than swapping I/O, but it can still have an impact on performance.

Investigating the Need to Add Memory

There is a recommended amount of memory for each Vantage hardware platform. See the Release Definition for your Vantage release for memory recommendations.

You can also monitor the use of FSG cache memory to determine if you should add more memory to assure full performance. See also [Managing FSG Cache](#).

1. Monitor your system during critical times to determine the ratio of logical to physical I/O.
2. After the lowering of the FSG cache percent to provide more memory for a memory consuming feature, again monitor your system during critical windows to understand the new ratio of logical to physical I/Os.
3. If the amount of FSG cache misses increases by more than 20% *and* the system has become I/O-bound, then adding more memory, if possible, is recommended.

ResUsage and Cache Hit Rates

ResUsage data can help determine cache hit rates by comparing total logical I/O requests to I/O requests that resulted in a physical I/O. Comparing these helps clarify any confusion that may arise when aligning DBQL data with ResUsage data, since DBQL data only tracks logical I/O.

An analogous circumstance occurs when attempting to differentiate between total BYNET message requests and BYNET message requests that result in a physical BYNET message being sent. Smaller systems see a greater difference between the two because of a higher percentage of point-to-point (PTP) messages with the sender and the receiver being the same node.

In ResUsageSpma table, a logical read count, that is, a count of the total number of requested disk segments for the interval, is represented in the FileAcqs column. The physical read count, or the number of requested disk segments that actually resulted in a read by the I/O subsystem, is represented in the AcqReads and the PreReads columns.

There are similar breakdowns for writes. BYNET logical messages are represented in the columns that begin with Msg, and the physical counterparts are represented with the columns that begin with NetMsg.

Monitoring Memory

Use the ResUsage tables to obtain records of free memory usage and FSG Cache.

Memory Type	Monitor With	Comments
Free Memory	ResUsage	See <i>Teradata Vantage™ - Resource Usage Macros and Tables</i> , B035-1099 for more information on using ResUsage macros and utilities to monitor free memory and FSG Cache.
FSG Cache	ResUsageSpma ResUsageSvpr	

Using Memory Effectively

Regardless of the amount of available memory on your system, you need to use it effectively. To determine if your system is using memory effectively:

1. Start with a value for FSG Cache percent. See [Managing FSG Cache](#).
2. Adjust the value based on available free memory requirements. See [Reserving Additional Free Memory for Applications](#) and [Managing Free Memory](#).
3. Consider adjusting other memory-related settings to optimize database function.
 - DBSCacheThr
 - DictionaryCacheSize
 - HTMemAlloc
 - IdCol Batch Size
 - PPICacheThrP
 - RedistBufSize
 - SyncScanCacheThr

For information on how to adjust DBS Control memory settings, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Managing Free Memory

Free memory is used by:

- Administrative programs, such as program text and data, message buffers, kernel resources, and other applications such as FastLoad that require memory use.
- Vprocs for non-file system activity, such as:

Activity	Description
AWT	Pieces of AMP logic used for specific AMP tasks
Parser tasks	Pieces of PE logic responsible for parsing SQL
Dispatcher tasks	Pieces of PE logic responsible for dispatching work

Activity	Description
Scratch segments	Temporary work space
Messages	Communication between vprocs
Dictionary cache	Dictionary steps for parsing
Request cache	Temporary space used when executing steps

Determining Available Free Memory

The ResNode macro displays limited information on free memory. The ResNode report includes the Free Mem% column, which is the percentage of unused memory.

Adjusting for Low Available Free Memory

When the amount of available free memory dips too far below 100 MB (25,000 pages), some sites have experienced issues. 40 MB of free memory is the lowest acceptable amount. You can usually avoid problems if you configure your AMPs to have at least 135 MB of OS-managed memory per AMP. You can adjust the amount of available free memory by performing the following:

- Use `ctl` to adjust the FSG Cache percent downward to make more memory available to free memory. If the system takes too much memory for FSG Cache and the OS does not use that memory, the free memory is wasted.
- If available free memory goes below 100 MB during heavy periods of redistribution, lower the value of the `RedistBufSize` field in the DBS Control Record (see “RedistBufSize” in *Teradata Vantage™ - Database Utilities*, B035-1102).

Assured Minimum Non-FSG Cache Size

Teradata recommends these guidelines for minimum non-FSG Cache size per AMP.

- 135 MB per AMP when all nodes are up.
- 112 MB per AMP when 1 node is down in a clique.
- 90 MB per AMP when the maximum number of nodes allowed down are down in a clique.

These configuration guidelines help avoid performance issues with respect to memory swaps and paging, memory depletion, and CPU starvation when memory is stressed.

Performance Management Recommendations

Many sites may require more free memory than the default calculated on [Shared Memory](#). Teradata recommends that you provide additional memory per AMP to free memory by setting the FSG Cache percent to a value less than 100%.

Use the following calculation:

FSG Cache percent = (FSG Cache - 39 MB * # AMPs) / FSG Cache

Memory Size (GB)	Memory for Baseboard Drivers & 10 AMPs /2PE Vprocs	FSG Cache (MB)	Less 58 MB per AMP Vprocs	FSG Cache Percent
6.0	1854	4290	3822	89%
8.0	1914	6278	5810	92%

For large configurations, consider one or more of the following options to resolve I/O bottlenecks or excessive memory swapping:

- Consider using aggregate join indexes to reduce calculations during query processing.
- Set RedistBufSize to an incremental value lower; for example, from 4 KB to 3 KB
- Set FSG Cache percent to less than 100%, taking into account total memory size.
- Consider modifying the application to reduce row redistribution.
- Ask your support representative to reduce the internal redistribution buffer size.

Note:

This is an internal tuning parameter, *not* the user-tunable RedistBufSize.

Potential Problems

A possible problem is when an application on a large configuration generates many messages over the BYNET with concurrent row redistributions involving all nodes.

The following are not a problem:

- Row duplications
- Merging of answer sets

Row Redistribution Memory Requirement

To avoid the overhead of sending lots of little messages across the BYNET, buffers are used to batch up individual rows during the row redistribution process. Both load utilities and queries involve such redistribution, but their approach to outbound buffering is different.

Row redistribution for query processing uses separate single buffers per AMP for each node in the system. This means that the amount of memory required for redistribution in a node grows as the system grows.

The DBC Control Record RedistBufSize field controls the size of redistribution buffers. See [RedistBufSize Performance Implications](#).

- Default query redistribution buffer size = 32 KB per target node
- Total memory for one sending AMP = 32 KB * number of nodes in system
- For eight AMPs per node, total memory required per node =
8 * 32 KB * number of nodes in system

Redistribution Buffer Resizing

The following example provides the calculations for resizing the redistribution buffer based on a configuration of 8 nodes at eight AMPs per node. (The system reserves 32MB per AMP).

- Single node requirement (single user) = 32 KB * 8 = 256 KB
- Multi-user (20 concurrent users) = 20 * 256 KB = 5 MB (less than the 32 MB default)

The following example provides the calculations for a configuration of 96 nodes at 8 AMPs per node and shows that the requirement exceeds the system default:

- Single node requirement (single user) = 32 KB * 96 = 3072 KB (3 MB)
- Multi-user (20 concurrent users) = 20 * 3072 KB = 64 MB (far exceeding 32 MB per AMP)

Performance effects of high-volume redistribution processing include:

- Excessive memory paging/swapping
- Possible I/O bottleneck on BYNET I/O

Managing FSG Cache

System Usage of FSGCache

The file system manages FSG Cache, which is used by:

- AMPs on the node
- Backup activity for AMPs on other nodes

The file system uses FSG Cache for file system segments such as:

- Permanent data blocks (includes fallback data and SIs)
- Cylinder Indexes for permanent data blocks
- Cylinder statistics for Cylinder Read
- Spool data blocks and Cylinder Indexes for spool
- WAL space, including Transient Journal (TJ) rows and WAL REDO records
- Recovery journal rows

Space in FSG Cache

Space in the FSG Cache is not necessarily evenly distributed among AMPs. It is more like a pool of memory; each AMP uses what it needs.

FSG cache contains the most recently used database segments. When Vantage needs to read a data block, it checks and reads from cache instead of from disk, whenever possible.

The system performs optimally when FSG Cache is as large as possible, but not so large that not enough memory exists for the database programs, scratch segments, and other operating system programs that run on the node.

Calculating FSG Cache Size Requirements

The FSG Cache percent field controls the percentage of memory to be allocated to FSG Cache. You can change the value in FSG Cache percent using the ctl utility. See the section on setting variables in “Control GDO Editor (ctl)” in *Teradata Vantage™ - Database Utilities*, B035-1102.

First, configure sufficient operating system memory, using the guidelines discussed in [Managing Free Memory](#). Then let the remaining memory be allocated to FSG Cache.

Calculating FSG Cache Read Misses

To calculate if FSG Cache read misses have increased, use the following formulas:

- FSG Cache read miss = physical read I/O divided by logical read I/O

Physical read I/O counts can be obtained from ResUsageSpma table by adding FileAcqReads + FilePreReads.

Logical I/O counts can be obtained from ResUsageSpma table column FileAcqs.

- Increase in FSG Cache misses = FSGCacheReadMissAfter divided by FSGCacheReadMissBefore

While Vantage cannot guarantee a particular improvement in system performance, experience has shown gains of 2-8% when adding 1GB of memory per node in such instances.

Using DBS Control Fields to Manage Memory

The DBS Control utility displays and allows modification of various database-level configuration parameters. Several of these can be used to tune memory management and in that way affect system performance.

DictionaryCacheSize Performance Implications

The default value allows more caching of table header and database object access rights information, and reduces the number of I/Os required. It is especially effective for workloads that access many tables (more than 200) and for those that generate many dictionary seeks.

Increase the size of the dictionary cache to allow the parser to cache additional data dictionary and table header information.

For tactical and Online Complex Processing (OLCP) type workloads, maintaining a consistently short, few-second response time is important. These workloads may benefit from a larger dictionary cache, particularly when their query plans have not been cached in the request cache. A larger dictionary cache will allow more dictionary detail, needed for parsing and optimizing, to remain in memory for a longer period of time. For query workloads with a response time of more than one minute, there may be no measurable difference when this field is set to a higher value.

IdCol Batch Size Performance Implications

The IdCol Batch Size setting involves a trade-off between insert performance and potential gaps in the numbering of rows inserted into tables that have identity columns.

A larger setting results in fewer updates to DBC.IdCol in reserving batches of numbers for a load. This can improve the performance of bulk inserts into an identity column table. However, because the reserved numbers are kept in memory, unused numbers will be lost if a database restart occurs, resulting in a gap in the numbering of identity columns.

PPICacheThrP Performance Implications

Under most use cases, the default value for PPICacheThrP is adequate, and should not be changed. However, if there are performance issues that might be addressed by adjusting this value, consider the information in this section.

The current data block for the corresponding partition (or buffer, in the case of inserts to column-partitioned tables) is associated with each context. The current set of data blocks or buffers (one for each context) are kept in memory, if possible, to improve the performance of processing the set of partitions at the same time. If there is a shortage of memory, these blocks or buffers may need to be swapped to disk. Excessive swapping, however, can degrade system performance.

Larger values may improve the performance of partitioning operations, as long as the following occur:

- Data blocks or AMP buffers for each context can be kept in memory. When they can no longer be kept in memory and must be swapped to disk, performance may degrade.
- The number of contexts does not exceed the number of nonempty, noneliminated partitions for partitioning operations. (If they do, performance will not improve because each partition can have a context, and additional contexts would be unused.)

In some cases, increasing the value of PPICacheThrP above the default value can provide a performance improvement for individual queries that do these partitioning operations. However, be aware of the potential for memory contention and running out of memory if too many of these queries are running at the same time.

The default setting of 10 is conservative, and intended to avoid such memory problems. With 80 AMP Worker Tasks (AWTs) per AMP on a system with the default setting of 10, the maximum amount of FSG cache that could be used for these partitioning operations is 80% of FSG cache memory, if all AMPs are simultaneously executing partitioning operations, such as sliding-window joins for 80 separate requests. For configurations that have more than 80 AWTs defined as the maximum, the setting is scaled to the number AWTs. For example, at the default setting of 10, a cap of 80% of FSG cache memory per AMP would still be in effect on such systems.

For many sites, the default may be too conservative. All 80 AWTs might not be running partitioning operations at the same time. If, at most, 60 partitioning operations are expected to occur at the same time, the value of PPICacheThrP could possibly be raised to 15. If at most 40 are expected, the value could possibly be raised to 20, and so on. The best value for this parameter is dependent on the maximum concurrent users expected to be on the system and their workload. No one value is appropriate for all systems.

Also, consider that the number of concurrent partitioning operations, such as sliding-window joins, may increase as partition usage is increased. Increasing the value may increase performance now without memory contention or running out of memory but, in the future, as more partitioning operations run concurrently, performance may decrease, or out of memory situations may occur.

If less than 80 concurrent partitioning operations are expected for your site, and you think that better performance may be possible with an increased value for PPICacheThrP, you can experiment with PPICacheThrP settings to determine an increased PPICacheThrP setting that is optimal for your site and safe for your workloads. Measure pre- and post-change performance and degree of memory contention under expected current and future workloads to evaluate the effects of the change. If increasing the value to one that is reasonably safe for your site does not yield adequate performance for partitioning operations such as sliding-window joins, consider defining partitions with larger granularity, so fewer partitions are involved in the sliding-window join.

RedistBufSize Performance Implications

To avoid the overhead of sending many small messages across the BYNET, buffers are used to batch individual rows during the row redistribution process. RedistBufSize setting determines the size in kilobytes for row redistribution buffers used during load utilities.

The default is 4 (which translates to 4 KB), and each AMP will have one such buffer in memory for each AMP in the configuration during a load job row redistribution.

If a system has relatively few AMPs, a larger redistribution buffer size usually has a positive effect on load performance. However, on larger systems with many AMPs, a large buffer size can consume excessive memory, especially if many load jobs are run concurrently.

Maintain the RedistBufSize at 4 KB for up to 48 nodes with 8 AMPs/node. As the system configuration grows larger, you can compensate by doing one or more of the following:

- Set RedistBufSize smaller in proportion to the increase in the total number of AMPs, that is, send more smaller messages)
- Add more memory to increase the total memory in the system to accommodate the redistribution buffers somewhat larger.
- Increase the amount of free memory available for redistribution buffers by setting FSGCache percent smaller

To help determine if RedistBufSize is too high, see if the minimum available free memory consistently goes below 100 MB during heavy periods of load utility redistribution. Also, check for significant swapping (more than 10/second) during this period. If this is the case, reduce RedistBufSize an incremental value lower, for example, from 4 KB to 3 KB.

Periodic Maintenance and Performance

Cleaning Up Logs

You should periodically remove old information from the DBC.Acctg table. For example, the following command will delete entries for September 2006:

```
DELETE FROM DBC.ACCTG WHERE SUBSTR(ACCOUNTNAME, 1, 6) = 'CB0609';
```


Note:

To minimize performance impact, perform any necessary maintenance to DBC tables when the system is quiescent or least busy. Cleaning certain tables, such as DBC.AccLogTbl lock the table and prevent access to the system.

DefragLowCylProd

A low value in this field reduces the performance impact of defragmentation. However, setting the value extremely low might cause cylinders to be consumed more quickly, which could cause more MiniCylPacks to run. Set DefragLowCylProd higher than MiniCylPackLowCylProd because defragmentation has a smaller performance impact than cylinder pack.

Capacity Planning

Monitoring CPU Usage

Vantage records both normalized and raw CPU measures. Normalized CPU time is derived by applying a CPU scaling factor to the node level raw CPU time.

The standard co-existence scaling factors for all node types are pre-defined in PDE startup parameter files in the Open PDE startup.txt file. The per-node values are added to the vconfig and tosgetpma() structures by PDE startup for use by other components. In this way, Vantage provides accurate performance statistics for mixed node systems, particularly with respect to CPU skewing and capacity planning, that is, usable capacity.

The formula is:

$$\text{Node_Scaling_Factor} * \text{Node_CPU_Time}$$

The scaling factor values are derived from:

- Measuring the raw capability of the Teradata compute node unconstrained by I/O or environment.
- Basing values on basic functionality, including data access rate, data load rate, row processing rate, raw transaction rate.

Currently, the 5100 is the base scaling factor value (1.00).

For AMP level reporting in DBQL and AMPUsage, the formula is:

$$\text{Node_Scaling_Factor} * \text{AMP_CPU_Time}$$

ResUsage and DBC.AMPUsage View

The DBC.AMPUsage view displays CPU usage information differently than the way usage information is displayed in ResUsage data.

This facility...	Provides...
ResUsage	metrics on the system, without making distinctions by individual user or account ID.
DBC.AMPUsage view	<p>AMP usage by individual user or account ID.</p> <p>AMPUsage counts logical I/Os, not physical.</p> <p>Some CPU seconds used by the system cannot be accounted for in AMPUsage. Therefore, ResUsage CPU metrics will always be larger than AMPUsage metrics. Typically, AMPUsage captures about 70-90% of ResUsage CPU time.</p>

For more information on the DBC.AMPUsage view, see *Teradata Vantage™ - Data Dictionary*, B035-1092.

Using the ResNode Macro Set

For capacity planning, generally only ResUsageSpma is required. This is the only table required to make use of the ResNode macro set.

Important information from ResNode includes:

- CPU utilization
- Parallel efficiency to show hot nodes or AMPs
- CPU to I/O balance
- OS busy versus DBS busy to see the characteristics of the workload
- Memory utilization
- Availability and process swapping (paging)
- Network traffic

Observing Trends

ResUsage data is most useful in seeing trends when there is a reasonably long history (more than a year) available for comparison. Use this data to answer questions, such as:

- How heavily is the system used at different times of the day or week?
- When are there peaks or available cycles in utilization?

CPU Busy

The ResNode macro reports CPU busy by second averages. Use it for general system analysis.

The macro contain the Avg CPU Busy column, which is the average CPU utilization for all nodes. Avg CPU Busy % is a measure of how often multiple CPUs in a node were busy during a log period.

- In a DSS environment, a small number of jobs can easily bring the CPU close to 100% utilization.
- High CPU utilization consists of a Avg CPU Busy % of 70% over significant periods of time.
- The CPU is stressed differently in DSS and transaction processing environments.

For ResNode columns, see *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

CPU Tasks in a DSS Environment

The following lists how CPU tasks are carried out on the node during DSS operations.

1. Prepare for read:
 - Memory management allocates memory for the data block.
 - Database software communicates with the file system.
 - File system communicates with the disk controller.
2. Qualify rows. Determine if the row satisfies the WHERE clause condition(s).

Most DSS operations require full table scans in which the WHERE clause condition check is relatively time-consuming. Full table scans generally result from SQL statements whose WHERE clause does not provide a value for an index or partition elimination.
3. Process rows:
 - Join
 - Sort
 - Aggregate
4. Format qualifying rows for spool output.

CPU Tasks During Transaction and Batch Maintenance Processing

The following describes how the CPU tasks are carried out on the node during a transaction batch maintenance processing.

Notice that the qualify rows activity is missing from the table. In transaction processing, it is more common for the WHERE clause to provide a value for the PI or USI. The read itself qualifies rows. Transaction processing typically avoids further conditional checks against non-indexed columns. All of these CPU tasks occur on the nodes.

1. Prepare for read:
 - Memory management allocates memory for the data block.
 - Database communicates with the file system.
 - File system communicates with the disk controller.
2. Update row:
 - Database locates row to be updated.
 - Memory management allocates memory for the new data block to be built.
 - Database updates the changed row and copies the old rows.
 - Database communicates with the file system.
 - File system communicates with the disk controller.

Parallel Node Efficiency

Parallel node efficiency is a measure of how evenly the workload is shared among the nodes. The more evenly the nodes are utilized, the higher the parallel efficiency.

Parallel node efficiency is calculated by dividing average node utilization by maximum node utilization. Parallel node efficiency does not consider the heaviness of the workload. It only looks at how evenly the nodes share that workload.

The closer parallel node efficiency is to 100%, the better the nodes work together. When the percentage falls below 100%, one or a few nodes are working much harder than the others in the time period. If node parallel efficiency is below 60% for more than one or two 10-minute log periods, Vantage is not getting the best performance from the parallel architecture.

Poor Parallel Efficiency

Possible causes of poor node parallel efficiency include:

- Down node
- Uneven number of AMPs per node
- Skewed table distribution
- Skewed join or aggregation processing
- Non-Vantage application running on a TPA node
- Coexistence system with different speed nodes

Poor parallel efficiency can also occur at the AMP level. Common causes of poor AMP parallel efficiency include:

- Poor table distribution (You can check this in DBC.TableSizeV.)
- Skewed processing of an SQL statement:
 - User CPU in seconds (You can check this in DBC.AMPusage.)
 - Spool (You can check this in DBC.DiskSpaceV.)

CPU Use

The following macros provide information on CPU use.

Macro	Table	Purpose
ResCPUByNode	SPMA	Report of how each individual node is utilizing CPUs
ResCPUByPE	SVPR	Report of how each Parsing Engine (PE) utilizes the CPUs on irrespective node
ResCPUByAMP	SVPR	Report of how each AMP utilizes the CPUs on the respective node

For information on these macros, see *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

Monitoring Disk Utilization

ResUsage and Disk Utilization

The ResNode macro reports disk utilization averages by second.

Parallel Disk Efficiency

Parallel AMP efficiency does not necessarily match parallel disk efficiency. Monitor parallel disk efficiency using ResUsageSvdsk and ResUsageSpdsk tables.

The following table lists common mistakes that can cause skewing and poor parallel efficiency and suggests solutions.

Mistake	Solution						
A user did not define a PI. The system uses the first column of the table as the default P1.	Define a PI with good data distribution.						
A user used a null as PI for the target table in a left outer join.	Perform any of the following: <ul style="list-style-type: none"> Choose a different PI. Handle NULL case separately. Use a multiset table. 						
A user performed a join on column with poor data distribution. For example, the user entered: <pre>SELECT A.colname, B.x, B.y FROM A, B WHERE A.colname = B.colname;</pre>	<ul style="list-style-type: none"> Identify column value and counts. For example, enter: <pre>SELECT colname, count(*) FROM T HAVING count(*) > 1000 GROUP BY 1 ORDER BY 1 Desc;</pre> The following displays: <table> <thead> <tr> <th><u>colname</u></th><th><u>count(*)</u></th></tr> </thead> <tbody> <tr> <td>codeXYZ</td><td>720,000</td></tr> <tr> <td>codeABC</td><td>1,200</td></tr> </tbody> </table> Break the query into two separate SQL statements. For example, handle codeXYZ only in one SQL statement; handle all other cases in another SQL statement. Collect statistics on the join column. 	<u>colname</u>	<u>count(*)</u>	codeXYZ	720,000	codeABC	1,200
<u>colname</u>	<u>count(*)</u>						
codeXYZ	720,000						
codeABC	1,200						

Monitoring Traffic

ResUsage and Host Traffic

The ResHostByLink macro analyzes the traffic flow between Vantage and the mainframe or workstation client and reports averages by second.

For ResHostByLink columns, see *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

ResUsage and BYNET Data Macro

Macro	Description	Purpose
ResNode	(System provided) by second averages	General system analysis

For ResNode columns, see *Teradata Vantage™ - Resource Usage Macros and Tables*, B035-1099.

Performance Considerations when Expanding or Upgrading the Database System

Expansion involves adding any combination of disk arrays, memory, vprocs, or nodes (with BYNETs).

Upgrade means upgrading the Vantage software.

Adding Disk Arrays

To determine if the system needs more storage, look at the ResUsageSvpr table for unusual disk activity, such as frequent:

- MiniCylPacks
- Defrags
- Packdisks

You may need to add more storage capacity to existing nodes when:

- Excessive disk activity (I/O) is impacting performance
- Application changes require additional spool space
- Database growth requires additional storage

Adding Vprocs

The following table lists reasons for adding vprocs.

IF you want to ...	THEN add ...
increase storage capacity	<p>disks, probably with AMPs, and perhaps nodes. When you add storage, you normally add AMPs.</p> <p>The number of AMPs you add depends on the number of ranks assigned to the existing AMPs. For example, if you add two disk cabinets to each of 20 ranks, you would add 10-20 more AMPs to your configuration.</p> <p>Note:</p> <p>Teradata recommends that you increase memory when you add AMPs. If your existing CPUs cannot handle the load caused by the additional AMPs, you may also need to add nodes.</p>
reduce single-user response time	<p>AMPs.</p> <ul style="list-style-type: none"> • Optimization of single-user throughput or response time is especially significant when there are fewer AMPs than CPUs per node. • In a concurrent workload, you might be able to achieve the desired throughput by adding more AMPs to existing nodes. • Be sure your memory is adequate for the additional AMP workload.

IF you want to ...	THEN add ...
increase the capacity for concurrent sessions	<p>PEs, perhaps nodes, and perhaps a mainframe connection.</p> <ul style="list-style-type: none"> For network sessions, add PEs if you have fewer than 10 PEs per node. If you already have 10 PEs per node, add another node. <p>Note:</p> <p>The session limit is 120 for each PE and 1200 for each gateway. Because each node runs just one instance of the gateway, more than 10 PEs per node does not provide increased network sessions.</p> <ul style="list-style-type: none"> The channel driver does not have a session limit, but normally one PE is configured for each channel connection. If your configuration has less than 4 channel connections, you can add another channel connection and another PE. Verify that there is enough CPU capacity to handle more PEs. If not, add nodes.

Adding Memory

When you add memory, you increase cache to maximize the capability of the CPUs. This is helpful when CPUs are processing faster than the disk contents can be read into memory (that is, when the system is I/O-bound).

The following table lists reasons to add more memory.

Condition	Description
Add vprocs to existing nodes	<p>Each vproc consumes 32 MB of memory. When you add vprocs to existing nodes, you probably should add memory.</p> <p>Additional vprocs can substantially reduce free memory, which can cause more I/Os because the system can cache fewer datablocks.</p>
Excessive paging/ swapping (thrashing)	<p>More memory means that more code and data can be cached, achieving less I/O for paging and swapping.</p>
Tables in memory	<p>Increased memory may reduce I/O by accommodating:</p> <ul style="list-style-type: none"> Tables that are currently too large to remain in memory during processing More small tables concurrently residing in memory during processing <p>I/O can be affected by the size of data blocks. See the DATABLOCKSIZE option for the CREATE TABLE statement in <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144.</p>

Scaling

Your user applications should be scalable.

For example, assume that table BigT is a very large table but its rows are hash distributed based on only 16 unique values. Applications using BigT perform well and with high parallelism on a system with 1 or 2 nodes and 8 to 16 AMPs.

If you then expand the system to 128 AMPs, the rows of BigT still hash to only 16 unique values, and so are still distributed among only 16 AMPs. Thus, applications do not perform any better, and perhaps not as well.

To ensure scalability of your applications, try to make your PI a unique or nearly unique index. If a single column does not provide uniqueness, combine two or more columns to make up the index. You can define up to 64 columns per PI.

The result is many more unique hash combinations, and your applications should continue to perform well as your system expands.

Checking the Performance of an Upgrade

Monitor system behavior to make sure a Vantage software upgrade is producing the expected performance increases, as follows:

- Recollect statistics for the existing configuration if they are not current.
- Create a baseline test suite.
- Run the baseline on the system before the upgrade.
- Save the EXPLAINS for the base queries.
- Run the test bed again after the upgrade, and get new EXPLAINS.
- Compare the two EXPLAINS.
- Check for faster/slower response on any of the test queries. If slower, look at the before and after EXPLAIN. Check the statistics.
 - If stale, recollect.
 - If not, report this as a performance regression problem to Teradata Support immediately.

Performing these simple activities, even with a small group of queries (5-10 minimum are recommended), will help validate your performance results and expectations following any sort of important system change.

Teradata System Limits

This section provides the following limits.

- System limits
- Database limits
- Session limits
- System-derived and system-generated column data types

The reported limits apply only to the platform software. Platform-dependent client limits are not documented.

System Limits

The system software specifications in the following tables apply to an entire Teradata Vantage configuration.

Miscellaneous System Limits

Parameter	Value
Maximum number of combined databases, users, and zones.	4.2 x 10 ⁹
Maximum number of database objects per system lifetime. A database object is any object whose definition is recorded in DBC.TVM. Because the system does not reuse DBC.TVM IDs, this means that a maximum of 1,073,741,824 such objects can be created over the lifetime of any given system. At the rate of creating one new database object per minute, it would take 2,042 years to use 1,073,741,824 unique IDs.	1,073,741,824
Maximum size for a table header.	1 MB
Maximum size of table header cache.	8 x 10 ⁶ bytes
Maximum size of a response spool row.	<ul style="list-style-type: none"> • 1 MB on large-cylinder systems • 64 KB on small-cylinder systems
Maximum number of change logs that can be specified for a system at any point in time.	1,000,000
Maximum number of locks that can be placed on aggregate online archive logging tables, databases, or both per request. The maximum number of locks that can be placed per LOGGING ONLINE ARCHIVE ON request is fixed at 25,000 and cannot be altered.	25,000
Maximum number of 64KB parse tree segments allocated for parsing requests.	12,000
Maximum number of nodes per system configuration.	1,024

Parameter	Value
Limits on vprocs of each type restrict systems with a large number of nodes to fewer vprocs per node. Systems with the maximum vprocs per node cannot approach the maximum number of nodes.	
Maximum size of a query band.	4,096 UNICODE characters

Message Limits

Parameter	Value
Maximum number of CLlv2 parcels per message	256
Maximum request message size (client-to-database)	7 MB This limit applies to messages to and from client systems and to some internal Vantage messages.
Maximum response message size (database-to-client)	16 MB
Maximum error message text size in a failure parcel	255 bytes

Storage Limits

Parameter	Value
Total data capacity	~ 12 TB/AMP
Minimum data block size with small cylinders	~9KB 18 512-byte sectors
Default data block size with small cylinders	~127KB 254 512-byte sectors
Maximum data block size with small cylinders	~256KB 512 512-byte sectors
Minimum data block size with large cylinders	~21KB 42 512-byte sectors
Default data block size with large cylinders	~127KB 254 512-byte sectors
Maximum data block size for a large cylinder system that does not use 4KB alignment	1,023.5KB 2,047 sectors
Maximum data block size for a large cylinder system that uses 4KB alignment	1,024KB 2,048 sectors

Parameter	Value
Maximum number of data blocks that can be merged per data block merge	8
Maximum merge block ratio block size for a table.	100% of the maximum multirow block size for a table.

Gateway and Vproc Limits

Parameter	Value
Maximum number of sessions per PE.	Multiple. This is true because the gateway runs in its own vproc on each node. See <i>Teradata Vantage™ - Database Utilities</i> , B035-1102 for details. The exact number depends on whether the gateways belong to different host groups and listen on different IP addresses.
Maximum number of sessions per gateway.	Tunable: 1 - 2,147,483,647. 1,200 maximum certified. The default is 600. See <i>Teradata Vantage™ - Database Utilities</i> , B035-1102 for details.
	30,720 This includes the sum of all of the following types of vproc for a configuration: <ul style="list-style-type: none"> • AMP Access Module Processor vprocs • GTW Gateway Control vprocs • PE Parsing Engine vprocs • RSG Relay Services Gateway vprocs • TVS Teradata Virtual
Maximum number of AMP vprocs per system.	16,200
Maximum number of GTW vprocs per system. This is a soft limit that Teradata Support personnel can reconfigure for you. Each GTW vproc on a node must be in a different host group. If two GTW vprocs are in the same host group, they must be on different nodes.	The default is one GTW vproc per node with all of them assigned to the same host group. The maximum depends on: <ul style="list-style-type: none"> • Number of IP addresses assigned to each node.

Parameter	Value
	<ul style="list-style-type: none"> Number of host groups configured in the database. <p>Each gateway on a node must be assigned to a different host group from any other gateway on the same node and each gateway needs to be assigned a disjoint set of IP addresses to service.</p> <p>This does not mean that all gateways in a system must be assigned to a different host group.</p> <p>It means that each gateway on the same node must be assigned to a different host group.</p> <p>Gateways on different nodes can be assigned to the same host group.</p>
Maximum number of PE vprocs per system. This is a soft limit that Teradata Support personnel can reconfigure for you.	2,048
Maximum number of TVS allocator vprocs per system. This is a soft limit that Teradata Support personnel can reconfigure for you.	2,048
Maximum number of vprocs, in any combination, per node.	127
Maximum number of AMP vprocs per cluster.	8
Maximum number of external routine protected mode platform tasks per PE or AMP. This value is derived by subtracting 1 from the maximum total of PE and AMP vprocs per system (because each system must have at least one PE), which is 16,384. This is obviously not a practical configuration. The valid range is 0 to 20, inclusive. The limit is 20 platform tasks for each platform type, not 20 combined for both. See <i>Teradata Vantage™ - Database Utilities</i> , B035-1102 for details.	20
Maximum number of external routine secure mode platform tasks per PE or AMP. This value is derived by subtracting 1 from the maximum total of PE and AMP vprocs per system (because each system must have at least one PE), which is 16,384. This is obviously not a practical configuration.	20
Size of a request control block	~ 40 bytes
Default number of lock segments per AMP vproc. This is controlled by the NumLokSegs parameter in DBS Control.	2
Maximum number of lock segments per AMP vproc.	8

Parameter	Value
This is controlled by the NumLokSegs parameter in DBS Control.	
Default size of a lock segment. This is controlled by the LockLogSegmentSize parameter in DBS Control.	64 KB
Maximum size of a lock segment. This is controlled by the LockLogSegmentSize parameter in DBS Control.	1 MB
Default number of locks per AMP	3,200
Maximum number of locks per AMP.	209,000
Maximum size of the lock table per AMP. The AMP lock table size is fixed at 2 MB and cannot be altered.	2 MB
Maximum size of the queue table FIFO runtime cache per PE.	<ul style="list-style-type: none"> • 100 queue table entries • 1 MB
Maximum number of SELECT AND CONSUME requests that can be in a delayed state per PE.	24
Amount of private disk swap space required per protected or secure mode server for C/C++ external routines per PE or AMP vproc.	256 KB
Amount of private disk swap space required per protected or secure mode server for Java external routines per node.	30 MB

Hash Bucket Limits

Parameter	Value
Number of hash buckets per system. This value is user-selectable.	<p>The number is user-selectable per system. The choices are:</p> <ul style="list-style-type: none"> • 65,536 • 1,048,576 <p>Bucket numbers range from 0 to the system maximum.</p>
Size of a hash bucket	<p>The size depends on the number of hash buckets on the system if the system has:</p> <ul style="list-style-type: none"> • 65,536 hash buckets, the size of a hash bucket is 16 bits. • 1,048,576 hash buckets, the size of a hash bucket is 20 bits. <p>Set the default hash bucket size for your system using the DBS Control utility (see <i>Teradata Vantage™ - Database Utilities</i>, B035-1102 for details).</p>

Interval Histogram Limits

Parameter	Value
Number of hash values.	4.2 x 10 ⁹
Maximum number of intervals per index or column set histogram. The system-wide maximum number of interval histograms is set using the MaxStatsInterval parameter of the DBS Control record.	500
Default number of intervals per index or column set histogram.	250
Maximum number of equal-height intervals per interval histogram.	500

Database Limits

The database specifications in the following tables apply to a single database. The values presented are for their respective parameters individually and not in combination.

Name and Title Size Limits

Parameter	Value
Maximum name size for database objects, for example, account, attribute, authorization, column, constraint, database, function, GLOP, index, macro, method, parameter, password, plan directive, procedure, profile, proxy user, query, role, procedure, table, transform group, trigger, UDF, UDM, UDT, using variable name, view. See <i>Teradata Vantage™ - SQL Fundamentals</i> , B035-1141 for other applicable naming rules.	128 UNICODE characters
Maximum system name size. Used in SQL statements for target level emulation. See <i>Teradata Vantage™ - SQL Data Manipulation Language</i> , B035-1146.	63 characters
Maximum SQL text title size.	256 characters

User Limits

Parameter	Value
Maximum number of external roles.	50

Table and View Limits

Parameter	Value
Maximum number of journal tables per database.	1
Maximum number of error tables per base data table.	1

Parameter	Value
Maximum number of columns per base data table or view.	2,048
Maximum number of columns per error table. This limit includes 2,048 data table columns plus 13 error table columns.	2,061
Maximum number of UDT columns per base data table. The same limit is true for both distinct and structured UDTs. The absolute limit is 2,048, and the realizable number varies as a function of the number of other features declared for a table that occupy table header space.	~1,600
Maximum number of LOB and XML columns per base data table. This limit includes predefined type LOB columns and UDT LOB columns. A LOB UDT or XML UDT column counts as one column even if the UDT is a structured type that has multiple LOB attributes.	32
Maximum number of columns created over the life of a base data table.	2,560
Maximum number of rows per base data table.	Limited only by disk capacity.
Maximum number of bytes per table header per AMP. A table header that is large enough to require more than ~64,000 bytes uses multiple 64KB rows per AMP up to 1 MB. A table header that requires 64,000 or fewer bytes uses only a single row and does not use the additional rows that are required to contain a larger table header. The maximum size for a table header is 1 MB.	1 MB
Maximum number of characters per SQL index constraint.	16,000
Maximum non-spool row size.	1 MB
Maximum internal spool row size.	~ 1MB
Maximum size of the queue table FIFO runtime cache per table.	2,211 row entries
Maximum logical row size. A logical row is defined as a base table row plus the sum of the bytes stored in a LOB or XML subtable for that row.	67,106,816,000 bytes
Maximum non-LOB column size for a nonpartitioned table. This limit is based on subtracting the minimum row overhead value for a nonpartitioned table row (12 bytes) from the system-defined maximum row length (64,256 bytes).	64,244 bytes
Maximum non-LOB column size for a partitioned table. This limit is based on subtracting the minimum row overhead value for a partitioned table row (16 bytes) from the system-defined maximum row length (64,256 bytes).	64,240 bytes
Maximum number of values (excluding NULLs) that can be multivalued compressed per base table column.	255
Maximum amount of BYTE data per column that can be multivalued compressed.	4,093 bytes

Parameter	Value
Maximum amount of data per column that can be multivalue compressed for GRAPHIC, LATIN, KanjiSJIS, and UNICODE server character sets.	8,188 characters
<p>Maximum number of columns that can be compressed per primary-indexed or primary-AMP-indexed table or join index using multivalue compression or algorithmic compression.</p> <p>This assumes that the object is not a NoPI table or join index or a global temporary trace table. All other tables, hash indexes, and join indexes must have a primary index or a primary AMP index. Primary indexes and primary AMP indexes cannot be either multivalue compressed or algorithmically compressed. Because of this, the limit is the maximum number of columns that can be defined for a table, which is 2,048, minus 1.</p> <p>The limit for multivalue compression is far more likely to be reached because of table header overflow, but the amount of table header space that is available for multivalue compressed values is limited by a number of different factors.</p> <p>Join index columns inherit their compression characteristics from their parent tables.</p>	2,047
Maximum number of columns that can be compressed per NoPI table using multivalue compression or algorithmic compression. Column-partitioned NoPI join index columns inherit their compression characteristics from their parent tables.	2,048
Maximum number of algorithmically compressed values per base table column.	Unlimited
Maximum width of data that can be multivalue compressed for BYTE, BYTEINT, CHARACTER, GRAPHIC, VARCHAR, and VARGRAPHIC data types.	Unlimited
Maximum width of data that can be multivalue compressed for data types other than BYTE, BYTEINT, CHARACTER, DATE, GRAPHIC, VARCHAR, and VARGRAPHIC.	Unlimited
<p>Maximum width of data that can be algorithmically compressed.</p> <p>The maximum data width is unlimited if you specify only algorithmic compression for a column.</p> <p>If you specify a mix of multivalue and algorithmic compression for a column, then the limits for multivalue compression also apply for algorithmic compression.</p>	Unlimited
Maximum number of table-level CHECK constraints that can be defined per table.	100
Maximum number of primary indexes or primary AMP indexes per table, hash index, or join index that is not a NoPI database object.	1
Minimum number of primary indexes per primary-indexed table, hash index, or join index.	1
Minimum number of primary AMP indexes per primary-AMP-indexed table or join index.	1
Maximum number of primary indexes per primary-AMP-indexed or NoPI or join index or global temporary trace table.	0
Maximum number of columns per primary index or primary AMP index.	64
Maximum number of column partitions per table, including two columns partitions reserved for internal use).	2,050

Parameter	Value
Minimum number of column partition numbers that must be available for use by an ALTER TABLE request to alter a column partition.	1
Maximum partition number for a column partitioning level. Maximum number of column partitions for that level + 1	Maximum number of column partitions for that level + 1
Maximum combined partition number for a single-level column-partitioned table or column-partitioned join index.	The same as the maximum partition number for the single partitioning level.
Maximum number of rows per hash bucket for a 44-bit uniqueness value.	17,592,186,044,415
Maximum combined partition number for a multilevel partitioning for 2-byte partitioning.	65,535
Maximum combined partition number for a multilevel partitioning join index for 8-byte partitioning.	9,223,372,036,854,775,807
Maximum number of ranges, excluding the NO RANGE, UNKNOWN, and NO RANGE OR UNKNOWN and UNKNOWN partitions, for a RANGE_N partitioning expression for 2-byte partitioning. This value is limited by the largest possible INTEGER value.	65,533
Maximum number of ranges, excluding the NO RANGE, UNKNOWN, and NO RANGE OR UNKNOWN and UNKNOWN partitions, for a RANGE_N partitioning expression for 8-byte partitioning. This value is limited by the largest possible BIGINT value.	9,223,372,036,854,775,805
Minimum value for <i>n</i> in a RANGE#L <i>n</i> expression.	1
Minimum value for <i>n</i> in a RANGE#L <i>n</i> expression for 2-byte partitioning.	15
Minimum value for <i>n</i> in a RANGE#L <i>n</i> expression for 8-byte partitioning.	62
Maximum number of partitions, including the NO RANGE, UNKNOWN, and NO RANGE OR UNKNOWN partitions, for a single-level partitioning expression composed of a single RANGE_N function with INTEGER data type.	2.147.483.647
Maximum number of ranges for a single-level partitioning expression composed of a single RANGE_N function with INTEGER data type that is used as a partitioning expression if the NO RANGE and UNKNOWN partitions are not specified.	65,533
Maximum number of ranges for a single-level partitioning expression composed of a single RANGE_N function with BIGINT data type that is used as a partitioning expression if the NO RANGE and UNKNOWN partitions are not specified.	9,223,372,036,854,775,805
Maximum number of ranges for a single-level partitioning expression composed of a single RANGE_N function with BIGINT data type that is used as a partitioning expression if both the NO RANGE and UNKNOWN partitions are specified.	9,223,372,036,854,775,807
Maximum value for a partitioning expression that is not based on a RANGE_N or CASE_N function. This is allowed only for single-level partitioning.	65,535

Parameter	Value
Maximum number of defined partitions for a column partitioning level.	The number of column partitions specified + 2. The 2 additional partitions are reserved for internal use.
Maximum number of defined partitions for a row partitioning level if the row partitions specify the RANGE_N or CASE_N function.	The number of row partitions specified.
Maximum number of defined partitions for a row partitioning level if the row partitions do not specify the RANGE_N or CASE_N function.	65,535
Maximum number of partitions for a partitioning level when you specify an ADD clause. This value is computed by adding the number of defined partitions for the level plus the value of the integer constant specified in the ADD clause.	9,223,372,036,854,775,807
Maximum number of partitions for a column partitioning level when you do not specify an ADD clause and at least one row partitioning level does not specify an ADD clause.	The number of column partitions defined + 10.
Maximum number of column partitions for a column partitioning level when you do not specify an ADD clause, you also specify row partitioning, and each of the row partitions specifies an ADD clause.	The largest number for the column partitioning level that does not cause the partitioning to be 8-byte partitioning.
Maximum number of partitions for each row partitioning level without an ADD clause in level order, if using the number of row partitions defined as the maximum for this and any lower row partitioning level without an ADD clause.	The largest number for the column partitioning level that does not cause the partitioning to be 8-byte partitioning.
Maximum partition number for a row-partitioning level.	The same as the maximum number of partitions for the level.
Minimum number of partitions for a row-partitioning level.	2
Maximum number of partitions for a CASE_N partitioning expression. This value is limited by the largest possible INTEGER value.	2,147,483,647
Maximum value for a RANGE_N function with an INTEGER data type.	2,147,483,647
Maximum value for a RANGE_N function with a BIGINT data type that is part of a partitioning expression.	9,223,372,036,854,775,805
Maximum value for a CASE_N function for both 2-byte and 8-byte partitioning.	2,147,483,647
Maximum number of partitioning levels for 2-byte partitioning.	15

Parameter	Value
Other limits can further restrict the number of levels for a specific partitioning.	
Maximum number of partitioning levels for 8-byte partitioning. Other limits can further restrict the number of levels for a specific partitioning.	62
Maximum value for n for the system-derived column PARTITION#L n .	62
Minimum number of partitions per row-partitioning level for a multilevel partitioning primary index.	1
Minimum number of partitions defined for a row-partitioning level.	2 or greater
Maximum number of partition number ranges from each level that are not eliminated for static row partition elimination for an 8-byte row-partitioned table or join index.	8,000
Maximum number of table-level constraints per base data table.	100
Maximum size of the SQL text for a table-level index CHECK constraint definition.	16,000 characters
Maximum number of referential integrity constraints per base data table.	64
Maximum number of columns per foreign and parent keys in a referential integrity relationship.	64
Maximum number of characters per string constant.	31,000
Maximum number of row-level security constraints per table, user, or profile.	5
Maximum number of row-level security statement-action UDFs that can be defined per table.	4
Maximum number of non-set row-level security constraint encodings that can be defined per constraint. The valid range is from 1 to 10,000. 0 is not a valid non-set constraint encoding.	10,000
Maximum number of set row-level security constraint encodings that can be defined per constraint.	256

Spool Space Limits

Parameter	Value
Maximum internal spool row size.	~ 1MB

BLOB, CLOB, XML, and Related Limits

Parameter	Value
Maximum BLOB object size	2,097,088,000 8-bit bytes
Maximum CLOB object size	<ul style="list-style-type: none"> 2,097,088,000 single-byte characters

Parameter	Value
	<ul style="list-style-type: none"> 1,048,544,000 double-byte characters
Maximum XML object size	2,097,088,000 8-bit bytes
Maximum number of LOB rows per rowkey per AMP for NoPI LOB or XML tables	~ 256M The exact number is 268,435,455 LOB or XML rows per rowkey per AMP.
Maximum size of the file name passed to the AS DEFERRED BY NAME option in a USING request modifier	VARCHAR(1024)

User-Defined Data Type, ARRAY Data Type, and VARRAY Data Type Limits

Parameter	Value
Maximum structured UDT size. This value is based on a table having a 1 byte (BYTEINT) primary index. Because a UDT column cannot be part of any index definition, there must be at least one non-UDT column in the table for its primary index. Row header overhead consumes 14 bytes in an NPPI table and 16 bytes in a PPI table, so the maximum structured UDT size is derived by subtracting 15 bytes (for an NPPI table) or 17 bytes (for a PPI table) from the row maximum of 64,256 bytes.	<ul style="list-style-type: none"> 64,242 bytes (NoPI table) 64,241 bytes (NPPI table) 64,239 bytes (PPI table)
Maximum number of UDT columns per base data table. The absolute limit is 2,048, and the realizable number varies as a function of the number of other features declared for a table that occupy table header space. The figure of 1,600 UDT columns assumes a FAT table header. This limit is true whether the UDT is a distinct or a structured type.	~1,600
Maximum database, user, base table, view, macro, index, trigger, procedure, UDF, UDM, UDT, constraint, or column name size. Other rules apply for Japanese character sets, which might restrict names to fewer than 30 bytes. See <i>Teradata Vantage™ - SQL Fundamentals</i> , B035-1141 for the applicable rules.	30 bytes in Latin or Kanji1 internal representation
Maximum number of attributes that can be specified for a structured UDT per CREATE TYPE or ALTER TYPE request. The maximum is platform-dependent, not absolute.	300 - 512
Maximum number of attributes that can be defined for a structured UDT. While you can specify no more than 300 to 512 attributes for a structured UDT per CREATE TYPE or ALTER TYPE request, you can submit any number of ALTER TYPE requests with the ADD ATTRIBUTE option specified as necessary to add additional attributes to the type up to the upper limit of approximately 4,000.	~4,000
Maximum number of levels of nesting of attributes that can be specified for a structured UDT.	512
Maximum number of methods associated with a UDT.	~500

Parameter	Value
There is no absolute limit on the number of methods that can be associated with a given UDT. Methods can have a variable number of parameters, and the number of parameters directly affects the limit, which is due to Parser memory restrictions. There is a workaround for this issue. See ALTER TYPE information in <i>Teradata Vantage™ - SQL Data Definition Language Detailed Topics</i> , B035-1184 for details.	
Maximum number of input parameters with a UDT data type of VARIANT_TYPE that can be declared for a UDF definition.	8
Minimum number of dimensions that can be specified for a multidimensional ARRAY or VARRAY data type.	2
Maximum number of dimensions that can be specified for a multidimensional ARRAY or VARRAY data type.	5

Macro, UDF, SQL Procedure, and External Routine Limits

Parameter	Value
Maximum number of parameters specified in a macro.	2,048
Maximum expanded text size for macros and views.	2 MB
Maximum number of open cursors per procedure.	15
Maximum number of result sets a procedure can return.	15
Maximum number of columns returned by a dynamic result table function. The valid range is from 1 to 2,048. There is no default.	2,048
Maximum number of dynamic SQL requests per procedure.	15
Maximum length of a dynamic SQL request in a procedure. This includes its SQL text, the USING data (if any), and the CLIV2 parcel overhead.	Approximately 1 MB
Maximum combined size of the parameters for a procedure	1 MB for input parameters 1 MB for output (and input/output) parameters
Maximum size of condition names and UDF names specified in a procedure.	30 bytes
Maximum number of parameters specified in a UDF defined without dynamic UDT parameters.	128
Maximum number of parameters that can be defined for a constructor method for all types except ARRAY/VARRAY	128
Maximum number of parameters that can be defined for a constructor method of an ARRAY/VARRAY type	<i>n</i>

Parameter	Value
	where n is the number of elements defined for the type.
Maximum number of combined return values and local variables that can be declared in a single UDF.	Unlimited
Maximum number of combined external routine return values and local variables that can be instantiated at the same time per session.	1,000
Maximum combined size of the parameters defined for a UDF.	1 MB for input parameters 1 MB for output parameters
Maximum number of parameters specified in a UDF defined with dynamic UDT parameters. The valid range is from 0 to 15. The default is 0.	1,144
Maximum number of parameters specified in a method.	128
Maximum number of parameters specified in an SQL procedure.	256
Maximum number of parameters specified in an external procedure written in C or C++.	256
Maximum number of parameters specified in an external procedure written in Java.	255
Maximum size of an ARRAY or VARRAY UDT. This limit does not include the number of bytes used by the row header and the primary index or primary AMP index of a table.	64 KB
Maximum length of external name string for an external routine. An external routine is the portion of a UDF, external procedure, or method that is written in C, C++, or Java (only external procedures can be written in Java). This is the code that defines the semantics for the UDF, procedure, or method.	1,000 characters
Maximum package path length for an external routine.	256 characters
Maximum SQL text size in a procedure.	Approximately 1 MB
Maximum number of nested CALL statements in a procedure.	15
Maximum number of Statement Areas per SQL procedure diagnostics area. See <i>Teradata Vantage™ - SQL Stored Procedures and Embedded SQL</i> , B035-1148 and <i>Teradata Vantage™ - SQL External Routine Programming</i> , B035-1147.	1
Maximum number of Condition Areas per SQL procedure diagnostics area. See <i>Teradata Vantage™ - SQL Stored Procedures and Embedded SQL</i> , B035-1148 and <i>Teradata Vantage™ - SQL External Routine Programming</i> , B035-1147.	16

Query and Workload Analysis Limits

Parameter	Value
Maximum number of columns and indexes on which statistics can be collected or recollected at one time. 512 or limited by available parse tree memory and amount of spool.	512
Maximum number of pseudo indexes on which multicolumn statistics can be collected and maintained at one time. A pseudo index is a file structure that allows you to collect statistics on a composite, or multicolumn, column set in the same way you collect statistics on a composite index. This limit is independent of the number of indexes on which statistics can be collected and maintained.	32
Maximum number of sets of multicolumn statistics that can be collected on a table or join index if single-column PARTITION statistics are not collected on the table or index.	32
Maximum number of sets of multicolumn statistics that can be collected on a table or join index if single-column PARTITION statistics are collected on the table or index.	31

Secondary, Hash, and Join Index Limits

Parameter	Value
Number of tables that can be referenced in a join.	128
Minimum number of secondary, hash, and join indexes, in any combination, per base data table.	0
Maximum number of secondary, hash, and join indexes, in any combination, per base data table. Each composite NUSI defined with an ORDER BY clause counts as 2 consecutive indexes in this calculation. The number of system-defined secondary and single-table join indexes contributed by PRIMARY KEY and UNIQUE constraints counts against the combined limit of 32 secondary, hash, and join indexes per base data table.	32
Maximum number of columns referenced per secondary index.	64
Maximum number of columns referenced per single table in a hash or join index.	64
Maximum number of rows per secondary, hash, or join index.	Limited only by disk capacity.
Maximum number of columns referenced in the fixed part of a compressed join index. Vantage implements a variety of different types of user-visible compression in the system. When describing compression of hash and join indexes, compression refers to a logical row compression in which multiple sets of nonrepeating column values are appended to a single set of repeating column values. This allows the system to store the repeating value set only once, while any nonrepeating column values are stored as logical segmental extensions of the base repeating set.	64

Parameter	Value
Maximum number of columns referenced in the repeating part of a compressed join index.	64
Maximum number of columns in an uncompressed join index.	2,048
Maximum number of columns in a compressed join index.	<ul style="list-style-type: none"> • 64 for repeating • 64 for nonrepeating

Reference Index Limits

Parameter	Value
Maximum number of reference indexes per base data table. There is a maximum of 128 Reference Indexes in a table header, 64 from a parent table to child tables and 64 from child tables to a parent table.	64

SQL Request and Response Limits

Parameter	Value
Maximum SQL text size per request. This includes SQL request text, USING data, and parcel overhead.	1 MB
Maximum request message size. The message includes SQL request text, USING data, and parcel overhead.	7 MB
Maximum number of entries in an IN list. There is no fixed limit on the number of entries in an IN list; however, other limits such as the maximum SQL text size, place a request-specific upper bound on this number.	Unlimited
Maximum SQL activity count size.	8 bytes
Maximum number of tables and single-table views that can be joined per query block. This limit is controlled by the MaxJoinTables DBS Control field.	128
Maximum number of partitions for a hash join operation.	50
Maximum number of subquery nesting levels per query.	64
Maximum number of tables or single-table views that can be referenced per subquery. This limit is controlled by the MaxJoinTables DBS Control field..	128
Maximum number of fields in a USING row descriptor.	2,536
Maximum number of bytes in USING data. This value does not include Smart LOB (SLOB) data.	1,040,000
Maximum number of open cursors per embedded SQL program.	16
Maximum SQL text response size.	1 MB
Maximum number of columns per DML request ORDER BY clause.	64

Parameter	Value
Maximum number of columns per DML request GROUP BY clause.	64
Maximum number of fields in a CONSTANT row.	32,768

Row-Level Security Constraint Limits

Parameter	Value
Maximum number of row-level security constraints per table.	5
Maximum number of hierarchical row-level security constraints per user or profile.	6
Maximum number of values per hierarchical row-level security constraint.	10,000
Maximum number of non-hierarchical row-level security constraints per user or profile.	2
Maximum number of values per non-hierarchical row-level security constraint.	256

Session Limits

The session specifications in the following table apply to a single session:

Parameter	Value
Maximum number of sessions per PE.	120
Maximum number of sessions per gateway vproc. See <i>Teradata Vantage™ - Database Utilities</i> , B035-1102 for details.	Tunable: 1 - 2,147,483,647. 1,200 maximum certified. The default is 600.
Maximum number of active request result spools per session.	16
Maximum number of parallel steps per request. Parallel steps can be used to process a request submitted within a transaction (which can be either explicit or implicit).	20
Maximum number of materialized global temporary tables that can be materialized simultaneously per session.	2,000
Maximum number of volatile tables that can be instantiated simultaneously per session.	1,000
Maximum number of SQL procedure diagnostic areas that can be active per session.	1

Handling Teradata Crashdumps: Operational DBAs

This section provides an overview of Teradata crashdumps. If you encounter any problems that result in a Teradata crashdump being generated, contact Teradata Support.

Teradata Vital Infrastructure and Crashdumps

Under normal circumstances, Teradata Support personnel handle crashdumps directly using the TVI.

Each system at your site *must* be registered with its own site ID so that Teradata will have all the proper information on the configuration of each specific system on your site.

If you have created and registered a site ID for each system at your site, you should have TVI software already installed so that it can begin monitoring your database. When a 3610 error or an 033-12123-00 error occurs, this triggers an incident to be created. If you need to open an incident:

1. Go to <https://support.teradata.com>.
2. Log in.
 - A 3610 error indicates an internal error produced by a request sent to Vantage which has caused a snapshot dump and aborts the given request. This is done instead of restarting the system.
 - A 033-12123-00 message represents a task snapshot dump and an event is logged when a program requests a snapshot dump on itself.

TVI, through the TVI Manager and Auto Incident Create (AIC) server, will automatically notify Teradata Support personnel who will try to troubleshoot the problem to determine the root cause or may work with field engineers if there is a hardware problem. They will engage your local site team and may also escalate the incident to Teradata developers, if necessary.

Every customer site team is different: some local site teams may start collecting crashdumps and have them already ready in the event that your Teradata Support personnel asks for the crashdump to be sent in to Teradata. Other site teams will wait until instructed by Teradata Support to send the crashdump information.

If there are multiple users on the system at once and you are not sure who caused it or where the bad request or error came from, Teradata Support personnel may need to extract the problem SQL from the crashdump and ask you to run TSET on that SQL and send in the TSET pack file to continue the investigation.

TSET File Pack and the DBC.TSETQueryText Table

The TSET pack file is a set of Teradata Support files that can help in troubleshooting a crash. These files are intended for Teradata Support personnel so you do not need to create or manage these files.

Part of the TSET file pack includes information that comes from the DBC.TSETQueryText table. This table is a globally available system table created during installation. If a 3610 error occurs in the Parser, the system will attempt to log a row into this table. However, depending on the severity of the 3610 error, the system may not be able to populate the TSETQueryText table.

While the information from DBC.TSETQueryText does not directly populate the crashdumps database, the contents of the table can be used to generate TSET information by Teradata Support personnel for crash resolution. The DBC.TSETQueryText table captures problematic queries on the system that result in a 3610 error.

After the problematic query is captured, TVI provides Teradata and the DBA with a command to generate and retrieve the necessary TSET information based on the contents of the DBC.TSETQueryText table. This allows Teradata support analysts to reproduce the query immediately on a test system and saves time of waiting for a crashdump to finish saving as well as the time it takes to send (FTP) the crashdump to Teradata Support.

Note:

As an alternative method to the TSET Perl scripts for extracting TSET information based on the DBC.TSETQueryText table, Teradata Support personnel can use the external stored procedure named ExportTSET. However, the TSET information produced by the external stored procedure is in the form of BTEQ scripts and is currently not compatible with the TSET GUI client.

Crashdumps Overview

This section is an overview of Teradata crashdumps concepts and terminology. Information on Operating System (OS) dumps is provided only for clarification and differentiation from Teradata crashdumps.

Dump Types

The following table describes the types of dumps on your system.

Crashdump Type	Description
Operating System (OS) dump	<p>An OS dump is the unprocessed contents of memory at the time of an operating system crash. The content of the dump depends on your platform OS.</p> <p>An OS dump is the result of a single node OS crash and contains the entire contents of the memory of a node. It may also be referred to as a system dump, kernel dump, or node dump. On Linux, an OS dump is called a kernel panic dump.</p> <p>When an OS crash occurs, the node writes the contents of memory to a system dump area and restarts. An OS dump is typically the size of memory. An OS dump is independent of Teradata or any particular application.</p>
Teradata Crashdump	<p>A Teradata crashdump, also known as a PDE dump or simply crashdump, is a result of an unrecoverable Teradata error. When Teradata crashes, PDE on each node writes selective data to its own PDE dump directory, called a raw PDE dump. How much and which of this data is actually written also depends on the type of error. After the raw PDE dump is captured, Teradata restarts, but the system does not restart.</p>
Snapshot Dump	<p>A snapshot dump occurs when a process fails on a node and the failure does not cause a restart. PDE writes selective data to the dump directory, and then the database may abort or retry the request.</p>

Teradata Crashdumps Processes

After Vantage restarts, the CSP utility saves the raw PDE dump of each node as either rows in a crashdump table in the DBC.Crashdumps database or flat files (streams) in compressed binary format. Depending on dump control parameters, CSP may start automatically or you can start it manually. Since Teradata is a parallel system, a full Vantage crashdump is a system-wide entity, consisting of data from each node. Specialized tools exist to examine the crashdump.

The procedure for crashdumps is as follows:

1. An event occurs that triggers a crashdump.
2. DMP runs at the time the fault or reset is detected and saves a crashdump to the PDE raw dump directory.
3. After a restart, what happens next depends on the setting of the Save Dumps field of Screen Debug in the Ctl utility. If the field is set to:
 - On, CSP starts automatically and saves the crashdump from the raw dump directory to a table in the DBC.Crashdumps database. Although Save Dumps is On by default, Teradata recommends that you set this field to Off to conserve system resources and save crashdumps only when you need to.
 - Off, you start CSP manually and save the crashdump to the DBC.Crashdumps database. Manually saving crashdumps rather than having the system save them automatically is a best practice (see [Manually Saving Teradata Crashdumps to the Crashdumps Database](#)). Another choice is to copy the raw crashdump to a second Teradata system running the same version of the operating system and Vantage. See [Manually Saving Crashdumps to Another Teradata System](#).

You can also save crashdumps to flat files (stream files) on each node. The stream files are in compressed binary FastLoad format and can be either sent directly to the Teradata Support Center or moved to another system and then loaded into the Crashdumps database. For more information about saving to stream files, see [Manually Saving Teradata Crashdumps to Stream Files](#).

Because Vantage is a parallel system, a full crashdump is a system-wide entity in a table format consisting of data from each node. All nodes work in parallel to write their piece of the crashdump into a common table in DBC.Crashdumps: they write the contents of memory to internal disk and save the raw PDE dump into the DBC.Crashdumps database on all nodes of the system at approximately the same time.

Note:

In some cases, Teradata Support may choose to debug crashdumps directly from the dump area. For more information, see [Debugging Crashdumps](#).

4. Copy the crashdump from the table in DBC.Crashdumps and send the crashdump data to Teradata Support either:
 - Through FTP services.
 - Or, if you cannot upload your crashdumps using standard FTP services, copy the crashdumps onto external media using the Dump Unload/Load (DUL) utility. (See [Saving Crashdumps to Disk](#).)

For more detailed information on how to handle crashdumps for your specific OS, see [Handling Teradata Crashdumps](#).

Teradata Crashdump Formats

PDE first writes the data in a raw PDE dump and this information is saved (or can be saved) into either of the following formats:

- A table in the DBC.Crashdumps database
- Flat files (called stream files), which you can either move to another system or upload directly to the Teradata Support Center.

The CSP SAVE dump command requires two options: “source” and “target.” The default format for -source is “dump” (which is the PDE dump directory) and the default -target is “table” (the DBC.Crashdumps database).

For more information, see the CSP pdehelp. See [Manually Saving Teradata Crashdumps to the Crashdumps Database](#) for examples of the SAVE command. Also see [Manually Saving Teradata Crashdumps to Stream Files](#).

Teradata Crashdump Sizes

An OS dump is different from a Teradata crashdump in not only content but also size. Typically, a Teradata crashdump is smaller than the sum of the memory for all the nodes in the database configuration.

The size of raw PDE dumps still in “dump” format (*before* the crashdumps are saved in table format) depends on various factors such as the type of error, number of vprocs, amount of memory, number of processes running when Teradata crashed and so on. So the size of raw PDE dumps can vary greatly from 10 MB to 4 GB or even larger. For example, on a Linux node that is 4 GB with 10 vprocs, the size of the Teradata crashdump could range from 1.5 to 2 GB.

Because a full crashdump saved to the DBC.Crashdumps database is the collection of the raw PDE dumps from all the nodes, a full crashdump is approximately twice the size of the raw dump multiplied by the number of nodes for your system. The DBC.Crashdumps database, which is allocated from the permanent space of DBC, should be large enough to hold 4 to 5 crashdumps.

Note:

Dump creation fails if the number of dumps saved on a node exceeds the limit in the control GDO. After each dump, Teradata checks the number of dumps saved on the node and writes a warning to the system messages log if there is space for only one more dump or if the saved dump limit has been reached.

To prevent space issues, actively manage dumps by saving raw PDE dumps from the dump directory to DBC.Crashdumps, unloading crashdumps from the DBC.Crashdumps database to tape, and deleting unwanted crashdumps from both the dump directory or DBC.Crashdumps database.

Viewing Teradata Crashdump Messages

Messages are written to the log file located at `/var/log/messages`.

Locating Crashdumps

For the most part, if you encounter a problem on your system that results in a crashdump, Teradata Vital Infrastructure will notify Teradata Support if you have TVI installed on your system. Otherwise, you should contact the Teradata Support Center.

- Raw PDE crashdumps are saved to `/var/opt/teradata/tddump`.
Teradata recommends that the TDTEMP and TDDUMP partitions be identified as `/var/opt/teradata/tdtemp` and `/var/opt/teradata/tddump`, respectively. You can use the `dmpconfig` utility to view or change the default locations. For more information on this utility, type `dmpconfig -h`.
- Crashdumps in table format are located at `DBC.Crashdumps.Crash_YYMMDD_HHMMSS_NN`.
- Crashdumps in stream format are located at `/var/opt/teradata/tddump/pdedumps/stream`.

Note:

When you first install Vantage on your SUSE Linux system, you must partition the disk and save a slice for dumps. For more information, see *Teradata® Tools and Utilities for Linux Installation Guide (Amazon Linux 2, CentOS, OEL, RedHat, SLES, Ubuntu)*, B035-3160.

Configuring the Crashdumps Database

The `DBC.Crashdumps` database is an internal database initially installed by the `DIPCRASH` scripts when the system is installed. Because it is a database, it requires permanent space. Space is allocated and managed similar to other databases and can be modified using Teradata DML.

The size of the `DBC.Crashdumps` database is configurable and depends on available space and your administrative preferences. Just as the PDE dump directory should be large enough to hold approximately 4 to 5 raw dumps, the `DBC.Crashdumps` database should be large enough to hold 4-5 full crashdumps (a full crashdump is the total of all raw PDE dumps from each node).

By default the database is `NO FALLBACK`, and no dumps are saved if an AMP goes down. If you specify `FALLBACK`, you gain fault tolerance, but the crashdump will take significantly more time and resources.

Note:

You cannot use the `NO FALLBACK` option and the `NO FALLBACK` default on platforms optimized for fallback.

Crashdumps space is allocated from current PERM space for DBC. Reducing crashdumps space makes more free space available in DBC. Increasing Crashdumps space reduces DBC space.

Ad-Hoc Creation of DBC.Crashdumps with DIPCRASH

Sometimes you may want to run the DIPCRASH script from the DIP utility because the script may not have been properly run and set up during system installation.

To verify that Crashdumps exists, submit the following statement:

```
HELP DATABASE CRASHDUMPS;
```

If the database table exists but contains no data, the following message appears:

```
Empty HELP information returned.
```

If the database table does not exist, an error message appears warning you that the table does not exist. If you want to create the database table, run the DIP utility and execute the DIPCRASH script.

You can start the DIP utility from the Supervisor window of DBW.

Note:

Make sure DBC has enough space to create Crashdumps and still retain enough for a maximum-sized transient journal plus overhead.

The amount of PERM space allocated to Crashdumps is deducted from the currently available space of user DBC.

For instructions on running DIP, see “Database Initialization Program (DIP)” in *Teradata Vantage™ - Database Utilities*, B035-1102.

Checking DBC.Crashdumps Space Allocation and Usage

When your system was first installed with Vantage, there was a very small default amount of PERM space allocated to DBC.Crashdumps. You should adjust the space based on what you find is required for your workload and your site. For example, you may have to also increase or decrease the amount of space based on how many crashdumps you want to actually retain at one time. If the default amount is not large enough, the best thing to do is run some tests and increase or decrease PERM as required.

As described in [Teradata Crashdump Sizes](#), a full crashdump is the size of the raw dump multiplied by the number of nodes for your system. The DBC.Crashdumps database, which is allocated from the PERM space of DBC, should be large enough to hold 4 to 5 crashdumps.

To first check the amount of space used by DBC.Crashdumps, use the following query:

```
sel databasename,
sum(maxperm),
sum(currentperm)
from dbc.diskpaceV
```



```
where databasename='crashdumps'
with sum(maxperm) (title 'Total'),sum(currentperm)
group by 1;
```

Result:

```
*** Query completed. 2 rows found. 3 columns returned.
*** Total elapsed time was 1 second.
```

DatabaseName	Sum(MaxPerm)	Sum(CurrentPerm)
-----	-----	-----
Crashdumps	21,474,836,424	991,296,000
	-----	-----

In this example 21,474,836,424 (21 GB) are allocated to the crashdumps database and 991,296,000 (1 MB) is used.

Allocating More Space to DBC.Crashdumps Database

To increase the permanent space allocated to DBC.Crashdumps, submit a MODIFY USER statement similar to the following where the amount of PERM space equals the amount you would like to increase DBC.Crashdumps:

```
modify user crashdumps as PERM = 245000000;
```

Result:

```
*** Database/User has been modified.
*** Total elapsed time was 1 second.
```

Then submit the following statement to verify the new size and confirm that space was properly allocated:

```
sel databasename, sum(maxperm) from dbc.allspaceV where databasename =
'crashdumps' group by 1;
```

Result:

```
*** Query completed. One row found. 2 columns returned.
*** Total elapsed time was 1 second.
```

DatabaseName	Sum(MaxPerm)
-----	-----
Crashdumps	245,000,000

Note:

If DBC does not have enough space to increase Crashdumps, you need to free up space or add more disks. Crashdumps are saved in compressed format by default.

You can also use the `-compress` and `-nocompress` options on the `csp` command line to override the default setting. To save space, use the `-compress` option.

Automatic Versus Forced Crashdump

The following section discusses when and why automatic crashes occur and why you might want to force a crash to occur. For example, you can force a crash or restart to recover from a system hang. The forced crashdump may provide useful information about why the system hung.

Automatic Teradata Crashdumps

The following table describes the events that cause a crashdump.

This event ...	Can occur ...
error restart	for many reasons. Most commonly, a restart is due to a fatal error condition in Vantage. The Teradata crashdump capture occurs before the restart on each node to preserve the error information in node memory.
Forced restart (with Teradata crashdump)	at any time, using restart commands. Only use restart commands for maintenance, so you do not interrupt jobs that are running. See Commands for Forcing a Teradata Crashdump .

Forced Teradata Crashdumps

You can control the crashdump type (OS dump or Teradata crashdump) by the way you force the crashdump. Select the type of crashdump you need from the following table.

IF one or more of these conditions exist ...	THEN use ...
<ul style="list-style-type: none"> The node is not responding The system is not making a Teradata crashdump. (The system does not respond to input while it is making a Teradata crashdump). 	OS dump.
<ul style="list-style-type: none"> Database sessions are not progressing New sessions cannot start The AWS is responding 	Teradata crashdump.

Commands for Forcing a Teradata Crashdump

You can restart without causing a reboot using one of the following methods (also see [Forced Teradata Crashdumps](#)):

- In DBW, open the Supervisor window icon and enter the following command:

```
restart tpa dump=yes comment
```

where *comment* is text explaining why there is going to be a restart.

The system performs a Teradata crashdump and a cold or coldwait restart. (For details, see “RESTART” under “Vproc Manager (vprocmanager)” in *Teradata Vantage™ - Database Utilities*, B035-1102)

- You can access the Teradata Command Prompt window and enter the tpareset command:

```
tpareset -d comment
```

The tpareset command has basically the same effect as the restart command in DBW.

The -d option forces a Teradata crashdump.

The system performs a cold restart and a Teradata crashdump.

Whichever command you choose to execute, the system prompts you for a confirmation.

Handling Teradata Crashdumps

Use the Debug screen of the ctl utility to verify Teradata crashdumps control settings. The Debug fields should read as the following table describes.

ctl Field Name	Value	Purpose
Save Dumps	Off	<p>Although the default setting for this field is On, which enables the automatic transfer of Teradata crashdumps from the dump directory to DBC.Crashdumps database, it is a best practice to set this field to Off.</p> <p>It is a best practice to set this field to Off because allowing the system to automatically save crashdumps can be resource intensive. You can manually save the crashdumps if you need to by using the -force option of CSP. For more information, see Manually Saving Teradata Crashdumps to the Crashdumps Database or Manually Saving Teradata Crashdumps to Stream Files.</p>
Maximum Dumps	5	<p>Set the maximum number of raw PDE dumps per node that the system will save on the dump directory. This includes snapshot dumps.</p> <ul style="list-style-type: none"> The value of -1 means that the number of dumps is limited only by the space available on the disk containing the dump directory. This is not recommended. A value of 0 means no dumps will be saved. The default is 5.

ctl Field Name	Value	Purpose
		If the maximum value has been reached and another dump occurs, the latest dump is not captured and an error message is printed to the messages log.
Snapshot Crash	Off	A snapshot dump logs information about a recoverable error without requiring the system to restart. This field should only be changed under the direction of Teradata Support personnel for diagnosing errors.

Save Considerations

A Teradata crashdump table cannot be automatically saved if:

- The ctl Save Dumps setting is off.
- An AMP is down and FALLBACK is not defined for Crashdumps database. This will not happen on platforms optimized for fallback because fallback is defined automatically.
- The system is not up.
- There is not enough space.

You can, however, manually save crashdumps using the CSP utility. For more information, see [Manually Saving Teradata Crashdumps to the Crashdumps Database](#).

Listing Teradata Crashdumps Using CSP

Using the CSP utility, you can list views in either the dump directory or the DBC.Crashdumps database.

- To list the available crashdumps saved in the DBC.Crashdumps database, use the following CSP command:

```
# csp -mode list -source table
```

- To list the available raw PDE dumps, use the following CSP command:

```
# csp -mode list -source dump
```

Note:

The -source parameter defaults to the dump directory. Therefore submitting “csp -mode list” is equivalent to “csp -mode list -source dump”.

The output will show the fields listed in the following table.

Note:

The first line shows if the output is from the dump directory or the database:


```
csp: Searching for dumps in database Crashdumps on the local system
```

or

```
csp: Searching for dumps in raw dump directory
```

CSP Output Column	Description
Sel	If an asterisk for the crashdump is listed for this row, the crashdump is eligible or selectable for the specific list, save, or clear operation. If there is no asterisk, it is not included as part of the specific list, save, or clear operation.
ID (date-time-token)	The identifier assigned by CSP to the crashdump. For example, the ID 2008/11/08-15:55:26-24 means the crashdump was taken: <ul style="list-style-type: none"> On the date of November 8th, 2008 At the time of 15:55:26 With the token 24
Nodes	Number of nodes from which crashdumps information has been collected. If one of four nodes is down, that node is not included.
Event	The TDBMS/PDE event code which caused the crash.
Instigator	The trigger for the crashdump. The format is vproc number/Partition ID/Task ID. In the examples that follow the table: <ul style="list-style-type: none"> The vproc number is 16384 The Partition IDs are 0 and 11 The Task IDs that triggered the crashdump are 11480, 15160, 28272 and 6418).
Status	The status of the crashdump. If a crashdump is corrupted or is from the wrong software version, this column will report: "Required files are missing or corrupted" or "Dump belongs to a different PDE version." If the field is blank, the crashdump is usable and valid.

You can constrain the CSP results with criteria such as listing all dumps from a specific date:

```
# csp -mode list 2008/09/09
csp: Searching for dumps in raw dump directory /var/opt/teradata/tddump
csp: 4 dumps found, 1 dump to process
csp: Sel   ID (date-time-token)   Nodes   Event   Instigator   Status
csp: ---   -
csp: *    2008/09/09-17:03:13-02   1       10196   16384/0/(11480)
csp:      2008/08/14-11:35:58-01   1       10196   16384/0/(15160)
csp:      2008/07/11-14:41:58-01   1       10416   1/11/(28272)
csp:      2008/07/01-11:58:42-05   1       10196   16384/0/(6418)
#
```


Note:

Despite the date restriction for searching only for dumps on 2008/09/09, the list command returns *all* dumps including ones from other days such as 2008/08/14, 2008/07/11, and 2008/07/01. However, the crashdump from the desired date criteria of 2008/09/09 has an asterisk in the Sel field, which shows that it matches the command criteria.

If a node is inaccessible (for example, it is down because the BYNET or the PDE is not running), CSP will not be able to access the dump directory on that node. It will report the raw PDE dumps only one nodes that are up.

```
# csp -mode list
csp: Searching for dumps in raw dump directory /var/opt/teradata/tddump
csp: 4 dumps found, 4 dumps to process
csp: Sel      ID (date-time-token)      Nodes      Event      Instigator      Status
csp: ---      -
csp: * 2008/09/09-17:03:13-02          1    10196    16384/0/(11480)
csp: * 2008/08/14-11:35:58-01          1    10196    16384/0/(15160)
csp: * 2008/07/11-14:41:58-01          1    10416    1/11/(28272)
csp: * 2008/07/01-11:58:42-05          1    10196    16384/0/(6418)
#
```

Manually Saving Teradata Crashdumps to the Crashdumps Database

CSP is controlled by the SCREEN DEBUG command of the ctl utility. Use CSP to save crashdumps manually:

1. Submit the following command to list the available raw PDE crashdumps:

```
#csp -mode list -source dump
```

2. Save the raw PDE dumps into table format in the DBC.Crashdumps database.

- The basic form of the CSP command for saving crashdumps is:

```
#csp -mode save -source dump -target stream
```

CSP reads any files created by the DMP program in the Dump area and writes them into the Crashdumps database.

- You can also use the following command to save all dumps found by CSP:

```
#csp -mode save -force
```

```
Dump 2007/11/26-09:54:25-01 is about to be saved
Are you sure? [y,n,q,?] >
```

A question will be displayed for each dump and the user must answer each one with

yes, no, or quit (individual letters such as y, n or q can be used).

```
csp: Searching for dumps in configured PDE dump directory on all nodes
csp: 3 dump found, 3 dumps to process
```

```
csp: Saving dump 1998/11/05-18:55:26-24 to table Crashdumps.Crash_200711
05_185526_24
```

```
csp: Writing data rows for dump 2007/11/05-18:55:26-24
```

```
csp: Dump 2007/11/05-18:55:26-24 saved
```

```
csp: Dump 2007/11/05-18:55:26-24 cleared
```

```
csp: Saving dump 2007/11/04-17:37:53-23 to table Crashdumps.Crash_200711
04_173753_23
```

```
csp: Writing data rows for dump 2007/11/04-17:37:53-23
```

```
csp: Dump 2007/11/05-18:55:26-24 saved
```

```
csp: Dump 2007/11/05-18:55:26-24 cleared
```

```
csp: Saving dump 2007/11/04-17:23:27-22 to table Crashdumps.Crash_200711
04_172327_22
```

```
csp: Writing data rows for dump 2007/11/04-17:23:27-22
```

```
csp: Dump 2007/11/05-18:55:26-24 saved
```

```
csp: Dump 2007/11/05-18:55:26-24 cleared
```

If the SCREEN DEBUG setting has the Save Dumps flag set to not save crashdumps, use the -force option to allow CSP to override the Save Dumps setting. For example, if you submit:

```
#csp -mode save -source dump -target stream -force
```

By default, the system saves the crashdumps in Last In, First Out (LIFO) order. The most recent crashdumps are saved first. After the crashdump is saved to DBC.Crashdumps, the crashdumps are automatically cleared from the PDE dump directory.

3. Verify that the crashdumps were properly saved to DBC.Crashdumps by using the following command:

```
# csp -mode list -source table
csp: Searching for dumps in database Crashdumps on the local system
csp: 1 dump found, 1 dump to process
csp: Sel  DumpName          Nodes  Event  Instigator          Status
csp: ---  -
csp: *    Crash_20080807_142628_01    1  10196  16384/2/(21600)
#
```

You can run CSP commands from any node in the system, but the utility will read the dump directory on *all* nodes.

Note:

For more information on CSP commands, type `csp -h` at a Teradata Command Prompt or see the resource file located at `/usr/pde/lib/defaults/Libcsp`.

Manually Saving Crashdumps to Another Teradata System

After Teradata restarts following a crashdump, you can save the dump to another Teradata system if it is running the same version of the operating system and Vantage. To do this:

1. Copy the contents of the raw dump directory to the other system.
2. Save all dumps CSP finds by using this command:

```
csp -mode save -force
```

3. Verify that the crashdumps were properly saved to DBC.Crashdumps by using the this command:

```
# csp -mode list -source table
csp: Searching for dumps in database Crashdumps on the local system
csp: 1 dump found, 1 dump to process
csp: Sel DumpName          Nodes Event Instigator      Status
csp: ---  -----
csp: *   Crash_20080807_142628_01      1 10196 16384/2/(21600)
#
```

Manually Saving Teradata Crashdumps to Stream Files

You can save crashdumps to flat (stream) files on each node involved in a crashdump. The stream files are in compressed binary FastLoad format. Stream files can either be transmitted directly to the Teradata Support Center or moved onto another system at the customer site, where debugging can occur. The `cspstreamend` program loads the dump data in the stream files into a table in the Crashdumps database for analysis.

DBAs can save crashdump files all at once or a few nodes at a time. Both SMP and MPP platforms can save crashdump stream files. Saving to stream files has the following benefits:

- You do not need to save to or export from the Crashdumps database on the production system. This saves time.
 - Stream files enable you to move a dump to an on-premises test system for debugging. This frees the production system for other tasks.
1. Decide if you want a crashdump to be saved to stream files by default or if you want to specify the save destination for each crashdump.
If you want to make stream files the default, do the following:
 - a. Create a custom resource file in `/etc/opt/teradata/tdconfig/defaults/namedCsp.custom` and add this line to the file: `Target:stream`.
 - b. Issue a `tpareset` command.

Subsequent `csp -mode save` commands automatically save to stream files. The customized resource file remains after an upgrade.

- When a crashdump occurs, save the raw dump files as stream files.
For example, the following command tells CSP to save to stream files:

```
> csp -mode save -target stream -force
Dump 2017/05/12-05:22:30-06 is about to be saved
Are you sure? [y,n,q,?] > y
```

- Collect the stream file from each node into a single directory.
In the following example, the DBA collects the stream dump files named `CRASH_XXX` into the directory named `/tmp/dump`:

```
pcl -collect /var/opt/teradata/tddump/pdedumps/stream/
CRASH_20170512_052230_06 /tmp/dump
All 3 node(s) have connected
    byn001-3:1023: collect completed: 234921696 bytes sent (1
files/0 directories)
    byn001-2:1023: collect completed: 257842344 bytes sent (1
files/0 directories)
    byn001-1:1022: collect completed: 308514708 bytes sent (1
files/0 directories)
```

- Decide where to send the crashdump stream files next:
Choose one of the following:

- The Teradata Support Center
- A test or QA system

If you want to load the stream files into the Crashdumps database on a test system, run the `cspstreamend` program to merge the stream files and load them onto the test system.
In the following example, the DBA loads the data in the stream files from `/tmp/dump` into the Crashdumps database table named `demodump` on the test system named `<Harold>`.

```
> cspstreamend -path /tmp/dump -dumpname demodump -system <Harold>
```

An alternative method of loading the stream files onto a test server is to copy the stream files to a directory on the test server. You can then run `cspstreamend` on the test server to load the dump to the Crashdumps database on the test server.

Handling an Interrupted Dump Save Operation

If a dump save in progress is interrupted, for example the DBS restarts, the dump save is not resumed automatically and the dump does not appear with the `csp -mode list` command. If a dump save was interrupted, check the messages log to get the name of the crashdump and then use `csp -force` option to save the crashdump.

If a dump save is run while a node is out of the configuration, the crashdump is saved from the nodes that are currently available but not from the node that is out of the configuration. No error displays if this occurs.

The resulting saved crashdump is eventually a usable crashdump. You can run the following command on the saved crashdump to verify that it is usable:

```
#csppeek -i -d crashdumpname
```

where crashdumpname is usually "Crash_yyyymmdd_hhmmss_nn".

However, the final crashdump does not include the piece of the crashdump from the down node. In other words, the final crashdump will be missing the crashdump information from the down node that is not part of the configuration.

Error Logging

Error messages related to saving Teradata crashdumps are logged in /var/log/messages. For more information, see [Viewing Teradata Crashdump Messages](#) and [Troubleshooting Crashdumps](#).

Saving Crashdumps to Disk

If you cannot use FTP services to send your crashdumps to Teradata and must save your crashdumps to disk, use the DUL utility. DUL is a separate package that you must order and install individually. For more information on DUL, see *Teradata Vantage™ - Database Utilities*, B035-1102.

You can unload crashdump information to several sequential files, which are:

- Rows of data dumped from the AMPs
- Internal load module correspondence information
- Applicable rows from Software_Event_LogV

Mainframe systems have only one copy of each of these files. System with multiple nodes have one copy for each node involved in the crashdump.

By default, crashdump files are named Crash_yyyymmdd_hhmmss_nn where:

Variable	Definition
yyyymmdd	Year, month, and day of the crashdump
hhmmss	Time (hour, minute, and second) of the crashdump
nn	Sequential number associated with the crashdump

You can change the file name using BTEQ, but remember that the timestamps are important. For more information on DUL, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Sending Crashdumps to Teradata

To use FTP services to send crashdumps directly to Teradata for analysis, contact your Teradata Support personnel for more information.

You can also access the latest instructions from knowledge articles, including the latest file paths and server addresses. Topics include how to offload and upload database crashdumps to Teradata. To search for crashdumps:

1. Go to <https://support.teradata.com>.
2. Log in.

Debugging Crashdumps

Teradata Support may need to debug crashdumps. Teradata Support can use the system debugger to access the raw dumps *before* they are saved to the Crashdump table (even if the PDE and DBS are not up and running). This saves time and helps Teradata personnel identify problems more quickly.

If someone from Teradata Support needs to immediately debug a problem by looking at a raw dump rather than wait for the system to save it to the Crashdumps table, they can use the raw flavor commands of the system debugger.

If your assistance is required, a Teradata Support representative may direct you to use the commands listed in the following table.

Command	Description
set crash-database	Sets the Teradata system to the one on which you would like to view saved dumps. Note: This does not set the location for raw dumps.
info rawdumps	Displays a list of dumps in the raw dump directory of the node.
info dbmdumps	Displays a list of dumps in the dbm dump repository of the node.
info dbsdumps or info dumps	Displays a list of the dumps on the node. Note that the “set crash-database” command may have been used to point to a crashdumps database on a different machine than the one on which the debugger utility is running. The info dbsdumps command is only available with the DBS flavor of dumps.
attach rawdump <i>crashdumpname</i>	Attaches the named raw dump in the node's raw directory.
attach dbsdump <i>crashdumpname</i> or attach dump <i>crashdumpname</i>	Attaches the named DBS dump or crashdump in the Teradata DBS crashdumps database. For example, (gdb) attach dump Crash_20040704_160628_01

Attaching a Raw Dump and Displaying the Backtrace

First type the `csp -mode list` command to allow CSP to display what dumps are available.

```
foosystem:/var/opt/teradata/tddump # csp -mode list
csp: Searching for dumps in raw dump directory /var/opt/teradata/tddump
csp: 2 dumps found, 2 dumps to process
csp: Sel  ID (date-time-token)      Nodes  Event  Instigator  Status
csp: ---  -
csp: *    2008/09/16-14:00:40-05      1    10416  1/11/(20096)
csp: *    2008/09/16-13:53:07-04      1    10416  0/11/(14975)
```

Then run the Teradata system debugger utility (`/usr/pde/bin/gdb`) to start up a Coroner session to debug one of the raw dumps.

Note:

It is not recommended to debug a raw dump if csp is currently saving dumps.

```
foosystem:/var/opt/teradata/tddump # gdb
GNU gdb 6.0/13.10.00.00
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to
change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "x8664-suselinux-teradata".
(gdb) i raw
//:
2008/09/16-14:00:40-05
2008/09/16-13:53:07-04
(gdb) at raw 2008/09/16-14:00:40-05
Attaching to crash dump
//: cdbopen: *** Processing data from raw dump, please wait... ***
//: Unloading files to: /var/opt/teradata/tdtemp/crn-daeqV1
//: Unloading file libm.so.6
//: Unloading file libcliv2.so
//: Unloading file libdl.so.2
//: Unloading file libelf.so.0
//: Unloading file libgcc_s.so.1
//: Unloading file libpthread.so.0
//: Unloading file libnsl.so.1
```



```

//: Unloading file libc.so.6
//: Unloading file libtdusr.so
//: Unloading file libthread_db.so.1
//: Unloading file ld-linux-x86-64.so.2
//: Unloading file libnss_files.so.2
//: Unloading file emffile
//: Unloading file emffile.idx
//: Unloading file libstdc++.so.6
//: Unloading file libaio.so.1
//: Unloading file librt.so.1
//: Unloading file libbz2.so.1
//: Unloading file libz.so.1
//: Unloading file libcrypt.so.1
//: Unloading file libresolv.so.2
//: Unloading file libgssapi_krb5.so
//: Unloading file libkrb5.so.3
//: Unloading file libk5crypto.so.3
//: Unloading file libcom_err.so.2
//: Unloading file libkrb5support.so.0
  *** Selecting Aborting task(s), if any
Software Versions:
PDE      = 13.10.00.00
TDBMS    = 13.10.00.00
PDEGPL   = 13.10.00.00
TGTW     = 13.10.00.00
TDGSS    = 13.10.00.00
Reading symbols for task actmain...Using host libthread_db library "/lib64/
libthread_db.so.1".

('libdb1.so', 2aaaaabc7000) ('libummm1.so', 2aaaaaee9000)
('libevl1.so', 2aaaaafec000) ('libfil1.so', 2aaaab105000)
('libudf.so', 2aaaab455000) ('libudt.so', 2aaaab558000)
('libuci.so', 2aaaab6c3000) ('libamp1.so', 2aaaab7cf000)
('libamp4.so', 2aaaabbbf3000) ('libstp.so', 2aaaabf02000)
('libpdt.so', 2aaaac28d000) ('libtvssam.so', 2aaaac3c0000)
('libjil.so', 2aaaac50a000) ('libnetpde.so', 2aaaac623000)
('libpde.so', 2aaaac75d000) ('libemf.so', 2aaaac8b1000)
('libpdesym.so', 2aaaac9dd000) ('libbz2.so.1', 2aaaacb06000)
('libpthread.so.0', 2aaaacc17000) ('libelf.so.0', 2aaaacd2e000)
('libnsl.so.1', 2aaaace44000) ('libalgapi.so', 2aaaacf5b000)
('libgeos.so.2', 2aaaad0d4000) ('libgeos_c.so.1', 2aaaad3e0000)
('libproj.so.0', 2aaaad4f6000) ('libgdal.so.1', 2aaaad651000)
('libogrx.so', 2aaaadd1d000) ('libm.so.6', 2aaaade4e000)
('libcrypt.so.1', 2aaaadfa3000) ('libc.so.6', 2aaaae0dc000)

```



```

('libstdc++.so.6', 2aaaae30e000) ('libgcc_s.so.1', 2aaaae50c000)
('libaio.so.1', 2aaaae61a000) ('libdl.so.2', 2aaaae71b000)
('libthread_db.so.1', 2aaaae81f000) ('ld-linux-x86-64.so.2', 2aaaaaab000)
('libz.so.1', 2aaaae928000) ('librt.so.1', 2aaaaea3c000) done.
1//20871: Event number 33-10416-00 (severity 50, category 10), occurred on
1//20871: Tue Sep 16 14:00:40 2008, at 001-01 (Vproc 1, partition 11, task
1//20871: 20871) in system foosystem in Module , version
1//20871: PDE:13.10.00.00,TDBMS:13.10.00.00,PDEGPL:13.10.00.00,TGTW:13.10.
1//20871: 00.00,TDGSS:13.10.00.00
1//20871: Task aborted abnormally.
1//20871:
1//20871:
1//20871:
[Switching to 1//20871]
1//20871: ---Type <return> to continue, or q <return> to quit---
0x00002aaaae16dec1 in __nanosleep_nocancel ()
    from /var/opt/teradata/tdtemp/crn-daeqV1/libc.so.6
(gdb) bt
#0 0x00002aaaae16dec1 in __nanosleep_nocancel ()
    from /var/opt/teradata/tdtemp/crn-daeqV1/libc.so.6
#1 0x00002aaaae16ddae in sleep ()
    from /var/opt/teradata/tdtemp/crn-daeqV1/libc.so.6
#2 0x00002aaaac77368e in tsksuspend (utcbp=0x2aaaaebd7d80)
    at tsk/tskport.c:404
#3 0x00002aaaac794407 in logeventreset (logmsg_p=0x2aab1441a260, resetflags=0)
    at log/logusrlib.c:959
#4 0x00002aaaabf6e1b2 in awtcheck (MessagePtr=0x2aab1441a260)
    at ../../amp/awt/awtmain.c:1244
#5 0x00002aaaac794e19 in logevent_com (eventcode=10416,
    location=<value optimized out>, length=<value optimized out>,
    dumpkind=0 '\0', resetflags=0) at log/logusrlib.c:658
#6 0x00002aaaac78a315 in prgfaultcatch (signal=339177112,
    siginfo=0x2aab12fe1000, contextp=<value optimized out>)
    at prg/prgport.c:2268
#7 <signal handler called>
#8 evlfaultgen (a=1) at ../../par/evl/evllib.c:892
#9 0x00002aab144bb678 in ?? ()
#10 0x00002aaaaac4f20f in evlinterp (cm=0x2aab14377f30, context=0x2aab143acdf8)
    at ../../par/evl/evlinterp.c:2033
#11 0x00002aaaaac7784c in evlbuild (EvlCtxt=0x2aab143acdf8,
    OutRowLoc=46914267187568, OutRowFld5=
    {fldptr = {loc = 46914268279680, ptr = 0x2aab144bb380}, statusptr =
    0x2aab144bb380, ptr = 0x2aab144bb380, loc =
    46914268279680}, Sequent=0x2aab143ad0e8,

```



```

DumRowFlag=<value optimized out>) at ../../par/evl/evlbuild.c:222
#12 0x00002aaaabfc05a8 in WalkTbl (LINK=0x2aab14378e10)
    at ../../amp/stp/stpret.c:2108
#13 0x00002aaaabfc55d4 in stpret () at ../../amp/stp/stpret.c:3613
#14 0x00002aaaac10427e in awtstphd () at ../../amp/awt/awtstphd.c:264
#15 0x00002aaaabf7106c in awtmain (argc=296775680, argv=0xabf6e615)
    at ../../amp/awt/awtmain.c:2844
#16 0x00002aaaac77404a in tsknewthread (threadargp=<value optimized out>)
    at tsk/tskport.c:1185
#17 0x00002aaaacc1d143 in start_thread ()
    from /var/opt/teradata/tdtemp/crn-daeqV1/libpthread.so.0
#18 0x00002aaaae19b74d in clone ()
    from /var/opt/teradata/tdtemp/crn-daeqV1/libc.so.6
(gdb)

```

Deleting Teradata Crashdumps

Deleting unwanted Teradata crashdumps from your disk and archiving crashdumps you want to keep may help free up space. It may help prevent crashdumps from using up the space on your system and causing problems.

First determine how many crashdumps you want and how long you want to keep the crashdump. Depending on the size of your system and the size of the crashdumps themselves, you can consider keeping some of the crashdumps as historical records and delete them later when they are no longer necessary.

If you are having space problems due to crashdumps, consider limiting the “Max Dumps” parameter in `ctl` to a smaller number of crashdumps which will help prevent the crashdump area from filling up. If you are unsure about what to do, contact Teradata Support.

If you decide to delete all the crashdump tables in the Crashdumps database, submit:

```
DELETE DATABASE CRASHDUMPS;
```

You may also specify a single crashdump to be deleted using:

```
DROP TABLE CRASHDUMPS.tablename;
```

The following procedure describes how to use the CSP utility to delete crashdumps from the raw dump directory.

Note:

For more information on CSP commands, type `csp -h` at a Teradata Command Prompt or see the resource file located at `/usr/pde/lib/defaults/Libcsp`.

1. List the available raw PDE crashdumps.

Use the following command from the Teradata Command Prompt:

```
csp -mode list -source dump
```

The `-source` parameter reads the raw crashdumps from the PDE dump directory. The dump directory is also the default source, so it is equivalent to the command `csp -mode list` (without the `-source dump`).

```
csp -mode list -source -table
```

Use the following command from the Teradata Command Prompt:

```
csp -mode list -source dump
```

The `-source` parameter reads the raw crashdumps from the PDE dump device. The dump device is also the default source, so it is equivalent to the command `csp -mode list` (without the `-source dump`).

```
csp -mode list -source -table
```

For more options and syntax, type `csp -help` from the Teradata Command Prompt.

For CSP, the `-nodes <nodelist>` option allows you to choose specific nodes from which to list crashdumps. Use the `-nodes` option to target specific nodes for save, list, or clear operations.

2. Delete (clear) unwanted raw PDE crashdumps.

Use a command similar to the following:

```
csp -mode clear 2008/12/06-14:19:34-02
```

The raw PDE crashdumps cleared are those selected with an asterisk in the “Sel” field.



NOTICE

If you do not type a name, you will delete all crashdumps. However, you will be prompted to answer ‘y’, ‘n’, or ‘q’ for each crashdump to be cleared.

Note:

The `csp` default setting for `-source` is `dump` (the PDE dump directory). The “`csp -mode clear`” command is run from one node but deletes the raw crashdumps on all accessible Teradata nodes.

Fault Isolation Diagnostics

Fault isolation features capture information specific to the fault without requiring a reset. These failure diagnostics facilitate early analysis of problems before the associated dump is available for examination.

Content and abort handling depends on whether the fault being isolated occurred in the parser, an AMP, or Dispatcher, as described in the following table.

Type of Fault Isolation	Description
Dispatcher	<p>The Dispatcher prequalifies a fault for fault isolation.</p> <p>If the fault qualifies for fault isolation, the Dispatcher captures diagnostic data and then aborts the transaction. If the transaction abort completes successfully, then the system avoids a DBS reset.</p> <p>The diagnostic data consists of a snapshot dump, user session related data, stack backtrace, and the SQL request if it is available. (The last three items listed are captured in the system error log.)</p> <p>Note, however, that a DBS reset occurs if the fault fails to qualify for fault isolation, or if the system cannot complete the abort of the transaction successfully.</p>
Parser	<p>The parser issues a transaction abort and the Snapshot Dump facility captures the cause of the problem.</p> <p>The data capture includes the dispatcher worker task containing the query text.</p>
AMP (Spool Table)	<p>Teradata attempts to recover from the spool error without a reset.</p> <p>Typically, such an abort is accompanied by an optional Snapshot Dump to help Teradata Support analyze the reason for the AMP failure.</p> <p>Aborts generated by the AMP Fault Isolation feature:</p> <ul style="list-style-type: none"> • Return a detailed text description of the AMP failure to the associated client session. • Log diagnostic details of the failure to the system error log. AMP Fault Isolation also performs this error-logging when the system reset cannot be avoided. <p>Note:</p> <p>If you need to tune AMP Fault Isolation, contact Teradata Support. They can help modify the settings to:</p> <ul style="list-style-type: none"> • Reduce the cases for which the aborts are enabled. • Disable the feature entirely to force a Teradata reset and a dump on the next occurrence of the failure.

Troubleshooting Crashdumps

The following section is intended for Teradata Support personnel and lists some common questions regarding how to troubleshoot crashdumps.

1. What are the most common errors encountered while handling crashdumps and why do these occur?

Question	Answer
What does an error 2644 indicate?	<p>DBC.Crashdumps database is full.</p> <p>Delete unwanted tables to free up more space or add additional space to the Crashdumps database.</p>

Question	Answer
What does an error “Filesystem full” indicate?	<p>The raw PDE crashdump directory is full.</p> <p>You should delete unwanted crashdumps from the dump directory using the clear option of the CSP utility. For more information, submit:</p> <pre>csp -h</pre> <p>at a Teradata command prompt.</p>
What does the error “Control GDO maxdumps exceeded; no more dumps will be captured” indicate?	This error occurs when the maxdump parameter has been reached and another crashdump occurs.
I notice a crashdump with the following name: Crash_yyyymmddhhmmss_nn_Part What does this mean?	<p>This is a partial crashdump. If a crashdump is not saved to the Crashdumps database yet, it is considered a partial crashdump. You should simply wait and give the system more time to process the crashdump. If you have waited but are not sure the crashdump is okay to use, run the csppeek utility on the saved crashdump.</p> <p>If a crashdump is not saved completely in the dump directory, the crashdump is considered an incomplete crashdump.</p>

You can find the crashdump error messages in the log for your operating system (see [Viewing Teradata Crashdump Messages](#).)

2. What does it mean when there are “.Err” files present?

If you submit the following command in BTEQ:

```
HELP DATABASE CRASHDUMPS;
```

you might see .Err files, for example .Err1 and .Err2 files.

This may be normal if the system is *still* in the process of saving a crashdump. These error files will be cleared when the save completes. However, if the logs show that CSP did not successfully save the crashdump and the .Err files are left over (that is, they remain uncleared), this indicates that something went wrong while saving the dump.

3. What should I do if I am unable to save crashdumps?

If a CSP save failed, always check the log first. Then check the following items:

Item To Check	Description
Are your host entries properly configured? Is the Host Group ID defined on all of the hosts?	<p>The COP entries are needed in the hosts file for a client to login to the database. The copname must be configured in the /etc/hosts file.</p> <p>In addition to the hostname entries, you must also define the hostname COP alias on each node for all the crashdumps tools accessing crashdumps. Set up is normally done during staging or configuration of client utilities. Consider the following for setting up COP entries:</p> <ul style="list-style-type: none"> The cop alias entries should follow standard Teradata Network CLI protocols.

Item To Check	Description
	<ul style="list-style-type: none"> One of the cop entries in the file must be dbccop1. This name is used as a default for CLI. If you do not have dbccop1 in the hosts file then you get the following error: <pre>MOSI: ER_HOST (117): DBC name not found - possible HOSTS file problem. *** ERROR 204: Can't Init Things. Exiting!</pre> The COP entries must be sequential or CLI will stop looking for COP entries in the first COP entry that does not return successfully. For example, if dbccop1, dbccop2, dbccop3, dbccop5 are entries in the hosts file, CLI will not use dbccop5 to connect to the server, because dbccop4 is missing. If a system can connect to four COPs, the COP entries in the hosts file should look as follows: <pre>[IP address of NODE1] COPNAMEcop1 [IP address of NODE2] COPNAMEcop2 [IP address of NODE3] COPNAMEcop3 [IP address of NODE4] COPNAMEcop4</pre> You can have multiple COP entries per node as long as the COPNAME is different. For example: <pre>39.64.8.9 smp001-4 dbccop1 hotwmacop1 wmahotcop1 39.64.8.10 smp001-5 dbccop2 hotwmacop2 wmahotcop2 39.64.8.11 smp001-6 dbccop3 hotwmacop3 wmahotcop3 39.64.8.12 smp001-7 dbccop4 hotwmacop4 wmahotcop4</pre>
Have you made any tuning changes to throttle the number of FastLoad sessions running at any given time?	If a user FastLoad is running and crashdumps attempts to log on its session while using a PE-based host group, there may not be enough sessions available.
Does the Crashdumps database have enough space to save the dumps?	See Configuring the Crashdumps Database for information on how to check how much space Crashdumps has and how to allocate more space. You may also need to delete unwanted crashdumps to make more room.
Is CSP unable to allocate more than the minimum number of sessions per node in the specified host group?	You cannot save crashdumps in this situation. Find a different time window when the system is more quiescent.

4. What if CSP runs out of sessions?

To save a crashdump, the system must be able to log on a minimum number of sessions as defined in the resource file for CSP. The default is 4 sessions per node. There is also a maximum number of internal sessions set on the system per PE.

If you notice that CSP runs out of sessions, run CSP at a different time when the system is more quiescent.

5. How can I tell if a crashdump is okay to use?

- Use the CSP utility to check the raw PDE crashdumps with the following command:

```
csp -mode list -source dump
```

The output will appear as follows:

```
# csp -mode list
csp: Searching for dumps in raw dump directory /var/opt/teradata/tddump
csp: 4 dumps found, 4 dumps to process
csp: Sel ID (date-time-token) Nodes Event Instigator Status
csp: ---
csp: * 2008/09/09-17:03:13-02 1 10196 16384/0/(11480)
csp: dump_20080909_180130 1 16384 Required files are
missing or corrupted
csp: 2008/08/14-11:35:58-01 1 10196 16384/0/(15160)
csp: * 2008/07/11-14:41:58-01 1 10416 1/11/(28272)
csp: * 2008/07/01-11:58:42-05 1 10196 16384/0/(6418)
#
```

If the Status column reports “Required files are missing or corrupted” or “Dump belongs to a different PDE version,” the crashdump should *not* be used. If the Status column is blank, however, this means that the crashdump is okay to use.

- Use the csppeek utility by submitting the following command at a Teradata command prompt for a saved crashdump:

```
csppeek -i -d crashdumpname
```

where crashdumpname is in the form of Crash_yyyymmdd_hhmmss_nn.

If csppeek reports information about the crashdump as follows:

```
Dump (version 20) contains 1 node, 10 vprocs, 35 pids, 839 tids, 0 memos
nodeids: 16384(I)

Software versions:
PDE      13.10.00.00
TDBMS    13.10.00.00
TGTW     13.10.00.00
TCHN     13.10.00.00
TDGSS    13.10.00.00

NODE 16384 is system noun
Dump token: Crash_080911_080707_01 instigator: 33//3268
Caused by logevent: 10196-60 reported by /32382720 (3268/16384/0)
```



```
node 16384 contains vprocs 16384(N)
16383(P)
16382(P)
10238(V)
10237(V)
8192(G)
3(A)
2(A)
1(A)
0(A)
```

This means the crashdump is okay to use. If it does not report anything or reports an error, this indicates a problem with the crashdump because it could not read the crashdump information and you should *not* use the crashdump.

6. How do I get CSP to resume automatically saving crashdumps when it reports a 2644 failure (no more space in crashdumps database)?

When CSP reports this failure, CSP and cspslave will go to sleep until they are awakened by a utility named cspwake or by a tpareset. After you add more space to the Crashdumps database, you can run cspwake at a command prompt without any arguments to resume automatic dump saving. If automatic dump saving was not being done, then CSP would have to be manually started with the `–force` option instead.

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	Helpful resources in one place: <ul style="list-style-type: none">• Support requests• Account management and software downloads• Knowledge base, community, and support policies• Product documentation• Learning resources, including Teradata University
https://www.teradata.com/University/Overview	Teradata education network
https://support.teradata.com/community	Link to Teradata community